An Evolved Neural Network/HC Hybrid for Tablature Creation in GA-based Guitar Arranging

Daniel R. Tuohy and W.D. Potter
Artificial Intelligence Center, University of Georgia
daniel2e@uga.edu

Abstract

In this paper we describe a technique for creating guitar tablature using a neural network. Training data was parsed from an online repository of human-created tablatures. The contents of both the input layer and the set of training data have been optimized through genetic search in order to maximize the accuracy of the network. The output of the network is improved upon with a local heuristic hill-climber (HC). We implement this model in an existing system for generating guitar arrangements via genetic algorithm (GA). When compared to the original system for generating tablature, we note modest improvement in tablature quality and drastic improvements in execution time.

1 Introduction

Tablature is a notational system that describes to a guitarist how a piece of music should be played. Because notes can be played at several different positions on the instrument, playing directly from sheet music requires tedious decision making about where best to play each note. Tablature frees the performer from this responsibility by displaying fretboard finger positions in lieu of actual notes. We employ an evolved neural network model with local heuristic search for the purpose of generating guitar tablature for a piece of guitar music.

We have implemented this model for tablature generation into a system for automatically arranging pieces of music not originally written for the guitar. The process of arranging frequently requires eliminating notes in the original composition so that the arrangement is not too difficult to perform. We use a genetic algorithm to find the most relevant selection of notes from the original piece that does not burden the performer excessively. In doing so, the GA requires our model for tablature generation in order to create tablatures corresponding to candidate arrangements. These tablatures are necessary in order to assess the difficulty of the arrangements.

In section two we describe previous work on both the task of tablature creation and that of arrangement generation. Section three details the evolved neural network model for generating tablature and a hill-climbing search for improving those tablatures. In section four we discuss the implementation of the tablature generation model as part of a genetic algorithm for automatically arranging music. Finally, we summarize the improvements the new model has made in terms of both speed and reliability.

2 Background

Generating tablature for the guitar that is competitive with tablature created by human experts is a difficult task. It has often been treated as an NP-hard graph-search problem, based on the research of Samir Sayegh (Sayegh 1989), (Radicioni and Lombardo 2005), (Radisavljevic and Driessen 2004). Another method used genetic algorithms to evolve playable tablature (Tuohy and Potter 2005). These models for tablature creation are dependent upon an evaluation function. Defining such a function for this domain is exceedingly difficult because of the innate complexity of the human hand (Heijink and Meulenbroek 2002). We conjecture that learning how to create tablature directly from human-defined tablature will yield a more reliable, human-like method.

We do, however, require an evaluation function to assess the difficulty of the tablatures generated by this model. Previously, we developed a GA for generating arrangements, specifically reductions, of music to allow the music to be played on guitar (Tuohy and Potter 2006b). This GA evaluates arrangements partially on the basis of the difficulty of the corresponding tablature. For the current project, we evaluate tablature difficulty with the function originally used for fitness evaluations in a GA for generating tablature. This function will also guide a brief local search on the tablatures generated by our neural network method.

3 Neural Network Model

The approach to tablature generation presented here constructs tablature sequentially with a neural network (i.e. one note at a time). A danger inherent in this approach of processing tablature sequentially is that when placing a note on the

fretboard, it is often beneficial to be aware of where subsequent notes will be played. With this in mind, we are careful to construct a set of network inputs that encompasses as much relevant information about the context of a note as is feasible. This includes data describing the notes both preceding and, crucially, those succeeding a note. We have chosen to implement a standard three layer feed-forward network with sigmoid activation function trained with error back-propogation.

The network has 20 outputs with a continuous range between 0 and 1. These correspond to 19 frets on the guitar and the open strings. The fretboard selected for a note by the network is the viable position nearest to the output with the highest value. Networks were constructed using a Neural Network package from the University of Georgia (Smith 2004).

