

# Takterkennung mit neuronalen Netzen

Matthias Petermann

13. November 2005

## 1 Aufgabenstellung

In der Lehrveranstaltung Neuroinformationsverarbeitung wurde umfangreiches Wissen über Methoden der künstlichen Intelligenz - insbesondere zu Funktionsweise und Anwendungsgebieten von neuronalen Netzen - vermittelt. Im Sinne einer Prüfungsleistung erhält jeder Student die Möglichkeit einen Beleg zu einer praktischen Anwendung von neuronalen Netzen anzufertigen. Meine Aufgabe bestand darin zu untersuchen, ob eine akustische Takterkennung von musikalischen Werken mit Hilfe neuronaler Netze realisierbar ist.

## 2 Grundlagen

### 2.1 Was ist ein Takt?

Alle Melodien haben einen Rhythmus. Dieser kann fließend verlaufen - wie etwa in streicherdominierten klassischen Werken - oder auch in sogenannten Takten gemessen werden. Letzteres ist besonders deutlich in der Musik der Neuzeit festzustellen. Als Takt bezeichnet man einen regelmäßig wiederkehrenden Schlag.

“Innerhalb des Taktes bestimmt ein metrisches Schema aus Längen und Kürzen die rhythmische Struktur des Liedes. Diese Längen und Kürzen werden durch Noten unterschiedlicher Notenwerte dargestellt.“ Ein Notenwert definiert die Dauer des Tones im Verhältnis zur Dauer des ganzen Taktes bzw. einer ganzen Note.

### 2.2 Taktarten

Die Taktart eines Liedes wird in der Partitur durch die Taktvorzeichnung angegeben. Die Taktvorzeichnung besteht aus zwei übereinanderstehenden Zahlen. Die untere nennt die Zeiteinheit im Verhältnis zu einer ganzen Note. Die obere Zahl gibt an, wie viele Einheiten es pro Takt gibt. Beispiel: Ein  $\frac{4}{4}$ -Takt erfordert vier Viertelnoten pro Takt. Der Takt ist vollständig, wenn die Summe der enthaltenen Noten  $\frac{4}{4}$  - also einer ganzen Note - entspricht. Die Taktvorzeichnung eines  $\frac{4}{4}$ -Taktes ist erfüllt wenn er vier Viertelnoten

enthält - jedoch auch, wenn er aus zwei Viertelnoten und einer halben Note zusammengesetzt ist. Weiterführende Erläuterungen sind in der einschlägigen Musikkultur zu finden.

## **2.3 Erkennen von Taktarten**

Liegt das Notenblatt (Partitur) eines Werkes vor, so kann die Taktart ohne jeglichen Aufwand davon abgelesen werden. Selbst wenn die Taktvorzeichnung fehlt, können meist auch über das Verhältnis der Noten in den Takten Rückschlüsse auf die Taktart gezogen werden. Schwieriger wird es bei unbekanntem Stücken, zu denen keine Noten vorliegen. Viele Menschen sind nicht in der Lage, durch bloßes Hören eines Liedes den Takt herauszuhören. Die Bedeutung dieser Fähigkeit wird spätestens dann klar, wenn man einen Standard/Latin-Tanzkurs besucht. Die Zählweise der Tanzschritte orientiert sich an der Taktart der gespielten Musik. Im Idealfall sollte man schon am ersten Takt erkennen um welche Taktart es sich handelt, um folgerichtig den passenden Tanz zu wählen. Erkennungsmerkmal für den Takt ist - wie bereits im Vorangegangenen Abschnitt erläutert - ein charakteristisches Verhältnis aus Längen und Kürzen der gespielten Noten.

## **2.4 Digitalisierung von Musik**

### **2.4.1 Musik als mechanische Welle**

Musik kann wie alle akustisch wahrgenommenen Einflüsse im physikalischen Sinne als eine sich zeitlich und räumlich ausbreitende mechanische Welle beschrieben werden. Trägermedium dieser Welle sind Luftmoleküle. Soll Musik im Computer verarbeitet werden, muss sie in ein für den Rechner verwertbares Format gebracht werden - man spricht dabei von Digitalisierung.

### **2.4.2 Aufzeichnung von Musik**

Zunächst aber muss die mechanische Welle in ein analoges elektronisches Signal gewandelt werden. Das geschieht in der Regel mit Hilfe eines Mikrofones, das durch eine druckempfindliche Membran (kapazitiv oder induktiv) eine Änderung eines angelegten Messstromes herbeiführt. Das erhaltene analoge Signal dient als Eingangssignal für einen Analog/Digital-Wandler, wie er auf den Soundkarten von PCs verbaut ist.

### **2.4.3 Analog/Digital-Wandlung**

Das kontinuierliche Eingangssignal muss für die digitale Speicherung und Verarbeitung diskretisiert werden. Bei der Analog/Digital-Wandlung wird deshalb die Amplitude des kontinuierlichen Eingangssignales schnell aufeinander folgend mit großer Häufigkeit gemessen (abgetastet). Die Frequenz, mit der das geschieht, wird als Abtastfrequenz  $f$  bezeichnet. Bei diesen Messungen erhält man also  $f$  Amplitudenwerte für den Zeitabschnitt einer Sekunde. Bei der Wahl der Abtastfrequenz ist das Shannon'sche Abtasttheorem zu

