Prüfer:	Prof. Dr. Andreas Reuter
Betreuer:	Dr. Andreas Zell

begonnen am:	01. Dezember 1991
beendet am:	23. April 1992
CR-Klassifikation:	D.2.2, I.2.6, I.3.3

Graphische Visualisierungs–Hilfsmittel für einen Simulator neuronaler Netze

Günter W. Mamier

Diplomarbeit Nr. 880

Fakultät Informatik
Institut für Parallele und
Verteilte Höchstleistungsrechner
Universität Stuttgart
Breitwiesenstraße 20–22
D 7000 Stuttgart 80

Anmerkung

Dieses Dokument stimmt nicht exakt mit der von Günter Mamier erstellten Diplomarbeit überein. Es enhält zusätzlich Beschreibungen von Funktionen, die erst nach Abgabe der Diplomarbeit zu SNNS hinzugefügt worden sind.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

Der Forschungsbereich der neuronalen Netze, auch konnektionistische Systeme genannt, obwohl schon vor ca. 30 Jahren bekannt, gewann erst im Laufe der letzten Jahre immer mehr an Bedeutung und wird heute als ein wichtiges Gebiet der Informatik betrachtet. Hier werden in den nächsten Jahren große Fortschritte erwartet.

Schon sehr früh nach der Entwicklung der ersten Computer durch Pioniere wie Conrad Zuse kam die Idee auf, nicht nur arithmetische Aufgaben zu lösen, sondern auch die Vorgänge in einem natürlichen Gehirn elektronisch nachzuahmen (McCulloch und Pitts 1943).

In einem biologischen Nervenzentrum sind viele Millionen Nervenzellen miteinander verbunden (beim Menschen ca. 10^{11}). Diese Nervenzellen stellen einfache informationsverarbeitende Systeme dar. An den sogenannten Dendriten (Eingabestellen oder *Sites*) erreichen analoge Signale die Nervenzelle. Nach einer zeitlichen und/oder räumlichen Koppelung und Verarbeitung dieser Eingangssignale mit einer Aktivierungsfunktion wird die Zelle bei Überschreiten eines Schwellwertes „aktiviert". Aktiviert bedeutet dabei, daß sie ihrerseits ein analoges Signal über ihr Neurit mit einer Ausgabefunktion an nachfolgende Zellen weiterleitet. Die Verbindungen (*Links*) zwischen den Nervenzellen sind dabei nicht statisch, sondern verändern sich während des Lernprozesses. Es können sowohl neue Bahnen gebildet werden, als auch alte, nicht mehr gebrauchte, veröden. Dies kommt allerdings eher selten vor. Der Lernvorgang basiert überwiegend auf einer Änderung der Effektivität der synaptischen Kopplung. Da diese Vorgänge aufgrund ihrer sehr komplexen Dynamik bis heute nicht vollständig verstanden sind, ist die Erforschung konnektionistischer Systeme eine experimentelle Wissenschaft geblieben. Bis vor wenigen Jahren war die Rechnerkapazität, um diese Experimente durchzuführen, nicht vorhanden, und auch theoretische Erkenntnisse über die benötigten Lernvorgänge waren nicht vorhanden. Deshalb wurden die ersten frühen Ansätze zur Erforschung künstlicher neuronaler Netze nicht weiterverfolgt.

Der historische Ansatz, konnektionistische Systeme zu untersuchen um Erkenntnisse über die Funktion des menschlichen Gehirns zu gewinnen, war wohl die treibende Kraft solche Systeme zu erforschen. Gleichwohl ist eine völlig andere Motivation heute in den Vordergrund getreten. Eine Reihe aktueller Fragestellungen wie die Erkennung gesprochener Sprache oder die Erkennung von Mustern sind algorithmisch nur sehr schwer lösbar. Die Beobachtung, daß diese Aufgaben bereits von sehr jungen Menschen problemlos bewältigt werden, führte zu dem Versuch, sich bei der Lösung dieser Aufgaben vom Zwang fester Algorithmen zu trennen und verstärkt neuronale Netze zu verwenden.

Die erhofften Vorteile dieser neuronalen Netze liegen neben der effektiven Realisierung von paralleler Informationsverarbeitung vor allem in der Trainierbarkeit und Anpassungsfähigkeit der neuronalen Netze, wie auch in der Fehlertoleranz. Normale Algorithmen verlangen eine strikte, explizite Programmierreihenfolge aller notwendigen Schritte. Sie sind nicht in der Lage, auf neue Situationen zu reagieren oder sich ihnen anzupassen. Neuronale Netze hingegen werden nicht explizit programmiert sondern anhand von Beispielen trainiert. Dies bringt den Vorteil, daß sie auch neue Situationen in begrenztem Umfang meistern, und dafür Ausgaben liefern können. Da das Wissen des Netzes nicht in einzelnen Neuronen explizit, sondern über die gesamten Verbindungen verteilt implizit gespeichert wird, besitzen neuronale Netze auch ein hohes Maß an Redundanz. Diese Redundanz kann zu einer erhöhten Fehlertoleranz genutzt werden. Der Ausfall eines Neurons, oder einer Gruppe von Neuronen (im Gehirn oder auf einem VLSI-Chip) führt deswegen nicht automatisch zu einem Gesamtausfall des Systems. Es ist im Gegenteil die Möglichkeit gegeben, nach einer neuerlichen Trainingsphase die volle Funktionalität wieder zu erreichen. Diese Fähigkeit läßt sich für noch höhere Integrationsdichten beim Chipdesign hervorragend nutzen, bei dem höhere Toleranzgrenzen zu einer Reduktion von Produktionskosten führen können.

