Enrico Biermann (enrico@cs.tu-berlin.de)
Timo Glaser (timog@cs.tu-berlin.de)
Marco Kunze (makunze@cs.tu-berlin.de)
Sebastian Nowozin (nowozin@cs.tu-berlin.de)

WS 2002/03
15. 1. 2003

Resultate der Planungsphase

1 Einleitung

Das Dokument beschreibt die Resultate der ersten Phase des Projektes "Neuronales Netz" des C++ Programmier Praktikums vom Wintersemester 2002/2003 an der TU-Berlin [1]. Die erste Phase umfasst die Ausarbeitung mathematischer Grundlagen zur Problemstellung und das software-technische Design des Projektes.

Das Projekt selber besteht in der Realisierung einer einfachen Spracherkennung. Dabei werden dem Programm Spracheingaben geliefert, die zu klassifizieren sind. Die Eingaben bestehen ausschließlich aus Vokalen, die in fünf Klassen einzuteilen sind. Intern soll sich das Programm mehrerer neuronaler Netze bedienen, die erst trainiert und anschließend zur Klassifikation genutzt werden sollen. Die Interaktion mit dem Benutzer soll komplett über eine GUI realisiert werden.

2 Module und Moduldiagramm

Im Kern der Planungsphase stehen strukturelle Entscheidungen der Aufteilung der einzelnen Aufgaben. Diese Aufteilung wurde von uns gemeinsam getroffen und vier Programmteile wurden identifiziert. Die Module sind im einzelnen:

- Neuronales Netzwerk

  Mehrere neuronale Netzwerke bilden den Kern der Klassifizierungsalgorithmen. Eine flexible und robuste Klasse zum Arbeiten mit dem neuronalen Netzwerk ist daher Grundlage für dieses Projekt.

- Audioverarbeitung

  Die Sprachdateien liegen in speziell formatierten Eingabedateien vor, die eingelesen werden müssen. Weiter müssen die Audiodaten mehrere Verarbeitungsschritte durchlaufen, bevor sie durch die Vokalerkennung genutzt werden können.

- Vokalerkennung

  Die Vokalerkennung ist der eigentliche Mittelpunkt des Programms indem die Klassifikation statt findet. Dieses Modul stützt sich auf die zwei Säulen für die Eingabedaten und die Verarbeitung der Eingabedaten: der Audioverarbeitung und dem neuronalen Netz. Die Vokalerkennung ist damit Bindeglied zwischen den Sprachdaten und dem neuronalen Netz.

- Grafisches Benutzerinterface (Yava)

  Dieses Modul setzt auf die drei vorherigen auf und regelt die Interaktion mit dem Benutzer. Die grafische Benutzeroberfläche ist dabei so gestaltet, dass die einzelnen Teile des Programms erkennbar sind und dynamisch modifiziert werden können.

In Abbildung 1 wird das Moduldiagramm mit den vier oben besprochenen Programmteilen und deren Abhängigkeiten dargestellt.

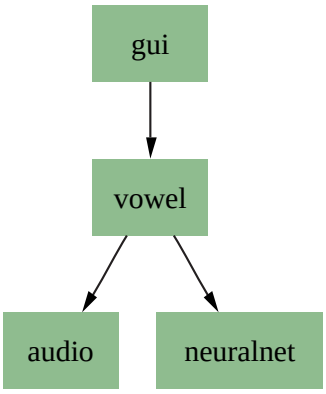

Abbildung 1: Grundlegende Modulstruktur des Programms

3 Datenfluss zwischen den Modulen

Wir haben den Datenfluss in einzelne Diagramme zerlegt, um eine übersichtlichere Darstellung zu erhalten. Dabei wird zu jeder Hauptaktion des Programms ein Datenflussdiagramm erstellt.

3.1 Hinzufügen der Vokale zu einem Set

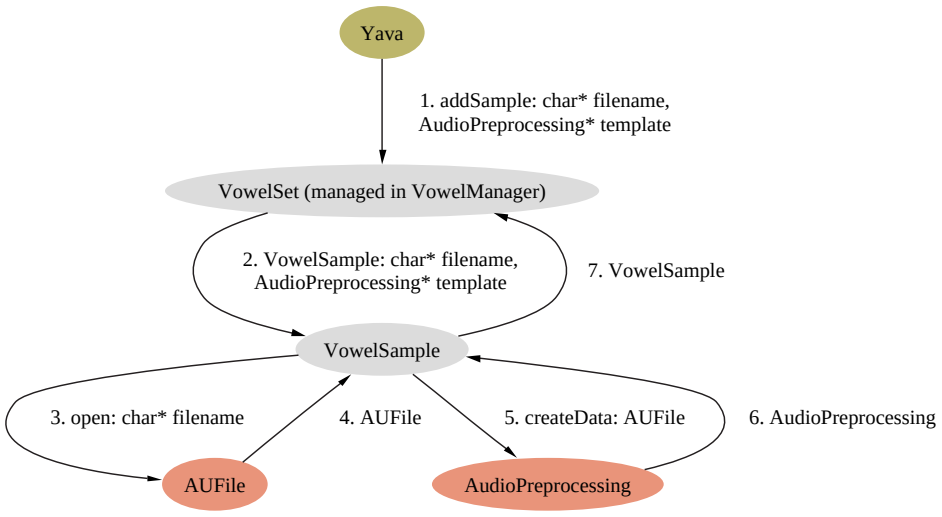

Abbildung 2: Hinzufügen der Vokale zu einem Set

Wird durch die GUI das Hinzufügen eines Vokals ausgelöst, so läuft transparent die Audiovorverarbeitung ab, und der resultierende `VowelSample` Typ wird zu einem `VowelSet` hinzugefügt. Dieser Vorgang ist in Abbildung 2 gezeigt.

