Long Short-Tunes Memory

Peter Schaldenbrand
Human-Computer Interaction Institute
Carnegie Mellon University
Pittsburgh, PA 15213
pschalde@andrew.cmu.edu

Yizhou He
Physics Department
Carnegie Mellon University
Pittsburgh, PA 15213
yhe2@andrew.cmu.edu

Husni Almoubayyed
Physics Department
Carnegie Mellon University
Pittsburgh, PA 15213
halmouba@andrew.cmu.edu

Abstract

Difficulties in learning music include the high dimensionality, poor evaluation methods, and lack of quantity and quality of data. Because of these difficulties, prior researchers would place many assumptions and constraints into their models to generate music. We generated music using a Long Short-Term Memory (LSTM) network that learned from MIDI files. We collected data from several sources that were diverse in terms of genre, tempo, key, and instrumentation, totaling over 100,000 midi files. We presented a novel custom loss function incorporating ideas from music theory. Despite the variety of data we used, our machine learning model was able to learn key signature, dynamics, tempo/rhythm, and recurrent structure of music. The generated music has a groove and sounds like music, however, it often plays notes off-scale, which could be considered not ideal. Our model could be further extended in the future to address these issues by placing constraints on the outputs or varying the look-back length.

1 Introduction

Music is not currently a common area of interest in the machine learning community. While much attention is given to text, speech, and images, music remains a domain that lacks the focus of many researchers. Among the many reasons for this neglect is the difficulty of learning from musical data. For instance, a piano has 88 keys which means that there can be up to 2^{88} different combinations of keys that an artist could strike (assuming only one note is played at a time). Across most representations, the dimensionality of musical data can be massive. Music, when it is stored in raw form such as .wav or .mp3 files, is often sampled at 48 kilohertz and it is common that each sample is 16 bits in size. Given this rate of at which music is stored, standard song lengths are explosive in data size when looking at raw music. MIDI is a communication protocol that specifies the pitch of a musical note, the velocity at which it is played, the instrument playing it, the tempo of the music, and a few other details. This makes a much more compact representation of music than raw data. But even so, the dimensionality of MIDI data can be very large with the possible number of combinations of notes, durations, and other features.

Another difficulty of using machine learning algorithms on music data is the number of samples to learn from. While raw data can be found in abundance, MIDI files and similar compact representations of music can be difficult to find in large quantities. [10] scraped the web for MIDI data and found

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

around 1000 classical pieces. Finding diverse genres of MIDI files is tedious and can often lead to finding poor quality data. Unlike raw data, MIDI files are not often created by professionals, and so the size and value of these files is often found to be poor.

The use of discriminatory machine learning algorithms on music has been of value to researchers and industry in recent years. Genre analysis [22], determining the name of a song given audio [23], and music recommendation algorithms [8] are all well studied problems in machine learning. Rather than using deterministic algorithms, in this paper we focus on generating music by learning from MIDI files.

Given the size and complexity of music data, we found it necessary to use deep learning algorithms to model the intricacies of music and its structure. We also noted that a model that was too complex could produce music that was too fine tuned to our training data, and therefore generate uninteresting results or just replay training data.

A more recent development in deep learning is the Generative Adversarial Network [15]. This algorithm is highly effective at generating data [16]. [14] created MuseGAN which generates music using a GAN. They used piano roll music which is similar in representation to MIDI, but the model can only generate four bars of music at a time and cannot connect multiple segments of four bars together aesthetically. MidiNet is similar in that it uses a GAN to generate music, but it also uses data from previous bars of music to generate new music, thereby being able to produce infinitely long sequences of music that should sound connected. MuseGan and MidiNet show promise in the generation of music, but fall short in producing believably human music.

While GANs can only produce a fixed output of generated music, Recurrent Neural Networks [17] are designed for time series data and can produce streams of data by predicting the next sound given a history of sounds in a song. Long Short-Term Memory (LSTM) [18] units can make an RNN more powerful by using an even longer and finer tuned history of the song to generate new segments of the song. In this paper, we will use an LSTM network to generate music since one of our goals is to produce a continuous stream of music.

