Deep Jammer: A Music Generation Model

Justin Svegliato and **Sam Witty**
College of Information and Computer Sciences
University of Massachusetts
Amherst, MA 01003, USA
{jsvegliato,switty}@cs.umass.edu

Abstract

Music generation remains an attractive and interesting application of machine learning since it is typically characterized by human ingenuity and creativity. Moreover, given the high-dimensionality of time-series data, it is difficult to construct a model that has the representational power necessary to capture the time- and note-invariant patterns throughout a musical piece. In this paper, we describe our implementation of a classical music generation model heavily influenced by previous work that uses deep neural networks, particularly Long-Short Term Memory networks (LSTMs), to capture the note and temporal patterns within a large dataset of classical pieces. We then use methods in transfer learning to train our classical music generation model on a small dataset of jazz pieces. Finally, we report and analyze the results of our experiments by comparing it to an existing music generation model that uses Markov Chains.

1. Introduction

Deep learning has recently had much success in many domains of machine learning, including text generation [23], language translation [3], image caption generation [17], object classification [14], game playing [22], and handwriting generation [9]. Moreover, within the particular domain of music, there has been much work in song, album, and artist classification [12] and song recommendation [7]. These tasks are ideal for deep learning approaches given the high-dimensionality and non-linearity of their underlying structure as well as the large amount of data readily available for training.

Another exciting yet challenging area within the particular domain of music is music generation. While music generation is another task characterized by high-dimensionality and non-linearity, it poses an additional challenge since it resides outside of the paradigm of traditional supervised learning techniques. It also requires a representation of historical states that must be leveraged to produce realistic music. Furthermore, the properties of temporal invariance and note invariance that are intrinsic to musical composition is difficult to capture by machine learning and deep learning techniques.

In this paper, we build a music generation model based on previous work by Daniel Johnson [11] that leverages Long-Short Term Memory networks (LSTMs) in order to capture the note and temporal patterns of music. The novel component of Johnson's architecture is the transposition of layer activations between a pairs of LSTM layers, which allows the recurrent layers to represent the sequential characteristics of music with respect to both harmonic and temporal dimensions. We then produce novel classical musical pieces by training our model on a large dataset of classical work. Subsequently, we generate novel jazz pieces by using methods in transfer learning to train our music generation model on a new dataset of jazz music. Finally, we conduct and assess experiments with various hyperparameters, such as layer size and dropout, as well as different transfer learning methods.

2. Related Work

While several machine learning methods have been used to explore musical domains, music generation has historically been a segmented and piece-wise endeavor. Significant progress has made in using machine learning techniques to discover structure in melody [21], chordal structure [4], ornamentation [8], and dynamics [13] independently, but little has been to done to integrate these components into comprehensive music generation.

The subject of algorithmic music generation has received some attention in the literature [20]. This work has predominantly been dominated by explicit knowledge-based methods [6], as well as parametric stochastic models such as Markov Chains [2]. These methods prove to be limited in their ability to produce human-like music.

Like many other domains, the application of explicit knowledge-based systems for music generation is limited

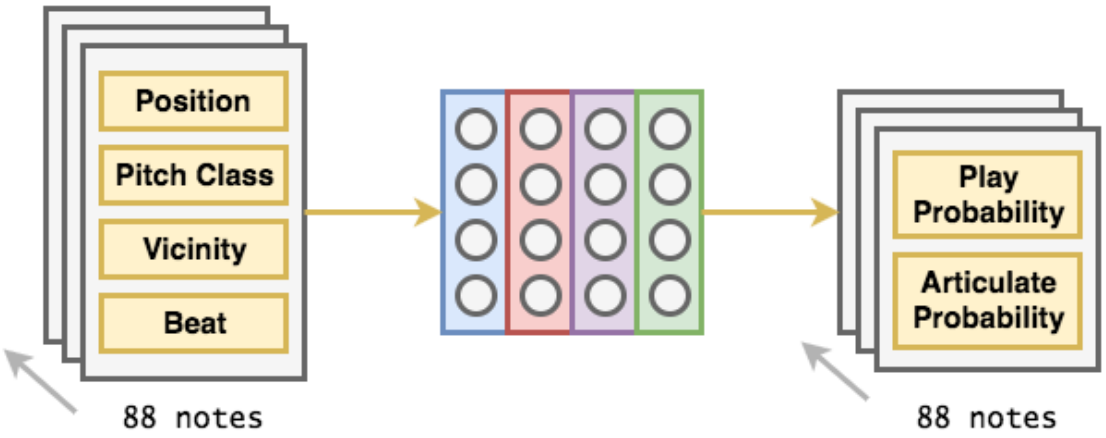

Figure 1: A black box diagram of the Deep Jammer music generation model. The input to the model is a set of 88 notes where each note has a position, a pitch class, a vicinity, and a beat attribute and the output of the model is a set of 88 notes where each note has a play probability and an articulate probability. Note that this only represents a single time step of the network. Moreover, the model is a four-layer neural network that will be described later.

by the challenge of formalizing consistent rules. In addition, this framework requires a human expert to encode the set of rules, inherently eliminating the ability of a model to *discover* previously unknown musical structure.

Unlike knowledge-based systems, which require explicit rules to generate music, stochastic methods such as Markov Chains can be used to generate music exclusively by example. However, these methods still rely on a human expert to explicitly define the state-space. This requirement poses significant challenges. If the state-space is too granular, the model may be limited in its ability to accomplish reasonable statistical inference due to the small number of training examples corresponding to each state. A state-space that is too broad, however, is also ineffective at representing the vast variability and expressiveness present in music.

Deep learning has shown some promise [1, 16] as a method for generating music using recurrent network architectures. The predominant advantage of these methods over knowledge-based or stochastic models is their ability to not only learn how to utilize features, but more importantly how to construct them with minimal feature-engineering. This allows deep learning models to discover the appropriate degree of complexity, balancing both predictive accuracy and broad expressiveness.

LSTMs have proven to be effective at representing both the short-term and long-term sequential dependencies in music. These methods have predominantly been developed for the purpose of analyzing text and language, which is inherently sequential in nature. Music shares many of these same sequential properties. In addition to a dependence on temporal sequences, an effective prediction must also consider the note invariance properties of musical composition. One method which has been proposed to account for this complex structure is known as a Biaxial LSTM [11]. This architecture serves as the foundation for our deep learning music generation model and is described in detail below.