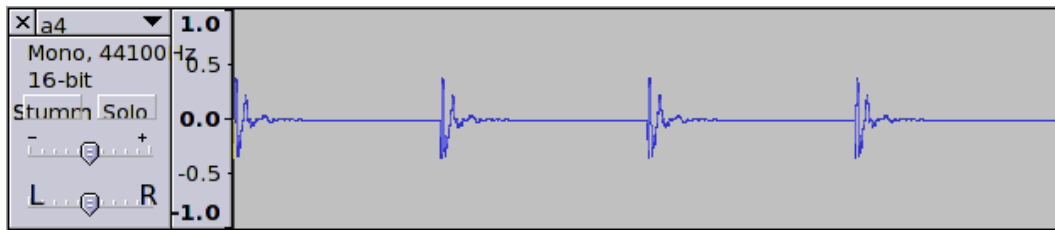


Abbildung 1: Darstellung von Amplitudenwerten auf der Zeitachse

beachten. Dieses sagt aus, dass zur eindeutigen Darstellung einer Schwingung der Frequenz  $f$  mindestens  $2f$  Perioden dieser Schwingung gemessen werden müssen. Mit einer Abtastfrequenz  $f = 44100\text{Hz}$  können also Frequenzen im Intervall  $f_{min} = 0\text{Hz} < f < f_{max} = 22050\text{Hz}$  dargestellt werden. Die vom menschlichen Gehör maximal wahrnehmbare Frequenz liegt wesentlich unterhalb von  $f_{max}$ . Dennoch werden gerade im Bereich der Digitalisierung von Audiosignalen typischerweise diese hohen Abtastfrequenzen verwendet, da die Abtastfrequenz neben ihrer Bedeutung für  $f_{max}$  auch ein Maß für die Qualität der Aufnahme ist.

Anschließend wird jeder ermittelte Amplitudenwert auf einer metrischen Skala abgebildet, deren Auflösung von der zur Verfügung stehenden Hardware und den verfügbaren Speicherkapazitäten bestimmt wird. Typischerweise steht hier eine ganzzahlige Skala im Intervall  $-32767 < y < 32768$  zur Verfügung, die sich im Computer als signed short integer abbilden lassen (16 Bit). Früher waren auch Auflösungen von 4 Bit und 8 Bit üblich, sind aber heute auf Grund der wesentlich schlechteren Qualität kaum noch von Bedeutung.

#### 2.4.4 Ergebnis

Mit den vorangehend beispielhaft verwendeten Parametern wäre das Ergebnis der Analog/Digital-Wandlung ein Datenstrom mit 44100 diskreten Werten des Datentypes signed short integer je Sekunde. Mit Hilfe derer lässt sich der zeitliche Verlauf der Amplitude des ursprünglichen Eingangssignales sehr detailliert rekonstruieren. Außerdem bilden sie die Grundlage für computergestützte Weiterverarbeitung und Analyse.

## 2.5 Analyse des Audiosignales

### 2.5.1 Ausgangspunkt

Das Ergebnis der Digitalisierung ist die Abbildung von aufeinander folgenden Amplitudenwerten auf einer Zeitachse. Setzt man die Amplitude  $y$  mit der Energie, beziehungsweise beispielhaft mit der Auslenkung einer Lautsprechermembran zum Zeitpunkt  $t$  gleich, so kann die Amplitude näherungsweise als Maß für die momentane Intensität des Signales (Lautstärke) gelten. Zu den zum Zeitpunkt  $t$  enthaltenen Frequenzanteilen (Frequenzspektrum) kann jedoch keine Aussage getroffen werden.

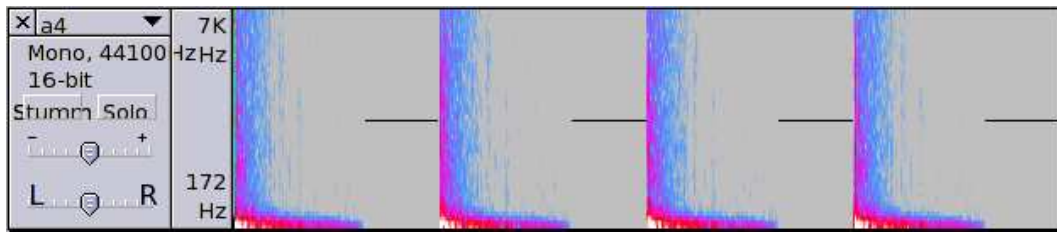


Abbildung 2: Darstellung des Frequenzspektrums auf der Zeitachse

### 2.5.2 Frequenzanteile in Musikstücken

Beim Hören von Musik wird schnell deutlich, dass allein die Lautstärke keinerlei Bedeutung für die Bewertung des Taktes hat. Vielmehr muss man sich auf das taktangebende Instrument konzentrieren. Bei elektronischer Musik könnte das eine Basedrum<sup>1</sup> sein, bei anderen Musikrichtungen aber auch ein Bass. Bei Solo-Stücken wird der Takt gar durch das Hauptinstrument angegeben - so zum Beispiel von einem Piano, dessen Frequenzbereich sehr variabel ist. Es ist also notwendig, zu jedem Zeitpunkt  $t$  eine Übersicht zu den anliegenden Frequenzen berechnen zu können. Dazu dient die Fourier-Transformation.

### 2.5.3 Die Fourier-Transformation

Die Fourier-Transformation transformiert eine Funktion von einer Zeitdarstellung durch Integration in einen reziproken Frequenzraum. Sie geht auf den französischen Mathematiker und Physiker Jean Baptiste Joseph Fourier zurück und ist eine der wichtigsten Grundlagen für die Signalverarbeitung. Zur Berechnung der diskreten Fouriertransformation wird oft die schnelle Fourier-Transformation (FFT) verwendet, ein Algorithmus, bei dem die Anzahl der Rechenschritte zur Berechnung der Fourierkoeffizienten wesentlich kleiner ist als bei einer direkten Implementation der Integration. Aufgrund der Komplexität des Verfahrens und des Vorhandenseins von freien, direkt benutzbaren Implementierungen der FFT wird an dieser Stelle nur auf weiterführende Literatur verwiesen.