The largest music generation projects is Magenta by Google [5]. Magenta uses LSTMs to generate music, and provides programs to extract melodies from MIDI files. Magenta uses an assumption that the music it is trained on and generating is from one source and instrument. The generated music is often not extremely interesting. We will try to use Magenta as a starting point and attempt to build a system with less assumptions about music and generate music that sounds more like a human produced it. We will build our model using a high-level neural networks API, Keras. Keras is a python module that is capable of running on top of TensorFLow, CNTK, and Theano. Although Keras is a high-level tool, it provides the ability to make fine grain changes to the machine learning model.

Often, one of the difficulties of learning music is the lack of large datasets. We pulled MIDI files from many different sources for our training data, and we ended up with a large dataset of around 100,000 MIDI songs. One of our goals was to not give the model an assumption on the genre of music to be generated, so we gathered a diverse dataset of music ranging from electronic dance music to classical.

Our choice to use MIDI files as the type of our training data was not random. Some of the latest research in generating music is using raw files to train very complex models. WaveNet [3] use a stacked convolutional layers without max pooling to create a very complex model that can generate audio. It works exceptionally well for text to speech. They have also trained this model on 200 hours of piano music. WaveNet was able to produce believably piano-created music, and given the complexity and size of raw music this is very impressive. However, the particular music that the model generated did not have interesting variations. Raw music data seems to be too complex for current machine learning algorithms to learn the nuances and beauty of the music. On the other hand is MIDI data which provides a much simpler representation of music. Even though this representation is simpler than raw music data, prior works such as [13], [11] and [12] use many assumptions about music to constrain their models on features such as tempo, genre, key, and rhythm. We would like to

Table 1: Attributes of music and their descriptions.

Aspect of Music	Description
Key Signature/Notes	The series of sharp or flat notes that a song should abide by.
Dynamics	How loud or quiet the music is. The differing intensities of the notes within a song.
Time Signature	An indication of rhythm that defines the beat as a division of a whole note giving the number of beats in each bar.
Recurrent Structure	Recurring segments of melodies and rhythms within a song.
Tempo/Rhythm	Speed of which the song is played and the divisions of time in which the notes are played.
Chords	Playing multiple notes at the same time. Often in sets of three or four.

meet in the middle of these two ends of the musical machine learning spectrum and produce a model that uses a simple representation of data such as MIDI files with very few assumptions about the data.

A fundamental difficulty with music generation stems from a philosophical question: what is music? Our model is trained on data that has been, by at least one human's standards, met the qualities of being called music. It is possible that what is musical to one person is not music to another and so our model might be trying to learn multiple notions of what music is. The previous works that have had good results with generating music have put many assumptions about music into the model, and the work that has put few assumptions into the model has not produced great music. Our goal is to have our machine learning model learn common aspects of music without assuming them. We define these commonalities of music as: key signature/notes, dynamics, recurrent structure, tempo/rhythm, and chords.

1.1 Related work

Early work in music generation algorithms predominantly put large assumptions into the models. To account for musics large dimensionality and lack of large, high quality datasets, researchers needed to fine tune their systems so they could learn the structure of melodies and rhythms easier. [11] take several commonalities of music and use them to their advantage to train their model. They only use blues music, the chords did not vary per song, and no rests were predicted, only quarter, half, and whole notes. Another paper that constrained its model to only be trained on one genre of data was presented in [12]. Their model was trained on a data set consisting only of MIDI files that were performances of Johann Sebastian Bach's music. They state that two issues with this model were that the model did not separate melody from accompaniment, and that it did not predict when notes ended, only when they began.

