

# Blues Improviser

Greg Nelson  
([gregoryn@cs.utah.edu](mailto:gregoryn@cs.utah.edu))

Nam Nguyen  
([namphuon@cs.utah.edu](mailto:namphuon@cs.utah.edu))

Department of Computer Science  
University of Utah  
Salt Lake City, UT 84112

## Abstract

Computer-generated music has long been an area of research. It is one subject that ties together two very different fields of academia: Computer Science and Music. We focus our study on a very specific format of jazz music: the twelve-bar blues in the key of C following a well known chord progression. This was chosen mainly because of the nature of blues music; that is, musicians typically invent melodies on the spot, rather than recite previously written ones. We set out to get a computer to do just that: improvise a blues solo. There have been a wide variety of ways to approach the problem, and this study examines two in particular: probabilistic ordering of notes and sequences of notes, and applications of neural networks.

## Introduction

There have been many different ways to attempt to create a machine that can generate music. There are as many techniques as there are kinds of music to create. This study is limited to the twelve-bar blues in the key of C-major driven by a well known chord progression: I I IV I IV IV I I V IV I V. The rhythm section (drums, bass, guitar/piano) is prewritten and fixed, and the study is further limited to just the solo voice of the generated song (piano, saxophone, trumpet, etc.). Actually, the walking bass line is also somewhat randomly generated as it just picks notes from the C-myxolydian scale, a common technique among jazz bassists. The crux of our study focuses on whether sequences of notes are better at capturing the structure of Blues music than single notes. As noted by Eck and Schmidhuber (2002), the Long Short Term Memory Neural Network model has shown the best results in learning the structure of Blues music. By using sequences, we hope to gain that same improvement using sequences. We even further restrict our study to just the ordering of the notes, and ignore duration. A method of generating the durations to be assigned to each note is developed that is similar to the NoteTable to be described below.

As specific as it is, we believe this creates quite a general foundation upon which a computer can create something novel; after all, blues music has thrived for a long time even though most of it doesn't wander far from this very foundation. Generating something novel is not enough, however, as a random sequence of notes chosen from the C-blues or C-myxolydian scales might actually sound decent. But to generate something that is quite beautiful is the challenge. Simply measuring the beauty of the output poses one of the greatest challenges.

## **The Alan Turing Test**

We propose a variation on one method for measuring the results. The Alan Turing test of Artificial Intelligence is basically as follows: a human examiner tries to discern whether the individual she is interacting with (probably by means of a computer terminal) is human or a computer. Both want to convince her that it is the human. The examiner is free to ask any questions she pleases. The machine is said to display characteristics of true artificial intelligence if it succeeds in fooling the examiner into thinking it was the human. We propose a variation on this test as follows. The examiner listens to two blues solos: one generated by a machine, and the other improvised by a musician (the musician should probably be chosen to be at the same musical "level" as the machine, so in our case, a beginner). The examiner is asked to decide which is which. In our case, the answer is all too obvious, usually, so we time the examiner on how long it takes her to make her decision. The longer the time, the better the results. This test is obviously very flawed in its bias, so the only measure of results that can really be trusted is your own judgment. Links are provided for each generation.

## **Giving a Machine an Ear for Music**

To objectify art is almost blasphemous to some! However, we believe that there is some objectivity behind it all. What makes a blues solo really sound good? What is it about a musician that makes someone say, "He's got soul!" There must be some universal or almost-universal structure behind the scenes that he is exploiting. Some of this universal structure has been discovered, such as frequency overtones. This paper, however, does not attempt to explain or uncover more in the field of psychoacoustics. We attempt to extract structure through other more indirect means.

How exactly does a musician learn to play something that sounds good? While it is the opinion of the author that much of his talent is innate, there still must be some things that are learned! Any musician will tell you that a good way to learn how to play good music is to listen to good music. We design our software to follow the same sort of "training" procedure. The methods of doing this will be explained in more detail later as we explicate the algorithms for generating the melodies.

## The JFugue MIDI Package

We used JFugue to create the MIDI files. JFugue is a Java API for programming MIDI. Its use of strings to represent MIDI events makes music programming very easy. It creates a layer of abstraction that removes the need to tinker with the MIDI byte codes. For example, to play a C note, you basically say `play("C")`. JFugue allows you to do just about anything possible with MIDI. David Koelle wrote JFugue, and more information can be found at <http://innix.com/jfugue>.

## The BLIMP package

The BLues IMProviser package contains a class called Blues which ties together many different MusicObjects to create a song. MusicObject is an interface, and any class that implements it represents one voice in the song. Such a class must implement a method which returns a JFugue string. In our package, the Drums, Bass, and Rhythm are all pre-written, so these classes aren't of much interest. The Melody class is the core of the project. There were many different generations of this particular class, and we describe each of them. It should be noted that when we say that we "trained" something on a song, we really mean that we extracted the solo instrument from the song, transposed it to the appropriate key, and trained it on the resulting string of notes. This is analogous to a human listener picking out the soloist, not paying much attention to the background, and being able to extrapolate the techniques she learned from the solo to other key signatures (a task which remarkably doesn't require much conscious effort on her part).

