

# Computer interpretation of piano music

Corrin Lakeland  
Supervisor : Dr. Anthony Robins

June 1997

## **Abstract**

This project is to produce a program that reads a musical score from a file and by using a neural network it interprets the score to produce an aesthetically pleasing corresponding sound file.

# 1 Introduction

Playing a musical instrument such as the piano requires a musician to hit and release keys. A musical score is intended to tell the musician which keys to hit, when to hit and release them and how hard to hit them. This project looks at presenting a musical score to a computer which would then synthesise the sound made by the notes specified in the score.

When a musician is playing a musical instrument their performance has significant variations in timing and loudness that are not specified in the score and it is these variations that create a good performance. This project is an attempt to produce a computer program which produces a similar performance to a musician.

## 1.1 Existing music synthesisers

Programs to play musical instruments on a computer have been available for over ten years. Such programs are used for theoretical study, presentations, games and enjoyment. Recently the quality of sound that a typical computer can produce has dramatically improved and listening to a compact disc on a computer proves computers are capable of accurately simulating a piano. However there is still a significant difference in the quality of sound produced by computer synthesisers to that produced by the musical instruments they are trying to simulate.

Most recent applications are based on the MIDI file format. A MIDI file contains a sequence of notes where each note can be played on any instrument with any duration and loudness. The only theoretical flaw in the MIDI protocol that its model of instruments is very weak with poor support for higher harmonics and how they are affected by loud notes. This means that it cannot sound like a piano even with a pianist's timing (Problem 1). In addition, while it is relatively simple to convert a piano score file to a MIDI file, without variation in volume and duration this results in an uninteresting performance (Problem 2).

In my project I hope to address both problems and produce a program that sounds similar to a CD recording of a pianist.

### 1.1.1 Enhanced MIDI

Several commercial programs such as Finale (?) have attempted to address Problem 2 by randomly changing the volume or duration specified in the score. This results in a more interesting performance but often the program chooses poorly and produces a strange sounding phrase.

### 1.1.2 KTH rules

Friberg, Fryden and Sundberg (?) developed an expert system for changing the volume and duration of notes. After examining pianists' performances they produced a series of 'expression rules', for example one of their rules states that 'A note longer than the previous note but shorter than its succeeding note is lengthened by 12%'. Their system can make sensible decisions about how to add expression and often does well addressing Problem 2. Unfortunately its rules are not perfect and so the system occasionally makes significant mistakes.

### 1.1.3 Melodia

Melodia (?) is equivalent to the KTH expert system except that it uses a neural network instead of an expert system for determining which parameters to change. Melodia usually produces good interpretations of a musical score and so solves Problem 2. However Melodia outputs a MIDI file and so it does not sound like a piano (Problem 1).

## 1.2 My approach

Alastair Thomson recently implemented an ANSI C++ general purpose neural network for his project in musical composition (?). I decided to use his network as a basis for my project. As described later, I have also obtained source code for performing fourier transformations, another neural network architecture and an Apple Macintosh prototyping program.

## 2 Progress

### 2.1 Neural network architecture

Starting with existing neural network code has saved a large amount of implementation time in designing data structures and especially in file I/O, however differences between its intended use and my project has lead to the problems described in Table 1. As explained in that table, most of the problems have now been solved and the neural network is expected to be finished soon.

Making changes to the neural network required writing or rewriting several thousand lines of code and clearly illustrated the need for a powerful general purpose neural network.

Problem	Solution
Alastair Thomson found that the learning algorithm he used (Alopex, (?)) was flawed and frequently fails to learn temporal patterns.	I have deleted the Alopex learning algorithm and I am modifying the source code for the Cascade learning algorithm (?) to work with the Thomson's architecture.
Alopex assumes that input is only presented to the network at the start of execution but I need to present the next input notes at each time slice.	I wrote a procedure that obtains input at every time slice and delays the output so that the output at time slice $n$ corresponds to the input at time slice $n$ .
The program did not have an option to save its weight space after learning.	I wrote procedures to save and load weight space files.
C++ was used to implement the project however C style programming and pointer manipulation was used extensively.	I rewrote the code without pointer manipulation, macro functions, and other complex C commands.