3.2 Initialisierung des neuronalen Netzwerkes

Wird das Klassifikationsystem vom Benutzer durch die GUI initialisiert, werden mehrere Parameter an die darunterliegenden Objekte mitgeteilt. Dies sind mehrere Netzwerklayouts, die Aktivie-

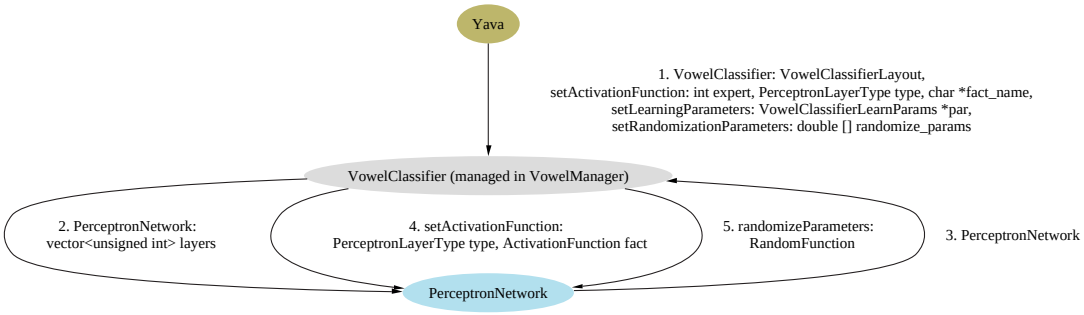

Abbildung 3: Initialisierung des neuronalen Netzwerkes

rungsfunktionen jedes Layers, Lernparameter des Systems und die Zufallseinstellungen, mit denen das Netz initialisiert werden soll. Die Datentypen und deren Interaktion zwischen den Modulen ist in Abbildung 3 gezeigt.

3.3 Klassifikation eines einzelnen Vokals

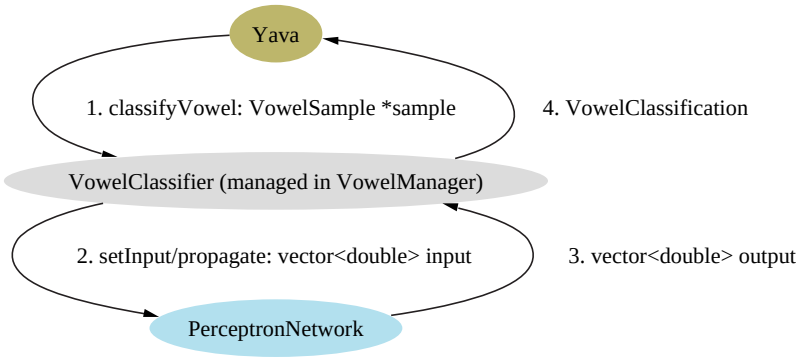

Abbildung 4: Klassifikation eines einzelnen Vokals

Durch die Audiospalte der GUI sind einzelne Vokale klassifizierbar, um den Erfolg des Klassifikationsmoduls zu überprüfen. Wird diese Aktion ausgelöst, so tritt der in Abbildung 4 gezeigte Ablauf in Kraft.

3.4 Lernen und Testen eines `VowelSets`

Die zentrale Aufgabe des Programms und einen grossen Teil der GUI in Anspruch nehmend ist das Lernen eines Sets von Vokalen. Dieser Prozess ist in Abbildung 5 aufgezeigt. Der `VowelManager` verwaltet dabei die einzelnen Vokale und implementiert sowohl Batch- als auch Onlinelearning auf sehr ähnliche Weise.

4 Inklusionsstruktur

Die in Abbildung 6 gezeigte Inklusionsstruktur gibt die Abhängigkeiten der Headerdateien an. Ein Pfeil von A nach B heißt dabei “A inkludiert B”. Der besseren Übersichtlichkeit willen wurde

