Learning Classes of Melodies with a Recurrent Neural Network

David D'Annunzio and Kenneth Ning

May 10, 2012

Department of Computer Science, Swarthmore College

Abstract

In this project, we create Elman-style recurrent neural networks that effectively learn certain classes of melodies, and then use this training to create new, similar melodies. The networks are trained on five different song inputs for each of the three melody classes we are considering: nursery rhymes, folk songs, and blues. We evaluate the neural networks based on their predictive performance on test songs as a quantitative measure. As a qualitative measure, we analyze the melodies created by these networks using common music theory techniques.

Keywords: Elman; neural networks; melody

1 Introduction

Evolutionary computing has had a vast impact on machine learning and adaptive algorithms. A research subfield has emerged, titled evolutionary music or computer music, that attempts to use these algorithms to solve problems within the music domain. These aspects of machine learning have been implemented in a variety of ways to solve music-related problems, such as composition, transcription, and improvisation. Our paper will focus on the task of using a recurrent neural network to create compositions that are unique, structured, and informative.

Chun-Chi and Miikkulainen use recurrent neural networks to create melodies in a style that is similar to the compositions of 20th century composer, Béla Bartók. In their experiment, the range of musical input spans three octaves and there are five different rhythmic notations to choose from. Pitches are encoded using the relative pitch paradigm, where each node in the neural network corresponds to interval steps away from a given reference point. Fitness of a melody is constrained based on several composition rules from music theory and other rules that are specific to Bartók's compositions. Overall, the evolved melodies responded well to the constraints and the generated melodies were fairly diverse.

However, computer-generated music and the proper encoding of music is still a point of contention. In Mozer's experiment, many different aspects of music representation are taken into consideration regarding the construction of their CONCERT architecture. Similar to our experiment, CONCERT is a recurrent network that attempts to predict classes of melodies. It uses this network to compose something in a similar style. Mozer implements a fairly complex method of music representation that includes information on pitch, duration, and underlying chord structure. This architecture was tested on Bach compositions, European folk songs, and waltzes. Overall conclusions suggested knowledge of local themes that reflected the input space, but there was general disappointment with the global coherency in composed melodies. We will see many similarities between Mozer's analysis and the analysis of our composed melodies.

Franklin's paper takes on more of a comparison approach between different recurrent networks and music representations. However, the music domain considered in this paper is jazz-oriented with musical tasks such as learning chord tones, improvisation techniques, and a structured melody. Network architectures that were compared included Mozer's, an Elman net, and an LSTM (Long Short-term Memory) recurrent network. The music representations compared included a binary representation, similar to our representation, Mozer's method, and a newly developed method coined as Circle of Thirds. The authors argue that LSTM combined with their developed Circle of Thirds representation produces the best results in their tasks. Franklin's comparison approach demonstrates the wide range of approaches in solving problems in the music domain. As opposed to some of the other papers seen, Franklin's experiment is also more grounded in quantitative analysis (using error as a measure) than in qualitative analysis (determining how good a melody sounds).

The recurrent networks we are using are Elman nets. Regarding music representation, we use a binary representation where our encodings of melodies are constructed line by line, each line corresponding to a note in the melody. Each line is a sequence of 0's and a 1 placed in the position that corresponds to the correct note in the scale, which happens to be one of the representations mentioned in Franklin. However, we are restricting the pitch range of our inputs to an octave (i.e. eight notes), and we are not considering different rhythmic values for notes. Although our implementation is a simplification of some of the experiments discussed, our hope is that this simplified approach will provide more concise, comprehensible insight into the workings of the neural networks. Additionally, we anticipate seeing similar results documented in the other papers.

Section 2 discusses basic music theory that will be helpful in the discussion of the results. Section 3 discusses the architecture of our recurrent neural network. Section 4 discusses the collection of data. Section 5 details each of the three experiments on the different classes of melodies. In Sections 6 and 7, we analyze the melodies that were constructed by the neural network and discuss future work.

2 Music Theory

This project explores how to form different types of melodies from specific scales. A **scale** is a collection of musical notes that correspond to the **key** of the melody. For this project, every created melody was played in the key of C.

