

Rose-Hulman Institute of Technology

**Rose-Hulman Scholar**

---

Senior Projects - Computer Science & Software  
Engineering

Computer Science & Software Engineering

---

Spring 5-21-2021

## **Music Generation with Deep Neural Networks Using Flattened Multi-Channel Skip-3 Softmax and Cross-Entropy**

Jessica Myers

Follow this and additional works at: [https://scholar.rose-hulman.edu/computer\\_science\\_seniorproject](https://scholar.rose-hulman.edu/computer_science_seniorproject)

---

# MUSIC GENERATION WITH DEEP NEURAL NETWORKS USING FLATTENED MULTI-CHANNEL SKIP-3 SOFTMAX AND CROSS-ENTROPY

Jessica Myers

Senior Thesis, CSSE Department, Rose-Hulman Institute of Technology

myersjm@rose-hulman.edu

21 May 2021

## ABSTRACT

This paper presents an investigation of convolutional neural networks as a means of generating human-plausible, goal-oriented music, specifically pop melodies. Deep neural networks were chosen as a focus because their training seems to mimic the way in which a person passively learns music throughout life. The raw dataset of MIDI files was acquired from 17,216 song clips by 4825 artists from Hooktheory's TheoryTab Database. A custom dataset was created by encoding the MIDI files into sparse matrices and sliding a fixed window over each song to generate sequences. A novel approach within the domain of music generation was employed using a custom 'skip-3' softmax activation function, as well as a 'skip-3' cross-entropy loss function. The current results of generating music, given a seed, using a fully-connected network, a convolutional network, and a dilated convolutional network show some evidence of rhythmic and harmonic patterns, but lack melodic elements.

## 1. INTRODUCTION

Can there be an artificially intelligent music composer? The mere thought of one invites both curiosity and criticism, as the ultimate creation of such a system could reveal the inner workings of the brain, but at the same time could serve as competition to human composers. However, if an AI composer did exist, depending on the spectrum of autonomy, it has the potential to be more useful to humans than harmful. It could serve as a tool to inspire new ideas in composers, as well as help musicians who are not composers themselves develop unique pieces suited to their particular musical tastes. Such a system could possibly give insight into a more efficient composition process, or have applications in other creative fields that also depend on innovative solutions to loosely-bounded problems—and if the AI composer somehow became sentient enough to not

want to be treated as a tool, then perhaps it could re-invigorate the music industry with new, emerging styles of music as part of its own record label.

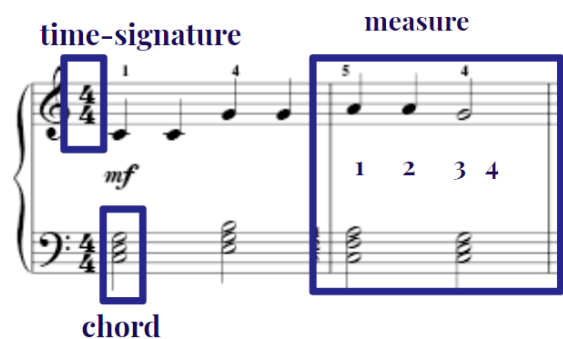


Figure 1: Example of music exhibiting structure

So how would one go about creating such a system? Music is governed by rules—the time signature restricts how many notes can fit in a measure (seen in Figure 1 above), and Western music has chord progression flow charts—so it only seems natural to code these rules to generate music. But what about creativity? Composers often break rules to be creative. This is where a rules-based perspective breaks down. Non-musicians don't enjoy music based on strict rule-following; yet they somehow know when something is off [1]. This implies a sense of passive learning of music throughout a person's life, similar to the way in which a neural network trains. This led to the investigation of deep neural networks as a means of generating music, specifically convolutional neural networks (CNN). CNNs are interesting to explore because they are traditionally used for image-based applications, yet music is not necessarily an image. A more common approach might be to use some form of recurrent neural network, or even a transformer, which are typical for Natural Language Processing. However, past work in this area using convolutional neural networks in combination with the idea that the progression of

music over time could be viewed as an image lead this to be a plausible investigation.

Although there are several challenging aspects to this problem, one of the most significant is the observation that more than one note can be played at a particular time. The work in this paper uses a symbolic encoding of music, rather than pure audio, representing each note as either beginning, held, or not played. This two-dimensional encoding of the state presents a challenge in constructing a network that predicts the next set of notes. The proposed solution involves novel ‘skip-3’ softmax and cross-entropy functions to handle this unique classification problem.

Another challenging aspect is scope—typically pop melodies are placed in context with their harmonies. This translates to reading two tracks, melody and chords, in the MIDI files, and having a network learn patterns in both. This paper presents models trained with melody and chords, as well as models trained with melody only. Dataset size also plays a role, as a significant amount of computing power is needed to preprocess and train networks on millions of examples.

Capturing the overarching patterns in those examples is also a challenge, because this data is ordinal and notes at the beginning of a melody impact notes at the end. The system explained in this paper seeks to capture those long-term dependencies by using convolutions and exploring dilated convolutions; however, this system was designed within the scope of a melody, not an entire song. The ultimate goal of this project would be to create a system to compose melodies which could later be integrated as modules into a larger piece of music, perhaps by use of a separate system.