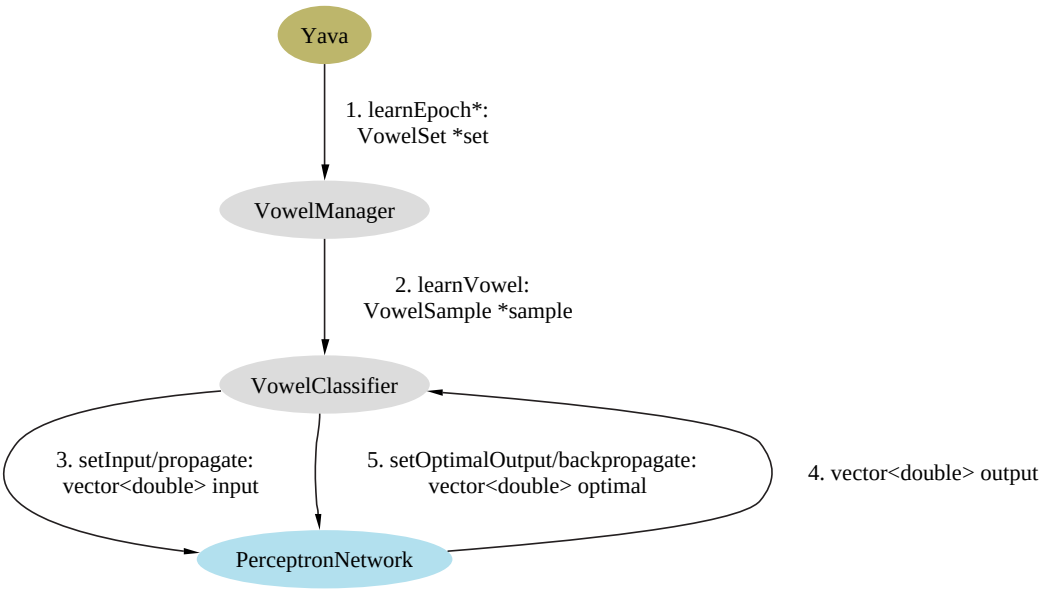

Abbildung 5: Lernen und Testen eines `VowelSets`

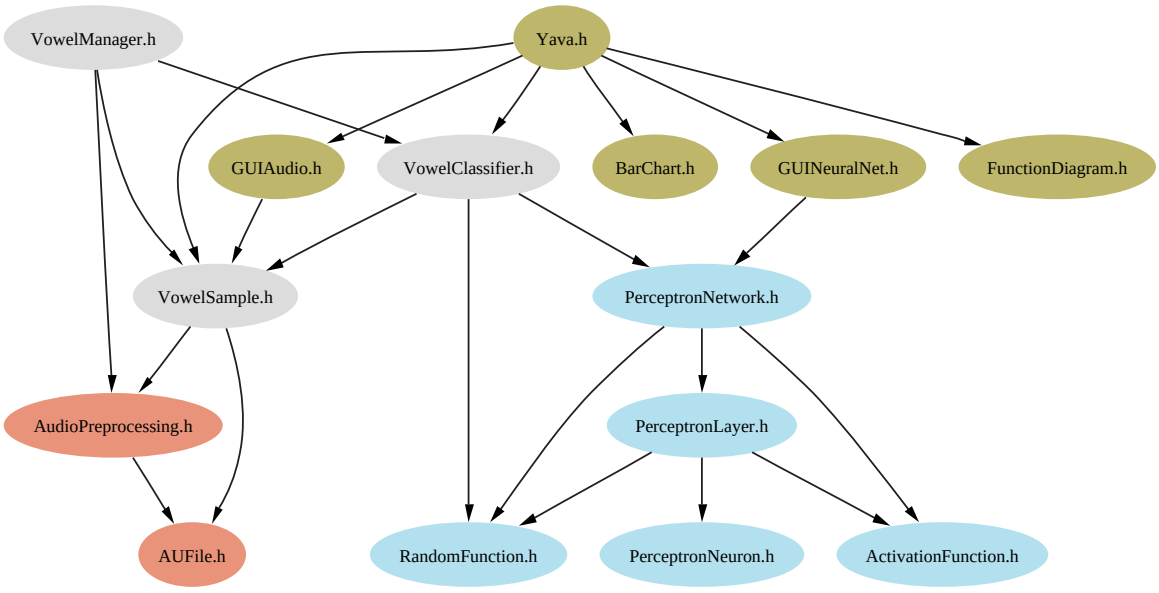

Abbildung 6: Inklusionsstruktur des Programms

jedem Modul eine Farbe zugeordnet. Man erkennt deutlich, die zunehmende Abstraktion nach oben hin.

5 Header Dateien und Makefile

Der komplette Verzeichnisbaum des Programms befindet sich im Tarball `Yava-Phase-1.tar.gz`.

6 Die einzelnen Module

6.1 Neuronales Netzwerk

Die mathematische Herleitung der hier verwendeten Algorithmen und der Beschreibung der verwendeten Lernverfahren befindet sich im Anhang “Grundlagen neuronaler Netzwerke”.

6.1.1 Implementation

Die Implementation des neuronalen Netzwerkes bildet die Struktur des mathematischen Models ab. Dabei werden im wesentlichen drei Klassen benutzt, die sich wie in Abbildung 7 dargestellt zusammensetzen.

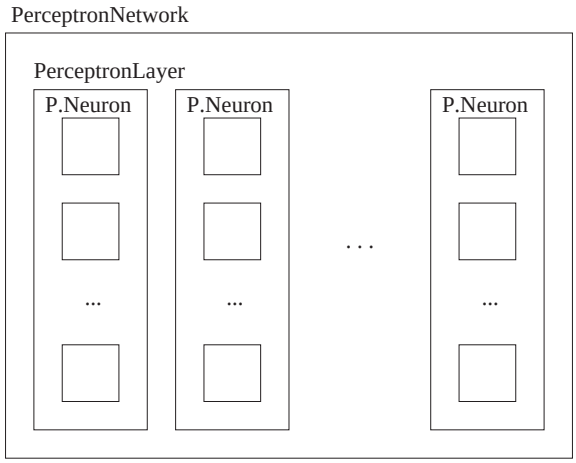

Abbildung 7: Klassen der Implementation des neuronalen Netzwerkes

Die drei Klassen sind:

- `PerceptronNetwork`

  Dies ist die einzige Klasse, die von Außen benutzt zu werden braucht. Sie bietet Methoden für alle vier Algorithmen die im Anhang besprochen werden. Weitere interessante Methoden bieten eine load und save Funktionalität um das gesamte Netz abzuspeichern und durch einen Konstruktor wieder laden zu können. Die Klasse wird von dem Vokalerkennungsmodul benutzt. Die Klasse `WeightMatrix` des GUI Moduls stellt interne Daten der Netzwerkklasse grafisch dar.