## 3 Konzeption des neuronalen Netzes

### 3.1 Netzstruktur

In Audiolyzer verwende ich ein dreischichtiges neuronales Netz mit Backpropagation.

---

<sup>1</sup>Schlaginstrument (Trommel) mit Frequenzgang nahe der Basis (Base, unterer Bereich) des Frequenzbandes

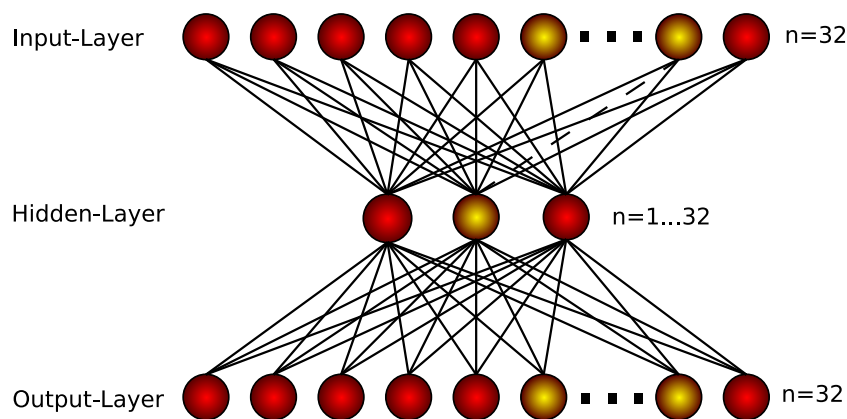


Abbildung 3: Schema der Netzstruktur von Audiolyzer

### 3.2 Eingangsschicht

Die Eingangsschicht (Input-Layer) besitzt eine feste Anzahl von Neuronen, auf die das normalisierte Pattern der zu erkennenden Taktart abgebildet wird. Das Zeitfenster, in dem das Pattern analysiert wird, wurde auf eine Sekunde begrenzt. Das Zeitfenster wird in 32 gleich große Einheiten aufgelöst, die als Eingangspegel für je ein Neuron dienen. Dazu wird im Analysezeitraum für jeden Frame des Samples der Pegel der für den Takt relevanten Frequenzen ermittelt. Über deren Mittelwert erfolgt dann eine Selektion, ob der Pegel als Taktsignal übernommen wird - d.h. nur die Pegel, die über dem Mittelwert liegen sind relevant. Dabei finden noch weitere Normierungen statt, die das Datenaufkommen auf 32 Einheiten pro Sekunde reduzieren. Dazu befinden sich Erläuterungen im Abschnitt zur Implementierung. Nach Abschluss der Bearbeitung in dieser Vorstufe steht für eine Sekunde Audio-Material ein Eingabevektor mit 32 Werten bereit. Die Anzahl 32 resultiert aus der Überlegung, dass die kürzeste erkennbare Note  $\frac{1}{16}$  Sekunde dauern kann. Um Töne und Pausen trennen zu können, wurde er verdoppelt. Im vorliegenden Programm kommen also 32 Neuronen in der Eingangsschicht vor.

### 3.3 Hidden-Schicht

Die Hidden-Schicht hat Einfluss auf das Lernverhalten des neuronalen Netzes. Die Anzahl der in ihr enthaltenen Neuronen beeinflusst maßgeblich die Fähigkeit der Generalisierung, d.h. die Bildung von Klassen und die Zuordnung von ähnlichen Pattern. Die Netzbibliothek von Audiolyzer erlaubt, die Anzahl der Neuronen in der Hidden-Schicht beliebig zur Laufzeit anzupassen. Diese Eigenschaft ist auch in der Benutzerschnittstelle zugänglich.

### 3.4 Ausgangsschicht

Die Ausgangsschicht besitzt wie die Eingangsschicht eine feste Anzahl von 32 Neuronen. Der Ausgangsvektor repräsentiert dabei sowohl die obere und untere Zahl der Taktvor-

zeichnung, wobei der Vektor gedanklich in der Mitte geteilt werden kann. Die ersten 16 Neuronen dienen der Darstellung der oberen Zahl, die folgenden 16 der unteren Zahl. Zur Darstellung einer bestimmten Zahl, wird nur der Pegel des ihr zugeordneten Neurons aktiviert. Das Ergebnis der Propagation eines angelegten Netzes kann jedoch recht unscharf ausfallen, weshalb Audiolyzer an dieser Stelle den höchsten Wert der Ausgangsneuronen als aktiv ansieht und davon die Taktzahl ableitet. Durch diesen 'mehrdimensionalen' Vektor konnte die Anzahl der Ausgangsneuronen vergleichsweise niedrig gehalten werden.

## 4 Spezifikation der Software

### 4.1 Allgemeine Beschreibung

Entsprechend der Aufgabenstellung wurde eine Anwendung mit grafischer Benutzeroberfläche implementiert. Interaktiv und menügeführt können mit dieser zu analysierende Samples eingeladen werden. Die Analyse erkennt - wenn vorhanden - das im Sample enthaltene Rhythmusmuster und gibt es grafisch aus. Die Ausgabe kann sowohl als Vorlage zum Anlernen neuer Taktarten als auch zum Erkennen eines Taktes auf Grundlage bereits gelernter Muster verwendet werden.