We will sometimes refer to notes in the scale not by their number, but by their solfège label. For the major scale, starting at scale degree 1 and ending at scale degree 8 the solfège labels are: $do, re, mi, fa, sol, la, ti, do$. do is used twice because the 1st and 8th scale degrees are the same note just an octave apart. The **tonic** of a scale is the 1st scale degree, *do*. The **dominant** is the 5th scale degree, *sol*.

3 Elman Networks

An Elman network is a special type of neural network that simulates memory. The way the network incorporates memory is by having recurrent connections, where nodes' outputs connect to their inputs. Elman networks have each hidden node output connect back to its input, giving a temporal memory of the output at the previous time step.

Our recurrent neural networks generally consist of eight input nodes, eight output nodes, and a variable number of hidden nodes.[1] The number of input and output nodes corresponds to the number of notes available in the scale being use. Figure 1 shows an example recurrent neural network.

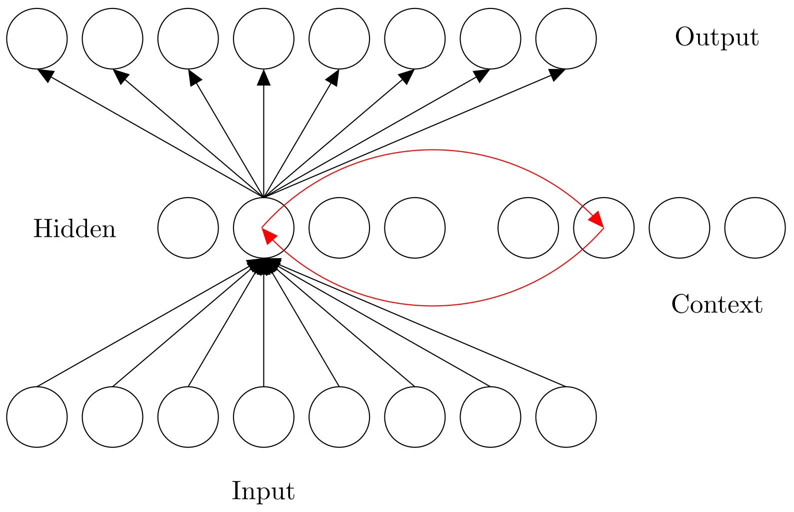


Figure 1: Elman Net Diagram. There are eight input and output nodes, and four hidden nodes. We also have an extra layer called the context layer. This layer stores the values of each hidden node from the previous time step. The context layer is the same size as the hidden layer and each hidden node has one corresponding context node.

4 Data Collection

Each test song was encoded by hand into a simple format readable by the neural networks. Figure 2 shows an example song before the encoding process. Figure 3 shows the same song after encoding.

[1]The number of hidden nodes varied based on our experiments

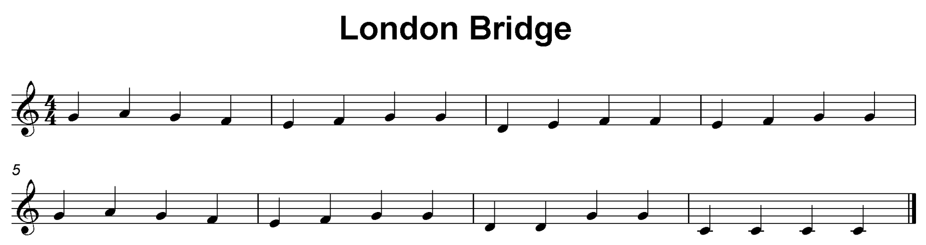


Figure 2: London Bridge Sheet Music

```
0 0 0 0 1 0 0 0        0 0 0 0 1 0 0 0
0 0 0 0 0 1 0 0        0 0 0 0 0 1 0 0
0 0 0 0 1 0 0 0        0 0 0 0 1 0 0 0
0 0 0 1 0 0 0 0        0 0 0 1 0 0 0 0
0 0 1 0 0 0 0 0        0 0 1 0 0 0 0 0
0 0 0 1 0 0 0 0        0 0 0 1 0 0 0 0
0 0 0 0 1 0 0 0        0 0 0 0 1 0 0 0
0 0 0 0 1 0 0 0        0 0 0 0 1 0 0 0
0 1 0 0 0 0 0 0        0 1 0 0 0 0 0 0
0 0 1 0 0 0 0 0        0 1 0 0 0 0 0 0
0 0 0 1 0 0 0 0        0 0 0 0 1 0 0 0
0 0 0 1 0 0 0 0        0 0 0 0 1 0 0 0
0 0 1 0 0 0 0 0        - - - - - - - -
0 0 0 1 0 0 0 0        - - - - - - - -
0 0 0 0 1 0 0 0        - - - - - - - -
0 0 0 0 1 0 0 0        - - - - - - - -
```