There were two evaluations scores used in [12]; accuracy and F1. They point out that these evaluation functions do not necessarily correspond to human perception of music similarity. This inspired us to write our own loss function that might be able to capture a humans' perception of correctness of predicted pitch or note. Another paper that confirmed the usefulness of a unique loss function for the nature of musical data tried to predict melodies and used rules to determine how successful the generated melodies were. [19] used evolutionary algorithms to find a neural network that maximized the chance of generating good melodies. To compare melodies, they used composition rules on tonality and rhythm including rules inferred from Béla Bartók's music . [19] views music as a language and tries to represent specific melodies.

While most of music generation algorithms focus on learning a prediction function directly, one of the prior works that we came across makes an attempt to model the distribution of data. [9] use a restricted Boltzmann machine (RBM) to model the distribution of notes at each timestamp in music. On top of the RBM, an RNN is used to generate actual music. One particular shortcoming of this method was that the generated music was not able to capture longer-term musical structure.

A very new and powerful generative model is the GAN. MuseGAN and MidiNet both use this model on compact representations of music: piano roll and MIDI, respectively. A particular shortcoming of a GAN with respect to time-series data is that the output is of a fixed size. MuseGAN only generates four bars of music at a time and cannot make multiple of these outputs sound as if they

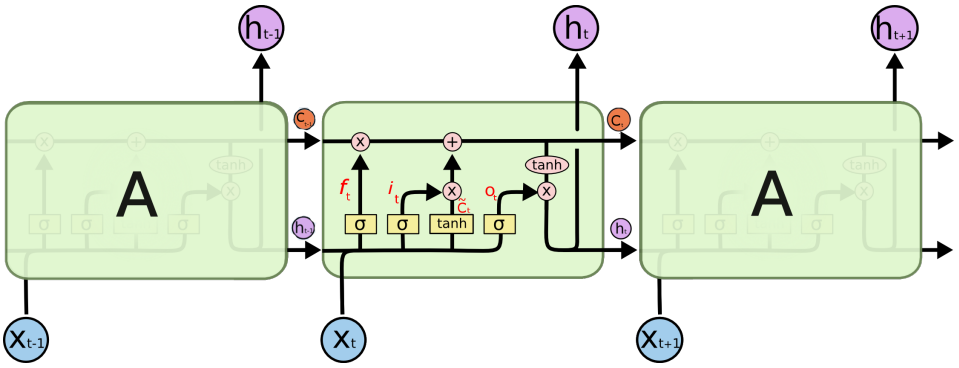

Figure 1: A representation of a general LSTM layer. Previous cell state C_{t-1} and output h_{t-1} will be passed to the next cell, together with new input x_t through forget gate f_t, input gate i_t and output gate o_t to update current cell state and new output. Adapted from an online article by Christopher Olah/Google Brain [26]

were connected. MidiNet uses some data from a previous output of the GAN to generate a new fixed size output that sounds related to the previous. The studies report that the generated music does not sound human-generated.

Magenta [7] is a tool for performing music generation. It provides functions to load and process MIDI files for training an LSTM by extracting the melody. Generated music from this tool has proven to that it can generate music that is similar to music produced by humans. Magenta also only works with one instrument at a time. We would like to generate better sounding music in our paper than the previous works that had used Magenta.

The most similar previous work that we found to ours is [10], who use a two layer LSTM to generate music. Their goals were to find a meaningful representation of music as a vector and to use a neural network to express the notions of harmony and melody. They also use MIDI files, however they normalized the ticks per beat in their data. The LSTM used cross-entropy loss and was only trained on music composed by Bach. They found that their model could not learn the "on-off" structure of MIDI files and therefore generated music with many rests or unappealing long notes.

2 Methods

2.1 Baseline Model

In order to determine the difficulty of predicting notes in a song, we created a baseline prediction model using logistic regression. Using the prior 10 notes in a song, could a model learn to predict the next note in the sequence? We used 20,000 songs, randomly selected from a diverse dataset of MIDI files for training our logistic regression classifier. We tested this classifier on 5000 different songs. The model predicted the correct note only 6.21% of the time. This displayed the difficulty in predicting notes in a song and signaled that a more complex model was necessary for this task.

