Universität Zürich / Wintersemester 2001/2002

Semesterarbeit

Virtuelles Labor für Neuronale Netze

vorgelegt von

Rolf Hintermann,
Dielsdorf, ZH, Schweiz,
Matrikelnummer: 98-706-575

Angefertigt am
Institut für Informatik
der Universität Zürich

Leitung: Prof. Dr. R. Pfeifer
Betreuer: H.P. Kunz, Verena Hafner

Abgabe der Arbeit: 3.05.2002

Inhaltsverzeichnis

1. Einleitung

In der Vorlesung „Neuronale Netzwerke“ werden verschiedene Typen von Neuronalen Netzwerken besprochen. Ziel dieser Arbeit ist es, diese verschiedenen Netzwerktypen und Visualisierungen derselben in Java zu implementieren, um Übungsbeispiele durchzuarbeiten oder Demos präsentieren zu können.

2. Merkmale von Neuronalen Netzwerken - four or five Basics

Alle Typen von Neuronalen Netzwerken funktionieren auf ähnliche Weise. Sie können mit „the four or five basics“ (siehe Skript, Kap. 2.) beschrieben und auch unterschieden werden. Diese „basics“ werden nachfolgend kurz beschrieben.

2.1. Knotencharakteristika

Jedes Neuronale Netz besteht aus einer bestimmten Anzahl Knoten, deren Funktionsweise wie folgt beschrieben werden kann:

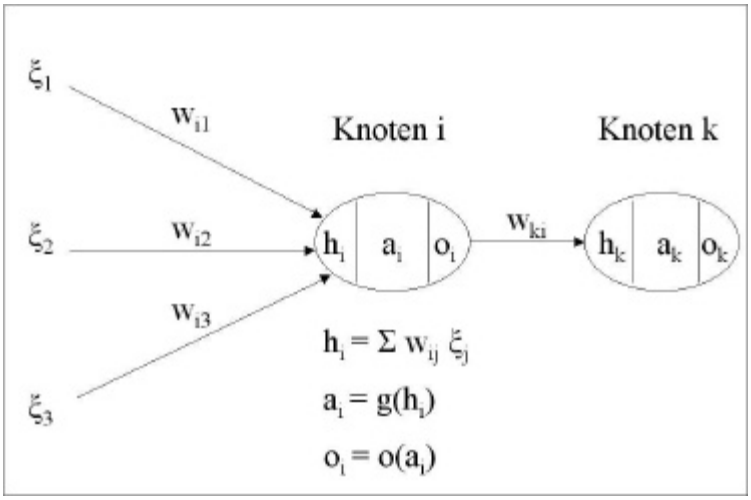


Abb. 2.0: Aufbau eines Knotens

a) ein Knoten *i* erhält einen Nettoinput h_i, der durch die Grösse der von dem zu ihm verbundenen Knoten eingehenden Inputs î, die entsprechenden Gewichte w_{ij} und durch die Propagierungsregel (siehe 2.3.) bestimmt wird.

b) der Knoten *i* erhält einen Aktivierungslevel a_i durch Anwendung der Aktivierungsfunktion $a_i=g(h_i)$.

c) der Knoten *i* gibt einen Output o_i an die mit ihm verbundenen Knoten weiter, der durch Anwendung der Outputfunktion $o_i=o(a_i)$ bestimmt wird. In den hier verwendeten Knotentypen entspricht die Outputfunktion jeweils der Identitätsfunktion.

2.2. Konnektivität

Die Konnektivität beschreibt, wie die einzelnen Knoten im Netzwerk untereinander verbunden sind. Zu einem Knoten *i* führt eine Verbindung von einem Knoten *j* mit dem Gewicht w_{ij}. Dieser Sachverhalt kann sehr gut in einer Matrix dargestellt werden.

Die hier behandelten Netztypen lassen sich in 2 Gruppen einteilen:

a) in Schichten aufgebaute Netzwerke, jede Schicht ist mit der nächsten voll verknüpft, die Verbindungen gehen nur in eine Richtung

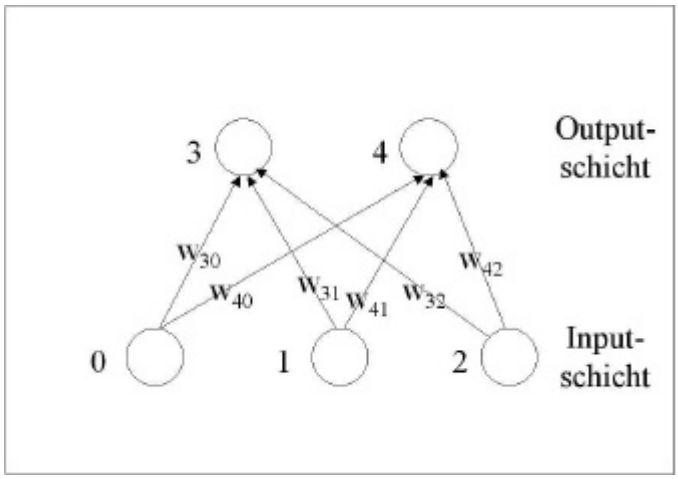


Abb. 2.1: ein einfaches Netzwerk, bestehend aus 2 Schichten

zu\von	0	1	2
3	w_{30}	w_{31}	w_{32}
4	w_{40}	w_{41}	w_{42}

Tab. 2.1: Gewichtsmatrix für das Netzwerk aus Abb. 2.1

b) Netze mit nur einer Schicht in der jeder Knoten mit jedem verknüpft sein kann, also auch rekurrente Verbindungen vorhanden sein können.