Figure 3: London Bridge Data File

Each number in a row indicates what note is active at time t.[2] Given an input song and its key, we can succesfully encode the notes of the song. In the example above, we encode the notes based on the C major scale: C, D, E, F, G, A, B, C. From the encoded version we see the active note at the first beat is the fifth scale degree, indicated by the 1 at the fifth position in the row. This is G, which is also shown in the sheet music. The next note is A, and therefore the encoded file shows the sixth scale degree as active. We can continue this process until the end of the melody at which point we have encoded the song.

In terms of meter, each row corresponds to a quarter note. If a melody contains a note longer than a quarter note, we interpret it as a sequence of quarter notes. In Figure 3, rows 7 and 8 are the subdivision of the half note into quarter notes. Lyrically, these rows correspond to the "down" in "Lon-don Bridge is fall-ing *down*". We also do not take rests into account.

5 Experiments

Nursery Rhymes

We trained the nursery rhyme neural networks on five input songs: Frere Jacques, Mary Had a Little Lamb, London Bridge, Row Row Row Your Boat, Twinkle Twinkle Little Star.

[2]More than one note could be active at the same time, resulting in possible chord structures, but for our project we limited the network to single note melodies.

The average length of each input song was around 30 notes. In creation, we had the neural networks create 32 note songs to maintain similarity with the inputs. The neural networks do not have a sense of time signature, but we transcribed every created melody as a 4/4 time signature.

During training of the nursery rhyme networks, we used cross validation. The reason to use cross validation is to lessen the risk of overtraining. Neural networks are intended to generalize to inputs they have never seen before, but sometimes with overtraining, the nets will only perform well for inputs they were trained on.

Cross validation works by providing two different data sets to the neural network, a training set and a cross validation set. The neural net trains on the training set, but instead of evaluating the finishing criteria on the error in the training set, we evaluate the finishing criteria based on the error in the cross validation set. The neural net never trains on the cross validation set, so using it as an error measure lets us see how well the network is generalizing. Once the error in the cross validation set stops decreasing, we stop training. Our cross validation set consisted of three songs: Pop Goes the Weasel, Yankee Doodle, Hot Cross Buns.

We created three separate networks where we performed 5 sweeps, 10 sweeps, or 20 sweeps on each individual song. We did this in a serial manner, clearing the context layer of the network when switching songs. After the individual song sweeps, we calculated the error in the cross validation set. We took the error in each song and then averaged the values. If the error was the lowest we had seen, we would save the weights as the current best network weights. Then we added the difference between the last cross validation error and the current cross validation error to a list. This was a 10 item list that kept track of the error difference between loop iterations. Initially, the error differences were negative because the network was getting better at predicting the songs. But as time continued and we approached the point of possible ovetraining, the network got worse at prediction and the error differences became positive. When the sum of this vector became positive we stopped training. Below we show simplified pseudocode for the training loop.

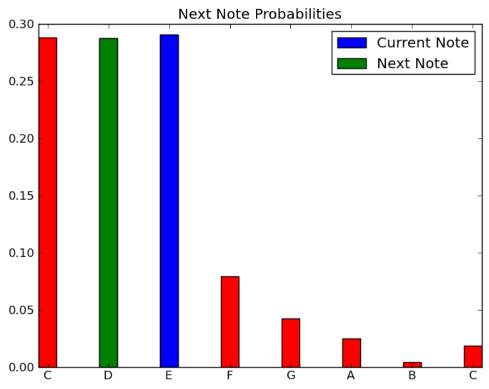
```
while still training
    train 5/10/20 sweeps on Frere Jacques, clear context layer
    train 5/10/20 sweeps on London Bridge, clear context layer
    train 5/10/20 sweeps on Mary Had a Little Lamb, clear context layer
    train 5/10/20 sweeps on Row Row Row Your Boat, clear context layer
    train 5/10/20 sweeps on Twinkle Twinkle Little Star, clear context layer

    perform cross validation
    determine if still training
```