- `PerceptronLayer`

  Einzelne Schichten im Netzwerk werden durch die `PerceptronLayer` Klasse modelliert. Dabei hat die `PerceptronNetwork` Klasse einen variabel grossen Vektor von Schichten. Jede Schicht bietet auf Schichtebene die vier besprochenen Algorithmen. GUI Klassen stellen Elemente dieser Klasse grafisch dar.

- `PerceptronNeuron`

  Was die Layer für das gesamte Netzwerk sind, sind die Neuronen für die einzelnen Schichten. Mehrere Objekt dieses Typs werden in einer `PerceptronLayer` Klasse verwaltet. GUI Klassen regeln die grafische Darstellung von Werten dieser Klasse.

Zusätzlich zu diesen Grundklassen werden zwei weitere Hilfsklassen eingesetzt. Die Klasse `ActivationFunctions` bietet mehrere Standard Aktivierungsfunktionen an, die zur Initialisierung des Netzwerkes verwendet werden können. Die Klasse `RandomFunctions` bietet Funktionen zur selektiven Erzeugung von Zufallszahlen, die zum Beispiel in einem bestimmten Bereich liegen.

6.2 Audioverarbeitung

Die Audiovorverarbeitung stellt der Vokalerkennung die Mittel zur Behandlung der Audiodateien bereit. Dabei wurde eine Aufteilung in zwei Klassen vorgenommen:

- `AUFile`

  Diese Klasse behandelt alle Operationen, die direkt mit den Audiodaten auf Dateiebene zu tun haben. Dazu gehören das Öffnen und Schreiben der Dateien, die Normalisierung und das Suchen der verwendbaren Samplebereiche. Benutzt wird diese Klasse nur von der Vokalerkennung.

- `AudioPreprocessing`

  Alle weiterführenden Methoden, die die Audiorohdaten speziell auf die Verwendung bei der Vokalerkennung vorbereiten, sind in dieser Klasse implementiert. Dazu gehören zum Beispiel Fensterung und Skalierung.

6.2.1 Maximaler Sampleabschnitt

Um einen geeigneten Abschnitt der passenden Länge aus der AU Datei auszuwählen, implementiert die `AUFile` Klasse die Methode `cutSampleRange`. Diese wählt aus den Gesamtsounddaten einen Abschnitt fester Länge aus, für den die Summe der Amplitudenbeträge maximal wird. Dazu wird ein Fenster der gewünschten Länge über die Audiodaten geschoben und die Summe der Amplitudenbeträge in diesem Fenster berechnet. Der Index, für den diese Summe maximal gross ist wird behalten. So lässt sich in linearer Zeit das Fenster im Sample bestimmen, was die grösste Amplitudenbetragssumme hat.

6.2.2 Fensterung

Die in der Klasse `AudioPreprocessing` implementierte Fensterung wird exakt nach den in der Projektbeschreibung gezeigten Formeln implementiert, daher sei auf eine Beschreibung hier verzichtet [1].

6.3 Vokalerkennung

Die Vokalerkennung benutzt drei neuronale Netze um Vokale zu klassifizieren. Das Gesamtproblem wird von vier wesentlichen Teilen gelöst:

- `VowelSample`

  Eine Klasse repräsentiert alle für einen Vokal benötigten Informationen. Dazu gehören die aus der Audiovorverarbeitung stammenden Daten und der korrekte Vokaltyp. Auch die Konvertierung zwischen dem aus der `AUFile` Klasse stammenden Userdaten-Bytearray und einer vom Programm nutzbaren C Strukturrepräsentation findet hier statt.

- `VowelClassification`

  Die Klassifikation eines `VowelSample` wird durch diese Klasse bewältigt. Hierzu werden drei neuronale Netze angesteuert, wie in der Beschreibung der Projektaufgabe gezeigt. Jedes neuronale Netz hat drei Ausgänge. Zum Treffen der Klassifikationsentscheidung wird eine recht einfache Strategie verfolgt: Der aktivste Ausgang jedes neuronalen Netzes bestimmt eine der drei möglichen Optionen. Es findet also eine Informationsreduktion statt, bei der viel Informationen verloren gehen. Aus der Gesamtmenge der Möglichkeiten, 27 an der Zahl $(3 \cdot 3 \cdot 3)$ verbleiben fünf sinnvolle Entscheidungen, nämlich die der zu erkennenden Vokale.

  Für eine Verbesserung dieser Methode bieten sich vielleicht Bayesische Entscheidungsnetzwerke an, die mit den Wahrscheinlichkeiten - also den “Sicherheiten” der Teilnetze - bessere Entscheidungen über die möglichen Vokale treffen können. Eine Implementation ist vorerst nicht geplant, wenn zum Ende des Projektes noch Zeit dafür ist, werden wir dies aber tun.

- `VowelSet`

  Da das Gesamtsystem Vokalerkennung die Möglichkeit zum Verwalten mehrerer Vokale benötigt, ist hier eine `VowelSet` Klasse implementiert, die diese Aufgabe bewältigt. Sie bietet Methoden für das Hinzufügen, Austauschen und Entfernen einzelner `VowelSample` Objekte.

- `VowelManager`

  Die `VowelManager` Klasse stellt die grosse Verwaltungsklasse des Vokalerkennungsmoduls dar. Es existiert nur ein Objekt dieser Klasse im Programm, das grösstenteils von der GUI angesteuert wird. Dieses Objekt verwaltet zwei `VowelSet` Objekte, für die Trainings und Test Daten. Des weiteren wird ein `VowelClassification` Objekt verwaltet und ein Hilfsobjekt zum Verarbeiten von Audiodaten erzeugt. Die GUI interagiert hauptsächlich mit dieser Klasse.