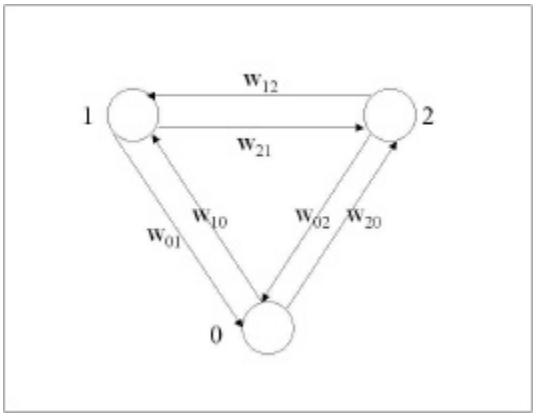


Abb. 2.2: ein einfaches Netzwerk bestehend aus nur einer Schicht

zu\von	0	1	2
0	w_{00}	w_{01}	w_{02}
1	w_{10}	w_{11}	w_{12}
2	w_{20}	w_{21}	w_{22}

Tab. 2.2: Gewichtsmatrix für das Netzwerk aus Abb. 2.2

2.3. Propagierungsregel

Die Propagierungsregel bestimmt, wie die Aktivierung im Netz weitergegeben wird, oder in anderen Worten, wie der Nettoinput eines Knotens berechnet wird.
In den behandelten Netzen wird der Nettoinput wie folgt berechnet:
$h_i = Ó\, w_{ij}\, o_j$

2.4. Lernregel

Die Lernregel bestimmt, wie die Gewichte im Verlauf des Lernprozesses geändert werden. Es lassen sich drei grundlegende Arten von Lernverfahren unterscheiden.

a) supervisierte Lernverfahren – der gewünschte Output zu einem Inputmuster ist bekannt, die Grösse der Gewichtsveränderung hängt vom Fehler des Netzwerks (gewünschter Output – effektiver Output) ab.

b) reinforcement Lernverfahren – dem Netz wird nach der Präsentation eines Musters nur mitgeteilt, ob die Klassifizierung korrekt war oder nicht.

c) nicht supervisierte Lernverfahren – der gewünschte Output zu einem Muster ist nicht bekannt. Die Hebb'sche Lernregel ist der bekannteste Vertreter der nicht supervisierten Lernverfahren.

2.5. Einbettung

Die Einbettung legt fest, wie ein Neuronales Netzwerk mit den Sensoren und Motoren eines Agenten verbunden ist, der mit seiner Umwelt interagieren soll.

3. Das Virtuelle Labor

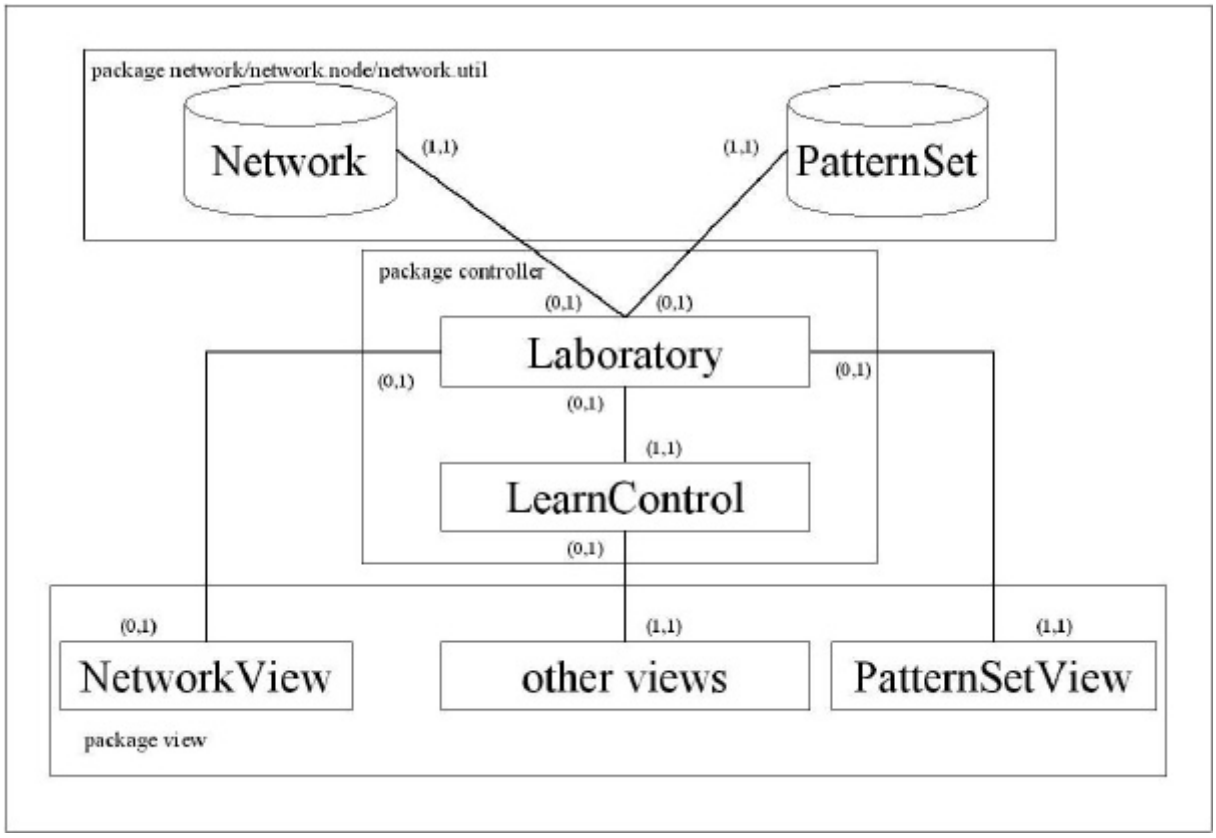


Abb. 3.1: Die Architektur des virtuellen Labors

Nachdem das Programm mittels Kommandozeile („java controller.Laboratory“) gestartet wird, erscheint folgendes Fenster:

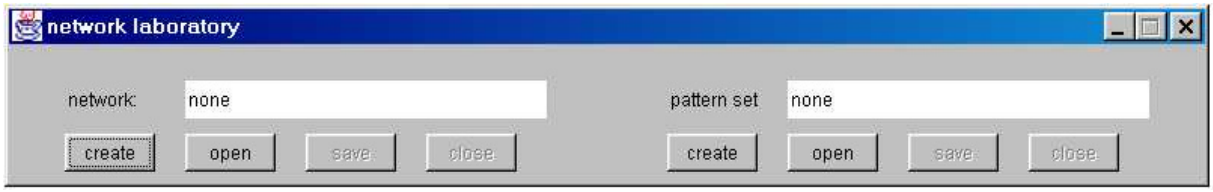


Abb. 3.2: das bei Programmstart erscheinende Fenster

Um im Virtuellen Labor mit einem Netz zu experimentieren

i) muss ein Netz erstellt werden

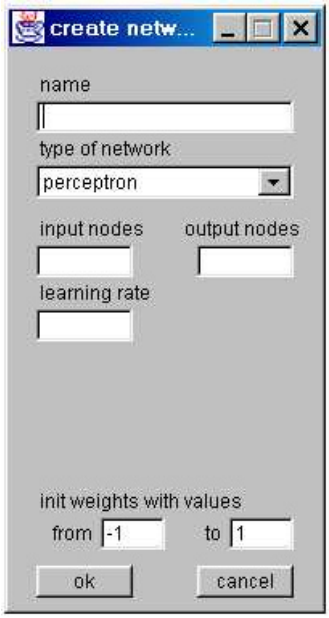
und

ii) müssen dazu passende Muster erstellt werden

Sind die oben aufgeführten Bedingungen erfüllt, wird eine Instanz der Klasse controller.LearnControl erstellt, mittels welcher der Lernvorgang gestartet werden kann (Trainingsmodus) oder dem Netz einzelne Muster präsentiert werden können (Testmodus).

3.1. Erstellen von Netzwerken

Ein Klick auf den „create“ Button auf der linken Seite des Fensters in Abb. 3.2 bewirkt das Erscheinen eines neuen Fensters, in dem der Typ des Netzwerks gewählt werden kann:



der Name des Netzwerks

Typ des Netzwerks

Für den gewählten Netzwerktyp erforderliche Parameter

minimale und maximale Anfangswerte der Gewichte

Abb. 3.3: Fenster zum Erstellen von Netzwerken

Erforderliche Parameter für die implementierten Netzwerktypen

Perceptron und Adaline

Name des Feldes	Inhalt	Sinnvoller Wertebereich
input nodes	Anzahl Knoten in input-schicht, bias nicht mitgezählt	Natürliche Zahl von 1 bis n
output nodes	Anzahl Knoten in output-schicht, bias nicht mitgezählt	Natürliche Zahl von 1 bis n
learning rate	Lernrate	Reelle Zahl von 0 bis 1, eher klein, z.B. 0.2

Perceptron- und Adaline Netzwerke können linear separable Abbildungen lernen. Beim Perceptron wird genau dann etwas gelernt, wenn die Klassifikation eines Musters nicht korrekt war. Die Gewichte, welche von den aktivierten Input Neuronen ausgehen, werden in diesem Fall um die Lernrate verändert, unabhängig von der Grösse des Fehlers. Bei Adaline Netzwerken, welche mit der Delta Lernregel lernen, hat die Grösse des Fehlers einen proportionalen Einfluss auf die Grösse der Gewichtsänderung. Die Delta Lernregel realisiert ein Gradientenverfahren.

Multi layer perceptron

Name des Feldes	Inhalt	Sinnvoller Wertebereich
Layers	Anzahl Knoten in jeder Schicht, beginnend bei der Input-Schicht, durch Kommas getrennt, bias jeweils nicht mitgezählt	Natürliche Zahlen von 1 bis n, z.B. 8,3,8
learning rate	Lernrate	Reelle Zahl von 0 bis 1, eher klein, z.B. 0.2
Momentum	Momentum	Reelle Zahl von 0 bis 1

Multi layer perceptrons können auch nicht linear separable Funktionen lernen. Es gibt jedoch keine Garantie, dass das globale Minimum der Fehlerfunktion gefunden wird. Im nicht linearen Fall können in der Fehlerfunktion lokale Minima existieren, die möglicherweise nicht

überwunden werden können. Um die Wahrscheinlichkeit zu verkleinern, in einem lokalen Minimum stecken zu bleiben, kann ein Momentum (Trägheits-) Term eingeführt werden, der einen Teil der Gewichtsänderung des letzten Lernschritts zu derjenigen des aktuellen Lernschritts dazu rechnet.