Table 1: Further development of Thomson's neural network

### 2.1.1 Algorithms

After examining the literature I decided that a non recurrent neural network architecture would be insufficiently powerful to learn musical expression. However there is some debate in this area due to the difficulties in training recurrent networks. NetTalk demonstrated that non recurrent networks are capable of learning temporal sound patterns (?) while other authors have shown that the temporal patterns in music require a recurrent network (?, ?).

I initially intended to use the Alopex recurrent learning algorithm (?) however this has been shown to be theoretically flawed (?) and so I decided to use the Cascade Correlation learning algorithm (?). This algorithm learns in very few epochs which is important because my training data takes a long time to present to the network.

### 2.1.2 Data structures

The data structures used to represent the neural network are described below. They correspond to the ‘textbook’ abstractions of a Neural network with a data structure to represent each of the three components in a network. They are each implemented as a separate C++ class in the files `Network.h`, `NodeLayer.h`, `Node.h` (appendix D).

**Network** implements a neural network. This has an array of `NodeLayers` and can be created, trained on input data, tested and saved to disk. There will only be one instance of this class in the project, representing the pianist.

**NodeLayer** implements a single layer of a neural network. This is intended to separate the `Network` from individual nodes in the network. It has an array of nodes and supports basic iteration.

**Node** implements the primitive nodes in a neural network. A `Node` has an array of connections from other nodes and can be printed, trained and saved to disk.

This data division comes from the original Alopex code and I have not changed it. The primary benefit of creating a `NodeLayer` data structure is that it allows the network designer to think conceptually about layers and for these designs to be implemented easily. The cascade architecture differs from this by only supporting one hidden layer.

The translation between music and these data structures is described in Section 2.3.

## 2.2 User interface

It is intended that the pianist program can be used without any understanding of neural networks. For this reason I have obtained an application for the Macintosh that I will use as a ‘front end’ to a trained neural network that can play music.

Music is generated by presenting each note in a musical score to the trained neural network and storing the fourier series that the networks outputs. This fourier series is then converted into sound using a inverse fourier transformation and saved as an AIFF file.

## 2.3 Input to the neural network

I decided to use a representation of music based on that used by Mozer in his neural network composer (?). His approach and the extensions I have made are described below. Theoretically it is possible to produce the correct output with distinct input, however it has been shown that preprocessing input improves network performance (?).

Because my representation must be able to support all music but his representation only needs to support the notes his system composes, his representation has sometimes been insufficient. In these situations I have added nodes to represent the additional information that I need. An example of this is that many composers specify notes should be played *staccato*. This is technically equivalent to a very short rest between notes but most pianists demonstrate individual interpretations.

### 2.3.1 Pitch

It would be theoretically possible to use a single node to represent the pitch of the current note. Mozer’s results imply that such a node is essential but not sufficient. He suggests that the network should also be presented with a ‘chroma circle’ and a ‘circle of fifths’ and cited psychological evidence for this representation. This representation means that the network is presented with similar input in the same situations as a human would say they hear a similar note.

**Pitch height** is the scaled logarithm of the current notes frequency. Using the logarithm means that it is intervals, rather than absolute frequency, that is being counted. This means the Network will generalise over tones more easily.

**Chroma circle** produces a similar value for two notes on the same key (e.g. C major) and so causes the network to treat notes in the same key similarly.

Note	Chroma Circle						Circle of fifths					
C	+	+	+	-	-	-	-	-	-	+	+	+
C#	+	+	+	+	-	-	+	+	+	+	-	-
D	+	+	+	+	+	-	-	-	-	-	-	+
D#	+	+	+	+	+	+	+	+	+	+	+	+
E	-	+	+	+	+	+	+	-	-	-	-	-
F	-	-	+	+	+	+	-	-	+	+	+	+
F#	-	-	-	+	+	+	+	+	+	-	-	-
G	-	-	-	-	+	+	-	-	-	-	+	+
G#	-	-	-	-	-	+	+	+	+	+	+	-
A	-	-	-	-	-	-	-	-	-	-	-	-
A#	+	-	-	-	-	-	-	+	+	+	+	+
B	+	+	-	-	-	-	+	+	-	-	-	-
rest	+	-	+	-	+	-	+	-	+	-	+	-

Table 2: Network representation of the central octave

**Circle of fifths** produces similar values for notes separated by a fifth, a frequency ratio of 2 : 3. This causes the network to treat notes separated by a fifth similarly.