2.2 Feature Vector & Scaling

Due to the large size and dimensionality of raw music data, we used MIDI file. MIDI structure is discussed in greater detail in section 3. These files were pre-processed into a feature vector that could learn the aspects of music we detail in table 1. We determined that there was a minimum of only three variables that were necessary to learn these aspects of music: note, velocity (how hard a note is struck), and the time between the current and previous note. Figure 2 shows an example of this pre-processing. The values in the feature vector were further pre-processed using a Standard Scaler in `scikit-learn`, which keeps the mean and variance to use it the inverse scaling afterwards on the predictions [24].

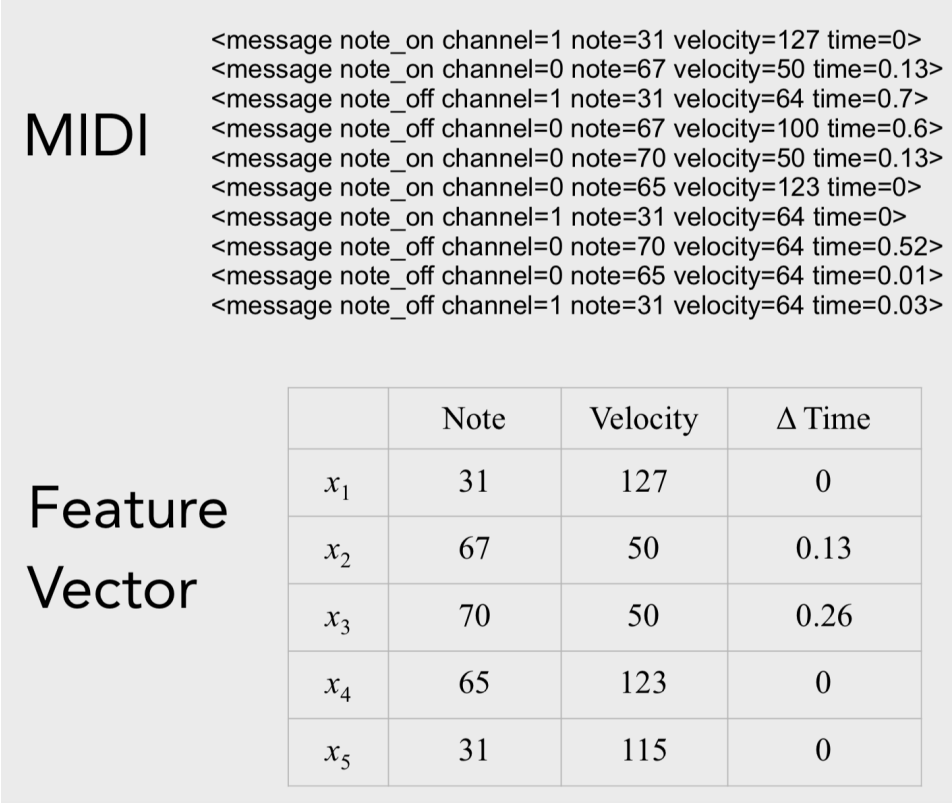

Figure 2: The structure of a typical MIDI file, consisting of a series of messages, indicating that a note should be turned on or off, which MIDI channel the message is broad-casted on, the note value (pitch), the velocity (how hard the note is played), and a time delta (how long the MIDI software should wait before executing the next message). The bottom part represents the feature vectors that we extract from the MIDI messages: we use note, velocity, and time deltas across channels.

2.3 Loss Function

Most typical loss functions used in machine learning reward close values and penalize far values from the truth. This is a meaningful approach to take with velocity and time deltas, and thus we used the mean squared error loss function for those two features. Musical notes, however, are usually on certain scales, and sound better when they stay on those scales. Usually, this means striking the true, 4th, or 7th note away from the true note will sound good, while others will not. This pattern also repeats every 12 notes (since this is a 12-base scale, the -5, and -8 would be good predictions, rather than -4 and -7.) Figure 3 shows the loss function which was created by fitting a high-order polynomial, using `scipy` [25], to a set of points that satisfy these conditions. The three losses were combined by adding them together after weighing the velocity and time delta losses as 20% of the amplitude of the note-loss, which was deemed more important.