Hebb, Oja, Sanger

Name des Feldes	Inhalt	Sinnvoller Wertebereich
input nodes	Anzahl Knoten in input-schicht, bias nicht mitgezählt	Natürliche Zahl von 1 bis n
output nodes	Anzahl Knoten in output-schicht, bias nicht mitgezählt	Natürliche Zahl von 1 bis n
learning rate	Lernrate	Reelle Zahl von 0 bis 1, eher klein, z.B. 0.2

Diese Netzwerktypen lernen die Hauptkomponenten einer Datenverteilung mit dem Mittelwert im Ursprung. Bei reinem Hebb'schen Lernen können die Gewichte nur wachsen, d.h. sie streben gegen unendlich. Die Oja und die Sanger Lernregel enthalten einen Zerfallsfaktor, der dafür sorgt, dass die Gewichte, falls sie zu Beginn normalisiert wurden, auch weiterhin normalisiert bleiben. Ein Oja Netz mit einem Output Knoten findet die Erste Hauptkomponente der Daten. Ein Sanger Netzwerk mit n Output Knoten findet die ersten n Hauptkomponenten der Verteilung, absteigend geordnet an den Ausgängen.

Kohonen map

Name des Feldes	Inhalt	Sinnvoller Wertebereich
input nodes	Anzahl Knoten in input-schicht, bias nicht mitgezählt	Natürliche Zahl von 1 bis n
map (CheckboxGroup)	Wahl, ob die einzelnen Dimensionen jeweils kreisförmig geschlossen oder entlang einer Linie verlaufen	„line“ oder „circle“
kohonen nodes	Anzahl Knoten je Dimension	Natürliche Zahl von 1 bis n
to the power of	Anzahl Dimensionen	Natürliche Zahl von 1 bis n
learning rate	Lernrate zu Beginn	Reelle Zahl von 0 bis 1, eher gross, z.B. 0.9
decay	Zerfallsfaktor, mit dem die Lernrate nach jeder Epoche multipliziert wird	Reelle Zahl von 0 bis 1, eher gross, z.B. 0.9
learn radius	Lernradius zu Beginn	Natürliche Zahl von 1 bis n, der Lernradius sollte zu Beginn einen grossen Teil der Karte abdecken. Der Abstand zwischen zwei benachbarten Knoten in der Kohonenschicht beträgt 1
decay	Zerfallsfaktor, mit dem der Lernradius nach jeder Epoche multipliziert wird	Reelle Zahl von 0 bis 1, eher gross, z.B. 0.9

In einem Kohonen Netzwerk soll ein Input Raum auf die Kohonen Schicht abgebildet werden, wobei die Topologie erhalten werden soll. D.h. benachbarte Knoten in der Kohonen Schicht sollen ähnliche Eingabevektoren repräsentieren. Um dies zu erreichen, wird nach der Präsentation eines Musters das Gewinner Neuron der Kohonen Schicht ermittelt. Sieger wird das Neuron, dessen Gewichtsvektor dem Eingabevektor am nächsten liegt. Anschliessend werden die Gewichte des Sieger Knotens und seiner Nachbarn innerhalb des Lernradius zu dem Eingabevektor hin verschoben. Die Grösse der Veränderung hängt einerseits ab von der

Lernrate, andererseits von der Nachbarschaftsfunktion, je kleiner der Abstand des Betreffenden Knotens zum Sieger (in der Karte), umso grösser die Änderung. Sowohl Lernradius als auch Lernrate werden im Verlauf des Lernvorgangs kleiner.

Hopfield

Name des Feldes	Inhalt	Sinnvoller Wertebereich
number of nodes	Anzahl Knoten	Natürliche Zahl von 1 bis n
update state	Wahl, ob die Aufdatierung der Knoten synchron (kann oszillieren) oder asynchron stattfinden soll	„synchronous“ oder „asynchrounous“

In ein Hopfield Netzwerk können verschiedene Muster bestehend aus 1 / –1 „gespeichert“ werden. Dabei werden die Gewichte mittels Hebb’schem Lernen verändert. Die Kapazität beträgt ca. 10% der Anzahl der Knoten. Wird dem trainierten Netz ein neues Muster präsentiert, so wird die Aktivierung des Netzwerks- vorausgesetzt, ein stabiler Zustand wird erreicht- dem gespeicherten Muster entsprechen, welches dem präsentierten am ähnlichsten ist. Zu beachten ist, dass mit jedem Muster, welches gespeichert wird auch das Negativ dazu später als sogenannter „spurious attractor“ ein möglicher Endzustand des Netzwerks sein kann.

Sind alle Felder korrekt ausgefüllt und wird der „ok“ Button geklickt, so wird ein Netzwerk erstellt und in neuen Fenstern dargestellt.