Neben der jetzt zur Verfügung stehenden Rechnerkapazität ist aber wohl auch eine gewisse Enttäuschung gegenüber den Fortschritten der „klassischen künstlichen Intelligenz" für die Rennaisance der neuronalen Netze verantwortlich.

Da Hardware-Realisierungen auf VLSI-Chips (noch) nicht sehr verbreitet sind und außerdem auch für Forschungsaufgaben oft zu inflexibel sind, findet ein großer Teil der heutigen neuronale Netze Forschung mit Simulatoren auf normalen Ein- oder Mehrprozessor-Computern statt. Diese Simulatoren sind Softwarepakete, die die inherent parallelen Prozesse des Netzwerks entweder sequentialisieren und auf einem Prozessor abarbeiten, oder auf mehrere Prozessoren[1] verteilen. Beispiele solcher Simulatoren sind der Rochester Connectionist Simulator, die PDP (Parallel Distributed Processing) Simulatoren, Neural Works Professional II Plus, der Pygmalion Simulator, Netsim und nicht zuletzt SNNS (Stuttgarter Neuronale Netze Simulator).

[1]Die Zahl der Prozessoren ist in der Regel bedeutend kleiner als die Zahl der simulierten Neuronen

Kapitel 2

Einführung in SNNS

SNNS – der Stuttgarter Neuronale Netze Simulator – ist ein leistungsfähiger Simulator, der seit 1989 an der Universität Stuttgart im Laufe mehrerer studentischer Projekte entwickelt wird. Er zeichnet sich durch hohe Flexibilität, einfache und effiziente Bedienung und leichte Portierbarkeit aus. Zweck des Projektes ist die Schaffung einer effizienten Simulatorumgebung zur Erforschung von verschiedenen konnektionistischen Systemen und deren Leistungsfähigkeit. Dazu steht SNNS auch anderen Forschungseinrichtungen über ftp kostenlos zur Verfügung.

Die Bausteine von SNNS gliedern sich in drei Module

- Simulatorkern
- Netzwerkcompiler NESSUS
- Oberfläche XGUI

Abbildung 2.1 gibt dazu einen schematischen Überblick.

2.1 Kern

Im Simulatorkern findet die Verwaltung und Veränderung der neuronalen Netze statt. Trainings– und Simulationsfunktionen sind hier implementiert. Auch die Speicherverwaltung wird vom Kern übernommen. Dieser Kern ist sehr portabel und bedarf meist nur einer neuen Compilierung um auf einem neuen Rechnertyp verwendet werden zu können. Der Simulatorkern ist auf fast allen Unix Workstations mit Ansi C lauffähig. Er wurde auf Unix-PCs und Workstations von Sun, DEC, HP und IBM entwickelt und getestet. Auch eine parallele Version des Kerns für eine MasPar MP-1216 steht seit kurzem zur Verfügung.

Im Moment sind im Kern die folgenden Lernverfahren implementiert: Backpropagation, Backpropagation mit Momentum, Counterpropagation, Quickpropagation und Backpercolation. Diese Liste läßt sich über eine Makro-Bibliothek vom Benutzer leicht erweitern. Ebenso können neue Site–, Ausgabe– und Aktivierungsfunktionen hinzugefügt werden.