## 2. RELATED WORK

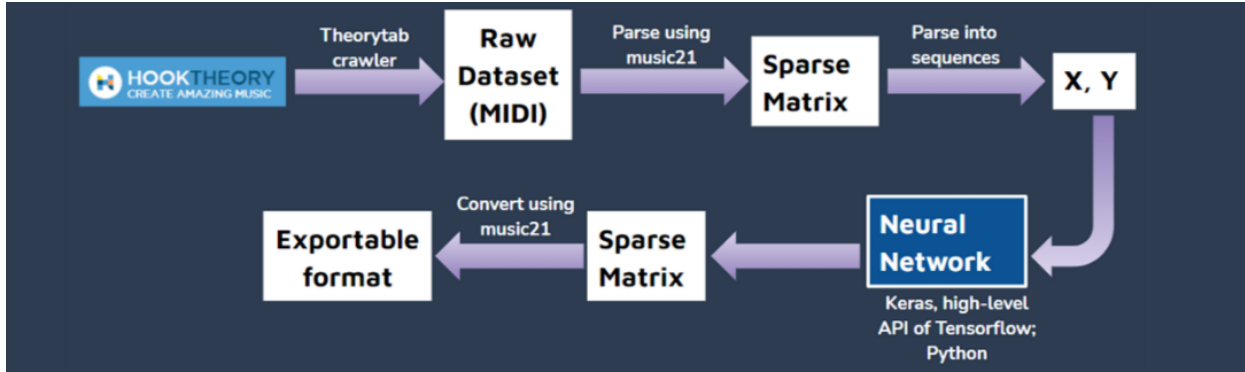
Research in the area of artificial intelligence and music generation, as well as in the broader multidisciplinary field of “Computational Creativity,” is not new, including the philosophical ideas surrounding the definition of creativity and whether it is possible to encode it [2]. Different levels of creativity have been identified: combinational creativity (creating something novel from the intersection of pre-existing ideas), exploratory creativity (finding valuable unexplored ideas within a

pre-existing conceptual space), and transformational creativity (having ideas outside of the pre-existing conceptual space) [2]. One of the foundational curiosities within the area of AI and music generation is whether a system can be made that exhibits any of these levels of creativity, and if so, which one.

Methods outside of deep neural networks have been used to generate music. Markov chains, as surveyed in [2], are probabilistic models known to be good at replicating syntax; however, generated outputs sometimes lack semantics, and higher order chains can risk plagiarizing pre-existing songs, rarely venturing beyond combinational creativity. Rules-based methods have also been investigated, though their tradeoff lies in the loosening of rules to allow for creativity, and the rules are limited by human imagination [2].

Recurrent neural networks (RNNs) and long short-term memory networks (LSTMs) are commonly used in Natural Language Processing applications and have a history in music generation, as surveyed in [2]. The issue with RNNs is that sometimes they do not remember the beginning of a melody by the end of generating it, which can limit the melody’s goal-oriented nature. They also suffer from the vanishing/exploding gradient issue, which LSTMs attempt to solve. For really long sequences, LSTMs are inferior to Transformers, which remember a global context and are the state of the art in Natural Language Processing. The focus of my approach is on melodies, not on composing a full song, which is why Transformers were not chosen. CNNs were chosen instead of LSTMs in my investigation because past work has been done with CNNs and music generation to suggest that CNNs are just as powerful as LSTMs, and also CNNs are relatively unexplored within this area which makes it an intriguing problem.

WaveNet is a CNN-based network that can generate raw audio waveforms, with applications in music and speech [3]. WaveNet uses dilated causal convolutions to capture the long-ranging sequential dependencies. The authors of WaveNet mention that “because models with causal convolutions do not have recurrent connections, they are typically faster to train than RNNs” [3]. The potential of WaveNet is



**Figure 2:** System diagram detailing the flow of data

that it can create new sounds; however, this is at the cost of the dataset taking up a large amount of memory. MIDI files are small in size so are more manageable to work with, which is one of the reasons they were chosen for my work in this paper. Additionally, audio also encodes performance and instrumentation, resulting in possibly unlimited versions of the same song. In this paper, I focus on the pure composition of music, removing the layer of performance.

WaveNet showed that CNNs are a viable method for generating music. Following WaveNet, the creators of MidiNet sought to use CNNs in a similar manner, but with a twist—the representation of the music is symbolic (MIDI), not audio waveforms—and it incorporates generative adversarial networks [4]. MidiNet only encodes pitches as being on/off (binary) and has the restriction that a maximum of one note can be generated at a time, emphasizing melody. In contrast, the work I am presenting encodes pitches in a ternary format (on/off/continued) and has no limit to the number of notes that can be played at a given time. MidiNet has the ability to impose a chord condition restriction to the net, allowing a melody to be composed within a certain chord progression. The authors of MidiNet conclude that a CNN-GAN based network can be a “powerful alternative to RNNs” [4].

Dilated convolutional networks with symbolic data have been investigated as well, with PianoNet [5]. PianoNet encodes music in a binary (on/off) representation and flattens the time and pitch dimensions into a one-dimensional array for training. One-dimensional dilated convolutions are used to encompass long-reaching dependencies, similar to WaveNet. This differs from my approach which

encodes the data in a ternary-manner and preserves the two-dimensional time-pitch format. The dilated convolutions I investigate are therefore two-dimensional, with the dilation rate not always being the same in each dimension.