3.1 Training Data

Our data are taken from a popular online library of tablature for classical guitar (Classtab.org 2006). We have selected excerpts from 75 different pieces of music, constituting a wide variety of styles and difficulty levels. We chose to use tablature as data because it is readily available in ASCII format, which allows for the generation of a large training base of patterns. It does however, deny us access to timing information, which is present in traditional sheet music but not in tablature notation. From the 75 excerpts we generated 1853 patterns, one for each note and corresponding fretboard position.

3.2 Network Inputs

A training pattern corresponds to information about the context of a single note and consists of 64 inputs. 41 of these pertain to notes played previous to or simultaneously with the note being processed. For each pattern, we generate the number of notes in the current chord, a number between one and four corresponding to the "octave" of the current note, a number between one and 12 corresponding to the notes C through D#, the octave and note of the highest and lowest notes in the chord, the average fret of the notes in the previous three chords, the fret and the string used the last time the note occured in the piece, the fret and string of the last four notes, the number of chords played since the last four notes were played, the number of notes in the previous two chords, a binary value for each string indicating which strings are already in use in the current chord, and all of the possible frets and strings of the current note.

23 additional inputs comprise data pertaining to notes that have yet to be assigned fretboard positions. These are the number of notes in the next two chords, the octave and note of the next six notes, the number of chords between the current note and the next six notes, and the fret and string furthest from the guitar body of the nearest subsequent limiting note. A limiting note is one which can only be played in one position, or can only be played on the sixth fret or higher. In this way, the network has information about future notes which require the hand to be at or above a certain place on the fretboard. The final input indicates how many chords remain until the next limiting note must be played. These inputs are all those which we thought could possibly be informative.

3.3 Evolving the Network Inputs and Training Data

By "evolve", we do not mean that the weights of our network are optimized with genetic search (Whitley 1995). Weights are set using a standard back-propogation algorithm. Rather, genetic search is used to select from among all possible inputs those which are truly informative, and to select from 75 possible excerpts which 65 result in the most accurate trained network when used as training data. We are, in effect, evolving the input layer of the network as well as the data set on which the network is trained. Irrelevant inputs and noisy training data can potentially confuse a neural network, and we have no reason to believe that our data are devoid of either. We use GAs to optimize bit strings denoting which inputs and which training excerpts are used. We have done so using GAs based roughly on Whitley's GENITOR model (Whitley and Kauth 1988). We have opted to use GAs rather than less computationally intensive methods because a network can be fully trained in less than three minutes.

The fitness of a bit string is the accuracy of the corresponding trained network. Accuracy is the percentage of fretboard positions selected by the network that coincide with those in published tablatures. To determine which fretboard position is selected by the network, we first determine which output node has the highest value. The fret corresponding to this node is then compared with all viable positions for that note, and the position nearest to this fret is considered to be the output of the network.

Each network is trained until no decrease in the mean squared error on an independent test set is found in 400000 learning events, calculating error at intervals of 200 events. One GA was used to optimize a bit string of length 64, corresponding to the 64 possible inputs. Another optimized a bit string of length 75, with the requirement that the bit string denote the inclusion of at least 65 excerpts as training data.

4 Implementing the Neural Network for Tablature Creation

In order to generate a tablature from a sequence of notes, we generate all of the data comprising an input sequence for the first note in the piece. These data are run through the feed-forward network and the node with the value closest to one is

taken to be the output. The fretboard position appended to the tablature is the one nearest to the output of the network. This process is repeated for every note in the piece of music, in sequence.

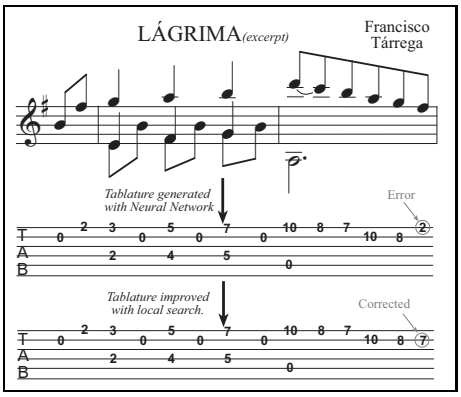

Figure 1: A tablature is created by neural network. The last note is "fixed" with local search.