3. Model

We proceed in this section as follows. First, we outline the structure of the input fed into the model. Next, we discuss the components of the output generated by the model. Thereafter, we describe the design of the model by providing a motivation and an explanation for each component. Finally, we enumerate the most significant hyperparameters of the model that can be adjusted to maximize performance.[1]

3.1. Input

The input to the model is a *segment* of a musical piece. In particular, a segment X is a three-dimensional matrix of 128 *time steps*, 88 *notes*, and 78 *attributes*. To put matters simply, we feed into the model 128 time steps where each time step is composed of 88 notes since there are 88 piano keys on a traditional piano. Then, for each note, we have 78 attributes that encapsulate various aspects of the corresponding note. See **Figure 1** for a depiction of the input.

However, to give a more formal description, the shape of a segment X is

$$(T, N, A) = (128, 88, 78),$$

[1]Note that we first implemented our model in Keras, a popular deep learning library in Python. After running into several issues during training, we then re-implemented our model in Theano, which is a lower-level Python framework that can be used for mathematical computation.

where T is the number of time steps, N is the number of notes, and A is the number of attributes. The attributes that describe each note is then a vector that can be represented as

$$\vec{a} = \langle p, c, v, b \rangle,$$

where the components p, c, v, and b are described below. Note that the size of each attribute is included in brackets. [2]

- **Position [1]**: A number p that represents the piano key position of the corresponding note. The position ranges from 1 to 88 given that there are 88 piano keys on a traditional piano. This gives the model the representational power to capture the difference between notes of the *same pitch*—e.g., two *A* notes—that are played in *different octaves*. Without the **Position** attribute, the model would lack the capacity to separate notes within different octaves of the same pitch. It is also important to note that the size of this range is necessary to capture every note contained within the training data. [3]

- **Pitch Class [12]**: A categorical array c where each element represents a distinct pitch in the chromatic scale.[4] This set of features contains a one-hot representation of pitch where all pitches not associated with the corresponding note contain a value of 0. While this feature is not essential to music generation, it encourages the model to learn a sense of note-invariance across different octaves.

- **Vicinity [50]**: An array v that contains the state of the pitches that neighbor the corresponding note. While the vicinity could be arbitrarily increased or decreased, we chose to capture the pitches for the upper octave and lower octave of the corresponding note. Moreover, we store two values for every neighboring note: namely, for each surrounding note, we store a 1 if the note was played at the last time step (and a 0 otherwise) and a 1 if the note was articulated [5] at the last time step (and, again, a 0 otherwise). Again, this feature is not essential to music generation. Rather, it encourages the model to learn a sense of sequential harmony and relative pitch. This is similar to the human auditory system, which emphasizes relative sequences of pitches rather than absolute pitches held in isolation.

- **Beat [4]**: A binary number b that represents the location of the corresponding note in the measure. For example, a note is associated with a binary integer ranging from 0000 to 1111 since our resolution is at 16th notes. This can be arbitrarily increased or decreased to capture the appropriate temporal scale for the classical pieces contained within the training set.

3.2. Output

The output of the model Y is a two-dimensional matrix of 88 *notes* and 2 *predictions*. For each of the 88 piano keys on a traditional piano, we have 2 probabilities. That is, we have a prediction vector

$$y = \langle \alpha, \beta \rangle$$

with the shape of

$$(N, P) = (88, 2),$$

where the components α and β are discussed below. Note that the size of each attribute is included in brackets. See **Figure 1** for a depiction of the output.

- **Play Probability [1]**: A probability α that represents whether the corresponding note will be played at the next time step.

- **Articulate Probability [1]**: A probability β that represents whether the corresponding note will be held at the next time step.

In a word, the model is a function that takes as input a segment X and then yields a prediction Y as output.

3.3. Design

Using a deep neural network, we implement our model with the input X and output Y described in the previous subsections. Remember that the shape of the input X is (T, N, A) and the shape of the output Y is (N, P) where T is the number of time steps, N is the number of notes, A is the number of attributes, and P is the size of the prediction. We also define the size of the **Time Layer** as S_t, the size of the **Note Layer** as S_n, and the size of the **Dense Layer** as S_d. The layers of our neural network are listed in order below. For the sake of brevity, we have deferred a thorough discussion of the various hyperparameters that can be tuned to a later subsection. See **Figure 2** for a depiction of the network's architecture.

[2] The data format for the input and output of all of the experiments uses the standard Musical Instrument Digital Interface (MIDI) format. MIDI encodes music as a collection of musical events, such as a given note being played at a given timestep. While the data processing and parsing is extensive to convert this representation into a useful form for music generation, these tasks have been omitted from this report for brevity. Specifically, the dataset of 325 classical songs and 50 jazz songs used in training were extracted from the Classical Piano MIDI Page [15] and Doug McKenzie Jazz Piano. [18]

[3] For clarity, pitch refers to the label of a note, such as C, whereas note refers to a specific position on the piano keyboard, such as middle C.

[4] The chromatic scale includes all twelve pitches.

[5] Articulation captures when a note is held, rather than being played and released.