### 4.2 Randbedingungen

Die Software hat Proof-of-Concept-Status. Sie verarbeitet momentan vorbereitete Samples mit konstruierten Testdaten, die als WAV-Dateien<sup>2</sup> vorliegen. Für eine ordnungsgemäße Funktion muss der erste Ton des zu analysierenden Taktes zum Zeitpunkt  $t = 0$  in der WAV-Datei beginnen.

### 4.3 Softwarevoraussetzungen

Die Software wurde in der Programmiersprache C unter NETBSD entwickelt und liegt im Source-Code vor. Für UNIX-kompatible Betriebssysteme wird ein passendes Makefile mitgeliefert. Neben dem GNU C-Compiler bestehen folgende Abhängigkeiten:

- GLib 2.6.x - *Eine Library mit erweiterten C-Funktionen*
- Gtk+ 2.6.x - *Eine Widget-Library für das X-Window-System*
- LibSndFile - *Eine Library zum Einlesen von Sounddaten verschiedener Formate*

Die abhängigen Libraries gehören zum Standard-Lieferumfang der meisten aktuellen Linux-Distributionen. Unter NETBSD bzw. FreeBSD können diese problemlos über das Paketmanagement installiert werden.

---

<sup>2</sup>44100Hz, 16 Bit, Mono

## 5 Implementierte Module

Das Projekt besteht aus einer Reihe von Modulen, die in der Programmiersprache C implementiert sind.

### 5.1 gui.c

Gtk+ implementiert ein quasi-objektorientiertes Widget-Set für das X-Window-System. Ähnlich von nativen objektorientierten Sprachen wird für jedes Widget ein Objekt erzeugt. Dieses Objekt kann in seinem Verhalten beeinflusst werden sowie bei bestimmten Aktionen des Nutzers sogenannte Callback-Funktionen aufrufen. Das Modul `gui.c` enthält eine einzige, ziemlich lange Funktion mit der Signatur `void construct_gui()`, deren Aufgabe es ist, die Objekte und Handler für Benutzeraktionen zu erzeugen.

### 5.2 callbacks.c

Dieses Modul enthält Callback-Funktionen, die entweder durch Handler der Benutzeroberfläche oder durch andere Callback-Funktionen aufgerufen werden. Ein großer Teil der Callback-Funktionen dient der Realisierung der Benutzeroberfläche selbst und wird daher hier nicht näher erläutert. Lediglich die bezüglich der Aufgabenstellung relevanten Funktionen werden behandelt.

Die Funktion `void button_start_analysis_click(GtkWidget*, GtkWidget*)` ist zentraler Einstiegspunkt für die Audioanalyse. Sie realisiert im Wesentlichen die Steuerung des Datenflusses aus den WAV-Dateien durch die FFT. Außerdem findet hier die Visualisierung der Audiodaten während der Analyse statt. An dieser Stelle bietet sich eine Erklärung des für die Analyse verwendeten Algorithmus an. Nach dem Öffnen der WAV-Datei wird diese eingelesen und entsprechend der in der Benutzeroberfläche gesetzten Framesize blockweise an die FFT weitergereicht. Die FFT liefert zwei Vektoren - einen der die diskreten Werte der Frequenzen an den Stützpunkten enthält, und einen, der die tatsächliche Signalstärke dieser Frequenzen angibt. Für jeden analysierten Frame wird die FFT berechnet und aus dem Vektor der Signalstärken diejenige herausgesucht, die am wahrscheinlichsten einem Ton entspricht (stärkstes Signal). Diese wird in einem Vektor auf der Zeitachse abgelegt. Synchron dazu wird immer wieder der Mittelwert der in diesem Vektor abgelegten Signalstärken gebildet und grafisch in Form einer wagechten Linie in der Visualisierung der Frequenzanalyse angezeigt. Nach der Analyse des letzten Frames wird mit Hilfe des Mittelwertes eine Klassifizierung vorgenommen. Alle Signale oberhalb des Mittelwertes gelten als Ton, alle unterhalb werden ignoriert. Um Störeinflüsse zu minimieren werden vor der Klassifizierung die Eingangswerte abermals in Gruppen zu je 8 Werten zusammengefasst und darüber der Mittelwert gebildet. Das Ergebnis der Klassifizierung ist ein lineares Muster, das als Eingangsbelegung für das neuronale Netz angelegt werden, oder als Trainingsmuster gespeichert werden kann.

Die Funktion `void button_traindata_learn_click(GtkWidget*, GtkWidget*)` realisiert das Training des neuronalen Netzes. Als Trainingsmuster gespeicherte Muster werden zyklisch an das neuronale Netz angelegt und propagiert. Die Ausgabe des neuronalen

Netzes wird mit den Soll-Werten verglichen, die ebenfalls Bestandteil der Trainingsmuster sind. Entsprechend der Abweichungen wird ein Fehler berechnet und zur Korrektur der Gewichte zurückpropagiert.

Die Funktion `void button_pattern_propagate_click(GtkWidget*,GtkWidget*)` propagiert ein angelegtes Muster durch das neuronale Netz. Die Ausgabe des Netzes wird interpretiert und in der Benutzeroberfläche als Ergebnis dargestellt.

### **5.3 fft.c**

Dieses Modul implementiert die Fast-Fourier-Transformation als anwendungsbereite Bibliothek. Der Algorithmus dazu wurde weitestgehend unverändert dem Script Multimedia-Technologien von Professor Bruns entnommen.