2.4 Model Architecture, Training, and Validation

All recurrent neural networks have the form of a chain of repeating modules of neural networks. In standard RNNs, this repeating module will have a very simple structure, such as a single `tanh` layer. LSTMs are capable of learning long-term dependencies which also have this chain-like structure, but the repeating module has a different structure. A common LSTM architecture is composed of a memory cell, an input gate, an output gate and a forget gate.
Figure 2 provides a visual representation of an LSTM cell. In the figure and the following equations: x_t, h_t refer to input and output vectors of the LSTM unit, f_t, i_t, o_t refers to activation vectors of forget gate, input gate and output gate respectively, C_t saves unit state information, and W, b are the weight matrices and bias vectors.

Table 2: Network architecture as implemented in `keras`. All parameters are trainable. The output shape of the last (third) layer is due to the prediction (and input) containing three features.

Layer (type)	Output shape	Parameter number
lstm_1 (LSTM)	(None, 48)	9984
dense_1 (Dense)	(None, 24)	1176
dense_2 (Dense)	(None, 3)	75

As seen in figure 2, a state, C, is passed from cell to cell and provides a mechanism for the network to learn long-term relationships in the data. Some information, however, is not needed for long-term usage and so the forget gate decides what information is thrown away from the previous cell state. Weights, W_f, are learned through the back-propagation algorithm, with a stochastic gradient descent optimizer, to determine which information should be forgotten from the cell state.

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f) \tag{1}$$

Input gates decide what new information we're going to store in the cell state. Two temporary variables are used to calculate the new cell state, C_t. The variable i_t represents which values in the cell state we are going to update. $\hat{C}_t$ is a vector of candidate values for the new cell state.

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i) \tag{2}$$

$$\hat{C}_t = \sigma(W_C[h_{t-1}, x_t] + b_C) \tag{3}$$

We can then calculate the new cell state, C_t, by deleting the unimportant information, and adding the new information.

$$C_t = f_t C_{t-1} + i_t \hat{C}_t \tag{4}$$

In order to calculate the output of the cell, we first take the sigmoid value of the current input and previous cell's output multiplied by learned weights, W_o. Then we multiply this vector by the cell state to obtain the output, h_t.

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \tag{5}$$

$$h_t = o_t \tanh C_t \tag{6}$$

Our neural network is built with one LSTM layer, followed by two dense (fully-connected) layers, as detailed in table 2. The network has a look-back of 3, thus using three previous feature vectors to predict the next. This architecture is chosen for multiple tests on its ability to predict correct notes after being fed with a sequence of feature vectors. We adopt this relatively simple network because it reaches the best balance of accurate prediction and consumption of computational resources.

3 Dataset

We decided to compile our own dataset for use in this project. The data is MIDI files collected online from sources under licenses allowing free usage for research purposes. A MIDI data file comprise of an array of messages, each consisting of the channel number (0 to 15); the note value; the velocity, representing how hard a note was hit; whether the message indicates that the note is being turned on or off; and a time-delta, representing the amount of time (in units of ticks) since the last message on the same channel. MIDI files also include other metadata such as the tempo and time signature. Figure 2 shows a representation of a typical MIDI file.

The data we collected has reached over 100,000 songs, and has been pushed on to github, at `https://github.com/hsnee/DeepLearning4Music/tree/master/data`, mostly organized by genre. We use the python library `mido` to read in the MIDI files [15].