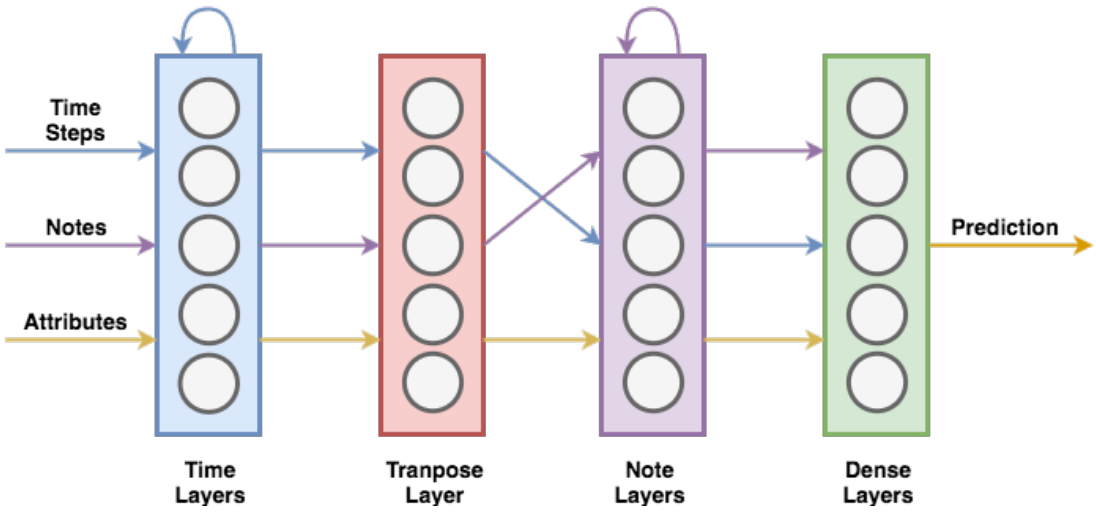

Figure 2: The neural network architecture of the Deep Jammer music generation model. The network takes in a matrix of the shape $(128, 88, 78)$ and outputs a prediction matrix of the shape $(88, 78)$. To be explicit, there is a Time Layer, a Tranpose Layer, a Note Layer, and a Dense Layer. As the input enters the Time Layer, the sequence dimension is the time dimension. On the other hand, as the input enters the Note Layer, the sequence dimension is the note dimension. The Dense Layer then summarizes the features of the Time and Note Layers.

1. **Time Layer** $(T, N, A) \to (T, N, S_t)$
   An LSTM layer that captures the temporal patterns of music. Since the sequence dimension of an LSTM is the leftmost dimension, this layer learns how the note attribute vectors $\vec{a}$ vary with the time step dimension t: it learns how the note attributes evolve with each time step.

2. **Transpose Layer** $(T, N, S_t) \to (N, T, S_t)$
   A layer that transposes the time step and note dimension. This sets the note dimension as the new sequence dimension, which prepares it to be processed by the **Note Layer** below.

3. **Note Layer** $(N, T, S_t) \to (N, T, S_n)$
   An LSTM layer that captures the note patterns of music by learning how the note attribute vectors vary with the note dimension. Simply put, this layer learns how the note attributes evolve with each note.

4. **Dense Layer** $(N, T, S_n) \to (N, S_d)$
   A fully-connected layer that converts the high-dimensional output from the previous layer to an *unnormalized prediction pair*. That is, for each note, we have an unnormalized prediction pair that contains the unnormalized play probability and the unnormalized articulate probability. This layer can be viewed as a summary of the features that the **Time Layer** and **Note Layer** have learned.

5. **Activation Layer** $(N, S_d) \to (N, S_d)$
   A layer that normalizes or—more colloquially—squashes each element of the unnormalized prediction pair from the **Dense Layer**. It passes the left and right elements through the *sigmoid* activation function, which is defined as

$$a(x) = \frac{1}{1 + e^{-x}}.$$

   This yields a *normalized prediction pair* that represents the play and articulate probabilities described in the previous section. While S_t and S_n can be chosen arbitrarily, S_d must always be 2 since the network outputs a 2-element vector for each note.

3.4. Hyperparameters

While the previous subsection discusses the basic architecture of the music generation model, there are many hyperparameters that can be tuned to optimize performance. We describe the most important hyperparameters of the model below.

Note that each hyperparameter is only applicable to the **Time Layer**, the **Note Layer**, and the **Dense Layer**: namely, there are no hyperparameters that can be adjusted for the **Tranpose Layer** and the **Activation Layer**. Although the activation function in the **Activation Layer** could be set to another activation function like *tanh*, *ReLU*, or *Leaky ReLU*, the *sigmoid* activation function is most relevant to our model since it generates probabilities by squashing a given input to a value between 0 and 1.

- **Layer Count**. While we previously highlighted that there was only a single **Time Layer**, **Note Layer**, and **Dense Layer** in the music generation model, we can instead chain many layers of the same type together so long as the dimensions remain consistent. For instance, we can have two or more consecutive **Time Layers** to increase the representational capacity of the model to detect note attribute patterns over time. We can also stack **Note Layers** in a similar fashion to produce analogous results over the note dimension. More generally, the music generation model can have n **Time Layers**, y **Note Layers**, and z **Dense Layers**. If the model has too many layers, however, we run the risk of partially memorizing the input data due to overfitting.

- **Layer Size**. For each layer, we can adjust the number of nodes in that layer. As an illustration, each **Time Layer** can have an arbitrary number of nodes to increase or decrease its representational power. In other words, the more nodes present in the layer, the more expressiveness it has, which means it can model more complex relationships over the sequence dimension. This is of course at the risk of memorizing the input data as well.

- **Dropout**. We can use *dropout* after each layer as a form of regularization [5]. This prevents the model from memorizing or, more formally, overfitting the test data. Since we can place dropout after a **Time Layer** or a **Note Layer**, we can regularize the patterns detected when either the time or note dimension act as the sequence dimension of the LSTM. As we increase the probability p of dropout, the neural network will memorize less and less of the training data, allowing it to compose more *creative* pieces at the risk of dissonance.

This section only serves as an overview to the various hyperparameters that can be tuned in our music generation model. A thorough examination of the results of each hyperparameter is discussed later.

4. Training

The model is trained similarly to state-of-the-art natural language processing techniques: the sequence is treated as a training example where the historical information is used as a set of features. This general methodology is used to train our network, with the inclusion of recurrence in both the temporal and the note dimensions. To be more precise, the model creates a prediction of each note and time step based on the individual example's features, the preceding time state, and the preceding note state. The use of LSTMs allows the network to retain relevant information across a wide range of time and note history.