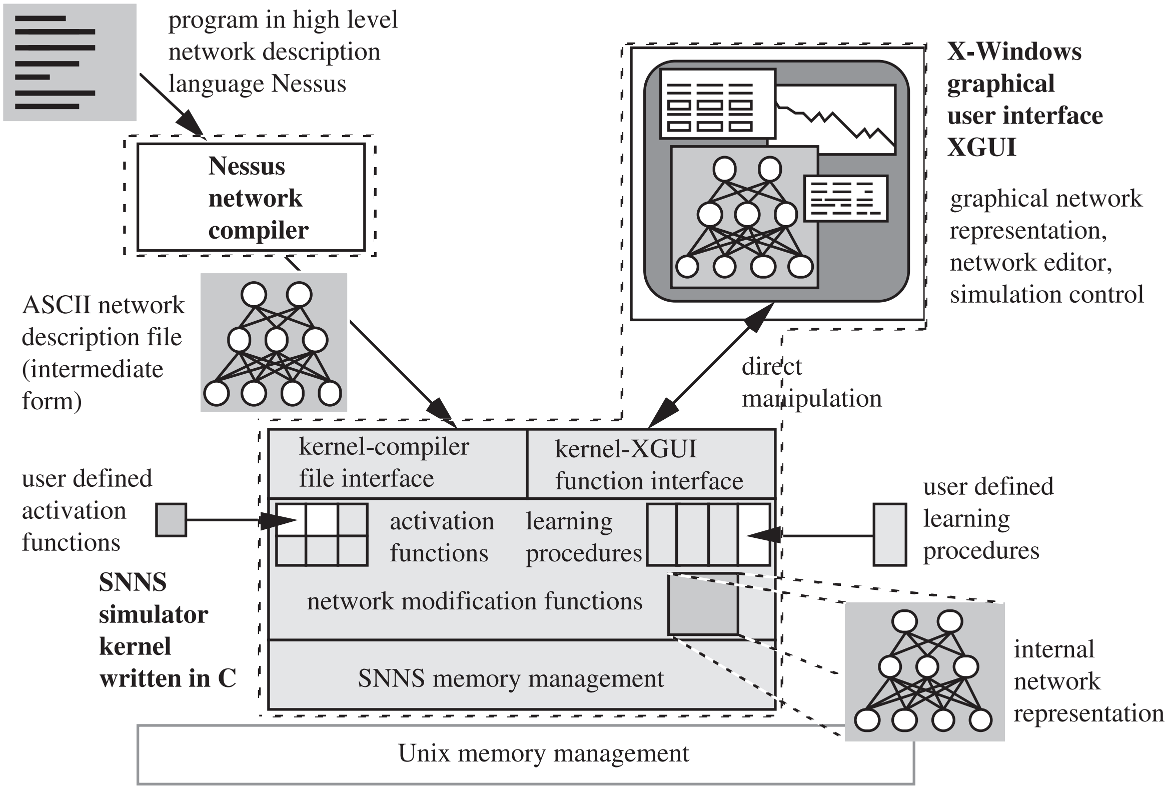

Abbildung 2.1: Die Bestandteile von SNNS

2.2 NESSUS

NESSUS (Netzwerk Spezifikations Sprache der Universität Stuttgart) erlaubt den Entwurf von mittelgroßen Netzen über eine einfache Netzwerkbeschreibung. Die vom NESSUS-Compiler erzeugte Datei kann vom Kern gelesen und weiterverarbeitet werden. Zur Netzwerkbeschreibung stehen Typdefinitionen für Zellen und Sites, Verbindungsstrukturen sowie ein Teilnetzkonzept zur Verfügung. Letzteres erlaubt den einfachen Zusammenbau größerer Netze mittels mehrerer NESSUS Programme.

2.3 XGUI

Die graphische Benutzeroberfläche XGUI (X-Graphical User Interface) ist der Hauptzugriff des Benutzers auf SNNS. Sie erlaubt unter anderem:

- Das Darstellen des Netzes. Entweder mit allen Zellen und Verbindungen, oder nur von ausgewählten Teilen.

- Das interaktive Erstellen von Netzen. Entweder Zelle für Zelle, und Verbindung für Verbindung, oder bei größeren Netzwerken (oder Netzwerkteilen) mittels der Funktion BIGNET.
- Die Modifikation eines mit NESSUS erstellten Netzes.
- Das Durchführen des Trainings.
- Das Testen von Mustern.

Der Bildschirm wird dabei in mehrere, vom Benutzer rufbare und schließbare Fenster unterteilt, die eine möglichst optimale Sicht des Netzwerkes und Bedienung des Simulators erlauben

Ein typischer Bildschirmaufbau ist in Abbildung 2.2 dargestellt.

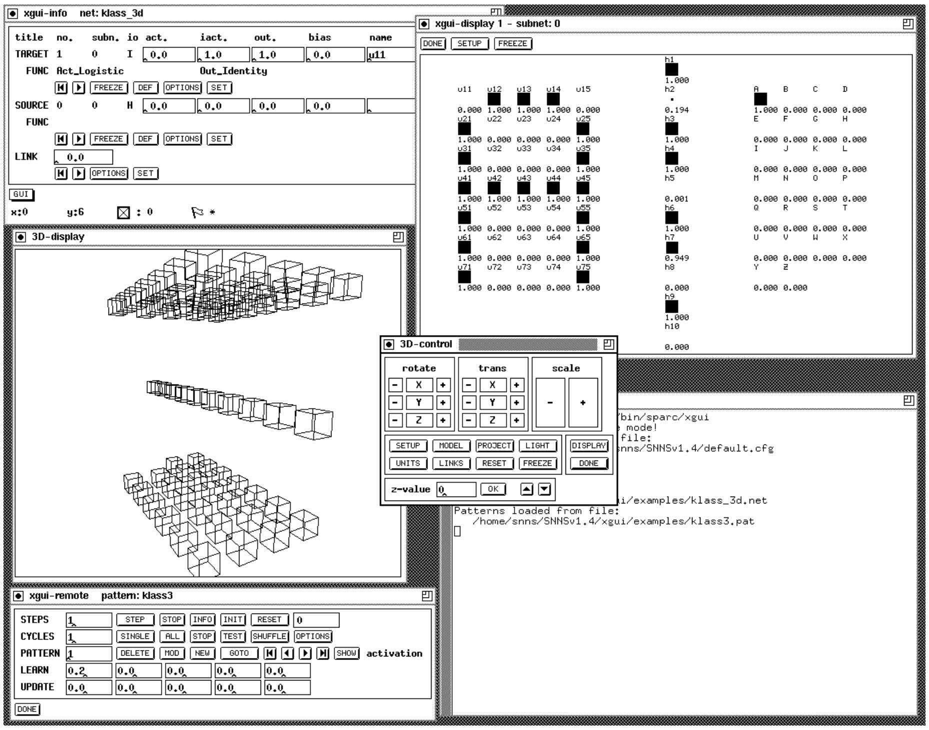

Abbildung 2.2: Ein typischer Bildschirmaufbau

Neben einem 2–dimensionalen Display–Fenster (oben rechts) steht auch eine 3D–Darstellung des Netzes zur Verfügung (mitte links). Diese 3D–Darstellung ist gut geeignet um größere, stark verbundene Netze übersichtlich darzustellen. Bei einer großen Zahl von Verbindungen würden in einer 2D–Repräsentation viele Units und ihre Werte von anderen Verbindungen überdeckt. In der Bildmitte befindet sich das Control–Panel für die 3D–Darstellung. Mit ihm wird die 3D–Darstellung gesteuert.