4 Results

We only used subsets of the datasets to train the network on, given the computational power necessary, and the large variations between different files in our dataset. Using 1000 songs that were

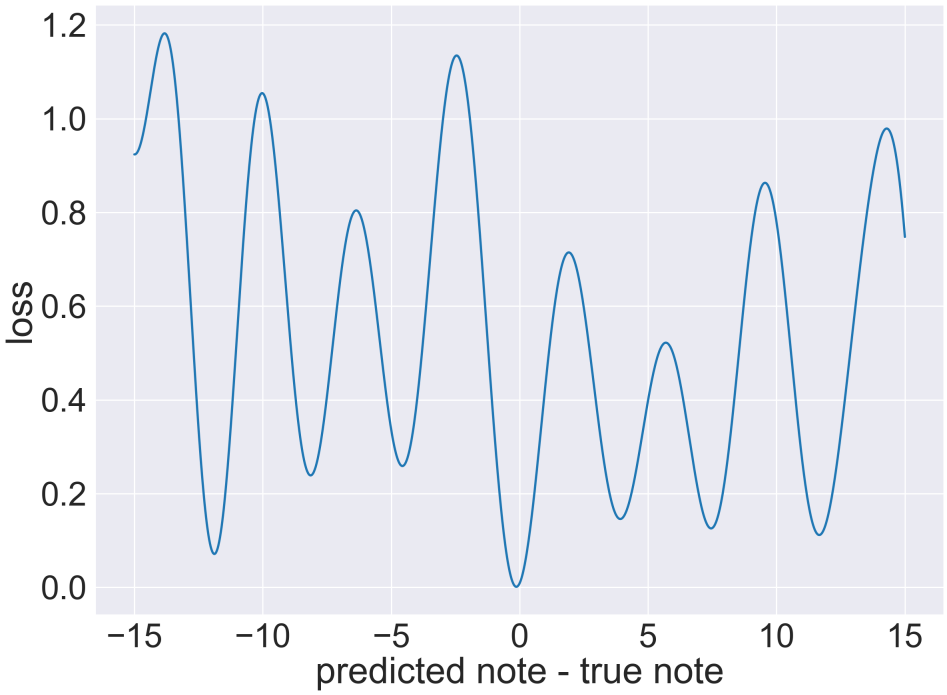

Figure 3: A plot of the loss function of the notes as a function of the difference between the predicted note and true note. In an effort to influence the predictions to stay on the same musical scale, the loss was manually chosen such that it has low values around 0, 4, 7. Since the musical scale has a base of 12, the loss values should repeat $\pm 12n$ for an integer n, with a small added loss to penalize predicting more than an octave away. All other values would be off-scale and were chosen to have significantly higher values. This was then fitted to a continuous and differentiable high-order polynomial.

mostly pop, we generated a few, long streams of music. Since our model generated music given a random seed, the initial parts of the music mostly sounded strange and not very much like music. However, after a few seconds, the songs started to show musical aspects.

Using the features in Table 1 as criteria, we evaluated our generated music to analyze what our model learned and summarized our results in Table 3. The first thing that one would notice about our generated music is that the note choices for the algorithm sound unlike other music. Especially considering the sound of our training data, the generated music sounds minor and does not seem to ever find a steady key or play notes in a single major scale. Figure 4 shows an excerpt of the generated music, and provides another representation of the outcomes.

Besides an optimal choice of musical notes, the algorithm was able to learn many aspects of music. The generated output often has a consistent beat and tempo. The music falls into a rhythm and is able to play recurring melodies and chords. These movements have recurring structure but evolve over time in the song.

5 Conclusion and Future Work

Our goal was to generate music using a machine learning model that could learn common aspects of music without us assuming them. Our results were positive in that we could product music that had dynamics, time signature, recurrent structure and chords, however, our model was not able to predict notes that were consistently on-key.

One feature that we have ignored learning in our model is the note duration, the importance of which is usually instrument-dependent. One enhancement of our model could be incorporating this feature since the duration of the note could help the model discern melody from chords and basslines.