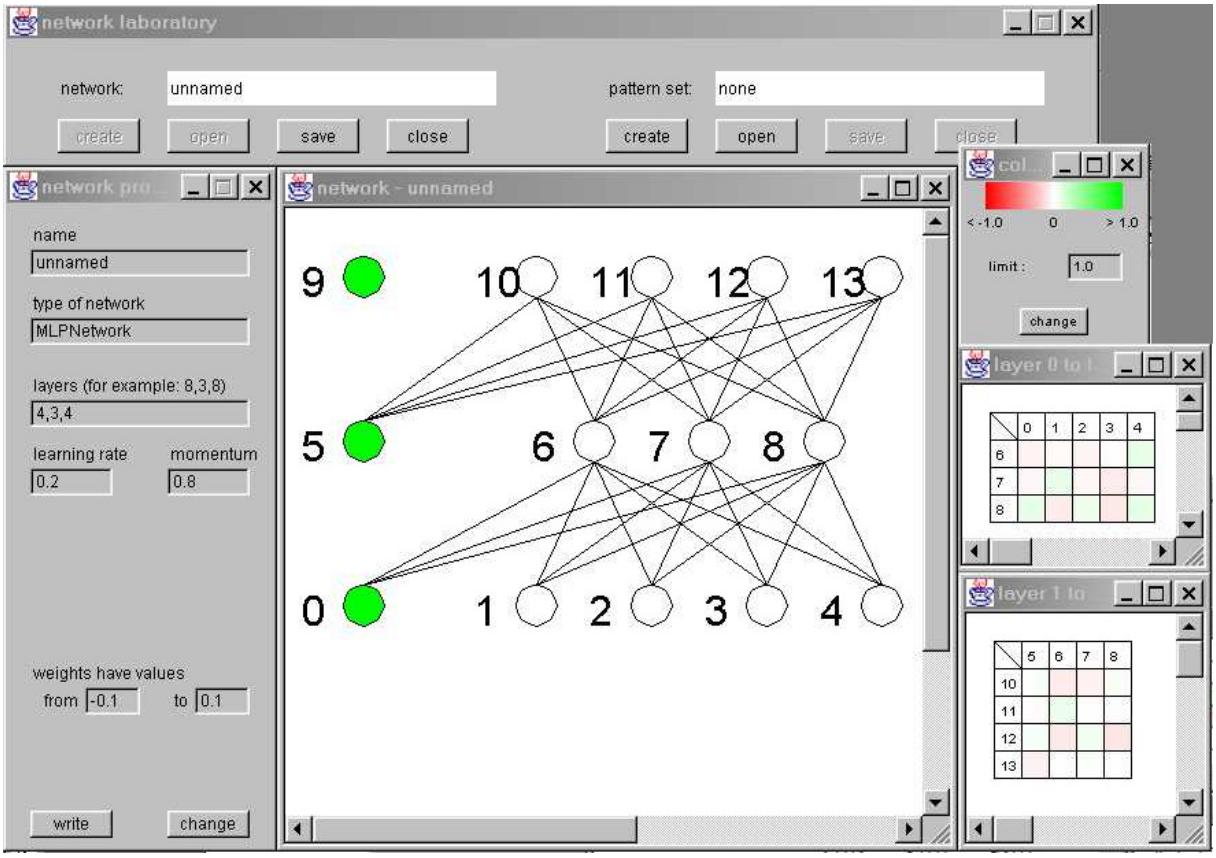


Abb. 3.4: Labor mit einem 4,3,4 multilayer perceptron

In Abb. 3.4 sieht man das Labor, nachdem ein multilayer perceptron erstellt worden ist. Wie man unschwer erkennen kann, sind jetzt sechs Fenster auf dem Bildschirm sichtbar. Das Hauptfenster oben zeigt auf der linken Seite den Namen des eben erstellten Netzwerks an. Im Fenster auf der linken Seite sieht man die Angaben für die Parameter dieses Netzes, mittels „change“ Button können sie geändert werden, jedoch nicht der Typ des Netzes oder die Anzahl Knoten in einer Schicht. Dazu müsste man das Netz mit dem „close“ Button im Hauptfenster schliessen und ein neues mit gewünschtem Typ und Grösse erstellen.
Der „write“ Button im Fenster auf der linken Seite dient dazu, die Gewichtsmatrix eines Netzwerks in ein Textfile zu schreiben.
Das Fenster in der Mitte der Abbildung stellt die einzelnen Knoten des Netzes und die bestehenden Verbindungen dar. Bei geschichteten Netzwerken sind die bias Knoten links angeordnet. Zuunterst befindet sich die Inputschicht, zuoberst die Outputschicht. Jede Schicht erhält automatisch einen bias, auch wenn dieser wie im Falle der Outputschicht eigentlich nicht benötigt wird. Die Farbe der Knoten spiegelt den Level der Aktivierung wider. Sattes Grün steht für eine Aktivierung der Höhe 1, Weiss für eine Aktivierung der Höhe 0. Um den genauen Wert abzulesen kann auf einen Knoten geklickt werden, woraufhin ein weiteres

Fenster erscheint. Es zeigt in der Titelleiste den Index des Knotens, in einem Label darunter die Aktivierung. In Abb. 3.5 handelt es sich um den bias Knoten der untersten Schicht, der mit 1 aktiviert ist.

Abb. 3.5: Aktivierung eines Knotens

Die zwei Fenster rechts zeigen die Gewichtsmatrizen zwischen den Schichten. Auf der vertikalen Achse sind die Indizes der Knoten zu denen eine Verbindung führt, auf der horizontalen die Indizes der Knoten, von denen die Verbindung ausgeht. Durch einen Klick auf einen Eintrag (farbiges Feld) in der Matrix kann in einem neuen Fenster der Wert des Gewichts abgelesen oder auf einen neuen Wert gesetzt werden. In der Titelleiste dieses Fensters sieht man, um welches Gewicht es sich handelt. In der Abb. 3.6 ist es das Gewicht zum Knoten 6 vom Knoten 1.

Abb. 3.6. Gewicht einer Verbindung

Da sich die Werte der Gewichte während des Lernvorganges stark verändern können, kann mit Hilfe der „ColorControl“- dem Fenster oben rechts in Abb. 3.4- die Skala für die Farbcodierung verändert werden. Es gilt jedoch immer, dass Grün für positive Verbindungen und Rot für negative steht.