While the problem of predicting sequences of pitches could be represented as a multi-class classification problem, our formulation relies on the more expressive representation as a collection of conditional probabilities. The following equation describes this representation, where the probability of a given note being articulated or held is a function of the time step (t), the note (n), and the state $(S_{t,n})$. For clarity, this state $(S_{t,n})$ refers to the combination of short-term and long-term memory from both the recurrent time-layers and the recurrent note-layers.

$$P(y_{t,n}) = f(t, n, S_{t,n})$$

The function $f(t, n, S_{t,n})$ can be thought of as the neural network itself, which takes as input a set of features representing the time step, note, and historical state. [6]

Consistent with this interpretation that the neural network is producing a probability density function we define the loss as the negative log-likelihood of the neural network model given the ground truth. Interpretting the output of the neural network as a Bernoulli random variable, the likelihood can be defined in terms of the observed Boolean at a given time step $y_{t,n}$ by the following equation, where σ is the output of the sigmoid layer.

$$\begin{aligned} L(\sigma|y_{t,n}) &= \prod_{t,n}(\sigma_{t,n}y_{t,n} + (1-\sigma_{t,n})(1-y_{t,n})) \\ &= \prod_{t,n}(2\sigma_{t,n}y_{t,n} - \sigma_{t,n} - y_{t,n} - 1) \end{aligned}$$

Therefore, the log-likelihood is given by the following equation.

$$loss = -\log(L(\sigma|y_{t,n})) = -\sum_{t,n}\log(2\sigma_{t,n}y_{t,n} - \sigma_{t,n} - y_{t,n} - 1)$$

Taking the derivative with respect to each output of the sigmoid layer yields the following.

$$\frac{\partial loss}{\partial \sigma_{t,n}} = \frac{1 - 2y_{t,n}}{2\sigma_{t,n}y_{t,n} - \sigma_{t,n} - y_{t,n} - 1}$$

These gradients with respect to the sigmoid layer are then propagated back through the neural network, resulting in a gradient of he loss with respect to each of the parameters in the fully connected and recurrent layers.

[6] Several of the features used as inputs to the first layer such as beat and pitch class, are engineered to accelerate the model's learning about important musical structure. Regardless, this simply provides the network with an indicator that these representations of time and pitch are significant, and not how they should be utilized to construct such a function.

The model was trained using the above formulation of the gradient with adadelta parameter updates[24], ensuring that the model training is robust to initial conditions. In order to optimize for GPU performance the training was conducted in batches of 5 full music segments simultaneously before conducting each set of parameter updates.

5. Generation

In order to generate music, we must take a different approach than the approach taken in training the model. The primary difference between training and composition lies in what we feed into the neural network.

On the one hand, we train the model by feeding in a batch of music segments—each of which has the shape (T, N, A) as defined above—and then backpropagating the gradient based on our loss function. This ensures that the model learns how to reproduce the ground truth as it continues to train.

On the other hand, we compose music by feeding in a single time step of a randomly selected music segment X: more precisely, the input is in the shape of $(1, N, A)$ in the case of generation rather than in the shape of (T, N, A) in the case of training. Observe that we only have a single time step in the input during generation as opposed to T time steps in training. The generation must proceed one time step at a time due to the recurrent nature of the network. That is, we want to generate a prediction only for the time step that follows the time step that we fed in as input.

In response to the segment X of shape $(1, N, A)$, the model generates a prediction matrix Y of the shape $(1, N, 2)$. The prediction matrix Y represents our prediction for the next time step: namely, for each note, we have a prediction vector y that contains the play probability α and articulate probability β. To determine whether each note is played or articulated, we generate a set of uniform random values R and compare to the prediction matrix Y. This comparison yields a boolean matrix B since it checks if each element of the prediction matrix Y is greater than or equal to the corresponding element in the random matrix R. The boolean matrix B represents whether each note should be played at the next time step.

Once we know which notes ought to be played at the next time step, we convert the boolean matrix B into a new music segment X'. We then repeat the entire process by feeding in the new music segment X' into the model to obtain the next boolean matrix B'. If we continue this process repeatedly, we will produce a sequence of boolean matrices, which represents the notes played and articulated at every time step in the generated piece. [7]

[7] As this method is not strictly deterministic in note selection, the same starting pitch has the potential to generate many distinct musical segments. Sampling also allows the network to play chords, which would not be possible if the generation was treated as a deterministic multi-class classification problem.

6. Transfer Learning

After training our music generation model on a large classical dataset, we use *transfer learning* to generate jazz music. Particularly, we train our pretrained music generation model on a small dataset of jazz music. Intuitively, since the music generation model has already learned weights that encompass the time and note patterns intrinsic to the general structure of music, the model can potentially learn how to play jazz music with little training. It is important to note that we test two methods in transfer learning:

- **Extended Training**: Train on a small dataset of jazz music without modifying any part of the pretrained music generation model. This is the most rudimentary method in transfer learning. Simply put, we train our music generation model on a large dataset of classical music until we produce realistic classical music and then continue to train on a small dataset of jazz music.
- **Fine-Tuning**: Train on a small dataset of jazz music only after re-initializing the weights of every **Dense Layer** of the pretrained music generation model. In other words, we follow a similar process to **Extended Training** after first re-initializing each **Dense Layer**.

7. Experiment

While many other applications of machine learning have been explored in detail, the implications of network design on a neural network's ability to generate music are relatively unknown. In that vein, we have conducted a series of experiments to better understand the effects of various network design considerations. [8]

Using the core architecture described above, we first constructed a baseline model to act as a reference for experimentation. The baseline model and the three subsequent experiments were trained on a batch of 5 music segments composed of 128 timesteps[9] from 325 classical piano songs for a total of 5000 epochs. Each of the transfer learning experiments described used the trained weights from the baseline model before additional training was conducted on a dataset of 50 music segments from jazz piano songs for a total of 2000 epochs.

[8] These experiments are not intended to be a completely comprehensive investigation into the effects of network design on music generation performance. Rather, we hope to provide the first step into this exploration. We feel that the extent of the experiments are appropriate for the available resources and project scope. Each experiment requires approximately 16 hours of training on a g2.2xlarge EC2 instance on AWS, limiting the full breadth of experimentation. If additional resources were available, we would also investigate the effect of individual hyperparameter selection, loss mechanisms on model performance, as well as other modifications.

[9] One timestep corresponds to a sixteenth note.

Each experiment is described in detail below. See **Table 1** for a summary of all experiments. Note that the baseline model is the Large Network, i.e., the first column of the table.