## Generation One: Random

This melody class is a base case against which all other generations are compared. It basically picks note values from the C-Blues scale in octaves 5, 6, and 7 randomly, although some effort is made to make sure the notes that are temporally close are also somewhat close in proximity (so as not to sound like a child banging violently on a keyboard). This is an important generation, because it keeps the listeners in check. Sometimes it helps to be reminded what a random output sounds like, because it is tempting to say that any novel melody is a positive result!

No training was done on this generation.

The songs generated by this generation are not completely astounding, and an example can be heard here: <http://www.cs.utah.edu/~gregoryn/blimp/g1-1.midi>

## Generation Two: NoteTable

The NoteTable class represented a 128x128 probability table. Each row represents the previous note and each column represents the next note. The entries represent the probabilities that the note corresponding to the column should follow the note corresponding to the row. (See Figure A). Since the table is essentially able to remember the previous note (and only the previous note), this algorithm is said to have a memory distance of one. In general with tables of this kind, an n-dimensional table will have a memory distance of (n-1). Since most blues phrases are at least four measures, capturing the “global” structure of such melodies might require a memory distance of anywhere from four to four thousand, depending on the song at hand. Creating an n-dimensional table requires memory space proportional to  $128^n$ . Similar methods have been tried before (Dodge & Jerse, 1985; Mozer, 1994). An alternative to probability tables, the SequenceGraph, is described below.

The (2-dimensional) NoteTable was trained by extracting probabilities from scales, chords, intervals, and songs.

Songs generated by this generation were only slightly better than generation one; they can be heard here: <http://www.cs.utah.edu/~gregoryn/blimp/g2-1.midi>

| <i>Current Note</i> | <i>Probability of Next Note</i> |          |          |          |
|---------------------|---------------------------------|----------|----------|----------|
|                     | <b>A</b>                        | <b>B</b> | <b>C</b> | <b>D</b> |
| <b>A</b>            | 0.25                            | 0.25     | 0.25     | 0.25     |
| <b>B</b>            | 1.0                             | 0        | 0        | 0        |
| <b>C</b>            | 1.0                             | 0        | 0        | 0        |
| <b>D</b>            | 0                               | 0        | 0        | 1.0      |

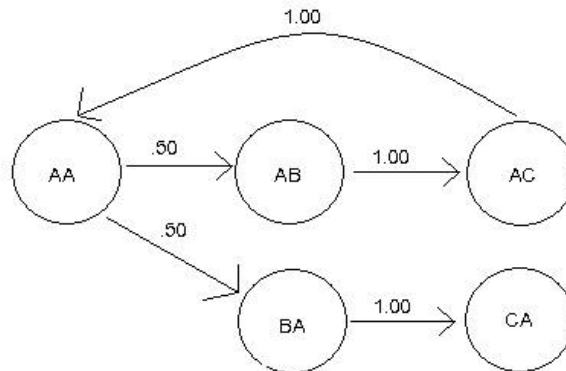
**Figure A**

The sequence AABACADDD would be parsed into something like this NoteTable. Notice that D will loop forever.

## Generation Three: SequenceGraph

The SequenceGraph is simply a directed graph. Each node represents a sequence of notes. The edge weights connecting one node to another represent the probability that that edge should be chosen as the next hop. A path through the graph would result in an ordering of sequences. This makes things much more flexible and general. A sequence of notes is treated as an atomic object, and the algorithm attempts to order the sequences rather than just notes. This technique essentially allows for a variable memory distance (equal to the sequence length or the number of notes in a sequence), but in our study we usually restricted it to two, three, four, or five. Figure B shows an example of a sequence graph of length two. A SequenceGraph made up of sequences that have a sequence

length of one would be identical to a NoteTable.



**Figure B**

The sequence graph of length 2 for the sequence AAABACAA

The SequenceGraph was trained similarly to NoteTable: sequences were extracted from scales, chords, and songs, and were compiled into a graph. The results were quite good, and examples can be heard here <http://www.cs.utah.edu/~gregoryn/blimp/g3-3.midi>

## Generation Four: NeuralNetwork

### The JOONE Package

This class was implemented using JOONE: Java Object Oriented Neural Engine by Paolo Marrone. More information can be found at <http://www.joone.org>. The network is a recurrent neural network containing four layers: one input, two hidden, and one output. The network uses a sigmoid function for the nodes' activation functions, and backward propagation for the learning algorithm. Input consists of a number in [0 1] that represents a sequence. Exactly how a number in [0 1] represents a sequence is described below.

### A Sequence Code

A sequence (like in the SequenceGraph) is an arbitrary ordering of an arbitrary number of notes. Notes are viewed as integers between 0 and 128. The sequence is basically viewed as a base-128 number which is plugged into the sine function, spitting out a number in [0 1]. These numbers are “semi-one-to-one” with the sequences, and one such number is henceforth known as the sequence's “code”. This choice of sequence representation turns out to be very counter-productive, and we discuss this problem in more detail later.

## The SequenceCollection

When the network is trained on a particular song, the sequences that are parsed out of the song are put into a set called SequenceCollection. Then, the network creates songs by choosing sequences from this set.