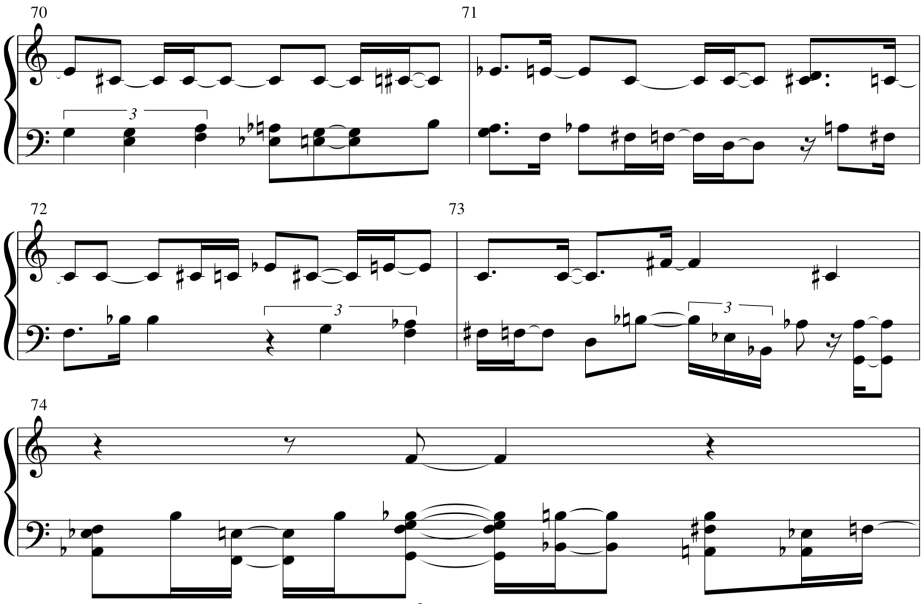

Figure 4: An excerpt of the music generated by the model. As can be seen from the high number of sharp and flat signs, the model struggles to predict values that are on the same scale. However, as can be seen from cell splittings, it seems comfortable with keeping a stable 4/4 time signature. As with most musical pieces, there usually are variations from one note to the next, but those variations do not fluctuate wildly. Finally, most of the bass parts have several (usually 3) notes playing at the same time, representing that the model has learned chords; while most of the treble parts are played in single notes, providing the song harmony, as is the convention in most musical pieces.

We have only been successful in generating single streams of music (i.e. one instrument at a time). This is in-line with the state-of-the-art algorithms, for example, Magenta [7] can only predict single streams as well. Another extension of our model could allow for several streams to be trained and predicted simultaneously, each representing a different instrument. Another interesting prediction our model could potentially make is instrument prediction; for example, predicting whether an instrument is a percussion or a melodic instrument.

Finally, there are still ideas to explore the enhancement of note prediction. We could attempt to vary the look-back length, or constrain the predictions to a single key scale.

Table 3: Status of the how well the model predicts the aspects of music presented in table 1.

Aspect of Music	Evaluation
Key Signature/Notes	Often fails to consistently stay on the same key
Dynamics	The model predicts dynamically varying velocities in the 60-100 range (on a scale of 1-126).
Time Signature	Consistently outputs the most common signature 4/4 numerator giving the number of beats in each bar.
Recurrent Structure	Shows recurrent melodies and rhythms, that evolve over time
Tempo/Rhythm	Often finds steady tempos, and familiar rhythms, but does deviate from both sometimes.
Chords	Learns the correct structure of chords, while keeping a good balance with single notes.

References

[1] Kim, J. (n.d.). (March 25, 2018) DeepJazz [Computer software]. https://deepjazz.io/

[2] Liang, F., Gotham, M., Tomczak, M., Johnson, M. & Shotton, J. (n.d.). (March 25, 2018) Bach Bot [Computer software]. https://www.bachbot.com

[3] Oord, A. V., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., . . . Kavukcuoglu, K. (n.d.). (March 25, 2018) WaveNet: A Generative Model For Raw Audio. https://deepmind.com/blog/wavenet-generative-model-raw-audio/

[4] Pachet, F., Roy, P., & Ghedini, F. (2013). Creativity through Style Manipulation: The Flow Machines project. Marconi Institute for Creativity Conference, 80.