### 3. PROCESS

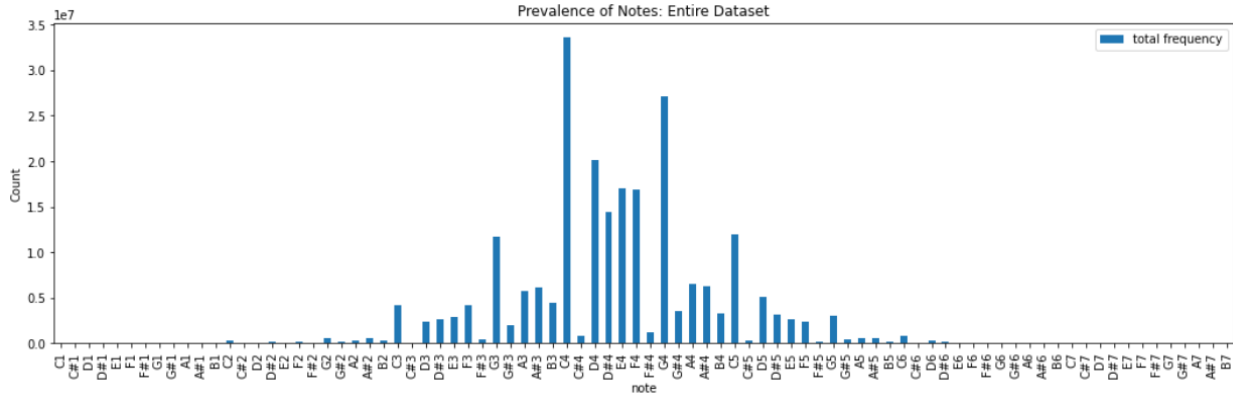
A system diagram is presented in Figure 2. It provides a general overview of the flow of data, from preprocessing to network training to music generation.

#### 3.1 Data Source

The raw dataset of MIDI files was acquired from Hooktheory’s TheoryTab Database [6] using a slightly modified version of a web crawler created by Wen-Yi Hsiao on GitHub under the project name “Lead-Sheet-Dataset” [7]. The raw dataset consisted of 17,216 song clips from 4825 artists. Only “no-key” (key of C) files were kept. A histogram detailing the distribution of pitches within the dataset can be seen in Figure 3 on the next page.

#### 3.2 Preprocessing

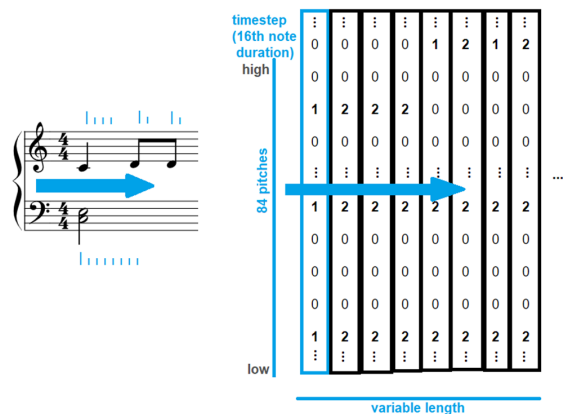
MIT’s Music21 Python library [8] was used to iterate over the melody and chord parts of each MIDI file. In this context, a timestep is defined as an array of size 84 representing 84 distinct pitches (7 octaves) with one 16th note duration, consisting of  $\{0, 1, 2\}$ , which respectively represent a rest, a pitch begins, and a pitch is held. For each note, chord, or rest object in the MIDI file, timestep(s) were created for that object’s duration, with subsequent timesteps receiving a 2 instead of a 1 due to the note being held



**Figure 3:** A histogram reflecting the prevalence of specific pitches within the dataset

to satisfy its complete duration. The two resulting sparse matrices (melody and chords) were combined by giving precedence to the melody if any notes overlapped, since this system is melody-focused.

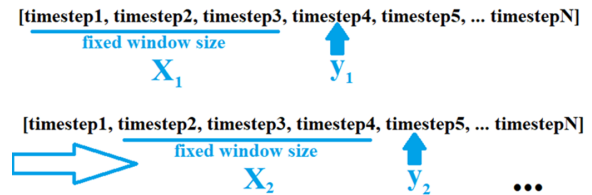
For example, in Figure 4 below, the melody and chord tracks for the music on the left are looped over independently. The index corresponding to the first note in the melody is looked up in a Python dictionary mapping pitch to index. The corresponding index in a timestep initialized with all zeros is assigned a 1. Because this first note is a quarter note, three subsequent timesteps are created, except a 2 is assigned instead of a 1 since the note is held across this portion of time. This process is repeated for the rest of the notes in the melody and for the chords track, and then the two matrices are combined to give the matrix located on the right side of Figure 4.



**Figure 4:** Encoding the MIDI file into a sparse matrix

After this sparse matrix was obtained, more preprocessing was necessary to yield an X, y pair

format that could be accepted by the network. The most logical choice for this was to let X be a sequence of consecutive timesteps and y be the subsequent timestep, in essence allowing the network to predict the next set of notes to be played (or continued), given all the notes that came before. In order to obtain these sequences, a window size had to be selected. For the current implementation, a window size of 4 measures (64 timesteps) was selected, since a typical melodic phrase tends to span this much time. It was conjectured that this size would encompass most of the melodic components, though future work could be done optimizing this hyperparameter. Padding consisting of 4 measures of rest was added to the front of each song to allow for the beginning patterns of a song to be learned by the network. The final dataset was then generated by sliding a fixed window over each song, generating (X, y) pairs, with the target being the next timestep after the 4 measures. An abstracted depiction of this process can be seen in Figure 5 below.