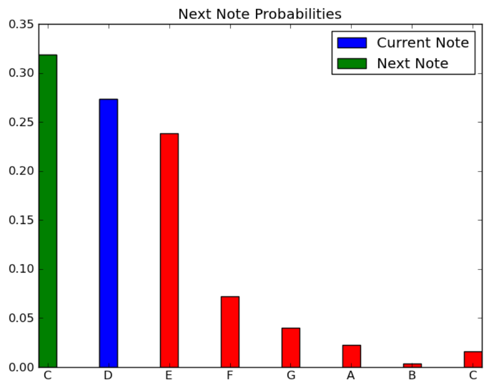
Once a neural network was trained we began song creation. Song creation began by giving the network a starting note, a vector of 0s with one 1, similar to the rows in Figure 3. We propagated this input through the networks and got an output of activation values, between 0 and 1. We normalized these values so that they summed to 1 and gave us a valid probability distribution. We then used roulette wheel selection to choose our next note based on this probability distribution. Once a note had been chosen, it became the next input to the network. This process was repeated until the desired number of notes had been generated. Figure 4 shows example normalized activation values from an example prediction run.

For each melody, we stored the chosen notes and wrote them to a .wav file. For each neural network we created six melodies, three with a starting note on the 1st scale degree and three with a starting note on the 5th scale degree.[3]

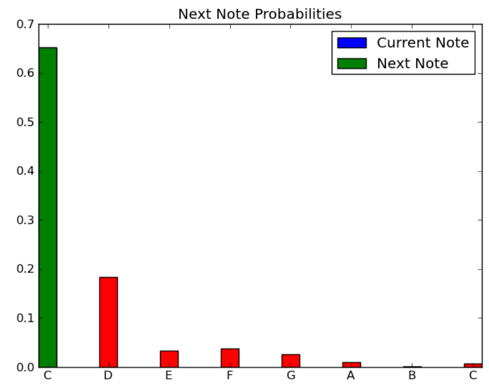
[3]To listen to created melodies visit www.sccs.swarthmore.edu/users/12/davidd/evolutionary_music



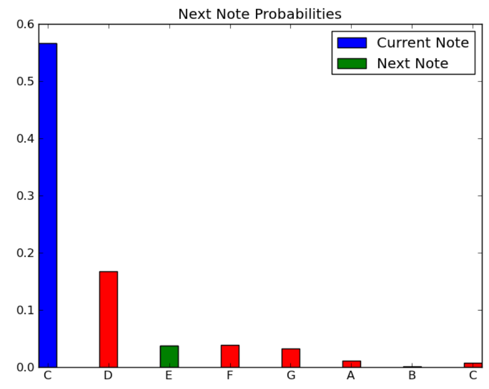
(a) Graph 1



(b) Graph 2



(c) Graph 3



(d) Graph 4

Figure 4: Activation Value Graphs for “Hot Cross Buns”. We give the starting note from “Hot Cross Buns” to the neural network, which outputs a vector of activation values, shown in Graph 1. This lets us view how well the neural net predicts the next note. Graph 2 shows the activation values after passing the second note, and so on.

Our test set contained the following three songs: Hot Cross Buns, Itsy-Bitsy Spider, I'm a Little Teapot.

Originally, we kept the number of hidden nodes at four and varied the number of individual song sweeps. The following tables show results based on three quantitative measures. The first measure is Euclidean distance. The output of our neural nets is a vector of values between 1 and 0, acting as the probability distribution from which to choose the next note. The true value is a vector of seven 0s and one 1, indicating which note is actually played next. We can take the Euclidean distance between the output of the network and the target output as a measure of prediction success.