## **5.4 Die Netzwerk-Bibliothek**

### **5.4.1 Hinweis**

Die von Audiolyzer benutzte Bibliothek zur Realisierung des neuronalen Netzes nutzt mehrere Routinen des unter der GPL veröffentlichten Projektes *lwneuralnet* von Peter van Rossum. Sie modelliert mehrschichtige neuronale Netze mit Backpropagation und wurde von mir speziell auf Audiolyzer zugeschnitten und angepasst.

### **5.4.2 network\_init.c**

Dieses Modul realisiert alle Funktionen, die zur Initialisierung eines neuen neuronalen Netzes erforderlich sind.

### **5.4.3 network\_admin.c**

Dieses Modul stellt Funktionen zum Manipulieren einzelner Parameter des neuronalen Netzes bereit.

### **5.4.4 network\_io.c**

In diesem Modul sind Routinen implementiert, die das Laden und Speichern von Netztopologien mitsamt ihren Gewichten in ASCII-Dateien ermöglichen.

### **5.4.5 network\_calc.c**

Die in diesem Modul enthaltenen Funktionen führen die Berechnungen durch, die für Forward- und Backpropagation notwendig sind. Außerdem sind Routinen der Fehlerberechnung und des Trainings implementiert.

## 5.4.6 Ein kleines Programm zur Demonstration

Die Bibliothek lässt sich sehr leicht in eigene Programme integrieren, um diese mit einer KI auszustatten. Das folgende kleine Beispiel soll das verdeutlichen:

```
#include "network.h"
#define MAX_TRAININGS 100000
#define MAX_ERROR 0.0

int main()
{
    int no_of_neurons[10];
    float inputs[2][5] = {1,1,1,1,1, 0,0,0,0,0};
    float targets[2][1] = {0, 0.5555};
    float output = 0;
    float error = 0, total_error = 0;
    int t = 0, i = 0, swap = 0;
    network_t *net;
    srandom(time(0));
    no_of_neurons[0] = 5;
    no_of_neurons[1] = 5;
    no_of_neurons[2] = 1;
    net = net_allocate_l (3, no_of_neurons);
    while ((t < MAX_TRAININGS) && ((total_error >= MAX_ERROR) || (t <= 10))) {
        net_compute(net, inputs[swap], &output);
        error = net_compute_output_error(net, targets[swap]);
        net_train(net);
        if(t==0) {
            total_error = error;
        }
        else {
            total_error = 0.9 * total_error + 0.1 * error;
        }
        t++;
        swap=!swap;
    }
    net_compute(net, inputs[0], &output);
    printf("1,1,1,1,1:%f\n", output);
    net_compute(net, inputs[1], &output);
    printf("0,0,0,0,0:%f\n", output);
    return 0;
}
```

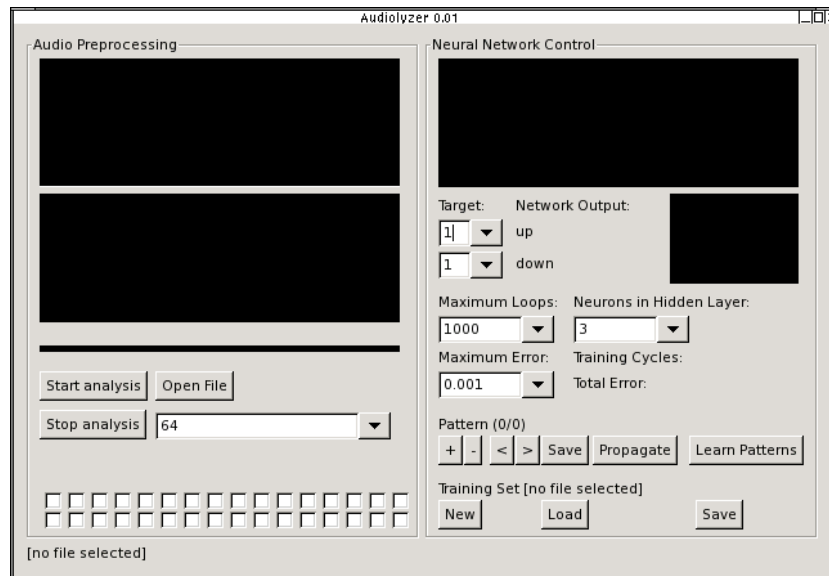


Abbildung 4: Audiolyzer direkt nach dem Start

## 5.5 main.c

Das Hauptmodul der Anwendung besteht nur aus sehr wenigen Zeilen Quellcode. Es wird die Funktion zum Erzeugen der Benutzeroberfläche aufgerufen und anschließend die Kontrolle an den Event-Handler von Gtk+ übergeben.

# 6 Bedienung der Software

## 6.1 Grundaufbau

Nach dem Starten des Programmes erscheint ein zweispaltiges Control Panel auf dem Bildschirm. Die Zweiteilung des Panels in die funktionellen Einheiten *Audio Preprocessing* und *Neural Network Control* fördert die Anschaulichkeit des Erkennungsprozesses. In der linken Spalte sind demnach alle Steuerelemente angeordnet, die die Vorverarbeitung der Audiodaten und das Errechnen der Eingabewerte für das neuronale Netz betreffen. In der rechten Spalte hingegen finden sich alle Steuerelemente für die Kontrolle des neuronalen Netzes und zur Steuerung des Lernprozesses.