[5] Sidor, S. (n.d.). (March 25, 2018) Magenta: Make Music and Art Using Machine Learning. https://magenta.tensorflow.org/

[6] Tang, A. & Shekar, P. (n.d.). (March 25, 2018)Gruv [Computer software]. www.gruv.io

[7] Sidor, S. (March 25, 2018) Magenta: Make Music and Art Using Machine Learning. https://magenta.tensorflow.org/

[8] Oord, A. V., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., . . . Kavukcuoglu, K. (March 25, 2018). WaveNet: A Generative Model For Raw Audio. https://deepmind.com/blog/wavenet-generative-model-raw-audio/

[9] Boulanger-Lewandowski, N., Bengio, Y. & Vincent, P. (2012). Modeling Temporal Dependencies in High-Dimensional Sequences: Application to Polyphonic Music Generation and Transcription. In Proceedings of the 29th International Conference on Machine Learning. Edinburgh. doi:http://www-etud.iro.umontreal.ca/ boulanni/ICML2012.pdf

[10] Huang, A. & Wu, R. (2016). Deep Learning for Music. https://cs224d.stanford.edu/reports/allenh.pdf.

[11] Eck, D. & Schmidhuber, J. (2002). USI-SUPSI Instituto Dalle Molle(Tech. No. IDSIA-07-02). doi:http://people.idsia.ch/ juergen/blues/IDSIA-07-02.pdf

[12] Liu, T. I., & Ramakrishnan, B. (2014). Bach in 2014: Music Composition with Recurrent Neural Network. https://arxiv.org/abs/1412.3191v2.

[13] Franklin, J. A. (2006). Recurrent Neural Networks for Music Computation. Journal on Computing, 18(3), 321-338. http://cs.smith.edu/ jfrankli/papers/Informs2006_18_03_0321.pdf

[14] Dong, H., Hsiao, W., Yang, L., & Yang, Y. (2017). MuseGAN: Multi-track Sequential Generative Adversarial Networks for Symbolic Music Generation and Accompaniment. https://arxiv.org/abs/1709.06298.

[15] Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., . . . Bengio, Y. (2014). Generative Adversarial Nets. Advances in Neural Information Processing Systems. http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf

[16] Radford, A., Metz, L., & Chintala, S. (2016). Unsupervised Representation Learning with Deep Constitutional Generative Adversarial Networks. https://arxiv.org/pdf/1511.06434.pdf.

[17] Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. Nature, 323(6088), 533-536. doi:10.1038/323533a0

[18] Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. Neural Computation, 9(8), 1735-1780. http://www.bioinf.jku.at/publications/older/2604.pdf

[19] Chen, C. J., & Miikkulainen, R. (2001). Creating Melodies with Evolving Recurrent Neural Networks. In Proceedings of the 2001 International Joint Conference on Neural Networks. http://nn.cs.utexas.edu/downloads/papers/chen.ijcnn01.pdf

[20] Chollet, F. (2015). Keras. https://github.com/keras-team/keras

[21] Bjorndalen, O. (2013). Mido (Version 1.2.8). https://mido.readthedocs.io

[22] Tzanetakis, G., & Perry, C. (2002). Music Genre Classification of Audio Signals. IEEE Transaction on Speech and Audio Processing, 10(5). https://dspace.library.uvic.ca/bitstream/handle/1828/1344/tsap02gtzan.pdf?

[23] Wang, A. L. (2003). An Industrial-Strength Audio Search Algorithm. https://www.ee.columbia.edu/ dpwe/papers/Wang03-shazam.pdf.

[24] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David

Cournapeau, Matthieu Brucher, Matthieu Perrot, Édouard Duchesnay. Scikit-learn: Machine Learning in Python, Journal of Machine Learning Research, 12, 2825-2830 (2011)

[25] Jones E, Oliphant E, Peterson P, et al. SciPy: Open Source Scientific Tools for Python, 2001-, http://www.scipy.org/ [Online; accessed 2018-05-07].

[26] Olah, C. (2005) Understanding LSTM Networks. http://colah.github.io/posts/2015-08-Understanding-LSTMs/