If our target vector is denoted as

$$\mathbf{t} = \{t_0 \quad t_1 \quad t_2 \quad t_3 \quad t_4 \quad t_5 \quad t_6 \quad t_7\}$$

and our output vector is denoted as

$$\mathbf{o} = \{o_0 \quad o_1 \quad o_2 \quad o_3 \quad o_4 \quad o_5 \quad o_6 \quad o_7\}$$

then our Euclidean distance is

$$dist = \sqrt{(t_0 - o_0)^2 + ... + (t_7 - o_7)^2}$$

We summed up the distance between every pair of target and output vectors, and then divided by the number of notes in the song to get the average distance (Table 1). We also calculated the distance between the actual target note and its corresponding probability. If the probability of the target note was 0.6, then the distance would be 0.4. We summed this single note distance for every pair of target and output vectors and again took the average (Table 2). Finally, we calculated the percentage of time where the probability of the target note was above 0.4 (Table 3). We performed the same error analysis while varying the number of hidden nodes. The results are shown in Table 4, 5, and 6.

	Training sweeps per song		
	5	10	20
Hot Cross Buns	.7381	.7378	.7266
Itsy-Bitsy Spider	.7642	.7703	.7663
Teapot	.8314	.8253	.8271

Table 1: Euclidean Distance; 4 hidden nodes

	Training sweeps per song		
	5	10	20
Hot Cross Buns	.5996	.6043	.5901
Itsy-Bitsy Spider	.6622	.6620	.6664
Teapot	.7455	.7446	.7437

Table 2: Distance between actual note and probability; 4 hidden nodes

	Training sweeps per song		
	5	10	20
Hot Cross Buns	.2667	.2667	.4000
Itsy-Bitsy Spider	.2295	.2131	.2623
Teapot	.1875	.1406	.1406

Table 3: Percentage of time target note probability is above 0.4; 4 hidden nodes

	Hidden Nodes				
	5	6	7	8	16
Hot Cross Buns	.741	.664	.701	.642	.635
Itsy-Bitsy Spider	.759	.740	.727	.747	.727
Teapot	.818	.778	.752	.758	.762

Table 4: Euclidean Distance

	Hidden Nodes				
	5	6	7	8	16
Hot Cross Buns	.599	.515	.516	.477	.464
Itsy-Bitsy Spider	.652	.612	.578	.614	.548
Teapot	.729	.646	.623	.601	.599

Table 5: Distance between actual note and probability

	Hidden Nodes				
	5	6	7	8	16
Hot Cross Buns	.267	.600	.533	.667	.667
Itsy-Bitsy Spider	.246	.443	.508	.410	.525
Teapot	.219	.391	.406	.469	.391

Table 6: Percentage of time target note probability is above 0.4

Because of time constraints, we did not perform the same quantitative analysis for the folk song genre or the blues genre.

Folk

We chose the following folk songs from *Popular songs of nineteenth-century America* by Jackson.

- Camptown Races
- Nelly Gray
- Old Black Joe
- Oh Susanna
- Long Long Ago

The average length of these songs was around 60 notes, so we chose 64 notes, another multiple of four, as our melody creation length. Unlike the nursery rhymes, we did not use cross validation. We trained using individual sweep values of 5, 10, and 20.

Blues

We chose blues songs from two sources, *Folk Blues* by Silverman and *Blues: An Anthology* by Handy.

- B♭ Shuffle
- Loveless Love
- Skinner Blues
- Depression Blues
- Prison Bound

Each input was 96 notes (12 bar phrase with 8 note subdivision), so we chose to mimic this in creation. We trained these in the same way that folk was trained, without cross validation.

The blues scale networks contain 11 input and output nodes as opposed to the 8 from nursery rhymes and folk songs. This is because the blues scale contains three extra "blue" notes. These notes are the flat 3, flat 5, and flat 7. When a note is flat, it is a half-step lower in pitch than its non-flat counterpart. In Figure 5, the three flat notes are E♭, G♭ and B♭. The natural versions of these notes, E, G and B, are notated with the ♮ symbol. The scale is shown in the sheet music below. The ♭ symbol identifies the blue notes.