**Figure 5:** Dataset generation using a fixed sliding window

Throughout this process, memory management was key due to the large amount of data and limited computing resources available. To handle this, the data was broken into halves at certain points,

processed, and then later recombined. The final dataset was split 80%/20% train/test and then shuffled. To reduce contamination between the train and test sets, all sequences belonging to a certain artist were kept in the same set. The final training set consisted of 2,527,499 sequences and the test set contained 631,875 sequences. A depiction of the dataset can be seen in Figure 6 below.

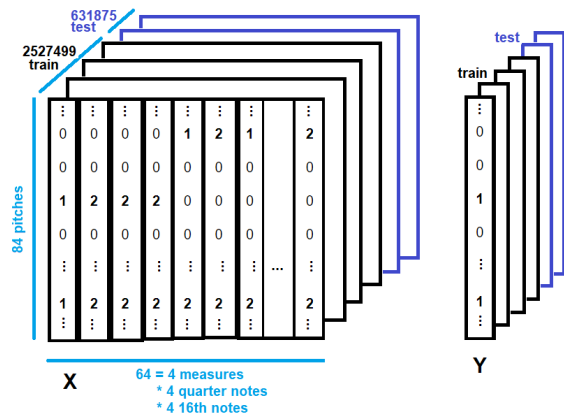


Figure 6: Final dataset after preprocessing

Note that later on, this process was repeated except with chords omitted to obtain a melody-only dataset as well.

### 3.2 Novel Loss and Activation Functions

While this is a clear classification problem, it is significantly different from a typical classification problem. A typical classification problem has a one-dimensional list of possible targets; for example, one might classify a picture of an animal as being a dog or a cat, or in hand-written digit recognition, the possible targets might be the numbers one through nine. With the problem presented in this paper, the target-space is two-dimensional—there are 84 notes and each could fall under three different categories: 0, 1, or 2. In the typical classification problem, the softmax activation function creates a probability distribution among its one-dimensional set of targets, and there is one most likely prediction. However, the problem presented in this paper requires a prediction between the categories of 0, 1, 2 for each of the 84 notes because more than one note could be played at the same time. Visualizing this as 3 channels for each of {0, 1, 2}, similar to a color picture’s RGB

channels, the goal was to take the softmax across these channels for each of the 84 pitches.



Figure 7: Flattening of the targets into a one-dimensional space

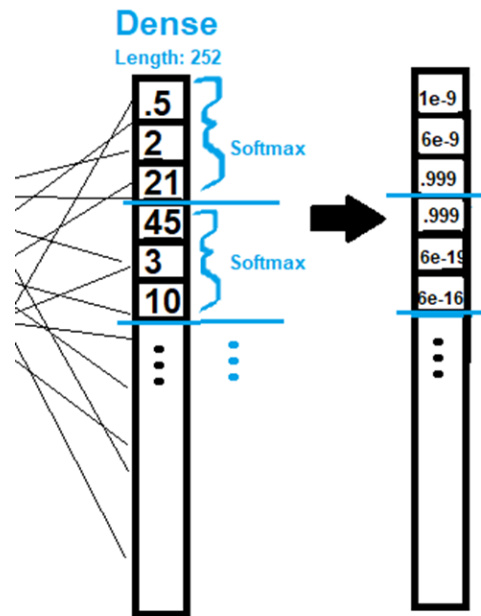


Figure 8: Skip-3 Softmax

Using TensorFlow Keras, the traditional softmax activation function did not work in this way. Thus, due to the nature of the unique encoding of the data, a novel approach was needed. The targets were flattened into a one-dimensional array of size 252 (84 \* 3), as can be seen in Figure 7. This translated to the last layer of the network being a fully-connected, flattened layer of 252 output nodes. Digging into the internals of Keras, a custom skip-3 softmax function was created that takes the softmax across every 3



output nodes, depicted in Figure 8 on the previous page. Additionally, in a similar manner, a custom skip-3 cross-entropy loss function was created that operates over every 3 entries (mimicking 3 channels), as well as a custom skip-3 accuracy. As far as the author knows, this is a novel approach in the domain of music generation due to this specific matrix representation of music.

### 3.3 Neural Network Architectures

Multiple neural network architectures were explored, beginning with simpler models. The first model consisted of several fully-connected layers and no convolutions. Its architecture can be seen in Figure 9 below. Note that the 64 in the input shape represents the 4-measure fixed sliding window (4 sixteenth notes in a quarter note \* 4 quarter notes in a measure \* 4 measures = 64 timesteps), and the 84 represents the 84 different pitches. This network, as well as all the others, use the custom skip-3 softmax activation function and skip-3 cross-entropy loss function. All the networks explored in this paper used the Adam optimizer and a learning rate of 0.00001, which was discovered through trial and error. The fully-connected network had 4,928,252 trainable parameters.

The second network, also seen in Figure 9, is a simple convolutional network consisting of one convolutional layer. No padding was used (termed 'valid' in Tensorflow) to avoid the false impression that some musical sequences taken from the middle of songs were preceded by rests. A kernel size of

(4,18) was used because 4 timesteps represents a quarter note duration, a slightly larger unit in the context of the melody, and 18 pitches spans a little more than one octave, possibly capturing that whole range. Max pooling was not used because it was conjectured that it might distort the encoding of the music. The convolutional network had 66,020,912 trainable parameters.

Both the fully-connected and the convolutional networks were trained with 500,000 sequences, a process that took several days. Due to limitations on available computing power, it was not possible to train any of the networks with the full dataset. The fully-connected and convolutional networks were trained using the melody and chords dataset.