3.2. Erstellen von Mustern

3.2.1 Neue Muster erstellen

Beim Erstellen von Mustern sollte darauf geachtet werden, dass diese die richtige Grösse für das Netz haben. Die Anzahl Einträge im Inputvektor muss gleich der Anzahl Inputknoten des Netzwerks sein. Für Supervisierte Netzwerke gilt zusätzlich, das die Anzahl Elemente im Targetvektor (gewünschter Output) mit der Anzahl Knoten in der Outputschicht des Netzwerks übereinstimmen muss. Für das Netz in Abb. 3.4 müsste also ein „pattern set“ wie folgt erstellt werden:

Abb. 3.7: Erstellung eines „pattern sets“

Nachdem die Angaben in Abb. 3.7 mit „ok“ bestätigt werden, erscheint das Fenster in Abb. 3.8

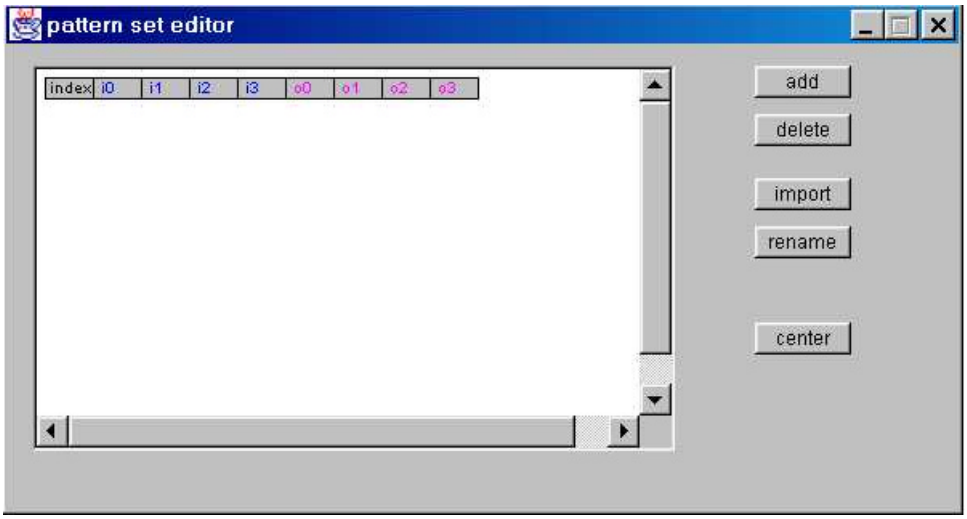


Abb. 3.8: der „pattern set editor“

Mit den Buttons „add“ und „delete“ können einzelne Muster hinzugefügt oder gelöscht werden. Die blauen Felder i0 bis i3 stehen für die Elemente des Inputvektors, die lila o0 bis o3 für die Elemente des Targetvektors.
Der Button „center“ verschiebt die Inputvektoren der einzelnen Muster so, dass ihr Mittelwert auf den Ursprung zu liegen kommt. Dies ist bei Oja und Sanger Netzwerken von Bedeutung, um die gefundenen Hauptkomponenten graphisch interpretieren zu können.

3.2.2. Einlesen von Mustern aus Textfiles

Mit dem „import“ Button kann ein Textfile eingelesen werden, das diese Form haben muss:
i) eine Zeile je Muster
ii) Vektorelemente durch Leerzeichen getrennt
iii) Inputvektor und Targetvektor durch Komma getrennt

Gültige Werte für Vektorelemente sind beliebige reelle Zahlen, jedoch kann der selbe Inputvektor nur einmal in einem „pattern set“ vorkommen.

3.3. der Lernvorgang

Nachdem ein Netz und ein „pattern set“ erstellt wurden, erscheint, falls die beiden zu einander passen, ein Kontrollfenster für den Lernvorgang. Je nach Typ des Netzwerks sieht dieses anders aus. Bei supervisierten Netzen können im Trainingsmodus Abbruchbedingungen festgelegt und schliesslich die Muster gelernt werden, im Testmodus einzelne Muster dem Netz präsentiert werden.
Bei kompetitiven Netzen (Hebb,Sanger,Oja,Kohonen) können im Trainingsmodus die Muster gelernt bzw. die Karte entfaltet werden, im Testmodus kann dem Netz ein Muster präsentiert werden und das Gewinner-Neuron wird aktiviert.
Bei Hopfield Netzen können im Trainingsmodus Muster gespeichert werden, im Testmodus kann ein einzelnes Muster präsentiert und aufdatiert werden.

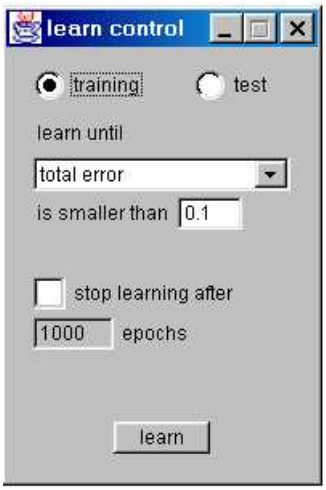


Abb. 3.9: Kontrollfenster für den Lernvorgang

4. Übersicht über die implementierten Netzwerktypen

im folgenden Abschnitt sollen die implementierten Netzwerktypen kurz mit Hilfe der vier „basics“ zusammengefasst werden.