## 6.2 Audio Preprocessing

Das Panel *Audio Preprocessing* besitzt zwei Monitor-Felder, die im Ausgangszustand komplett schwarz gefärbt sind. Der von oben gesehen erste Monitor zeigt den zeitlichen Verlauf des Eingangssignals nach der Fourier-Transformation als Kurve an. Während der Signalanalyse wird außerdem das gemittelte Signal durch eine horizontale, den Graphen schneidende Linie dargestellt. Das zweite Monitor-Feld stellt während der Analyse

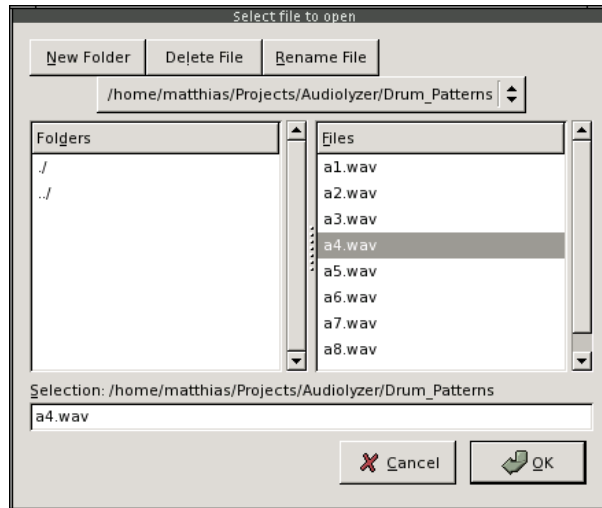


Abbildung 5: Dialog zum Öffnen einer zu analysierenden Sampledatei

die durch die Forier- Transformation ermittelten momentanen Frequenzanteile als Balkendiagramm dar. Direkt unter dem zweiten Monitor-Feld befindet sich ein schmaler schwarzer Streifen. Dieser dient der Visualisierung der Eingabedaten, die an das neuronale Netz geschickt werden. Dazu wird er in 32 Sektoren eingeteilt. Um eine Analyse durchzuführen muss zunächst eine zu analysierende Sampledatei geladen werden. Dazu kann mit dem Button *Open File* ein Dialog geöffnet werden (siehe Abbildung 5 auf Seite 11). Wurde eine gültige Sampledatei ausgewählt, kann durch Betätigen des Buttons *Start analysis* die Analyse in Gang gesetzt werden. Mit dem Button *Stop analysis* kann die Analyse, die insbesondere auf älteren Rechnern unter Umständen sehr zeintensiv sein kann, vorzeitig angehalten werden. Die Combobox dient dem Einstellen der Genauigkeit der Analyse. Sie beeinflusst direkt die Frame-Größe in Bytes, die der Fourier-Transformation vorgelegt wird. Obwohl die Combobox prinzipiell das Eintragen von benutzerdefinierten Werten erlaubt, ist die Verwendung der vordefinierten Werte empfohlen (optimal 256). Im Unteren Bereich des Panels befinden sich in zwei Reihen angeordnet 32 Checkboxes. Diese sind ebenfalls an die Eingänge des neuronalen Netzes gekoppelt und erlauben das Modifizieren der Eingabedaten vor dem Anlegen.

### 6.3 Neural Network Control

Das Panel *Neural Network Control* besitzt ebenfalls ein Monitor-Feld. In diesem wird während des Anlernens des Netzes die Lernkurve visualisiert. Eine steil abfallende Lernkurve signalisiert einen guten Lernfortschritt (abnehmende Fehlerkennung) - eine nahezu horizontale oder ansteigende Kurve hingegen ist Indiz dafür, dass die Lernkapazität des Netzes erschöpft ist, oder die zu lernenden Muster in sich einen Widerspruch darstellen (z.B. zwei gleiche Eingabemuster sollen auf zwei verschiedene Ausgabemuster trainiert werden). Unterhalb des Monitor-Feldes befinden sich zwei Comboboxen, mit denen für

ein Eingabemuster das gewünschte Ausgabemuster (Target) eingestellt werden kann. Davon rechts daneben wird beim Anlegen eines Eingabemusters an das Netz das ermittelte Ausgabemuster angezeigt. Das zweite (kleinere) Monitor-Feld zeigt die statistische Verteilung der Werte aller Ausgangsneuronen, die zur Ermittlung des Ausgabewertes herangezogen werden als Balkendiagramm. Mit Hilfe dessen kann sehr leicht eine Aussage zur Zuverlässigkeit der Ausgabe getroffen werden. Die mit *Maximum Loops*, *Maximum Error* und *Neurons in Hidden Layer* beschrifteten Comboboxen dienen der Parametrisierung des Netzes beziehungsweise des Lernvorganges. *Maximum Loops* setzt dabei eine Obergrenze für die Lernzyklen, nach deren Erreichen der Lernvorgang bedingungslos abgebrochen wird. Das Parameter *Maximum Error* stellt gleichwohl eine Grenze für den totalen Fehler dar, bei derer Unterschreitung der Lernvorgang unabhängig von der Anzahl der vergangenen Lernzyklen abgebrochen wird. Großen Einfluss auf den Lernvorgang und die Generalisierungsfähigkeit des Netzes hat die Anzahl der Neuronen im Zwischenlayer. Diese ist über die Combobox *Neurons in Hidden Layer* einstellbar. Unter den Steuerelementen zur Parametrisierung befindet sich eine Zeile mit Buttons, über die die Muster verwaltet, sowie das Lernen und Erinnern von Mustern gesteuert werden können. Die Button *+* und *-* erstellen ein neues beziehungsweise entfernen das aktuelle Muster aus der Liste. Es ist hierbei zu beachten, dass vor dem ersten Analysevorgang mindestens ein Muster erstellt werden muss. Die Buttons *j* und *j* erlauben das Blättern in der Musterliste. Mit *Save* wird die aktuell dargestellte Eingangs- und Ausgangsbelegung in den aktuellen Musterdatensatz gespeichert. Das ist insbesondere nach dem Ändern der Ausgangsbelegung und der Manipulation der Eingangsbelegung durch die Check-boxen nötig. Die Analyse hingegen führt eine automatische Speicherung des erkannten Musters in die Musterliste durch. Der Button *Propagate* legt das aktuelle Eingabemuster an das neuronale Netz an und propagiert die Werte durch. Die resultierende Ausgabe wird unter *Network Output* angezeigt. Der Lernvorgang kann mit dem Button *Learn Pattern* aktiviert werden. Unter Berücksichtigung der unter *Maximum Loops* und *Maximum Error* gesetzten Grenzen werden damit alle in der Musterliste enthaltenen Muster angelernt. Der Lernfortschritt wird durch die grafische Darstellung der Lernkurve visualisiert. Mit Hilfe der Buttons *New*, *Load* und *Save* kann die Musterliste gelöscht, eine neue Musterliste geladen oder die aktuelle Musterliste gespeichert werden.