The results of the fully-connected and convolutional networks indicated that some mechanism to capture longer-term dependencies might be necessary to produce more goal-oriented music; this led to the creation of a third type of network, the dilated convolutional network, shown in Figure 9 below. Dilated convolutions increase the size of the receptive field without increasing the number of parameters, and some past work has been done using dilated convolutions with music generation, so they seemed like a good next step. Exponentially dilated convolutions were used in several different network architectures, with one such example being shown in Figure 9 below. Instead of keeping the dilation rate the same in both dimensions, one dimension was doubled, and then the other, trading off to maximize

#### 1. Fully-Connected Network

```
model = Sequential()
model.add(Input(shape=(64,84,1)))
model.add(Flatten())
model.add(Dense(800,activation='relu'))
model.add(Dense(500,activation='relu'))
model.add(Dense(300,activation='relu'))
model.add(Dense(252,activation='skip3_softmax'))
```

#### 2. Convolutional Network

```
model = Sequential()
model.add(Input(shape=(64,84,1)))
model.add(Conv2D(20, kernel_size=(4,18),padding='valid',activation='relu'))
model.add(Flatten())
model.add(Dense(800,activation='relu'))
model.add(Dense(500,activation='relu'))
model.add(Dense(300,activation='relu'))
model.add(Dense(252,activation='skip3_softmax'))
```

```
model.compile(loss=skip3_cross_entropy,
optimizer=Adam(learning_rate=0.00001),
metrics=[skip3_accuracy])
```

#### 3. Dilated Convolutional Network

```
model = Sequential()
model.add(Input(shape=(64,84,1)))

model.add(Conv2D(4, kernel_size=2, padding='valid', activation='relu', dilation_rate=(1,1)))
model.add(Conv2D(8, kernel_size=2, padding='valid', activation='relu', dilation_rate=(2,1)))
model.add(Conv2D(12, kernel_size=2, padding='valid', activation='relu', dilation_rate=(2,2)))
model.add(Conv2D(16, kernel_size=2, padding='valid', activation='relu', dilation_rate=(4,2)))
model.add(Conv2D(20, kernel_size=2, padding='valid', activation='relu', dilation_rate=(4,4)))
model.add(Conv2D(24, kernel_size=2, padding='valid', activation='relu', dilation_rate=(8,4)))
model.add(Conv2D(28, kernel_size=2, padding='valid', activation='relu', dilation_rate=(8,8)))

model.add(Conv2D(32, kernel_size=2, padding='valid', activation='relu', dilation_rate=(1,1)))
model.add(Conv2D(36, kernel_size=2, padding='valid', activation='relu', dilation_rate=(2,1)))
model.add(Conv2D(40, kernel_size=2, padding='valid', activation='relu', dilation_rate=(2,2)))
model.add(Conv2D(44, kernel_size=2, padding='valid', activation='relu', dilation_rate=(4,2)))
model.add(Conv2D(48, kernel_size=2, padding='valid', activation='relu', dilation_rate=(4,4)))
model.add(Conv2D(52, kernel_size=2, padding='valid', activation='relu', dilation_rate=(8,4)))
model.add(Conv2D(56, kernel_size=2, padding='valid', activation='relu', dilation_rate=(8,8)))

model.add(Conv2D(60, kernel_size=2, padding='valid', activation='relu', dilation_rate=(1,1)))
model.add(Conv2D(64, kernel_size=2, padding='valid', activation='relu', dilation_rate=(2,1)))
model.add(Conv2D(68, kernel_size=2, padding='valid', activation='relu', dilation_rate=(2,2)))
model.add(Flatten())
model.add(Dense(252, activation='skip3_softmax'))
```

Figure 9: Three deep neural network architectures that were explored

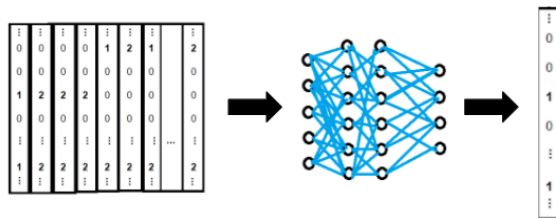
the number of convolutional layers. This idea that a large number of layers might produce better results comes from the idea that perhaps music composition consists of a large number of simpler functions, instead of a small number of complex functions, an idea from the literature [5]. The dilation rate expands up to a max of (8,8) before starting back over at (1,1). This was done to maximize the number of layers. The final layer of the network is fully-connected so that it can fit the format of the skip-3 softmax. This network had 722,224 trainable parameters.

The dilated convolutional network was trained using the melody-only dataset in an attempt to simplify the problem somewhat. It was trained with 500,000 sequences because the same limitations on computing power were in place as before.

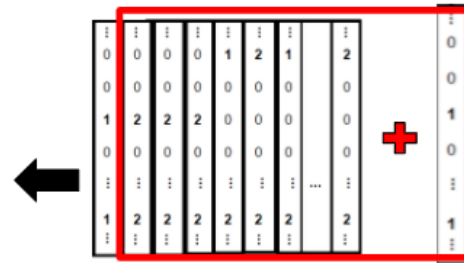
Early stopping with a patience of 5 epochs, as well as model checkpoints, were used for training all the networks. Additionally, a batch size of 32 was used. The number of epochs was set to a large number, 100, so that if necessary the model could train that long.