6.4 Grafische Benutzeroberfläche (GUI)

6.4.1 Das Design

- Die Audioverwaltung

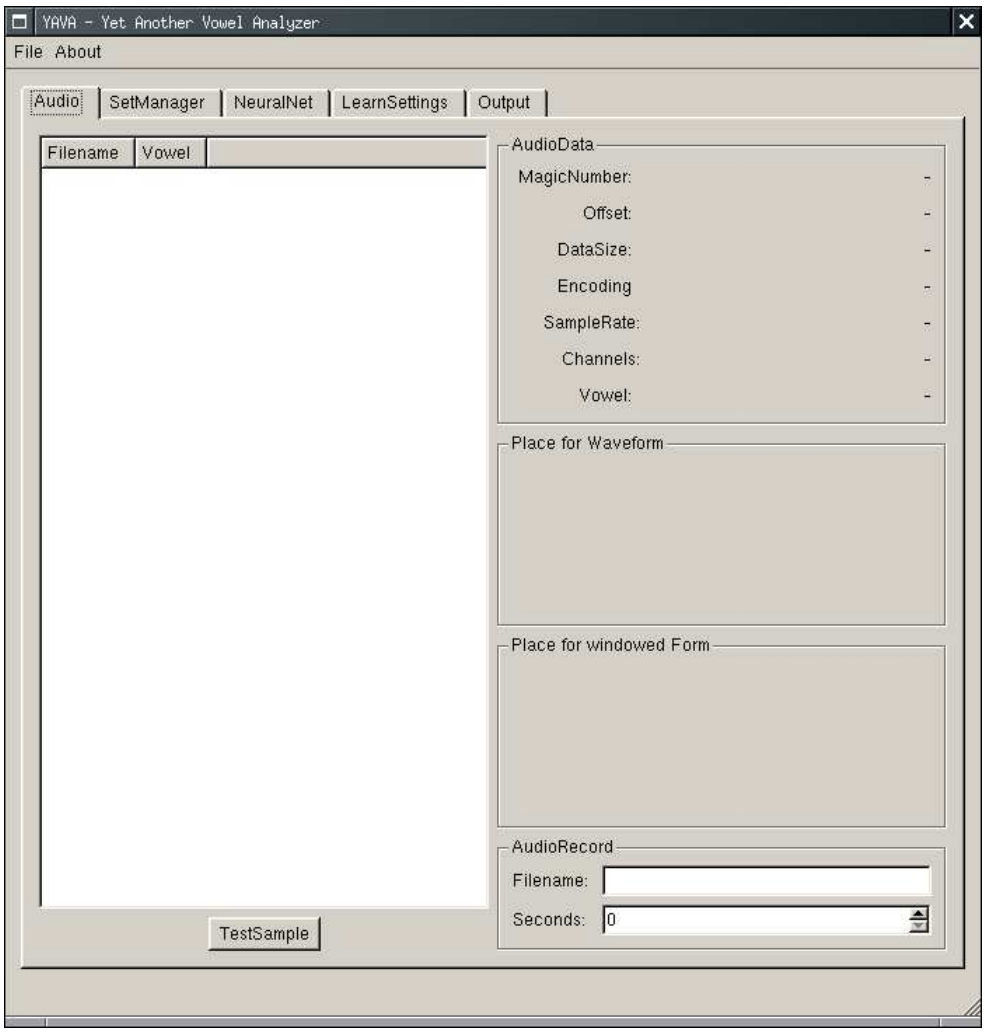

Abbildung 8: Audioverwaltung

Die Audioverwaltung dient zum Einsehen von Dateieigenschaften, sowie der Veranschaulichung der Vorverarbeitung von Audiodateien. Dazu bietet sie die Möglichkeit neue Samples aufzunehmen, und man kann die Dateien einzeln am neuronalen Netz testen.

Diese Tabulatorseite wird quasi komplett in der `GUIAudio.h` realisiert werden, abgesehen von der Dateiliste, da diese auch von dem Setmanager benutzt werden kann. Die Kommunikation der Daten wird komplett mit der `VowelSample.h` abgewickelt, da diese alle geforderten Informationen zur Verfügung stellen kann.