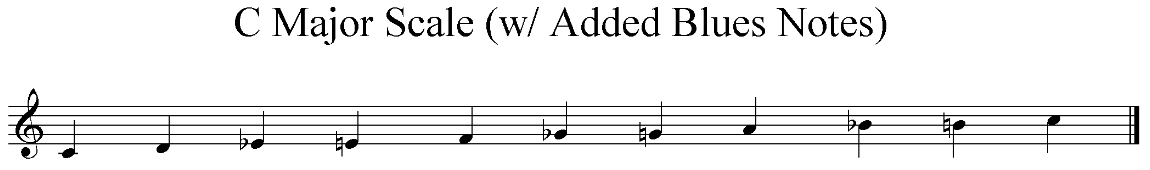


Figure 5: The Blues Scale

6 Discussion

6.1 Quantitative Results

When varying the number of training sweeps per song and maintaining the number of hidden nodes constant, we found very little difference in the resulting error in all three of our error metrics.

When varying the number of hidden nodes and performing only 1 training sweep per song, we had more noticeable decreases in error. Table 4 shows that the Euclidean Distance between the neural net output and the target vector decreased by 14%, 4%, and 7%, for Hot Cross Buns, Itsy-Bitsy Spider, and I'm a Little Teapot respectively. This decrease occurred when varying the number of hidden nodes from 5 to 16. We found similar trends in the Table 5, using the distance from the single active target note.

Our third error metric also supported the idea that having more hidden nodes means better prediction. Table 6 showed an increase in the average times the probability of the next target note was above 40%. For Hot Cross Buns, Itsy-Bitsy Spider, and I'm a Little Teapot, we saw an increase of 150%, 113%, and 34% respectively.

While these results give us a better idea of how well the networks predict notes in a song, it is difficult to say conclusively if this is a desirable attribute. Predicting a melody is not the same as creating one. Maybe by adding hidden nodes, we hurt the networks' ability to generalize. We want the neural nets to learn about genre specific qualities that are found in many songs. In addition, our data sets were small. If we increased the number of songs in all three data sets, we could get a more representative picture of the nursery rhyme genre and better evaluate the predictive skills of the created networks.

6.2 Qualitative Results

Figures 6, 7, and 8 are examples of a nursery, folk, and blues melody (respectively) that were composed by their respective neural networks. From a musical standpoint, each of these melodies contains interesting aspects that help illuminate the compositional process of the neural network.

The nursery rhyme melody in Figure 6 consists of themes that are common from a music theory perspective. For example, the melody finishes with a *2-5-1 progression*, which is a common line in music theory that involves those three degrees of the scale. In Figure 6, measures 7-8, the 2-5-1 progression is D-G-C. The ending line of "London Bridge" is also an example of a 2-5-1 progression. Additionally, the melody is primarily tonic-based, with the neural network consistently choosing tonic notes or notes around the tonic. However, there are interesting melodic jumps from the 3rd degree to the 8th degree (in measures 2 and 3), suggesting some unpredictability and creativity in the melody-creation process.

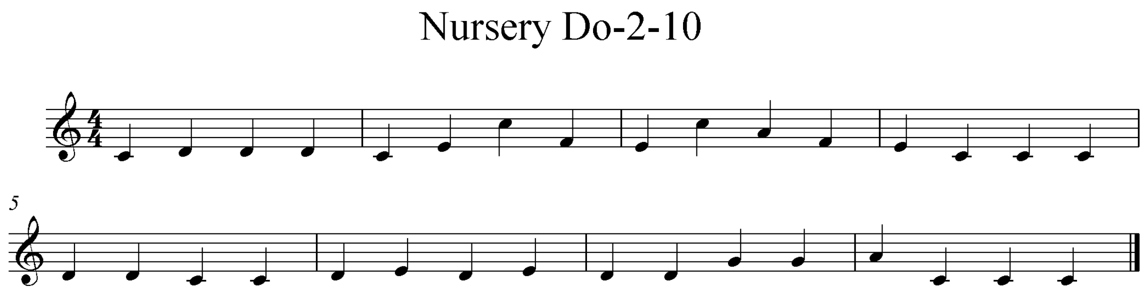


Figure 6: Composed nursery rhyme melody, with *do* starting note, 2nd run, and 10 sweeps

The folk melody in Figure 7 is slightly longer and more sophisticated. More interesting melodic themes appear; for example, in measures 8-10 and 14-16, a "mirror" pattern emerges where a descending scale is followed by the same notes, but ascending. A major triad theme (i.e. selecting the 1, 3, and 5 scale tones in a line) also appears in measure 2. In terms of note choice, the folk melody is primarily tonic and dominant based, which is a reflection on the overwhelming presence of those scale tones in most folk songs.