### 3.5 Music Generation

Music was generated by the network in the following manner: first a seed was chosen and given to the network's predict function, as seen in Figure 10. This seed could be any piece of music, or non-music (i.e. random noise), as long as it was 64 timesteps long. If it was shorter, it could be front-padded with silent timesteps. The network then produced a prediction, the next timestep. The seed was shifted to the left by one timestep, and the prediction was appended to the end, as shown in Figure 11. This process was then repeated for as many timesteps as the user wanted generated.



**Figure 10:** Predicting a seed's next timestep

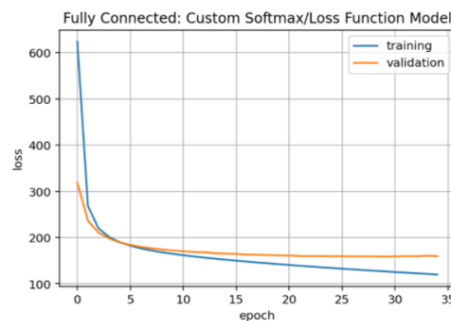


**Figure 11:** The seed was shifted to the left by one timestep and then the prediction was appended to the end.

The output from this process was a matrix, which is difficult to interpret on its own. Using the Music21 Python library and a process that was similar but the reverse of the earlier MIDI-encoding, this matrix was converted into a Music21 Stream, which could then be exported as a variety of formats, such as XML or MIDI. Music21 also allows audio playback of a Stream or visualization as sheet music.

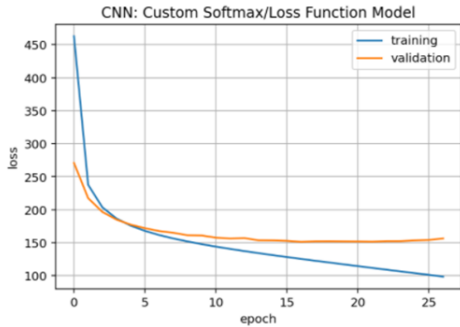
## 4. RESULTS AND DISCUSSION

The fully-connected and convolutional networks were trained with 500,000 sequences for several days, using the settings and hyperparameters described earlier. They stopped training after about 30 epochs, due to early stopping. Their associated learning curves can be seen in Figures 12 and 13.



**Figure 12:** Learning curves describing the training of the fully-connected network





**Figure 13:** Learning curves describing the training of the convolutional network

It is worth mentioning the idea of accuracy here. In some ways, it does not make sense to talk about accuracy—the purpose of this network is to produce music, a creative entity that walks the line between predictability and unpredictability. Should the network be rewarded for being predictable? It comes down to an ethical dilemma. Music is structured, so the network needs to learn to imitate, but is there a point at which it should be rewarded for making the wrong choice? In what weights is creativity encoded, or is it encoded at all? These are some of the questions that must be addressed in the future of research in this area. The networks trained in this paper are rewarded for having smaller losses, which makes the assumption that small loss equals better music. It is speculated that this might be true in terms of generating less random music, but as for the creative aspect, perhaps a different kind of loss function could be future work. The accuracies, computed as how many pitches’ {0, 1, 2} values were predicted correctly, were consistently approximately 98% or higher, even on the test set, across the different neural networks trained in this paper. This is somewhat uninterpretable for a few reasons. First of all, the baseline accuracy is unknown; however, it is most likely extremely small, given there are 84 possible notes and any number could be played or continued at the same time. Second, if most of the notes are not played, then it might be fairly easy to predict all the 0’s, getting the majority of the pitches’ states in a timestep correct. As a result, in the future, perhaps a different kind of accuracy could be created and computed to give less weight to getting 0’s correct.

Trial	Seed	CNN Result	Analysis	Fully-Connected Result	Analysis
A	"Twinkle Twinkle Little Star" 		Repeats seed's chord progression, but no melody. Last note resolves.		Frequent pauses, more melodic than CNN. Last note resolves.
B	Chords only (front-padded with rests) 		Keeps a consistent rhythm (no "melody" just chords)		Keeps a consistent rhythm, but doesn't resolve.
C	Unresolved Scale (stops on the leading tone) (front-padded with rests) 		Continues the leading tone, then resolves on the tonic. Silence, then plays the dominant (scale degree 5)		Continues the leading tone, then silence. Doesn't resolve.
***Note that any silence observed in the last 2 seconds of these mp3s is the result of the mp3 file format, not the actual song clip.					

**Table 1:** Results of generating music from various seeds for the convolutional and fully-connected networks that were trained on melody and chords data. Audio can be listened to at the following link: <https://docs.google.com/presentation/d/1NsCZnTX6somk5MIOIKw0xEs-CxUSWZsucA3OvpRLvBO/edit?usp=sharing>



**Figure 14:** (a) “Twinkle Twinkle Little Star” seed (b) CNN’s generated output using this seed

The main way the networks were assessed was by choosing seeds at random from the test and train sets, as well as by targeting specific examples outside those sets, and generating music for 64 timesteps. Note that this process was a manual one. It was not feasible to listen to ~3 million possible results by hand. Thus, the analysis given in this paper is based strictly on the examples that were looked at by hand; while it is believed these are representative of the entire set of possible results, it is still possible this might not be true. A few examples of generated music are analyzed in Table 1, with audio available at the provided link. These examples include a common tune, a sequence with only chords, and an unresolved scale, the latter of which is used to assess the tendencies of the network’s pitches, since in music theory some pitches tend to resolve on certain pitches more often than others, a foundational concept popular music is built upon. A sheet music example

of a seed and its generated output are shown in Figure 14.