Chords are processed recursively. Because the network could make a mistake in placing the first notes it encounters, we recursively generate every possible chord configuration that can be reasonably played. The chord inserted into the tablature is the one which is most consistent with the network's output. We judge consistency to be the cumulative difference between the assigned fretboard positions and the output of the network.

4.1 Improving network-generated tablatures with hill climbing

After evolving both the input layer and the training set, our network has an accuracy of 94.0% on the 65 excerpts selected via genetic search. This indicates that for 94.0% of the notes were assigned the same position on the fretboard that was present in the published tablature. This degree of consistency is such that the network reliably produces tablature similar or identical with that created by humans. However, the network is susceptible to rare cases of bad judgement, and tablatures occasionally include an obviously misplaced note or chord.

To correct this, we run a brief local search that passes over each note in the generated tablature a single time. For each note or chord, every possible fingering configuration is tested to see if it improves the tablature. To test improvement, we use an evaluation function previously used as the fitness function in a GA for tablature creation. A complete description of this evaluation function, which was optimized with a meta-GA to generate tablature consistent with published tablature, can be found in Tuohy and Potter (2006a). We only pass over the tablature one time so that the neural network's influence is not over-powered by that of the evaluation function. We have found that using the evaluation function for brief hill-climbing fixes obvious mistakes, but using it for a full hill-climbing optimization decreases consistency with published tablatures. Of particular interest is that even a single pass of the hill-climber decreased overall consistency on our data, although only slightly. This suggests that the network, despite it's occasional mistakes, is more adept at creating tablature than an algorithm for optimizing the evaluation function. However, we must acknowledge that hill-climbing is indeed useful because it corrects obvious mistakes. Inconsistency with published tablature that is introduced by the local optimization tends to be more harmless than the mistakes that are corrected. Figure 1 shows an example of a tablature generated for a piece using the neural network, with one of the fretboard positions "fixed" using local search.

5 Using generated tablatures in arrangement evaluation

In Tuohy and Potter (2006b) we describe a genetic algorithm that automatically generates arrangements for a piece of music. Fitness is a function of both playability (as defined by the aforementioned tablature evaluation function) and on "faithfulness" to the original composition. Briefly, "faithfulness" is determined on the basis of the number and importance of notes that are included. A note's importance is related to the part of the music it occupies. For example, notes in the melody are considered to be more important than those in the harmony. Fitness evaluation is also biased to promote continuity in moving lines, and to promote an appropriate distribution of notes throughout the arrangement (Corozine 2002), (Michael 2003). A complete mathematical account of this function can be found in Tuohy and Potter (2006b).

The playability of an arrangement is defined as the fitness of the corresponding tablature according to the same evaluation function we use for hill-climbing. Using orchestral scores as the source of potential notes, we then generated arrangements of excerpts from *Symphony No. 2* by Rachmaninoff, *Arioso* by Bach, and *Jupiter* by Gustav Holst. Figure 2 illustrates the process for arranging part of an excerpt from the Rachmaninoff, with notes omitted from the arrangement in gray.

6 Results

We compare our neural network model to a GA model for tablature creation because it is the only one for which we have significant performance data. All techniques were tested on a set of 65 excerpts constituting 1717 notes. Tablatures generated with a GA model that optimizes our evaluation function are 86.9% consistent with published tablature.

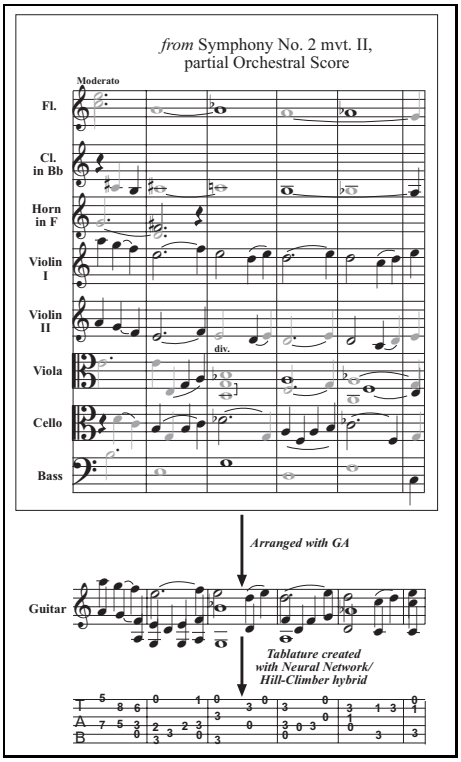

Figure 2: Generating an arrangement from a score.