Im 2D–Fenster findet die interaktive Generierung und Modifizierung des Netzwerks statt. Entweder durch direkte Eingaben von Buchstabensequenzen auf der Tastatur solange der Mauscursor sich in diesem Fenster befindet, oder durch Aufrufen von Menüs durch Klicken des linken Mausbuttons solange die Control–Taste gedrückt ist, können Befehle zur Netzwerkänderung eingegeben werden. Ferner besteht die Möglichkeit mehrere 2D–Displays zu öffnen und in jedem nur einen Teilaspekt des Netzes sichtbar zu halten. Eine Erstellung von 3D–Netzen erfolgt üblicherweise im 2D–Display mit einer anschließenden Übersetzung der Koordinaten.

Unten links befindet sich das Remote–Panel zur Steuerung des Trainings und der Definition der Muster. Vorhandene Muster können angezeigt und geprüft werden. Auch Aktivierungsfunktionen können hier definiert und initialisiert werden.

In der oberen linken Ecke ist das Info–Panel plaziert. Hier werden einerseits alle wichtigen Informationen über das Netz ausgegeben, andererseits kann der Benutzer aber auch direkt Werte einzelner Zellen oder Verbindungen setzen.

Als letztes Fenster befindet sich das normale Shell–Window, von dem aus SNNS aufgerufen wurde, in der rechten unteren Ecke. Hier werden Statusmeldungen und Informationen über das Netz ausgegeben.

Die Oberfläche ist unter dem X–Windows Fenstersystem entwickelt und implementiert worden. Sie verwendet den Athena Widget Kit und die Xt Intrinsic Funktionen. Da XGUI aber auch unter OpenWindows lauffähig ist, ist auch eine hohe Portabilität der graphischen Oberfläche gewährleistet.

2.4 Erweiterungsbedarf

Um die Vorgänge innerhalb des Netzes während und nach der Trainingsphase besser durchschauen zu können wurde die Entwicklung neuer Visualisierungswerkzeuge notwendig. SNNS war zwar ein effizienter Simulator für neuronale Netze, besaß aber außer dem Testen von Eingabemustern keinerlei Möglichkeit zur Analyse des Netzes. Der Bedarf, die internen Vorgänge analysieren zu können, bezog sich sowohl auf die einzelnen Komponenten des Netzes, als auch auf das Netz als Ganzes.

Es mußten deshalb zuerst Werkzeuge erstellt werden, um die internen Vorgänge graphisch darstellen, und Rückschlüsse auf das Verhalten des Netzes machen zu können.

Danach mußte die Entwicklung eines Analysemechanismus in Angriff genommen werden. Neben einer graphischen Ausgabe aller beteiligten Funktionen eines Neurons war vor allem die übersichtliche Darstellung der Verbindungsgewichte von Interesse, da diese ja praktisch das „Gedächtnis“ des neuronalen Netzes darstellen.

Auch die Frage, welche Eingabe, minimale Eingabe, oder maximale Eingabe ein trainiertes Netz zu einer gewünschten Ausgabe veranlassen war von großem Interesse und sollte durch die Entwicklung eines entsprechenden Tools zur Analyse des Netzwerks geklärt werden können.

Kapitel 3

Ziele dieser Arbeit

Ziel dieser Diplomarbeit war es, den oben erwähnten Erweiterungsbedarf zu befriedigen und die dafür notwendigen Funktionen zur Verfügung zu stellen. Insbesondere sollten graphische Tools entwickelt werden, die die interne Repräsentation des Netzes offenlegen und einer Analyse zugänglich machen. Im einzelnen war die Entwicklung folgender Hilfsmittel geplant:

- Die graphische Darstellung von Site–, Aktivierungs– und Ausgabefunktionen der Neuronen.
- Die Visualisierung der Gewichte des Netzes mit einem geeigneten Verfahren. Wenn möglich sollten Hinton Diagramme und WV–Diagramme Verwendung finden.
- Ein Analyseverfahren zur Ermittlung der Netzwerkstruktur und ihrer Auswirkung auf das Verhalten des Netzes, insbesondere unter Verwendung der sogenannten Inversionsmethode.

Alle Funktionen sollten in SNNS integriert werden und sich äußerlich an die bestehende Oberfläche anpassen.

Bestehende Low–Level–Routinen des Kerns und der Oberfläche wurden konsequent verwendet, um rechnerspezifische Funktionen zu vermeiden und das hohe Maß an Portabilität erhalten zu können. Das Modul zur Netzinversion (Feststellen der notwendigen Eingabe für eine bestimmte Ausgabe) ist zwar eine Funktion des Kerns, und ist auch dort implementiert, für den Benutzer aber nur über XGUI ansprechbar.