Table 2 is used for translating a note into input for the neural network. The pitch height (not shown) is the logarithm of the fundamental frequency of the note.

### 2.3.2 Key

It was found that some composers such as Bach expected different styles of playing in different keys (?). Because of this it is important to encode the current key so the network can play differently.

My representation of the current key is the representation of the current chord used by Mozer. Mozer graphed the apparent similarity of different chords as a tree and I have encoded this tree. This representation means that keys which are perceived as being similar have a similar activation. Table 3 on page 7 is the result of this encoding.

### 2.3.3 Duration

Mozer's representation of duration is similar to his representation of Pitch where a single node provides magnitude information and two 'circles' group notes of similar sounding duration. Activations are defined as follows:

Key	Activations													
$C$	-	-	-	-										
$C_7$	-	-	-	+										
$F$	-	-	+	-										
$F_7$	-	-	+	+										
$G$	-	+	-											
$G_7$	-	+	+											
$B_7$	+	-												
$E$	+	+	-	-										
$E_7$	+	+	-	+										
$A_7$	+	+	+	-	-									
$C_{dim}^\#$	+	+	+	-	+									
$E_m$	+	+	+	+	-	-								
$(G, B)_{aug}$	+	+	+	+	-	+								
$G_m$	+	+	+	+	+	-	-	-						
$G_{dim}$	+	+	+	+	+	-	-	+						
$A_{dim}^\#$	+	+	+	+	+	-	+							
$D_7$	+	+	+	+	+	+	-	-						
$D_m$	+	+	+	+	+	+	-	+	-	-				
$D_{m_7}$	+	+	+	+	+	+	-	+	-	+				
$D_{dim}$	+	+	+	+	+	+	-	+	+	-				
$B_{dim}$	+	+	+	+	+	+	-	+	+	+				
$F_m$	+	+	+	+	+	+	+	-	-					
$C_m$	+	+	+	+	+	+	+	-	+	-	-			
$A_{dim}$	+	+	+	+	+	+	+	-	+	-	+	-		
$G^\#$	+	+	+	+	+	+	+	-	+	-	+	+	-	
$C_{dim}$	+	+	+	+	+	+	+	-	+	-	+	+	+	-
$G_7^\#$	+	+	+	+	+	+	+	-	+	-	+	+	+	+
$A_m$	+	+	+	+	+	+	+	-	+	+	-			
$C_{aug}$	+	+	+	+	+	+	+	-	+	+	+			
$Rest$	+	+	+	+	+	+	+	+						

Table 3: Network representation of the current key

Name	Activation
sotto voce	-4
pianissimo	-2
piano	-1
mezzo piano	-.5
mezzo forte	+.5
forte	+1
fortissimo	+2
fff	+4

Table 4: Representation of the loudness of a given note

**Magnitude node** has an activation of  $\log_{12}$  of the number of beats for the note. This means the magnitude node will have a similar activation to other nodes.

**Third beat circle** counts  $beats \bmod \frac{4}{12}$ . This causes eighth notes and triplets to have the same representation.

**Quarter beat circle** counts  $beats \bmod \frac{3}{12}$ . This causes eighth notes and quarter notes to have the same activation.

**Style** Added a node to Mozer’s representation, this represent the duration style being applied to the note, eg *allegro* or fast.

**Change** Not from Mozer, this represents how the duration is changing which can be either *accelerando* (getting faster) or *rallentando* (slowing). This allows rules similar to those found in KTH to be developed.

This representation does have one disadvantage in that it does not adequately allow for trill notes because their duration will be too short to represent. However this representation only requires 4 nodes and is usually adequate.

### 2.3.4 Loudness

Loudness is represented by a single note which has an activation dependent on the current style being applied to the note, eg *fortissimo*. Table 4 describes the mapping used.

Changes in loudness are also important and represented in a musical score by either *decrescendo* (decreasing loudness) or *crescendo* (increasing loudness). Representing changes in loudness enables the network to use rules like those in KTH. A total of two input nodes are used to represent loudness.

### 2.3.5 Polyphonic music

Because very little piano music is monophonic I decided to support more than one note being played concurrently (*polyphonic* music). After experimenting with a variety of techniques I decided to present each note separately to the network along with a list of the intervals of other notes played concurrently. This means that the network only needs to determine the sound made by one note at once.