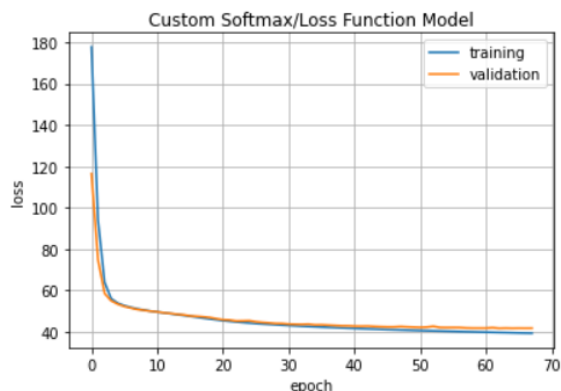
The following are some strengths that were observed among these fully-connected and convolutional networks. Both networks seem to generate continuing notes that belong to the same chords that were in the seed. In other words, it doesn't sound entirely random. All the chosen notes are within the same range as the seed (they are not extremely high or low in comparison). Both networks seem to continue or create rhythms on beat, fairly consistently. There is evidence of rhythmic patterns. There are hints of chord progressions, some continued exactly from the seed, and the networks seem to be good at endings, often resolving on the tonic.

On the other hand, there are some weaknesses: Both networks seem to want to end the song. Often, the generated music consists of a somewhat resolved ending to the seed, followed by silence that keeps being generated if it is left to keep generating. This can lead to abrupt, near-sighted endings. Sometimes it seems like the noise becomes amplified the longer the model generates a continuation. Both models tend to favor chords over melody. Sometimes there appears to be no sign of a melody, aside from the chords. As for creativity, the generated music seems to barely grasp musical intuition. Perhaps creativity is an emergent property that comes about when the basics are in place; thus, it is probably too premature to attempt to assess creativity in this context. At this point, the networks do not seem to be able to replicate a melody, which makes it difficult to analyze melodic properties. The generated music seems to depend largely on the seed given, with results varying widely, and sometimes the generated notes sound "wandering" and aimless. Sometimes silence is generated.





The dilated convolutional network was trained in a similar manner, but with the melody-only dataset in an attempt to simplify the problem. Again, 500,000 sequences were used due to limitations on available computing power and memory. A graph of the learning curves is shown in Figure 15.

A similar process of randomly choosing seeds was used to evaluate the dilated convolutional model. Of the results that were analyzed by hand, most of the generated outputs were a single continuing note or silence. There were a few interesting examples,

which can be listened to at the provided link in Table 2's caption.



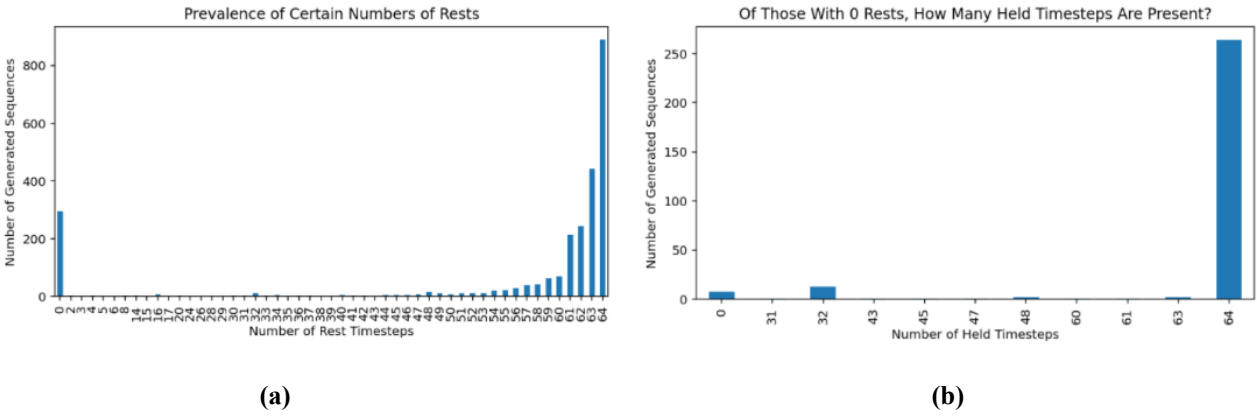
**Figure 15:** Learning curves describing the training of the dilated convolutional network

Interesting Results:	
Seed	Result
The Beach Boys: "I Get Around" 	 Rhythms on same note
Half of "Happy Birthday" 	 Rhythms on same unresolved note

**Table 2:** A few select results of generating music using the dilated convolutional network that were deemed intriguing due to rhythmic interest. Audio can be listened to at the following link: <https://docs.google.com/presentation/d/1NsCZnTX6somk5MIOIKw0xEs-CxUSWZsucA3OvpRLvBQ/edit?usp=sharing>

In an effort to assess the network's overall statistical performance in an automated way, approximately 2500 musical outputs were generated using seeds taken from the training set. Statistics including numbers of rest timesteps, held timesteps, and beginning timesteps, as well as numbers of different pitches used, were recorded for later analysis. A histogram was created in Figure 16a on the next page describing the number of generated sequences having specific numbers of rest timesteps. As can be seen from the figure, the majority of the sequences were primarily silent. Another peak occurs with about 300 of the 2500 sequences having no rests.