- Der Setmanager

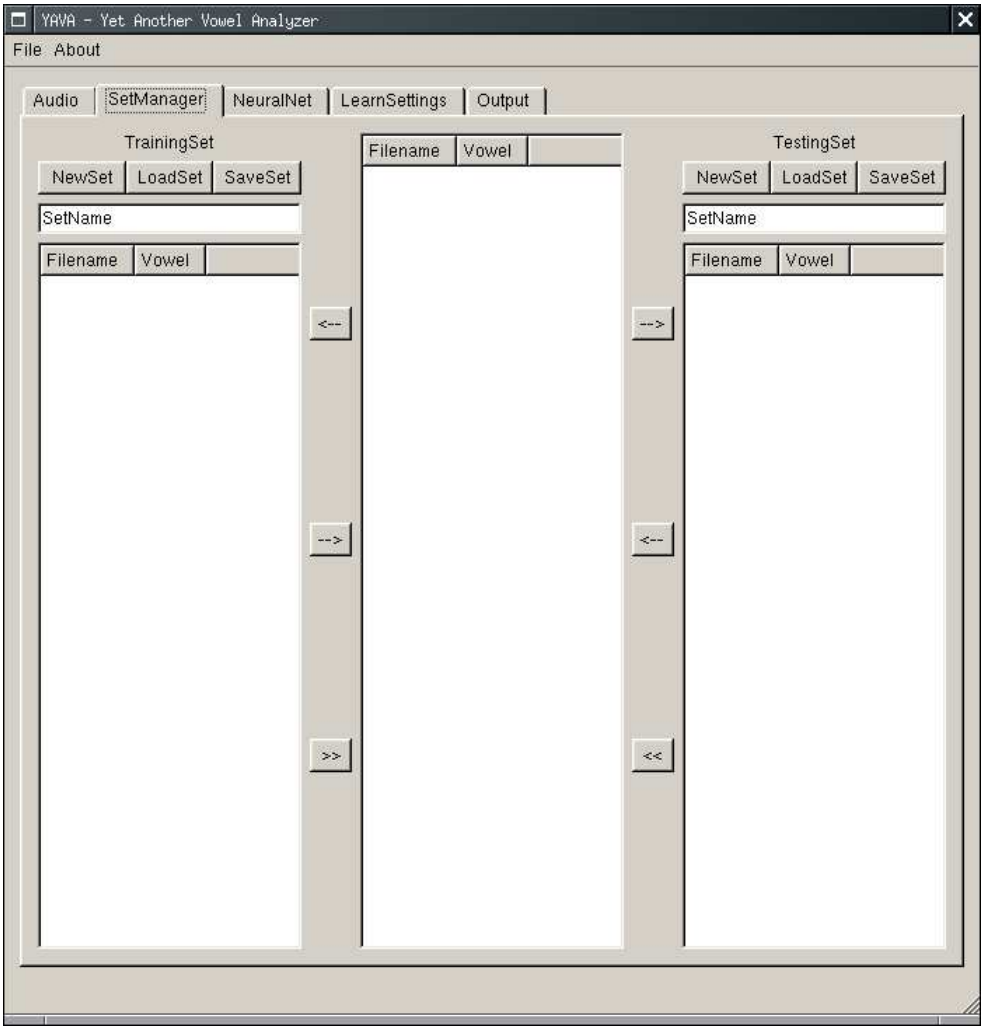

Abbildung 9: Verwaltung der Sets

Der Setmanager ist für die Verwaltung der Sets zuständig und bietet zwei Widgets für die Verwaltung des Test- und Trainingssets, die die Möglichkeit bieten, komplette Sets zu laden, speichern, erstellen, einzelne Dateien zu löschen oder untereinander zu verschieben sowie aus der Dateiliste in der Mitte neue Dateien hinzuzufügen.

- Die Einstellung des Neuronalen Netzes

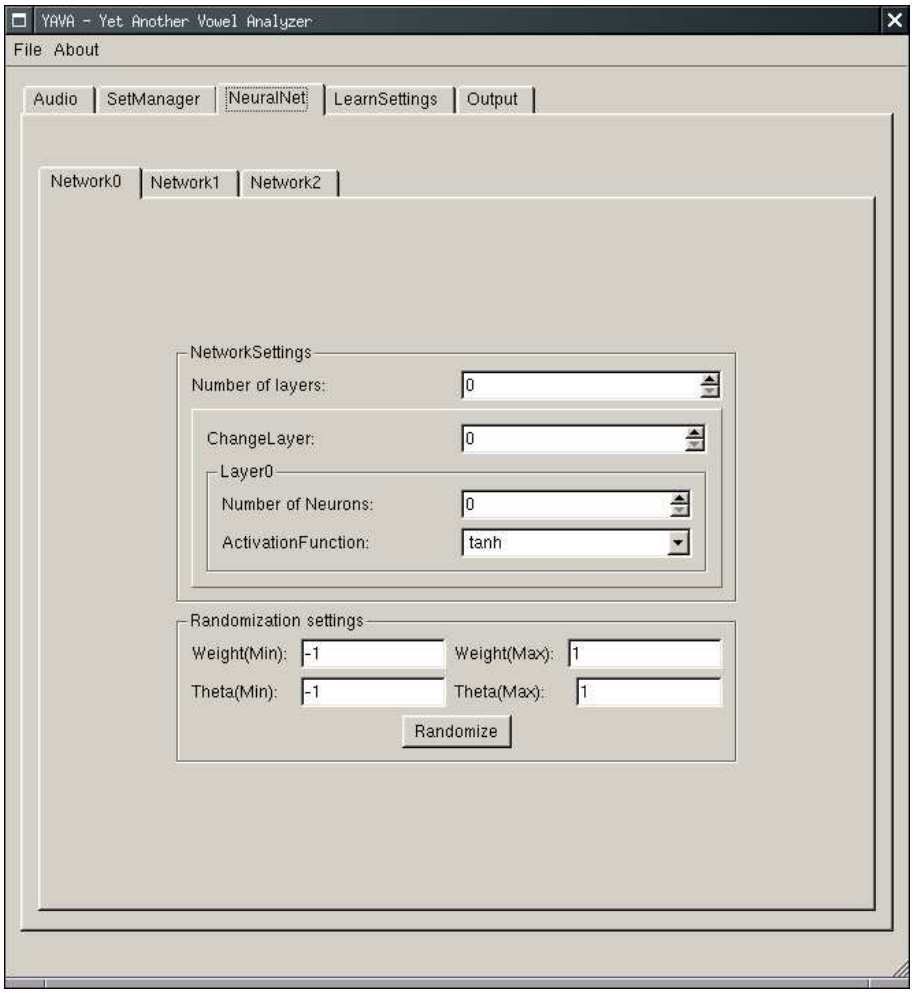

Abbildung 10: Einstellung des Neuronalen Netzes

Diese Tabulatorseite bietet drei weitere, jeweils für eines der drei Neuronale Netze. Jedes Widget zur Konfiguration eines Netzes ermöglicht die Einstellung der Anzahl der Layer, Anzahl der Neuronen innerhalb der Layer, Aktivierungsfunktion, Initialisierung der Schwellwerte und Gewichte.