This results in very few input units as only one note has to be described in detail while still providing enough contextual information to determine how to play a given note. If a note is over an octave away from the current note then the activation of its interval is logarithmically weighted on the number of separating octaves. This means that, for example, playing the note B3 concurrently to C3 will be emphasised more than playing the note B1 concurrently to C3.

A note may differ from the current note by  $\pm 6$  semitones and, remembering that C4 differs from C3 by zero semitones, 12 input nodes are required.

### 2.3.6 Context information

The activations of nodes in the network are based on following the melody. Because recurrent connections are not as effective as conventional input (?), the interval, loudness and duration of the proceeding and succeeding melody is also presented to the network.

This representation means that only 3 nodes are needed to represent the melody in a given time slice and so  $\pm 3$  time slices are presented, requiring 18 input nodes.

### 2.3.7 The pedals and other information

Pedals are represented by two nodes for each pedal. These nodes represent how far down the pedal is as well as how fast the pedal is moving. As pedals are used to change the sound produced their state must be presented to the network. Additionally, representing the movement of the pedal allows playing style to change as the pedal is depressed.

A variety of other playing styles such as *vivace* also need to be represented. These are summarised in Table 5 and are supported by the network to simplify translation of a musical score to an input file. A total of 9 nodes is required to represent the pedals and other information.

Name	Style
Glissando	Sliding run
Tremolo	Rapid repeat
Sforzando	Forced
Ritenuto	Held back
Vivace	Lively

Table 5: Other styles that are represented. Each style is represented by a boolean input node

### 2.3.8 Justification of this approach

To be effective the input to the neural network must present as much information as possible without unnecessary input nodes. This architecture uses ‘state of the art’ in music representation and has a total of 71 input nodes. This encoding is capable of representing almost all music and providing the network with information about the music in the most effective method. Clearly there is sufficient input nodes.

One question that must be asked is whether there is any unnecessary input. If there is then it will considerably decrease the rate of learning (?). The answer to this question will not be known until learning is tested without some of the input nodes.

## 2.4 Output from the neural network

I decided to have the neural network output the delay until the next time slice as well as a fourier series (see glossary) representing the sound that the current note generates. A fourier series allows totally arbitrary sound sequences to be represented and so avoids Problem 1 (p. 1) – the MIDI piano does not sound like a piano.

The delay until the next time slice can be approximated from the duration of the current note. However the pianists deviation from this time is extremely important and so the network uses two nodes to represent duration. One node represents the time until the next note and the other containing a small deviation to be made to this time.

There are a variety of advantages and disadvantages of using a fourier series (described below). I decided that the advantages of using a fourier series outweigh the disadvantages and hence I have decided to use 128 different output nodes corresponding to most of the first five harmonics of different notes. The choice of representing the first five harmonics comes from an analysis of the physics in

the piano (?) which showed that over 98% of the energy from a note is in the first five harmonics.

#### 2.4.1 Advantages of using a fourier series

The sound generated by a piano is from sinusoidal vibration and so can be easily and accurately represented as the sum of sinusoids. Additionally there are two facts which make using a fourier series more efficient:

1. Two notes an octave apart always have a frequency ratio of 2 : 1.
2. Notes are comprised of a fundamental frequency and a series of overtones with increasing frequency and varying energy. These overtones have frequencies a linear multiple of the fundamental frequency (so the third harmonic has a frequency three times that of the fundamental frequency).

Because of these two facts the  $n + 1$  harmonic of a note has the same frequency as harmonic  $n$  of the note an octave above. This means that very few nodes are required to accurately represent the fourier series.

#### 2.4.2 Disadvantages of a fourier series

There are two main disadvantages of a fourier series.

1. Because fourier series have high generality they require a lot of data to represent, making the learning process slow. Because a piano cannot produce most sound this inefficiency does not provide any benefit.
2. The algorithm for converting between a fourier series and sound is slow and requires the number of output nodes to be a power of 2.

### 2.5 Justification of this approach

There are a total of 130 output nodes from the network which would normally be considered too many. However it is possible to initially train the network with 18 of these nodes and relatively accurately interpolate between the weights generated and the weights in the 130 node network. This means that training the 130 node network will be relatively fast.