Da der innere Aufbau von SNNS nicht verändert werden sollte, war es nicht möglich, eine graphische Ausgabe der Site–Funktionen der Neuronen zu erstellen. Die anderen Visualisierungsziele konnten aber voll erreicht werden.

Kapitel 4

Neu entwickelte Funktionalität

Dieses Kapitel beschreibt die neu entwickelte Funktionalität. Dabei wird neben den theoretischen Grundlagen vor allem auf die Benutzersicht eingegangen. Die programmtechnische Beschreibung folgt in Kapitel 5.

4.1 Darstellung der Unit–Funktionen

Erstes Ziel dieser Diplomarbeit war die graphische Darstellung aller relevanten Funktionen einer Zelle. Diese Funktionsgraphen sollten einen raschen Eindruck von dem zu erwartenden Verhalten der Unit liefern, dabei aber relativ klein und übersichtlich gehalten werden. Randbedingung war, daß die Funktionen nur dargestellt werden, nicht aber in die Konsistenz des Simulators eingegriffen wird.

4.1.1 Site–Funktionen

Bestimmte Typen von Nervenzellen übernehmen die Signale ihrer Vorgängerzellen nicht am Zellkörper direkt, sondern bündeln sie vorher an bestimmten Eingabestellen. Diese Eingabestellen erlauben eine Vorverarbeitung der einlaufenden Information bevor sie an die eigentliche Zelle weitergeleitet wird.

Solche Eingabestellen, und die in ihnen stattfindende Vorverarbeitung, können in SNNS durch sogenannte Sites und Site–Funktionen an den Units simuliert werden.

Bedingt durch die innere Struktur des Simulator-Kerns war die geplante Darstellung der Site–Funktionen nicht möglich. Die Deklaration dieser Funktionen in der Funktionstabelle ist derart, daß sie beim Aufruf eine spezifische Unit als Parameter benötigen. Die Funktionen verarbeiten dann alle einlaufenden Verbindungen entsprechend ihrem mathematischen Modell.

Zur graphischen Darstellung wäre es erforderlich gewesen, der Funktion als Parameter einen Pointer auf eine Unit zu übergeben, deren Eingangsgewichte geändert würden. Ein Ändern von solchen Werten eines Pointers im Arbeitsbereich von SNNS würde aber der Forderung nach Konsistenzerhalt widersprechen, so daß eine Implementierung nicht möglich war.

4.1.2 Aktivierungsfunktionen

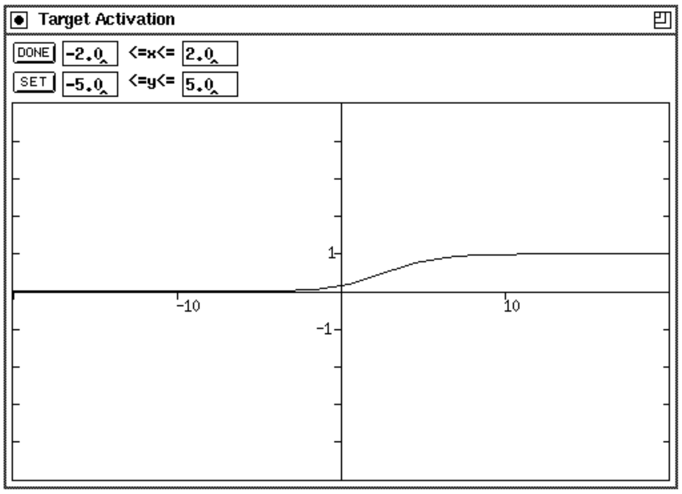

Abbildung 4.1: Eine logistische Aktivierungsfunktion

Nervenzellen ohne die in Abschnitt 4.1.1 erwähnten Eingabestellen nehmen die Informationen direkt am Zellkörper ab. Auch hier findet eine Verarbeitung und Koppelung der einlaufenden Signale statt, die darüber entscheidet, wie stark die Zelle aktiviert wird. Diese Verarbeitung erfolgt in SNNS mittels Aktivierungsfunktionen.

Abbildung 4.1 zeigt ein Beispiel einer Aktivierungsfunktion. Im Window Header wird angezeigt, ob es sich um die Aktivierungsfunktion der Source oder der Target Unit handelt. Da Eingabe–Units keine Vorgänger besitzen, kann der zur Darstellung notwendige Algorithmus nicht eingehalten werden. Daher kann von diesen Units keine graphische Darstellung der Aktivierungsfunktion erfolgen, obwohl auch für sie eine Definiton dieser Funktion möglich ist. Zur Darstellung einer Aktivierungsfunktion muß daher eine Hidden oder eine Output Unit als Source bzw. Target Unit gewählt werden.

Da dem Benutzer von SNNS die Möglichkeit gegeben wurde, jederzeit beliebige neue Funktionen zu definieren, ist ein optimaler x- und y- Bereich für die Darstellung nicht vorhersagbar. Es wurde deshalb die Möglichkeit geschaffen, den Bildausschnitt beliebig zu wählen. Dafür stehen im oberen Bereich des Fensters vier Dialogboxen zur Verfügung.