Parameter	Large	Medium	Small	Dropout
Time Layers (n)	2	2	2	2
Note Layers (y)	2	2	2	2
Dense Layers (z)	1	1	1	1
Time Layer 1 Size	300	150	75	300
Time Layer 2 Size	300	150	75	300
Note Layer 1 Size	100	50	25	100
Note Layer 2 Size	50	25	13	50
Dense Layer Size	2	2	2	2
Dropout Strength	0	0	0	0.5
Learning Rate	0.01	0.01	0.01	0.01
ϵ	1e-6	1e-6	1e-6	1e-6
ρ	0.95	0.95	0.95	0.95

Table 1: The experimental hyperparameters of the Large Network, Medium Network, Small Network, and Large network with Dropout. Note that the Large Network was then used for both transfer learning methods since it performed the best out of all the experimental architectures. Moreover, we found that a slow Decay Rate and 2 Time Layers, 2 Note Layers, and 1 Dense Layer had high performance without significantly amplifying training time and thereby chose to hold these as constant variables in our experiment.

7.1. Layer Size

Three distinct experiments were conducted to explore the effects of layer size. Specifically, all of the recurrent and fully connected layers were reduced in size to the specifications in **Table 1**. Intuitively, a network with more nodes should have the capacity to learn more complex non-linearity of the underlying structure of music. As discussed above, however, this also comes with a high computational cost and the risk of over-fitting.

7.2. Dropout

One experiment was conducted to explore the effects of adding dropout layers throughout the network. Specifically, dropout layers with a dropout parameter of 0.5 were added following each of the four recurrent layers. Dropout layers force redundancy in neural network learning representation, which results from individual activations being randomly set to 0 at each training step. Dropout is of particular interest because of its capacity to prevent overfitting in many domains. In the context of music generation, this likely means avoiding the excessive imitation of musical style found in the training data. However, this often comes with a trade-off in model performance, in this context resulting in lower quality generated music, especially if the domain is highly non-linear and complex.

7.3. Transfer Learning

Two experiments were conducted to explore transfer learning between musical genres. These experiments extract the layer weights of the Large Network before subsequently training for 2000 epochs on a dataset of 50 jazz segments with 128 time steps. Since the Large Network seemed to produce the best music, we used that as the model for transfer learning. For the extended training experiment, all of the layer weights were used to initialize the training on jazz music. For the fine-tuning experiment, all of the layer weights were used to initialize training on jazz music with the exception of the dense layer, which was randomly initialized.

8. Evaluation

Two distinct quantitative metrics were developed and implemented to assess the model's capacity to generate realistic and high quality music. These metrics include the loss trajectory to characterize the model's predictive accuracy during training as well as a blind survey of 26 participants to evaluate the generated music. We discuss each metric in the next two subsections.

In addition to our comparison of several deep learning models, we developed a simple Markov Chain model to use as an additional benchmark against the performance of the deep learning approach. We used a Markov Chain as a parametric model where the probability of generating a note y given that the previous note x is observed at the last time step is described by

$$P(y|x) = \frac{N_{y,x}}{N_x},$$

where $N_{y,x}$ is the number of times note y was played following note x and N_x is the number of times note y was played in total. In summary, the conditional probability of successive notes is approximated by their frequency.

Using the Markov chain, we generated music in an identical way to the neural network, replacing the full network with this simplified probability of generating each note at each time step t. In addition, we conducted a qualitative comparison between our network's output and a more sophisticated Markov model. [19]

8.1. Training Loss

While this is not a metric of predictive accuracy, we provide the training loss trajectories as an indicator of how different design decisions affect a model's capacity to learn musical representations. As the loss mechanism is consistent between experimental trials and does not include normalization, a comparison of training losses across provides

key insights as discussed in the analysis sections below. This metric is particularly interesting for the two transfer learning experiments since the loss at any given iteration can be thought of as the model's adaptation to the new domain.

8.2. Survey

Since musical quality is inherently subjective, we have assessed the model's performance by surveying a broad assortment of volunteers. Note that this was done through an online survey with embedded MP3 files using the service *Survey Gizmo*. Specifically, at least two generated segments from each type of network design that was tested as well as a single real classical piano segment and a single real jazz piano segment were provided to the survey participants for evaluation. Each participant was asked to assign a quality score on a scale from 1 to 10 (where 10 is the highest) for each musical segment. The participants were also asked if each segment seemed like it could have been composed by a human, which we call the piece's *believability score*.

We designed our survey in a way that minimizes bias in the assessment of the computer-generated music by omitting descriptive information as well as including the two real musical segments. The participants were falsely informed that every segment was generated by our algorithm. These segments provide a baseline reference for quantitative assessment, ensuring an effective comparison between music composed by a human and music composed by the algorithm. We have included the quantitative results in the following section.

9. Results

The following tables and figures represent the most interesting findings from the model training as well as important survey results. Please refer to the **Appendix** for all of the experimental results. We have omitted them for the sake of brevity. The generated music and source code can be found in the attached supplementary materials.

9.1. Training Loss

Figure 3 shows a typical relationship between training loss and epochs for the large network with fine-tuning on a small jazz dataset starting at (5). We generated music throughout the training process as shown in **Figure 4** in order to provide a qualitative understanding of what musical properties the network learns as loss evolves over time.

1. The music generated by the network before training has little harmonic and musical structure. Many notes are played all at once.

2. The network begins to learn appropriate rhythmic sparsity of classical music but harmonic structure remains undiscovered.

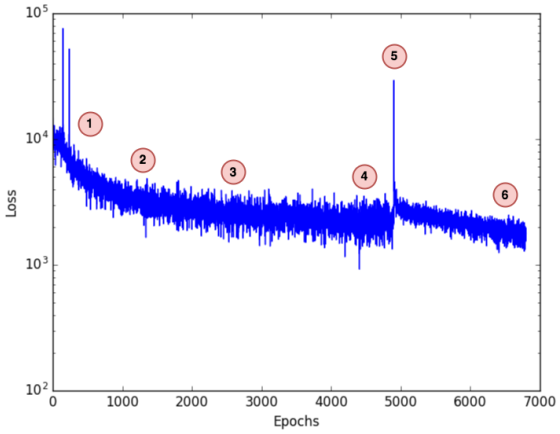

Figure 3: The training loss curve of the Large Network with Fine-Tuning. Note that each number is associated with an item in the list that describes how the music changes over time. Item (5) indicates when we begin to fine-tune the pretrained classical music generation network on a small dataset of jazz music.