- Die Einstellung der Lernparameter

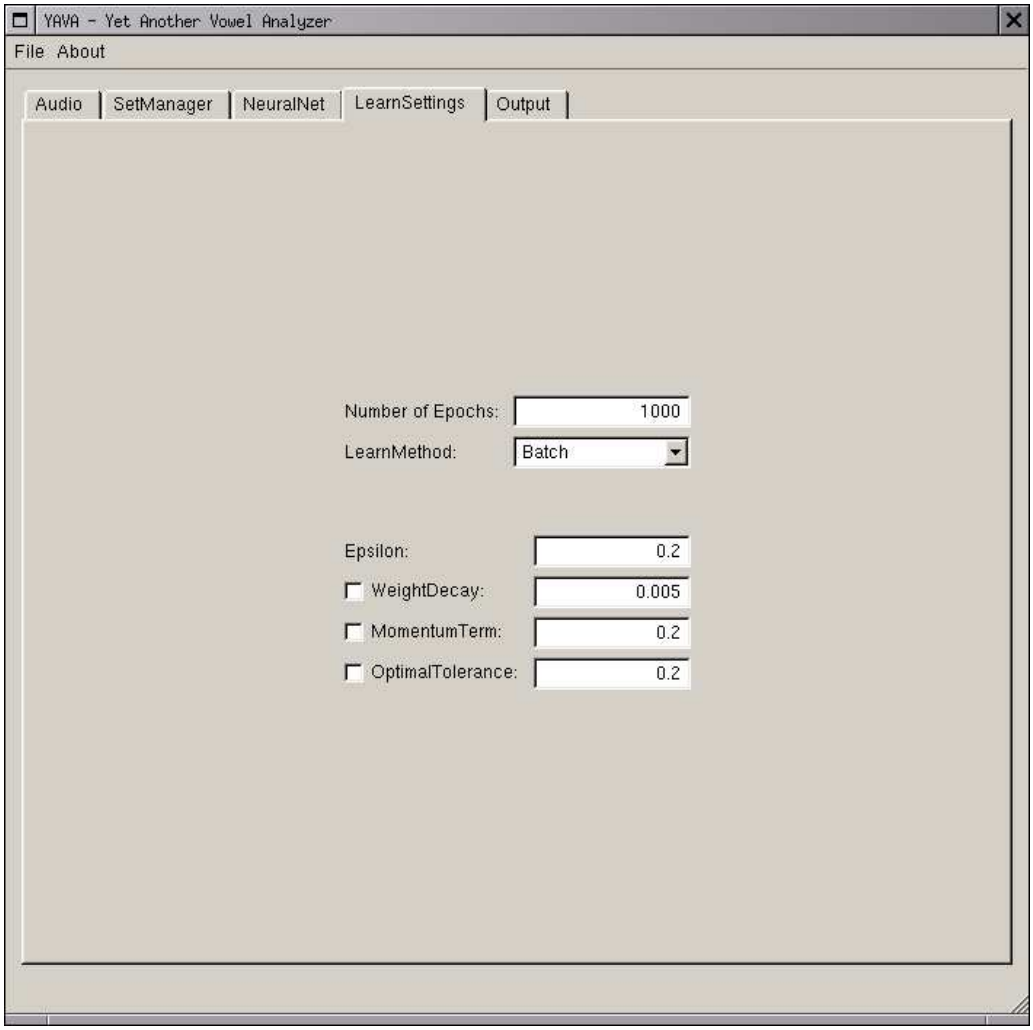

Abbildung 11: Einstellung der Lernparameter

Hier kann man die Lernparameter einstellen, wie zum Beispiel Schrittweite, Lernverfahren und ein paar Optimierungen.