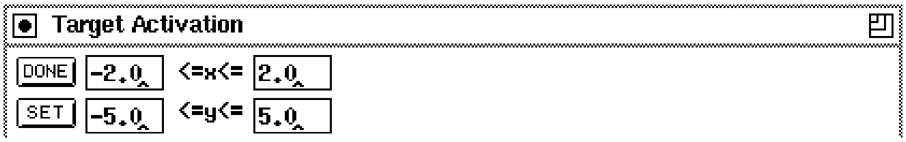

Abbildung 4.2: Die Dialogboxen in den Displays der Unitfunktionen

Die Voreinstellung wurde für die x-Richtung als das Intervall $[-2, 2]$, für die y-Richtung als das Intervall $[-5, 5]$ vorgenommen. Möchte der Benutzer andere Grenzen verwenden, muß er die gewünschten Werte lediglich in den entsprechenden Dialogboxen eintragen. Ein Anklicken des **SET** Buttons aktualisiert dann das Display mit den neuen Grenzen. Abbildung 4.3 zeigt dieselbe Funktion in neuen Grenzen. Die Default–Größe des Fensters ist relativ klein, da die Darstellung in erster Linie nur der Orientierung des Benutzers dienen soll. Wird eine exaktere Darstellung notwendig, kann das Fenster mit dem rechten oberen Window–Button zu jeder beliebigen Größe aufgezogen werden. In Verbindung mit den variablen Grenzen kann so jede beliebige Vergrößerung erreicht werden. Die Funktion wird abgeschlossen durch ein Anklicken des **DONE** Buttons.

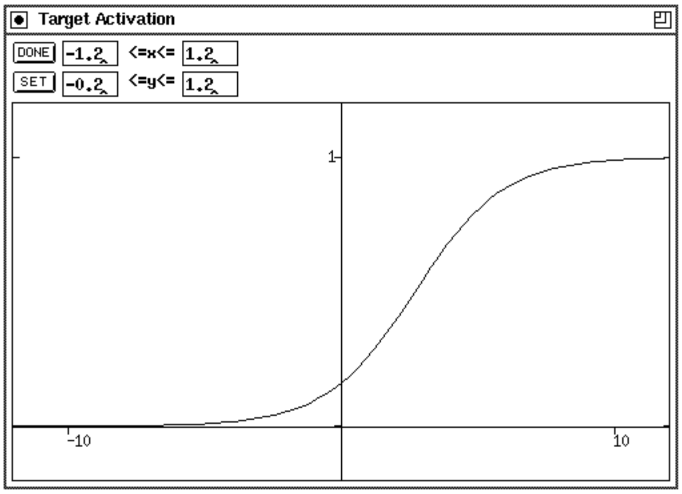

Abbildung 4.3: Die logistische Aktivierungsfunktion in neuen Grenzen

Auf eine Besonderheit der Beschriftung der x-Achse ist in diesem Zusammenhang hinzuweisen. Aufgrund der internen Struktur des SNNS müssen zur Darstellung der Funktion alle einlaufenden Verbindungen simultan geändert werden, um eine sinnvolle Darstellungssemantik sicherzustellen. Da eine gleichmäßige Anzahl von Verbindungen nicht gewährleistet ist, wird zur Beschriftung der x-Achse die Netzeingabe der Zelle verwendet. Die Netzeingabe ist die Summe aller einlaufenden Aktivierungen. Als Variablen in den Dialogfenstern für den x–Bereich werden hingegen immer die Grenzen für die Einzelaktivierungen angegeben.

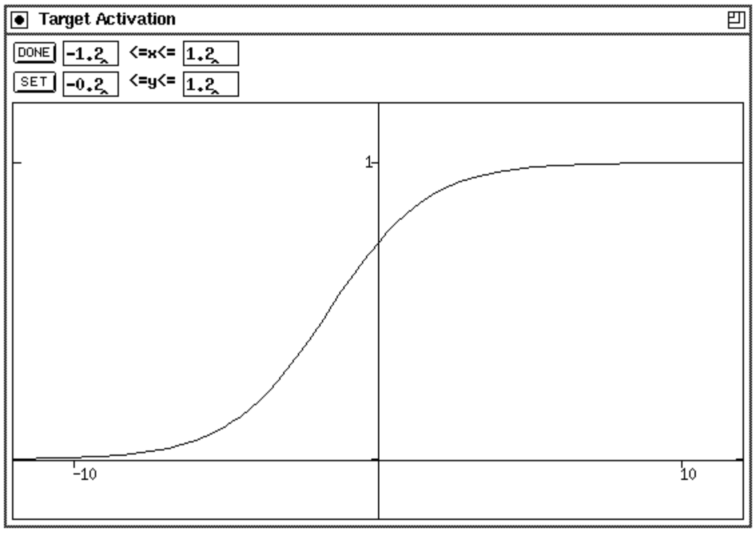

Abbildung 4.4: Die logistische Aktivierungsfunktion mit verändertem Bias

Der Benutzer sollte sich ebenfalls bewußt sein, daß der Schwellwert (Bias) der Unit bei vielen Aktivierungsfunktionen verwendet wird und deshalb auch einen Einfluß auf das Aussehen des Funktionsgraphen hat. Durch Setzen des Bias im Info–Panel und anschließendem Neuzeichnen der Funktion (Anklicken des **SET** Buttons im Funktions–Display) kann dieser Einfluß sichtbar gemacht werden. Abbildung 4.4 zeigt die Funktion aus Abbildung 4.3 mit verändertem Bias.