3. Sequences of notes from the generated music resemble common scales and harmonic themes in classical music but overall music is still dissonant and unpleasant.

4. The network generates original classical music with few dissonant notes and rhythmic abnormalities.

5. The last fully-connected layer of the network is reinitialized with random weights, and the network begins training on a smaller dataset of jazz music. The generated musical quality reverts back to the very early stages of training.

6. After a significant period of training the network has discovered jazz chords and rhythms but remains unable to generate high quality musical segments.

9.2. Survey

Table 2 provides a summary of the survey results. Twenty-six participants completed the survey, rating fourteen generated segments and two real classical and jazz segments. [10] The participants were instructed to rate the quality on a scale from 1 to 10 and report whether each segment

[10] All of the neural network models include two samples of generated music with the exception of the Large Network, which included 4 samples, since we deemed that to be the best quality music generator. Note that one real music segment was selected for classical and jazz as well. This gives us baseline that we can compare against.

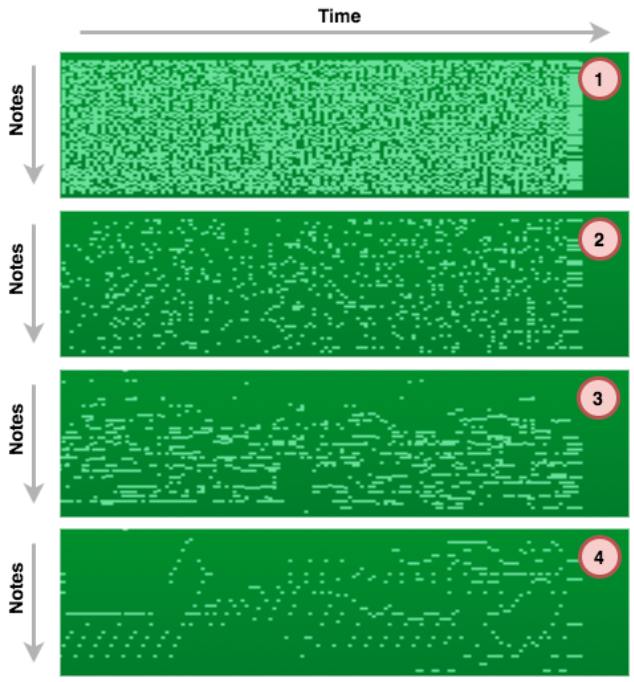

Figure 4: Music generated at various stages of training for the Large Network. As the training progresses, the network continuously learns to compose music with increasingly realistic rhythmic and harmonic structure. After approximately 4500 epochs additional training does not result in noticeably higher quality music.

could have feasibly been composed by a human. The percent of participants that answered affirmatively defines the believability score.

Model	R	σ	B
Large Network	6.7	2.1	76
Medium Network	6.2	2.0	73
Small Network	6.1	2.1	67
Large Network w/ Dropout	6.0	1.9	69
Extended Training	4.3	2.0	40
Fine-Tuning	4.7	1.7	48
Real Classical	8.1	1.4	100
Real Jazz	7.3	1.6	92

Table 2: The survey results of average rating (R), the standard deviation of the rating (σ), and the believability of the piece (B).

The average quality rating and believability scores demonstrate a strong positive correlation, which is consistent with our expectations. Of the three varying sizes, the Large Network generated music with the greatest average rating and believability scores followed by the Medium Network, and the Small Network. This is likely because the larger networks are able to extract a greater quantity and complexity of non-linear features from the input attributes and historical state, leading to a closer representation of musical structure. We expect that this phenomenon would continue with larger networks up until some threshold size, at which point the training would result in overfitting. This would be apparent if the network perfectly reproduced musical segments in the training set.

Dropout reduced the quality and believability of the generated music. We hypothesize that this is due to the fact that music is sufficiently complex to require the consistent activation of the vast majority of neurons. Perhaps if the network sizes were increased, as discussed above, the inclusion of dropout would result in expressive music that does not overfit to the training set. Moreover, the loss curve for the Large Network was generally lower than the loss curve for the Large Network with Dropout. Typically, lower loss has been an indicator of the quality of music; that is, the lower the loss, the better the music. This can be seen in the *Appendix* as well as the attached material.

Both of the transfer learning experiments, namely extended training and fine tuning, resulted in significantly lower quality music than the generated classical music. It is also important to note that transfer learning was still successful in adopting some of the characteristics of jazz music, particularly common chord structure, the overall sound of the piece, and rhythmic patterns. In addition, it surprisingly only took about a hundred epochs for the music generator to learn many features of jazz.

We hypothesize that the overall low quality of the music results are attributable to some combination of the following characteristics.

- Jazz music incorporates more complex rhythmic and harmonic structure than classical music, making learning inherently more challenging. While the layer sizes of the Large Network may have been sufficient for classical music, perhaps more are necessary to represent Jazz music.

- The network learned temporal and harmonic structure in the low-level recurrent layers during the pretraining phase that were unique to classical music. Extended training and fine-tuning are unable to reconcile these low-level feature representations, as gradient descent orients learning towards local minimum that are far from the global minimum.

As expected, the real compositions outperformed all of the algorithmic compositions for both classical and jazz compositions. However, this represents the average results of all of the surveys. Seven of the participants rated at least

one generated musical segment higher than the real musical segment.

While we did not include the Markov Chain model in the quantitative assessment of model performance, the qualitative results are indicative of the key differences in model expressiveness. As expected, the Markov chain created music that was often disharmonious and rhythmically disjointed. This contrasted starkly to the results of the deep learning model, which successfully captured the common harmonic and rhythmic structure of musical composition. [19]

10. Conclusion

In this report, we presented a novel method for using deep learning to generate musical segments. This network utilized LSTMs in both the time and the note dimension to capture the rhythmic and the harmonic structure of music. The wide variety of networks based on the same core architecture were trained on a dataset of 325 classical music compositions encoded through MIDI before generating original music segments.

In addition, we explored the efficacy of using this architecture for transfer learning to jazz music, utilizing both extended training and fine tuning. These methods were assessed qualitatively based on a progression of music generation throughout the training process, as well as quantitatively via a survey of willing participants.