## 7 Trainingsanleitung

Das Trainieren des Netzes geschieht in vier Schritten.

### 7.1 Aufnehmen der zu lernenden Samples

An erster Stelle steht die Gewinnung von verwertbarem Sound-Material, von dem die Taktarten bekannt sind beziehungsweise ermittelt werden können. Das Material muss als Wave-Datei mit einer Tonspur bei 44kHz Abtastfrequenz und 16Bit Auflösung vorliegen. Zu Demonstrationszwecken wurden mit Hilfe eines Drumcomputers<sup>3</sup> 9 verschiedene

---

<sup>3</sup>es kam das OpenSource- Programm Hydrogen zum Einsatz

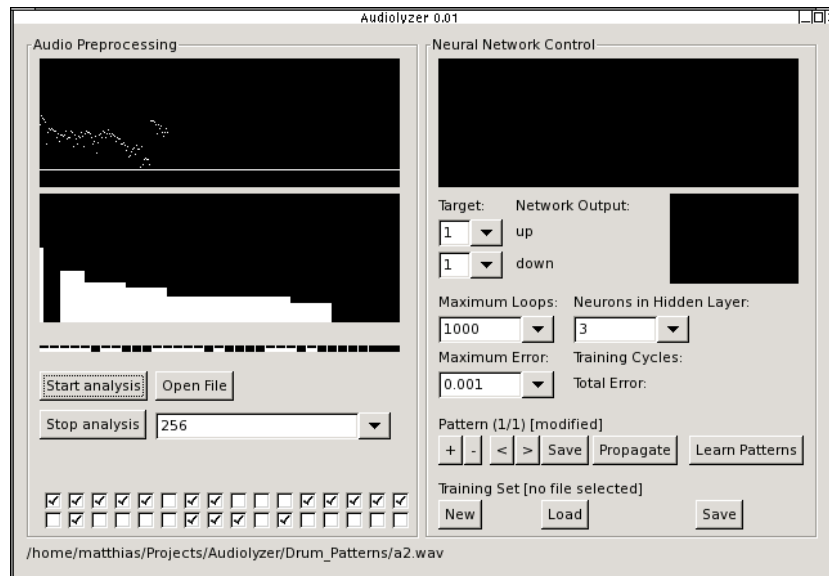


Abbildung 6: Signalanalyse mit dem Audio-Preprozessor

Muster eingespielt.

## 7.2 Analyse der vorliegenden Samples

Zur Umwandlung der Samples in ein Eingabemuster werden diese zunächst einzeln geladen und analysiert. Nach der Analyse ist die Eingabe in den aktuellen Musterdatensatz gespeichert. Es muss also vorher bereits ein Datensatz angelegt worden sein. Dieser Vorgang ist für jedes Eingabemuster zu wiederholen (Abbildung 6).

## 7.3 Bewertung der Eingabemuster

Nachdem alle Samples analysiert sind, können die Eingabemuster bewertet werden. Bewerten bedeutet in diesem Zusammenhang das Zuordnen einer konkreten Taktart. Die Taktart wird über die mit *Target* bezeichneten Comboboxen eingestellt. Nach der Einstellung muss der geänderte Datensatz mit *Save* zurückgeschrieben werden.

## 7.4 Starten des Lernvorganges

Mit dem Starten des Lernvorganges beginnt das automatisierte Training. Lernfortschritt und Fehler können an den Anzeigen um die Lernkurve abgelesen werden (Abbildung 7).

## 7.5 Propagation

Nicht mehr zum Trainingsvorgang gehört das manuelle Propagieren von Pattern. Dennoch eignet es sich gut zur stichprobenartigen Kontrolle des Lernerfolges. Dazu wird ein