Another histogram was made analyzing this subgroup and can be seen in Figure 16b. Of those non-silent generated sequences, the majority are entirely held,



**Figure 16:** (a) Histogram describing the number of generated sequences with certain numbers of rests. (b) Histogram describing the number of generated sequences with certain numbers of held timesteps in the subgroup that had no rests.

meaning they continue the last note of the seed indefinitely. There are smaller peaks at 0 and 32 held timesteps. This indicates the prevalence of some generated sequences with continuous sixteenth notes or continuous eighth notes, possibly like the results in Table 2. Perhaps this is representative of some statistic in music where if there are 0 rests, then the music’s purpose is purely rhythmic, keeping a consistent beat. It is also interesting that there are few odd numbers of held timesteps, which might indicate syncopation or unique rhythmic patterns that could be associated with a melody—however, these could still exist in the minority, having rests mixed in and being part of the more general Figure 16a above.

Overall, these two histograms confirm my original by-hand findings that most of the generated sequences from the dilated convolutional network were silent or consisted of a held note. Other dilated convolutional networks with slightly different architectures yielded similar results.

Because the results of the dilational neural network were significantly different from the results of the convolutional and fully-connected networks, the impact of the dilated convolutions is unknown. The switch to a melody-only dataset was a major change made at the same time as the switch to the dilated convolutions. To isolate the relative impacts of these two changes, earlier models would have to be trained with melody-only, and the dilated convolutional network would have to be trained with the melody and chords dataset.

## 5. CONCLUSIONS AND FUTURE WORK

In conclusion, a unique way of encoding MIDI files to capture the on, off, and held states was created for use in deep neural networks. Due to this unique encoding, a novel skip-3 softmax activation function and skip-3 cross-entropy loss function were developed. The current results of generating music, given a seed, using a fully-connected network, a convolutional network, and a dilated convolutional network show some evidence of rhythmic and harmonic patterns, but lack melodic elements. The effect of using dilated convolutions remains unknown.

Much future work could be done to explore these remaining unknowns. As previously stated, training the dilated convolutional network with the melody and chords dataset might allow the impact of dilated convolutions to be seen. If the dilated convolutions do not remedy the network’s nearsightedness, other techniques such as attention could be explored. Additionally, the fully-connected network and the convolutional network could both be trained with the melody-only dataset to focus more on melody-generation. Another aspect that was not investigated in this paper was using different fixed window sizes; perhaps a number other than 64 timesteps would be better-suited to the nature of these networks. Given more time, memory, and computing power, the entire dataset could be used in training instead of only 500,000 sequences. Furthermore, results of the convolutional network could be explored in more depth by looking at the particular

weights of each filter to try to figure out what properties of music theory the network is learning.

Now that the musical pipeline is fully established, there are some other specific areas for improvement along this pipeline. The development of a different kind of loss function and accuracy that takes into account the prevalence of zeros in the sparse matrices as well as the tradeoff between rewarding predictability and unpredictability could be a whole other project in itself.

Assuming this project were worked on for much longer, to the point that human-plausible melodies were being created, then a system could be built around the melody-generation one to compose large pieces of music. Melodies are building blocks that can be used to create a larger whole, and perhaps this has a method to it that neural networks could learn.

Can machines compose music? Can they pass the musical Turing test? Only time will tell.

## 8. REFERENCES

- [1] Bharucha, Jamshed J. and Peter M. Todd. "Modeling the Perception of Tonal Structure with Neural Nets." *Computer Music Journal*, 1989, vol. 13, no. 4, pp. 44-53.
- [2] Carnovalini, Filippo and Antonio Rodà. "Computational Creativity and Music Generation Systems: An Introduction to the State of the Art." *Frontiers in Artificial Intelligence*, 2020, vol. 3, no. 14.
- [3] Oord, Aaron van den, et al. "Wavenet: A Generative Model For Raw Audio." 2016.
- [4] Yang, Li-Chia, et al. "Midinet: A Convolutional Generative Adversarial Network For Symbolic-Domain Music Generation." *ISMIR*, 2017.
- [5] Angsten, Thomas. "Generating Piano Music with Dilated Convolutional Neural Networks." *Towards Data Science*, 2020.
- [6] <https://www.hooktheory.com/theorytab/artists/>
- [7] <https://github.com/wayne391/lead-sheet-dataset>
- [8] <http://web.mit.edu/music21/>
- [9] Chuan, Ching-Hua and Dorien Herremans. "Modeling Temporal Tonal Relations in Polyphonic Music Through Deep Networks with a Novel Image-Based Representation." *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.
- [10] Elbayad, Maha, et al. "Pervasive Attention: 2D Convolutional Neural Networks for Sequence-to-Sequence Prediction." 2018.
- [11] Briot, Jean-Pierre, et al. "Deep Learning Techniques for Music Generation—A Survey." 2019.
- [12] Briot, Jean Pierre and François Pachet. "Music Generation by Deep Learning—Challenges and Directions." 2018.
- [13] Fernández, Jose David and Francisco Vico. "AI Methods in Algorithmic Composition: A Comprehensive Survey." *Journal of Artificial Intelligence Research*, 2013, vol. 48, pp. 513-582.
- [14] Ariza, Christopher. "The Interrogator as Critic: The Turing Test and the Evaluation of Generative Music Systems." *Computer Music Journal*, 2009, vol. 33, no. 2, pp. 48-70.
- [15] Chuan, Ching-Hua. "From Context to Concept: Exploring Semantic Relationships in Music with Word2Vec." 2018.