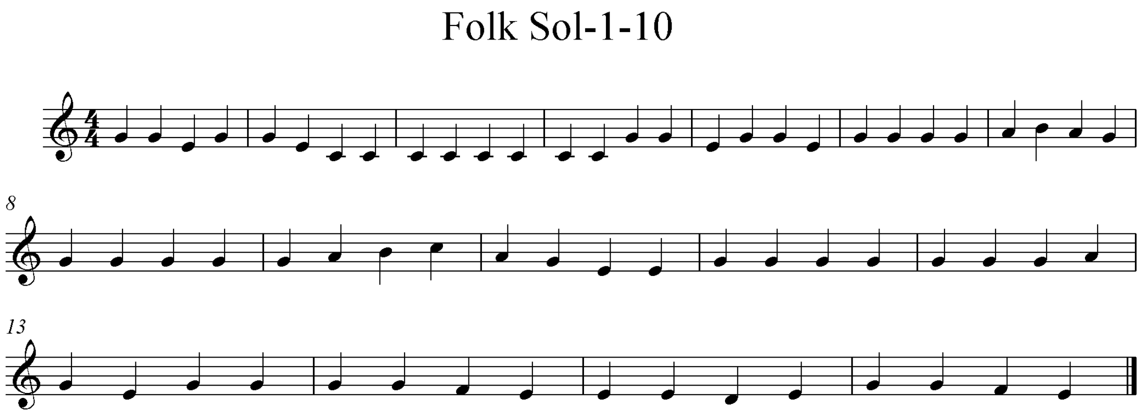


Figure 7: Composed folk melody, with *do* starting note, 1st run, and 5 sweeps

The blues melody in Figure 8 is perhaps one of our most unpredictable and unique melodies. Out of all of our blues compositions, this melody selected the most number of blues notes. Essentially, this melody contains lines of repeated notes (typically the tonic and dominant) punctuated by moving lines that consist of blues notes. Though these lines are sometimes quite complex, (e.g. measures 19-21), they do not sound very coherent and are most likely the result of some of the unpredictability in the neural network. Additionally, most of our blues melodies did not have a sense of when the actual blues “chord changes” occurred, suggesting a lack of global coherency in our composed melodies.