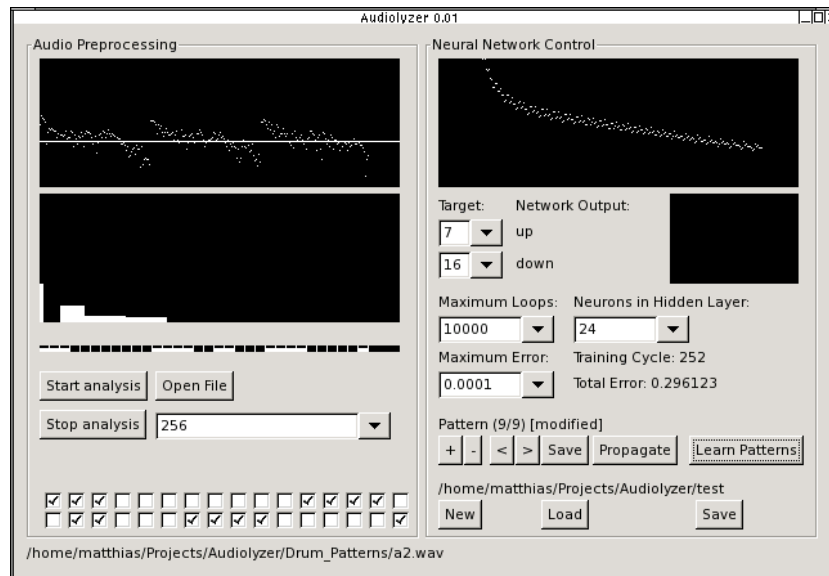


Abbildung 7: Anlernen des neuronalen Netzes und Darstellung der Lernkurve

beliebiges Muster ausgewählt und propagiert (Abbildung 8).

## 8 Beobachtungen und Erfahrungen

Anfangs verwendete ich eine feste Anzahl an Neuronen für den Hidden-Layer. Ich hielt 16 Stück für ausreichend - erhöhte später noch auf 32, was den Lernvorgang - also die Konvergenz des Fehlers - sehr beschleunigte. Allerdings stellte sich dabei heraus, dass die Generalisierungsfähigkeit des Netzes dadurch stark eingeschränkt wurde. Um besser experimentieren zu können, fügte ich einen Regler in die Benutzeroberfläche ein, mit dem sich die Anzahl der Neuronen im Hidden-Layer zur Laufzeit steuern lässt. Damit konnte ich sehr interessante Ergebnisse erzielen. So reichten bei einer entsprechend höheren Anzahl an Lernschritten bereits 3 Neuronen aus, um alle 9 Testmuster korrekt zu erkennen. Auch bei manueller Manipulation der angelegten Werte am Eingangslayer kamen stabile Ergebnisse heraus. Die Fähigkeit zu generalisieren konnte damit nachgewiesen werden.

## 9 Probleme und mögliche Verbesserungen

- Muster, die propagiert werden sollen, müssen vorher gespeichert werden.
- Die Frequenzanalyse funktioniert auf Grund von Geschwindigkeitsproblemen auf durchschnittlicher Hardware nicht in Echtzeit.
- Erkennung der Takte ist nur mit Test-Samples einer definierten Länge möglich. Hier könnte der Versuch unternommen werden, aus der Folge mehrerer aufein-

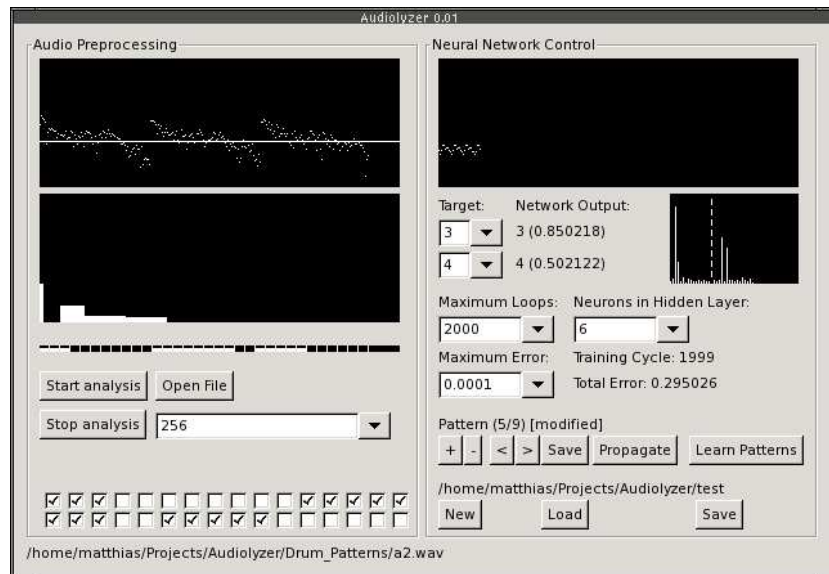


Abbildung 8: Darstellung der Werteverteilung der Ausgangsneuronen nach der Propagation

ander folgender Takte eine Periodizität zu erkennen. Dazu wäre allerdings eine intensivere Auseinandersetzung mit der Musiktheorie notwendig, als wie sie im Rahmen dieses Projektes möglich war.

## 10 Fazit

Die im Rahmen des Beleges erarbeitete Software ist so noch nicht produktiv einsetzbar. Sie zeigt jedoch, dass die Technologien der neuronalen Netze im Bereich der Musikanalyse anwendbar sind. Bei meinen Recherchen zur Thematik stieß ich zudem auf weitere interessante Anwendungen von neuronalen Netzen im Audibereich. Besonders hervorheben möchte ich an dieser Stelle die freie Software “Audacity“, die einen lernfähigen Rauschunterdrückungsfilter enthält. Als persönliches Fazit möchte ich anmerken, dass mir die Arbeit auf diesem Gebiet viele neue Erkenntnisse gebracht hat - nicht zuletzt weil der praktische Umgang mit neuronalen Netzen viele weitere Anwendungsmöglichkeiten offenbart hat.

## 11 Quellenangaben

- Holst, Imogen; Das ABC der Musik; 1992
- Brause, Rüdiger; Neuronale Netze - eine Einführung in die Neuroinformatik; 1991
- Grauel, Adolf; Neuronale Netze - Grundlagen und mathematische Modellierung; 1992
- Rossum, Peter van; <http://lwneuralnet.sourceforge.net/>