## **3 Future work**

I have four tasks to complete before this project is complete. There are three minor tasks and one major task. The three minor tasks are to complete the user interface, integrate the fourier and AIFF libraries and to implement the Cascade learning algorithm. The major task outstanding is detecting when the pianist has struck a key using a sound recording of a performance. This is needed so that the training data can be synchronised with the network.

### **3.1 Finishing the user interface**

The user interface needs to be completed and integrated with the neural network. If it is possible to generate a real time performance of a musical score then this would be a worthwhile extension.

### **3.2 Integrating the libraries**

I have obtained a library for translating sound from the frequency domain to the spatial domain and another library for saving sound in the spatial domain to an AIFF sound file. These need to be integrated with the network so that the network's output can be saved to a sound file.

### **3.3 Implementing the Cascade learning algorithm**

I have obtained the C code for the Cascade learning algorithm distributed by Carnegie Mellon University (?). However this source code relies on their implementation of a neural network and so before I can use it I will have to modify the source code slightly to support my network implementation.

### **3.4 Detecting a key press**

To train a neural network it must be presented with input and corresponding output so for my neural network this requires presenting every note from a musical score at the same time as that note in the recording of that score. To present the note from the recording requires finding when the pianist plays another note but there is no known algorithm for performing this task.

One simple solution to this problem is to assume that the pianist has perfect timing however if this is not the case then the teacher will lose synchronisation with the current note and the network will be unable to learn.

# Appendix

## A Glossary of musical terms

**Accelerando** gradually getting faster.

**Adagio** in a slow tempo.

**Allegro** fast.

**Chord** a group of notes sounded together.

**Crescendo** gradually getting louder.

**Decrescendo** gradually getting softer.

**Flat** a semitone lower than the specified pitch.

**Fortissimo** very loud.

**Fourier series** A procedure by which complicated periodic functions, such as sound waves, can be written as the sum of a number of simple wave functions.

**Harmonic** an overtone accompanying (and forming a note with) a fundamental at a fixed interval. (*The sound generated by a piano is the sum of all its harmonics*)

**Harmony** the combination of simultaneously sounded musical notes to produce chords and chord progressions, especially when creating a pleasing effect.

**Interval** the difference in pitch between two sounds

**Key** a system of notes related to each other and based on a particular note (key of C major).

**Legato** in a smooth flowing manner, opposite of Staccato.

**Major key** A key where notes are based on the penatonic scale where notes fall on the following semitones : 1-3-5-6-8-10-12-1

**Melody** single notes arranged to make a distinctive recognizable pattern; **tune**.

**Minor key** equivalent to the major key except the fifth note is one semitone lower. This results in the sequence : 1-3-4-6-8-10-12-1

**Monophonic** one sound – the music only contains one note being played at once.

**Music** the combination of sounds in a harmonious or expressive way.

**Note** a single musical tone of definite pitch. A key of a piano etc.

**Pitch** the frequency of vibrations producing a sound.

**Polyphonic** many sounds – the music contains more than one note being played at the same time.

**Phrase** a group of notes forming a distinct unit within a melody

**Rallentando** slowing, i.e. the notes are being played for longer.

**Scale** a set of notes at fixed intervals, arranged in order of pitch.

**Semitone** half a tone.

**Sforzando** with sudden emphasis.

**Sharp** a semitone higher than the specified pitch.

**Staccato** with each note sharply distinct, opposite of Legato.

**Timbre** the distinctive character of a musical sound apart from its pitch and volume. (*The sum of all harmonics except the first*)

**Tone** a musical sound of a definite pitch and character. Also an interval of a major second.

**Trill** a rapid alternation of played notes.

**Vivace** the passage is to be played in a lively manner.

## **B Aims and objectives (revised)**

### **B.1 Aim**

To produce an application that can read a musical score from a file and use a neural network to interpret the score to produce an aesthetically pleasing corresponding sound file.

## B.2 Objectives

- Complete the neural network architecture.
- Find or develop suitable training examples for the network.
- Produce an application which allows the network to be used easily.

## C References

## D Source code and word count

Following is the current source code for the neural network (containing just over 2000 lines of code). All files except `Network.lex` and `Network.yacc` are significantly different to Thomsons originals. Other source code (user interface, fourier transformations, Cascade learning algorithm and AIFF file manipulation) has not been printed because it is not primarily my own work

Word count (excluding figures and appendices): 2993 words.