4.1.3 Ausgabefunktionen

Aktivierte Nervenzellen geben ihre Aktivierung nicht unbedingt in derselben Form an ihre Nachfolger weiter. Sehr oft ist die Weitergabe in ihrem Betrag nach oben und unten beschränkt. Diese Nachverarbeitung der Zellaktivierung erfolgt in SNNS mit Hilfe der Ausgabefunktionen.

Abbildung 4.5 zeigt ein Beispiel einer Ausgabefunktion. Der Name der Funktion ist immer im Info–Panel angezeigt. Hier handelt es sich um die Clip_11–Funktion, bei der Aktivierungswerte zwischen -1 und 1 identisch weitergegeben werden, während größere und kleinere Werte der Zellaktivierung die Weitergabeintensität nicht über 1 steigen, bzw. nicht unter -1 sinken lassen. Das Display–Fenster ist in seinem Aufbau und seiner Handhabung mit dem Fenster der Aktivierungsfunktion identisch.

Um einen raschen Überblick über die zur Verfügung stehenden Ausgabefunktionen zu erhalten, empfiehlt sich folgendes Verfahren. Nach dem Öffnen des Funktions–Display–Fensters wählt man die Option „change output function“ des Popupmenüs unter dem **OPTION** Button. Ein Anklicken der Funktionsnamen in der dann erscheinenden Liste führt zu ihrer sofortigen Darstellung im Display. Somit kann man durch alle zur Verfügung stehenden Funktionen blättern. Diese Möglichkeit besteht natürlich auch für die Aktivierungsfunktionen.

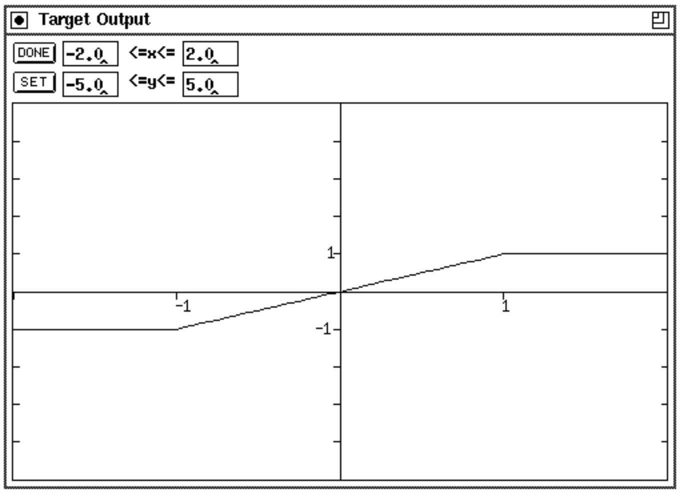

Abbildung 4.5: Die Clip_11–Ausgabefunktion

4.2 Darstellung von Verbindungsgewichten

4.2.1 Grundlagen

Wie bereits Eingangs erwähnt (siehe Kap. 1) speichern neuronale Netze ihre Information nicht explizit in einer oder mehreren Units, sondern vielmehr implizit, auf das gesamte Netz verteilt. Das „Wissen“ des neuronalen Netzes ist also in irgendeiner Form in den Verbindungsgewichten zwischen den Neuronen codiert. Es ist deshalb für jeden Anwender neuronaler Netze von größtem Interesse, wie sich die Verbindungen vor, während und nach dem Training verändern.

Das bloße Notieren der Gewichte neben den gezeichneten Verbindungen, wie es bisher in der XGUI Oberfläche ausschließlich möglich war, ist nicht geeignet um größere Untersuchungen durchzuführen, da keinerlei Übersicht über das Gesamtverhalten möglich ist. Auch eine Variation der Strichstärke der Verbindungen analog zu ihrem Gewicht ist nicht praktikabel. Zu schnell würden größere Verbindungsgewichte große Teile des Netzes verdecken und zu einem undurchdringlichen Knäuel verwachsen. Es muß daher ein weiteres, auf die Darstellung von Verbindungsgewichten spezialisiertes Display zur Verfügung gestellt werden.

Die darzustellenden Gewichte stellen eine Untermenge der reelen Zahlen dar. Es wird deswegen ein Mittel gesucht, reelle Zahlen anschaulich zu machen. Dabei gelten zwei Einschränkungen.

- Es müssen viele Zahlen gleichzeitig darstellbar sein.
- Die Darstellung muß auf einem Computerbildschirm möglich sein, d.h. in zwei Dimensionen stattfinden. (Zu 3D–Darstellungen siehe unten)

Daraus folgen im Prinzip drei Möglichkeiten zur Darstellung:

a) Länge, Größe, Dicke o.ä. eines geometrischen Objekts (Quadrat, Kreis, etc.).

b) Grauwert oder Farbe eines Pixels.

c) Die Position in einem Koordinatensystem.

Beispiele dieser Methoden für ein negatives r1 und ein positives r2 finden sich in Abbildung 4.6.