- Die Darstellung der Ausgabe des Neuronalen Netzes

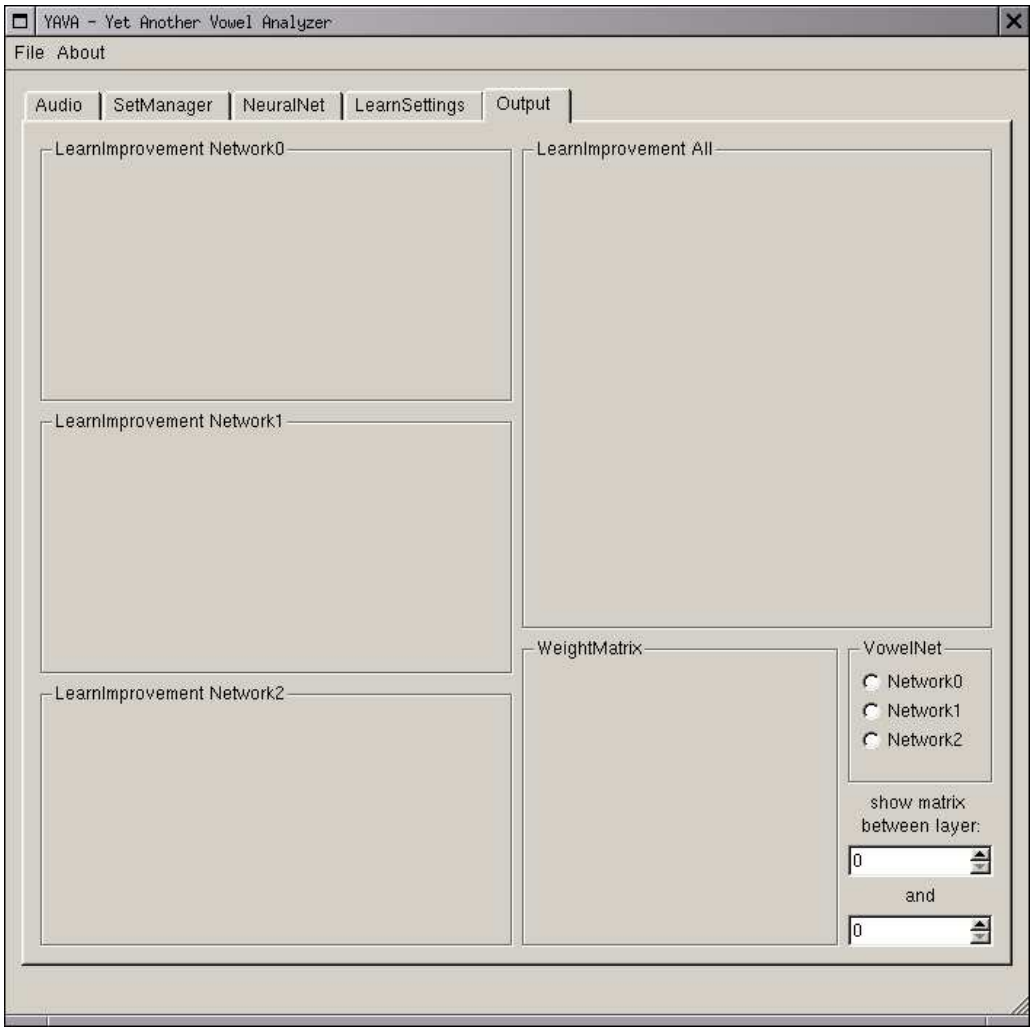

Abbildung 12: Darstellung der Ausgabe des Neuronalen Netzes

Letztendlich wird die Ausgabe des Neuronalen Netzes auf dem Bildschirm dargestellt. Dazu gehören die Fehlerdiagramme (hierbei wird es sich um Kurvendiagramme bei Sets handeln und um Balkendiagramme bei einzeln getesteten Dateien) und die Gewichtsmatrix. Bei dieser kann man zusätzlich einstellen, welches Netz dargestellt werden soll und welche Schicht.

6.4.2 Custom Widgets

- `BarChart`

  Mit Hilfe dieses Widgets werden Balkendiagramme gezeichnet. Diese werden bei der Visualisierung der Klassifizierung eines Vokals benutzt, um sowohl die Gesamtentscheidung des Systems anzuzeigen, als auch die Ausgaben der einzelnen neuronalen Netzwerke darzustellen.

- `FunctionDiagram`

  Der Lernerfolg jedes Netzwerkes und des Gesamtsystems wird jeweils durch zwei übereinanderliegende Funktionsgraphen dargestellt. Der eine Funktionsgraph zeigt die Anzahl der korrekt beziehungsweise falsch erkannten Vokale des Trainingssets an, ein darüberliegender

Funktionsgraph das gleiche für das Testset. Diese Darstellung wird vom `FunctionDiagram` Widget erledigt.

- `WaveForm` (in `GUIAudio.h`)

  Die Anzeige eines Vokals, sowohl als Audio Sample, als auch als vorverarbeiteter Eingabevektor, wird durch dieses Widget geregelt.

- `WeightMatrix` (in `GUINeuralNet.h`)

  Das Widget `WeightMatrix` ist identisch mit der in der Projektbeschreibung auf Seite 22 abgebildeten Matrix [1].

6.4.3 Die Hierarchie

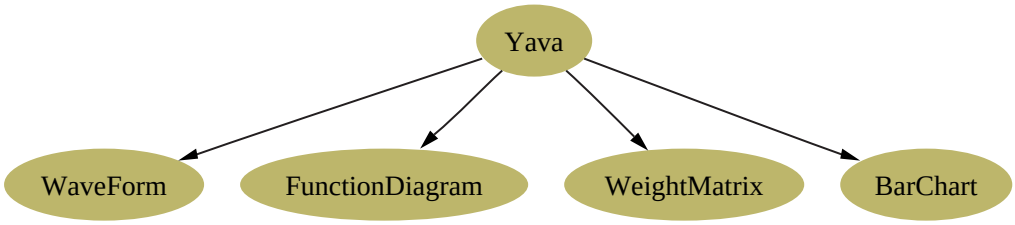

Abbildung 13: Widget-Hierarchie

Die GUI wird durch das `Yava` Modul realisiert werden, welches möglichst weitgehend mit dem QT-Designer erstellt werden soll. Die anderen Module liefern die speziell für die Anwendung erforderlichen Widgets wie zum Beispiel Waveform-Diagramme, Balkendiagramme und Funktionsdiagramme. Da wir im Moment vom QT-Designer noch recht erschlagen sind und uns die Stärken und Schwächen in Bezug auf weniger kleine Projekte noch nicht bekannt sind, kann davon ausgegangen werden, dass sich die GUI über die Zeit hinweg mit der Implementation noch entwickeln wird.

Literatur

[1] C++-Programmier-Praktikum, “Projektaufgabe Pj-NN”
`http://ni.cs.tu-berlin.de/lehre/C-Praktikum/data/Projekte/projekt_nn.pdf`