Overall, we found that large networks without dropout produced the highest quality music and transfer learning was unsuccessful at producing high quality jazz music but still was successful in adopting a subset of the the musical qualities of jazz.

Future work includes a more exhaustive analysis of hyperparameters since we were limited in our computing resources. In particular, it would be useful to test many more hyperparameter combinations, such as how music quality varies with additional layers, batch normalization layers, larger layers, as well as different update rules like Adam and Adagrad. While there are many more hyperparameters we could tweak, we only include a subset of them here for the sake of brevity.

Moreover, while our music generation model is effective at composing small segments of music, it cannot learn features about the long-term structure of music. That is, our model is only effective at generating music within a specific motif unlike real music, which typically includes multiple parts that express different ideas. To achieve this, we would most likely need to include attributes that encapsulate long-term information throughout a piece. Similarly, our model cannot capture the dynamics, i.e., how hard a key is pressed. This also limits the expressiveness of our music generation model since we can only play a note at a specific intensity. These two ideas contribute to the *emotion* of the pieces generated by the model. This would likely increase the believability score.

In summary, this project successfully explored and demonstrated the potential for deep learning in music generation. While this work is far from replacing human composers, the results imply that deep learning has much potential in supplementing artistic endeavors.

References

[1] Boulanger-Lewandowski, N., Bengio, Y., and Vincent, P. *Modeling Temporal Dependencies in High-dimensional Sequences: Application to Polyphonic Music Generation and Transcription*. Proceedings of the 29th International Conference on Machine Learning, (29), 2012

[2] Cambouropoulos, E. *Markov Chains as an Aid to Computer Assisted Composition*. Musical Praxis, 1 (1):4152, 1994.

[3] Cho, K., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation*. Arxiv. 2014

[4] Conklin, D. *Chord Sequence Generation with Semiotic Patterns*. Journal of Mathematics and Music, 10 (2):92-106, 2016.

[5] Dahl, G., Sainath, T., and Hinton, G. *Improving Deep Neural Networks for LVCSR using Rectified Linear Units and Dropout*. International Conference on Acoustics, Speech and Signal Processing, 2013.

[6] Ebcioglu, K. *An Expert System for Harmonizing Four-part Chorales*. Computer Music Journal, 12 (3):4351, 1988.

[7] Eck, D., Lamere, P., Bertin-mahieux, T., and Green, S. *Automatic Generation of Social Tags for Music Recommendation*. Advances in Neural Information Processing Systems, 20 (1):385-392, 2008.

[8] Giraldo, S. and Ramrez, R. *A Machine Learning Approach to Ornamentation Modeling and Synthesis in Jazz Guitar*. Journal of Mathematics and Music, 10 (2):107-126, 2016.

[9] Graves, A. *Generating Sequences with Recurrent Neural Networks*. Arxiv, 2014.

[10] Ioffe S., Szegedy, C. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. Arxiv, 2015.

[11] Johnson, D. *Composing Music with Recurrent Neural Networks*. Hexahedria, 2016.

[12] Knees, P., Pampalk, E., and Widmer, G. *Automatic Classification of Musical Artists based on Web-Data*. Oesterreichische Gesellschaft fuer Artificial Intelligence, 24 (1), 2004.

[13] Kosta, K., Ramrez, R., Bandtlow, O., and Chew, E. *Mapping between Dynamic Markings and Performed Loudness: A Machine Learning Approach*. Journal of Mathematics and Music, 10 (2):149-172, 2016.

[14] Krizhevsky, A., Sutskever, I., and Hinton, G. *ImageNet Classification with Deep Convolutional Neural Networks*. Advances in Neural Information Processing Systems, 25 (1):1097-1105, 2012.

[15] Krueger, B. 1996-2016. *Classical Piano MIDI Page*. http://www.piano-midi.de

[16] Liu, I. and Ramakrishnan, B. *Bach in 2014: Music Composition with Recurrent Neural Network*. Arxiv, 2014.

[17] Mao, J., Xu, W., Yang, Y., Wang, J., Huang, Z., and Yuille, A. *Deep Captioning with Multimodal Recurrent Neural Networks (m-RNN)*. Arxiv. 2014.

[18] McKenzie, D. 2012. *Jazz Piano MIDI Page*. www.bushgrafts.com/jazz/midi.htm

[19] Markov Music. 2015, Jun 18. *Markov Music Generator: Bach*. www.youtube.com.

[20] Papadooulos, G. and Wiggins, G. *AI Methods for Algorithmic Composition: A Survey, a Critical View and Future Prospects*. AISB Symposium on Musical Creativity, 110-117, 2016.

[21] Ponce de Leon, P., Inesta, J., Calvo-Zaragoza, J., and Rizo, D. *Data-Based Melody Generation through Multi-Objective Evolutionary Computation*. Journal of Mathematics and Music, 10 (2):173-192, 2016.

[22] Silver, D. et al. *Mastering the Game of Go with Deep Neural Networks and Tree Search*. Nature, 529 (1):484-489, 2016.

[23] Vinyalsm, O. and Le, Q. *A Neural Conversational Model*. Arxiv. 2015.

[24] Zeiler, M. *Adadelta: An Adaptive Learning Rate Method*. Arxiv. 2012.

Appendix

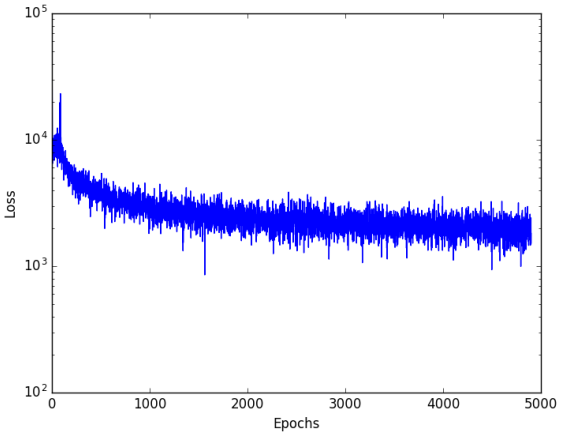

Figure 5: The training loss curve of the Small Network.