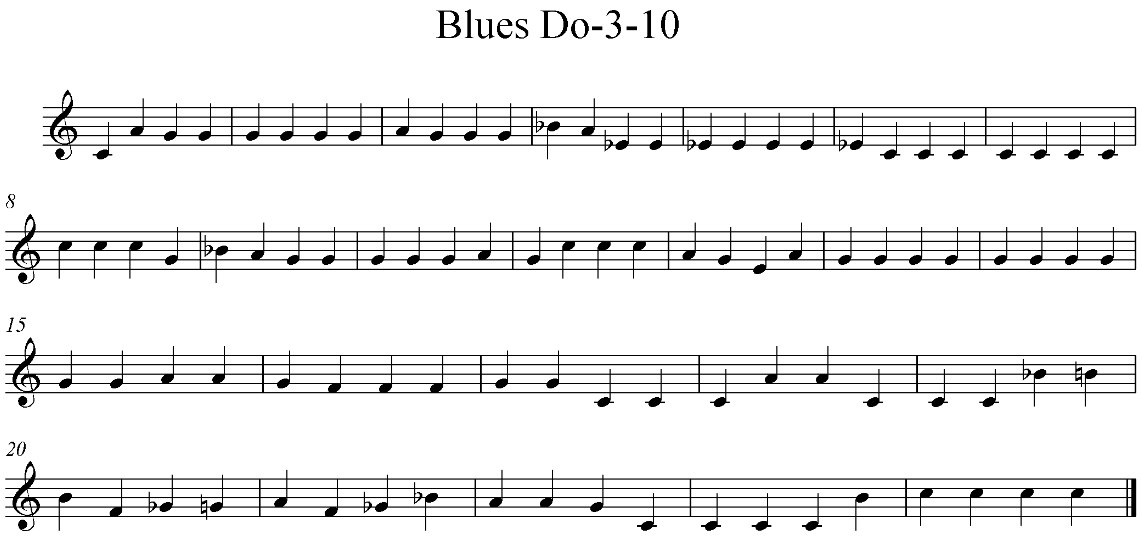


Figure 8: Composed blues melody, with *do* starting note, 3rd run, and 10 sweeps

These melodies generally had a local sense of memory, and many of these themes reflected musical characteristics of its input set. For example, tonic and dominant scale tones were overrepresented in almost all of our melodies in all genres, given their dominating presence in most of our input melodies. Additionally, the melodies were even able to learn some more sophisticated characteristics of the class of melodies such as incorporating melodic lines, the 2-5-1 progression, and selecting blues notes. Because the neural network selects the next node in the melody from a probability distribution, there is also a sense of unpredictability and randomness in the composed melodies that adds to the uniqueness of the melodic lines.

However, we should reiterate that our compositions are generally short-sighted. A given melody may sound pleasant at a specific section, but the networks lack ability at creating globally coherent composition. In other words, the networks need to improve on choosing notes that make the overall piece sound like a song, as opposed to just certain sections sounding like songs.

7 Future Work

While our results are novel and fairly interesting in a musical analysis sense, there is still work to be done to make our approach more robust. An immediately obvious challenge is to complexify the input and output melodies by adding in other factors such as note duration, rests, knowledge of time signatures/meter, and adding multiple notes at one time step (i.e. creating chords). This may require re-evaluating our current music representation format and consider alternatives that encompass more than the pitch of a melody.

A related challenge to consider is expanding our sample input space. Because of the restrictions imposed by our music representation, only certain melodies qualified as input candidates. Expanding our music representation complexity would allow inclusion of more melodies. Unfortunately, there would still be the labor-intensive issue of encoding these melodies. If there existed a large enough database of encoded melodies, or if we were able to develop an autonomous encoding algorithm, then we could potentially have a very rich, representative sample space of melodies for a given genre. Given this larger input space, the networks might be able to obtain an even more comprehensive grasp of a class of melodies.

Perhaps the most prominent issue to confront in this experiment, as well as in most other music compositional experiments, is the problem of global coherency. In this experiment and the others mentioned, the network is capable of reflecting local themes in the input space, but it is very incompetent at creating a melody that makes sense from a big-picture perspective. Experimentation with more hidden nodes, a different recurrent neural network, or a different musical representation may have promising results.

However, from a more theoretical standpoint in algorithmic music composition one must carefully consider the danger in having too many and too few music theory based restrictions. As Werner and Todd suggest in Biles's *Evolutionary Computer Music*, “More structure and knowledge built into the system means more reasonably structured musical output; less structure and knowledge in the system means more novel, unexpected output, but also more unstructured musical chaff” (p. 15). Thus, there are much larger questions for the music community to consider as a whole, regarding the appropriate balance between music theory and spontaneity in our music compositional paradigms. This larger discussion may become relevant in the further development of our project.

References

[1] M. C. Mozer. Neural network music composition by prediction: Exploring the benefits of psychoacoustic constraints and multiscale processing. Connection Science, 1994.

[2] J. A. Franklin. Recurrent Neural Networks for Musical Pitch Memory and Classification. Journal on Computing, Vol. 18, No. 3, Summer 2006, pp. 321-338.

[3] C. Chen and R. Miikkulainen. Creating Melodies with Evolving Recurrent Neural Networks. In Proceedings of the Internacional Joint Conference on Neural Networks, IJCNN 01. p.2241-2246, Washington, DC 2001.

[4] E. Miranda and J. Biles. Evolutionary Computer Music. Springer-Verlag London Limited 2007.

[5] Jackson, Richard. *Popular Songs of Nineteenth-Century America.* New York: Dover Publications Inc., 1976.

[6] Handy, W.C., ed. *Blues: An Anthology.* New York: The Macmillan Company, 1972.

[7] Silverman, Jerry. *One Hundred and Ten American Folk Blues.* New York: The Macmillan Company, 1958.