4.1. Perceptron

1) binäre Schwellwert Neuronen mit Werten 0/1

2) Input und Outputschicht mit je einem bias Neuron, volle Verknüpfung zwischen den Schichten

3) $h_i = Ó\, w_{ij}\, o_j$

4) Perzeptron Lernregel:

falls effektiver Output gewünschter Output

$Äw = çî^ì$

Wobei:

$ç$ = Lernrate

$î^ì$ = Input des Musters $ì$

4.2. Adaline

1) lineare Schwellwert Neuronen

2) Input und Outputschicht mit je einem bias Neuron, volle Verknüpfung zwischen den Schichten

3) $h_i = Ó\, w_{ij}\, o_j$

4) verallgemeinerte ÄRegel:

$Äw_{ij}^{\ ì} = ç(œ_i^{\ ì} - O_i^{\ ì})î_j^{\ ì}$

Wobei:

$ç$ = Lernrate

$œ_i^{\ ì}$ = gewünschter Output für Knoten i

$O_i^{\ ì}$ = effektiver Output des Knotens i

$î_j^{\ ì}$ = Input von Knoten j

4.3. Multi layer perceptron

1) sigmoide Neuronen

2) mindestens zwei Schichten mit je einem bias Neuron, volle Verknüpfung zwischen den Schichten

3) $h_i = Ó\, w_{ij}\, o_j$

4) Backpropagation:

$Äw_{ij}^{m} = çä_i^{m} V_j^{m-1}$

Wobei:

$ç$ = Lernrate

$ä_i^{m}$ = Delta des Knotens *i* in Schicht *m*

V_j^{m-1} = Aktivierung des Knotens *j* in Schicht *m-1*

4.4. Hebb

1) lineare Neuronen

2) Input und Outputschicht mit je einem bias Neuron (wird nicht benutzt), volle Verknüpfung zwischen den Schichten

3) $h_i = Ó\, w_{ij}\, o_j$

4) Hebb'sche Lernregel:

$Äw_{ij} = çî_i î_j$

Wobei:

$ç$ = Lernrate

$î_i$ = Output des Knotens *i*

$î_j$ = Output des Knotens *j*

4.5. Oja

1) lineare Neuronen

2) Input und Outputschicht mit je einem bias Neuron (wird nicht benutzt), volle Verknüpfung zwischen den Schichten

3) $h_i = Ó\, w_{ij}\, o_j$

4) Oja's Lernregel

$Äw_{ij} = çî_i(î_j - Ó\, î_j w_{kj})$

Wobei:

$ç$ = Lernrate

$î_i$ = Output des Knotens i

$î_j$ = Output des Knotens j

$Ó\, î_j w_{kj}$ = Summe $î_j w_{kj}$ über alle Outputknoten k

4.6. Sanger

1) lineare Neuronen

2) Input und Outputschicht mit je einem bias Neuron(wird nicht benutzt), volle Verknüpfung zwischen den Schichten

3) $h_i = Ó\, w_{ij}\, o_j$

4) Sanger's Lernregel

$Äw_{ij} = çî_i(î_j - Ó\, î_j w_{kj})$

Wobei:

$ç$ = Lernrate

$î_i$ = Output des Knotens i

$î_j$ = Output des Knotens j

$Ó\, î_j w_{kj}$ = Summe $î_j w_{kj}$ über Outputknoten k, bis zum Knoten i

4.7. Kohonen

1) binäre Schwellwert Neuronen mit Werten 0/1, die eine fixe Position in der Karte haben

2) Input und Kohonenschicht mit je einem bias Neuron (wird nicht benutzt), volle Verknüpfung zwischen den Schichten

3) $h_i = Ó\, w_{ij}\, o_j$

4) Lernregel:

$Äw_{i*} = çË_{i*}(\underline{î} - w_{i*})$

Wobei:

$ç$ = Lernrate

$Ë_{i*}$ = Nachbarschaftsfunktion

$\underline{î}$ = Inputvektor

w_{i*} = Gewichtsvektor

4.8. Hopfield

1) binäre Schwellwert Neuronen mit Werten –1/1

2) eine Schicht, Gewichte mit 0 initialisiert

3) $h_i = Ó\, w_{ij}\, o_j$

4) Hebb'sche Lernregel:

$Äw_{ij} = çî_i î_j$

Wobei:

$ç$ = Lernrate

$î_i$ = Output des Knotens i

$î_j$ = Output des Knotens j

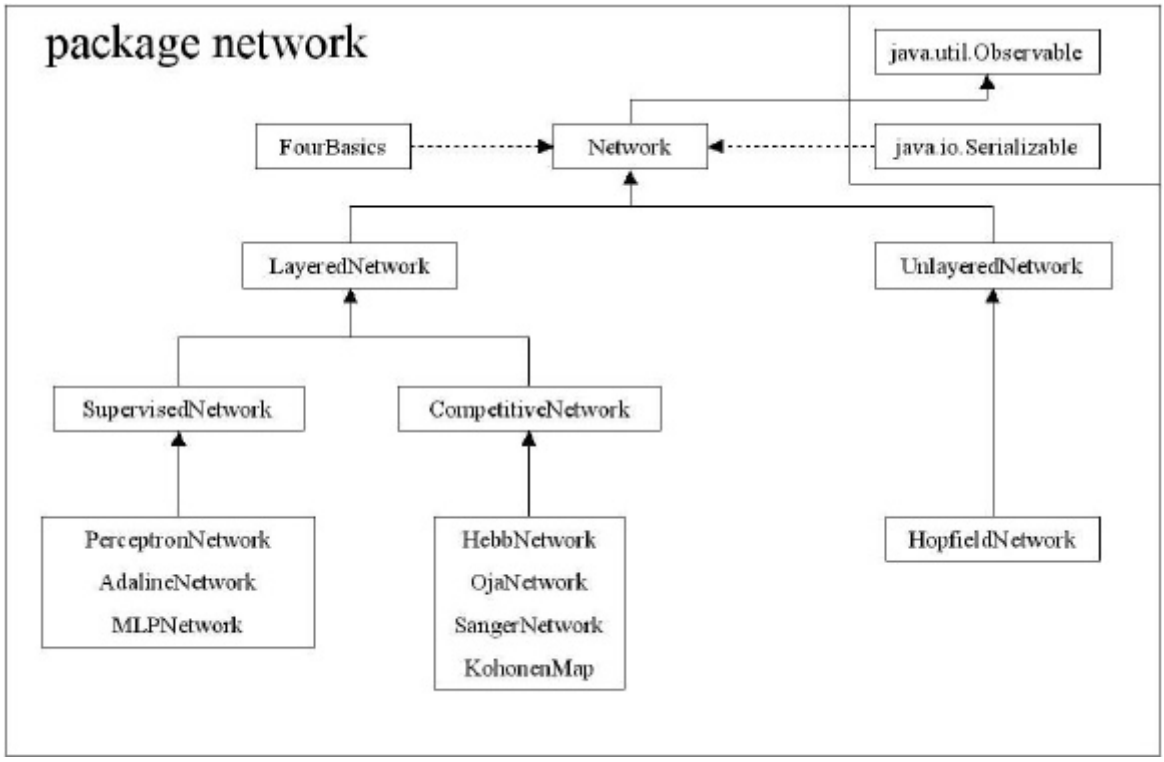
5. Verwenden der Klassen

Die Klassen, die benötigt werden, um ein funktionsfähiges Netzwerk zu erstellen befinden sich in den Packages network, network.node und network.util. Die Packages view und controller enthalten die Fensterklassen, mit welchen die Netze dargestellt und verändert werden können.

Es würde den Rahmen sprengen, hier auf einzelne Abschnitte im Code einzugehen. Eine Übersicht über die für die Netzwerke relevanten Packages findet sich im Anhang, der komplette Code und die Dokumentation auf der Diskette.

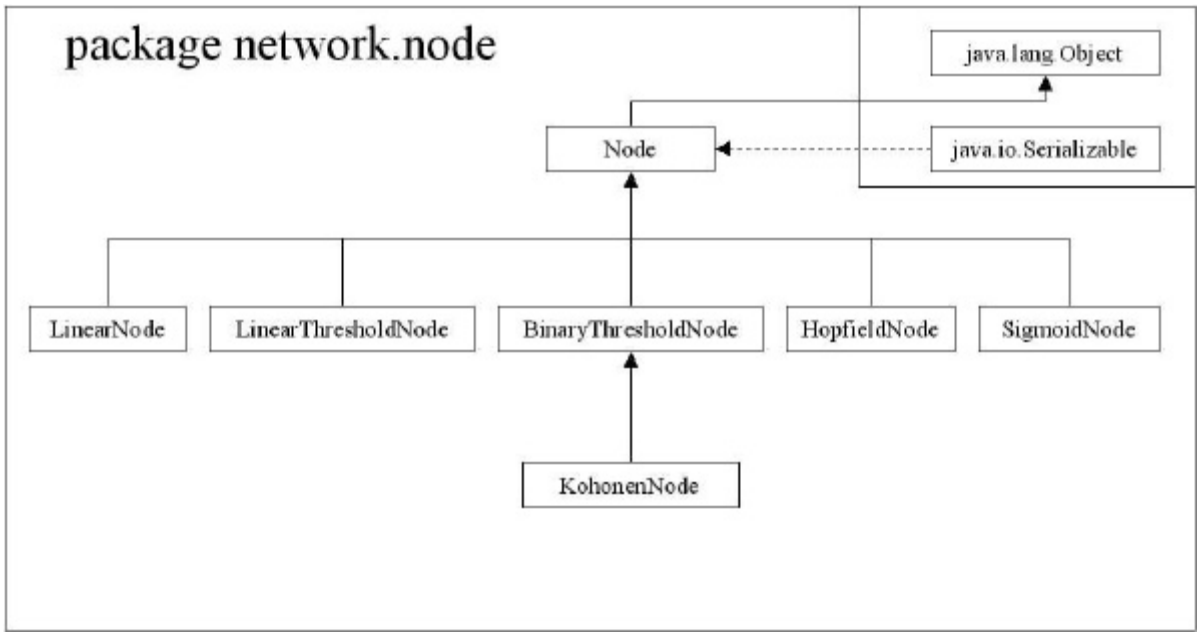
Anhang A – Für die Netzwerke relevante Packages

Package network



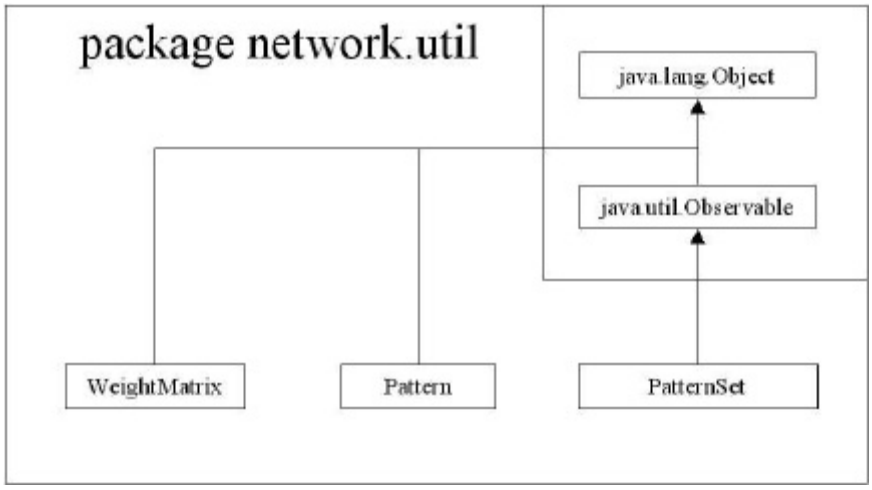
Klassen der package network

Package network.node



Klassen der Package network.node

Package network.util



Klassen der Package network.util