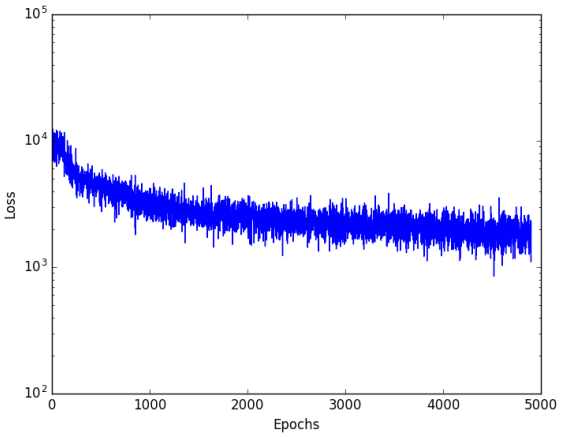

Figure 6: The training loss curve of the Medium Network.

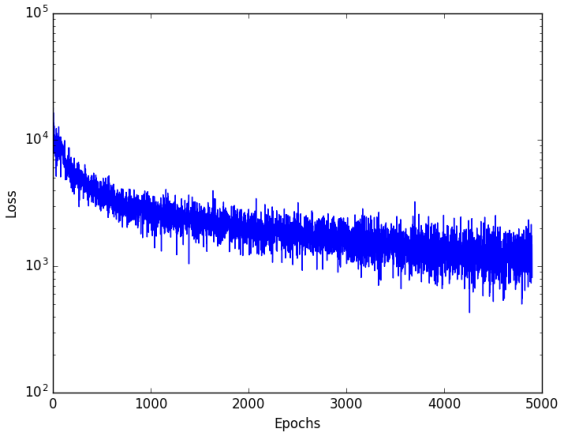

Figure 7: The training loss curve of the Large Network.

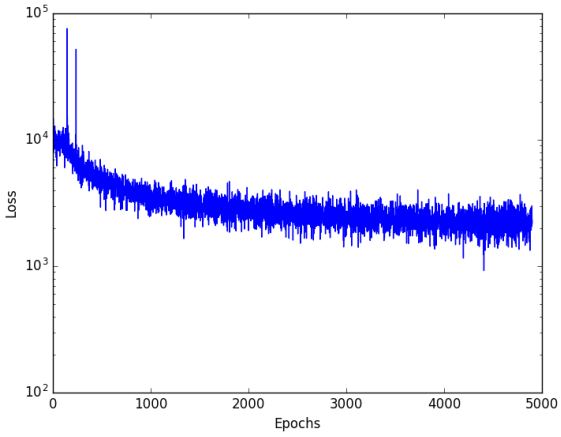

Figure 8: The training loss curve of the Large Network with Dropout.

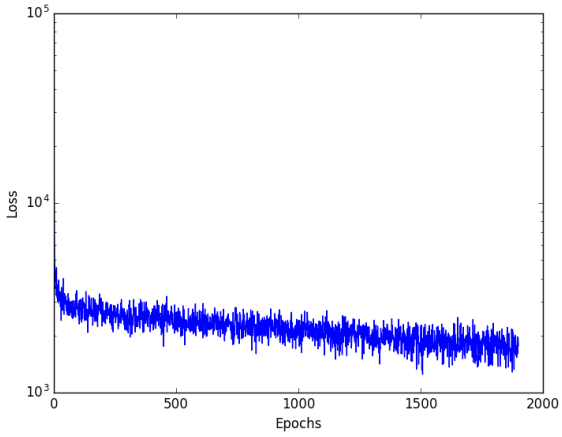

Figure 9: The training loss curve of Extending Training on the Large Network.

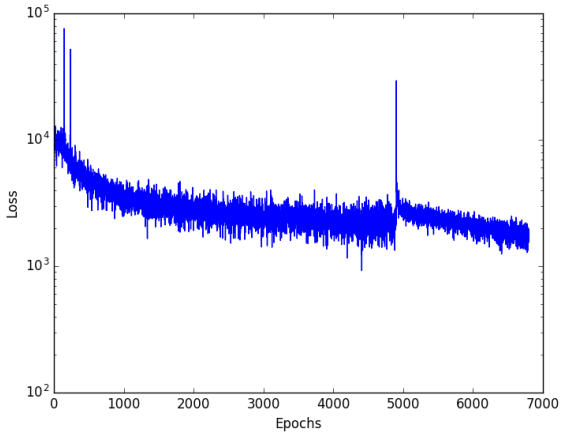

Figure 10: The training loss curve of Extending Training on the Large Network including the initial training loss curve.

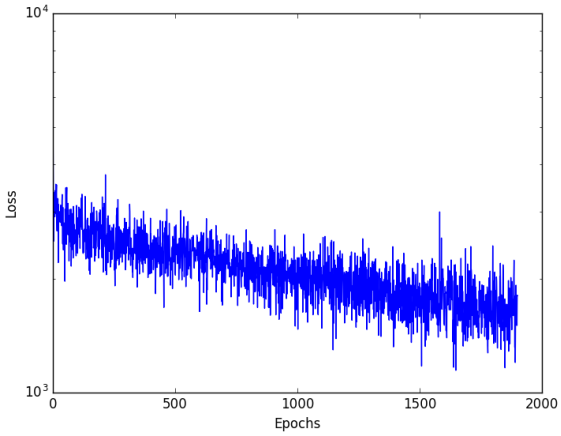

Figure 11: The training loss curve of Fine Tuning on the Large Network.

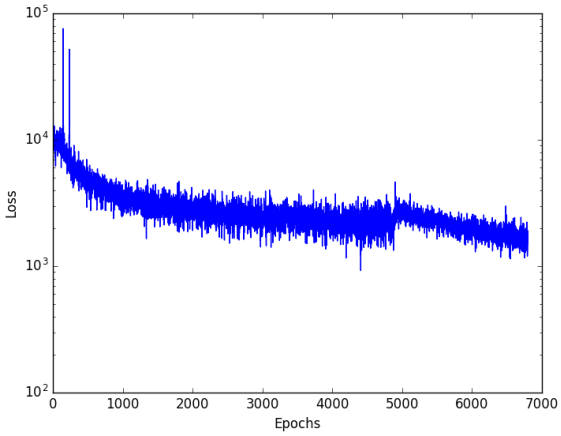

Figure 12: The training loss curve of Fine Training on the Large Network including the initial training loss curve.