This increases to 90.8% using the neural network with full hill-climbing. Using only brief hill-climbing, consistency increases to 91.1%. Using the network alone, consistency is even higher at 91.4% but has occasional obvious mistakes. This number is lower than the 94% achieved by the network on notes individually. This is because when a generated tablature diverges from a published tablature, the appropriate positions of notes subsequent to the point of divergence can be different than they would have been had there been no divergence.

The advantage in speed is more obvious. The average time necessary to create tablature for the Rachmaninoff piece in Figure 2 by GA was 5.7 seconds. To create tablature for the same piece using the neural network with local search was 55.6 milliseconds. This difference is much more noticeable when *arranging* the excerpt. An excerpt or phrase can be arranged by GA using a neural network for tablature creation in under a minute, while it would take an hour or more if tablature creation were handled with a GA.

Tablature and mp3 recordings of the arrangements created, as well as the tablature data set which we used, are available at http://www.ai.uga.edu/tuohy/excerpts.html.

7 Further Directions

We are pleased with the performance of our network, but there is room for improvement. Because we have trained our network on tablature, our network does not have access to timing information when making decisions. A network would likely perform better if it had tempo and note length information to work with, perhaps by extracting data from MIDI files and specifying correct tablature by hand (Wand and Tsai-Yen 1997).

References

Classtab.org (2006). Classical Guitar Tablature. http://www.classtab.org

Corozine, V. (2002). *Arranging music for the real world: classical and commercial aspects*. Pacific, MO: Mel Bay.

Heijink, H. and R. Meulenbroek (2002). On the complexity of classical guitar playing: functional adaptations to task constraints. *Journal of Motor Behaviour 34*(4), 339–351.

Michael, D. (2003). *Arranging for Open Guitar Tunings*. Anaheim Hills, CA: Center Stream Publishing.

Radicioni, D. and V. Lombardo (2005). Guitar fingering for music performance. In *Proceedings of the International Computer Music Conference*, pp. 527–530. International Computer Music Association.

Radisavljevic, A. and P. Driessen (2004). Path difference learning for guitar fingering problem. In *Proceedings of the International Computer Music Conference*. International Computer Music Association.

Sayegh, S. (1989). Fingering for string instruments with the optimum path paradigm. *Computer Music Journal 13*(6), 76–84.

Smith, B. (2004). Gail: Georgia artificial intelligence library neural network package. *University of Georgia*.

Tuohy, D. and W. Potter (2005). A genetic algorithm for the automatic generation of playable guitar tablature. In *Proceedings of the International Computer Music Conference*, pp. 499–502. International Computer Music Association.

Tuohy, D. and W. Potter (2006a). Generating Guitar Tablature with LHF Notation via DGA and ANN. In *Proceedings of the IEA/AIE*. International Society of Applied Intelligence.

Tuohy, D. and W. Potter (2006b). Ga-based music arranging for the guitar. In *Proceedings of the 2006 Congress on Evolutionary Computation (to appear)*. http://www.ai.uga.edu/tuohy/arranging.pdf

Wand, J. and L. Tsai-Yen (1997). Generating guitar scores from a midi source. In *Proceedings of the International Symposium on Multimedia Information Processing*.

Whitley, D. (1995). Genetic algorithms and neural networks. *Genetic Algorithms in Engineering and Computer Science*, 1–15.

Whitley, D. and J. Kauth (1988). Genitor: A different genetic algorithm. In *Proceedings of the Rocky Mountain Conference on Artificial Intelligence*, pp. 118–130.