### Utter Failure

There are several incarnations of NeuralNetwork. The first uses sequences of length one, the second uses sequences of length two, the third of length three, and so on until the eighth neural network engine. For all of these networks, the input is the previous sequence, and the output is a number in  $[0, 1]$  that is rounded to the nearest sequence existing in the SequenceCollection set. After several attempts at training and tweaking, the results are dismal. Every network converges to a local minimum. In other words, the network will always output the same sequence, over and over again.

### Discussion

The depressing results come as no surprise, however; at least, in retrospect. Designing a Neural Network to train on a set of floating point numbers (which reflect very little about their sequence counterparts), asking it to look for patterns, and then expecting it to compose music is quite ignorant. The humor in this fallacy was pointed out by one of the authors with the following analogy.

It's like giving a beginner band student a sheet of paper with a bunch of numbers written on it, and asking him to study it very closely for a very long time, looking for patterns, without actually informing him that it has anything to do with music! Then, furthermore, we give him an arbitrary number and ask him what should follow. So he gives us the best answer he can come up with, and we smack him upside the head! That's not very fair.

The failure of the network and the specific analyses of what could have caused it are discussed in more detail below.

The first culprit is the scaling function used to scale numbers to the interval  $[0, 1]$ . Sequences similar to each other, such as AABB and ABBB, could have codes that are far apart. This fails to capture any structure behind the sequences. In fact, the codes only encode sequences of integers, which themselves encode notes on a keyboard, which itself encodes frequencies in a spectrum, and so on. A lot is lost in all this encoding! As noted by Thom (1995), there is little relationship between every pair of chords. The same insight can be applied to sequences of notes. In fact, chords at least have *some* structure to them; sequences might not have any at all! There are myriad notes that can come after any given note. This hinders the use of discrete learning algorithms and neural networks.

Many representations have been proposed that capture a lot more behind the essence of a note, such as its pitch, harmonics, distance from other notes, etc... One ingenious method of called the PHCCCF representation presented by Shepard (1982) and utilized by Mozer (1994), incorporates a great deal of the structure behind the music. Using this sort of representation and generalizing it to sequences of notes would give much better results. If one is to expect a machine to pick out patterns related to the natural structure of music, one must first allow the machine to be aware of the structure!

The second culprit is the structure of the neural network. As noted by Russell and Norvig (1995, the art of choosing the number of hidden units needed to solve a given problem is not well understood. Even as we increase the number of hidden nodes, the results converge.

The third culprit is what is fed into the input layer of the network. We have one sequence as input. There may be two (or more) possible sequences that could be expected to follow it, and this causes the network to not be sure of what to output. The network attempts to minimize the error and it outputs a number between those two possibilities. After thousands of cycles of doing this, the network will always output numbers close to 0.5; it plays it safe by always being in the middle.

Attempts made to force the network to remember more and to take more input still result in utter failure. The songs can be heard here: <http://www.cs.utah.edu/~gregoryn/blimp/g4-1.midi>, notice it is the same sequence over and over again.

## **Future Research**

We believe that by working with sequences of notes rather than just the notes themselves reveals more about the global structure of a piece of music. A note is nothing without a context. The next step is to find a better way to integrate sequences with neural networks. We think that networks designed to deal with sequences would have a big improvement over networks designed to deal with single notes. Care must be taken to capture as much of the hidden structure of music as possible. Sequences must be represented in a clear and precise way. Accomplishing this last feat is a field of research in and of itself.

There are other improvements in the implementation of sequence algorithms as well. An example might be to allow sequences of various lengths. Also, the manner in which the sequences are parsed from the training set could be altered. And finally, it might be advantageous to input a single sequence into a more carefully designed neural network, and allow the network to output just a single note. We believe that there is a lot to be accomplished in the application of neural networks to computer blues improvisation.

## Conclusion

The most important finding of this study is that manipulating sequences of notes instead of single notes leads to better machine-generated blues solos. This might be because more of the structure of the song is captured this way. In the future, greater effort should be given to carefully constructing a representation of music that will capture as much of the structure of the music as possible. In doing so, the machine will be more “aware” of the underlying structure which we as humans take for granted.

## References

- Dodge, C., & Jerse, T. A. (1985). *Computer music: Synthesis, composition, and performance*. New York: Shirmer Books.
- Eck, D. and Schmidhuber, J. (2002). *A First Look at Music Composition using LSTM Recurrent Neural Networks*. Technical Report No. IDSIA-07-02, Istituto Dalle Molle di studi sull' intelligenza artificiale, Manno, Switzerland.
- Mozer, M. C. (1994). *Neural network composition by predication: Exploring the benefits of psychophysical constraints and multiscale processing*. *Cognitive Science*.
- Russell, S. and Norvig, P. (1995). *Artificial Intelligence: A Modern Approach* (pp. 579). Prentice-Hall, Inc.
- Shepard, R. N. (1982). *Geometrical approximations to the structure of musical pitch*. *Psychological Review*, 89.
- Thom, B. (1995). *Predicting Chordal Transitions in Jazz: The Good, the Bad, the Ugly*. IJCAI-95, Music and AI Workshop, Montreal, Canada.