Die Methoden a) und b) sind von Hinton und Pinz [Hinton90, PinBis90] vorgeschlagen worden. Dazu wird das Display als quadratisches Raster organisiert. Entlang der Längs– und der Querachse werden nacheinander alle Units aufgeführt. Besitzen zwei Units eine gemeinsame Verbindung, so wird am Schnittpunkt ihrer Rasterlinien ein Quadrat gezeichnet. Diese beiden Methoden fanden in dieser Diplomarbeit Verwendung.

Da jetzt die Position im Koordinatensystem bereits die Verbindung zweier Units repräsentiert, scheidet Alternative c) hier aus. Sie könnte aber z.B. in einem 3D–Koordinatensystem

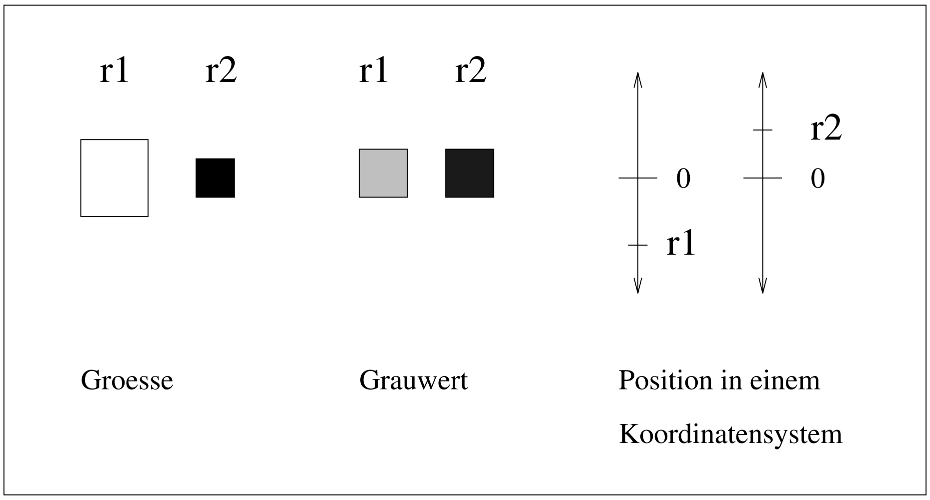

Abbildung 4.6: Darstellungsmöglichkeiten für zwei reele Zahlen r1 und r2.

als Höhe codiert werden und würde in der Darstellung eine Art Landschaft der Gewichte ergeben. Erreichbar wäre dies mit einer 3D–Simulation. Dieses Verfahren wird auch als *WV–Fläche* bezeichnet.

Die im Rahmen dieser Diplomarbeit implementierte Version der Visualisierungsmethoden a) und b) beinhaltet darüber hinaus noch einige Optimierungen. So wird automatisch eine optimale Fenstergröße gewählt. Ist das darzustellende Netz für eine vertretbare Fenstergröße zu umfangreich, wird von einem einfachen Display auf einen sogenannten Viewport umgeschaltet. Dieser Viewport erlaubt das Abscannen eines großen, virtuellen Fensters mittels Scrollbars. Bei sehr kleinen Netzen wird die Normgröße der Quadrate entsprechend erhöht. Der Benutzer kann auch über die rechte obere Ecke, wie bei X–Window Fenstern üblich, eine beliebige Fenstergröße einstellen. Eine eventuell notwendige Display/Viewport bzw. Viewport/Display Umschaltung wird dabei automatisch vorgenommen. Neben dem Einzeichnen von Raster-Hilfslinien, deren Gitterabstand vom Benutzer geändert werden kann, ist die automatische Erkennung von Schwarz–Weiß–, beziehungsweise Farbbildschirmen verwirklicht. Zur besseren Übersichtlichkeit werden die entlang der x–Achse aufgelisteten Units als Source Units interpretiert, während Units entlang der y–Achse Target Units repräsentieren. Dies führt dazu, daß für jede Verbindung nur ein Quadrat gezeichnet wird. Ein Eintrag in Spalte 2, Zeile 5 bedeutet also eine Verbindung von Unit 2 zu Unit 5. Bei Feed–Forward–Netzen werden die Eintragungen deswegen vor allem in der linken, unteren Ecke des Displays liegen. Bei großen Netzen ist oft nicht mehr erkennbar, welches Gewicht von einem bestimmten Quadrat dargestellt wird. Auch ist eine exakte Zuordnung zwischen Farbe/Größe und Gewicht nicht mehr möglich. Der Benutzer erhielt deshalb die Möglichkeit durch Anklicken eines Quadrates sich genauer über die zugehörige Verbindung zu informieren. Ein Anklicken eines Verbindungsquadrates öffnet das unten abgebildete Info–Fenster, welches beim Loslassen der Maustaste wieder verschwindet. Wird auf eine nicht existente Verbindung gezeigt, erscheint das in Abbildung 4.8 abgebildete Fenster.

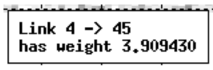

Abbildung 4.7: Info–Fenster zum Weight Visualisation Window

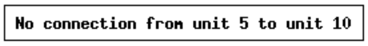

Abbildung 4.8: Info für fehlende Verbindung

4.2.2 Schwarz–Weiß–Darstellung

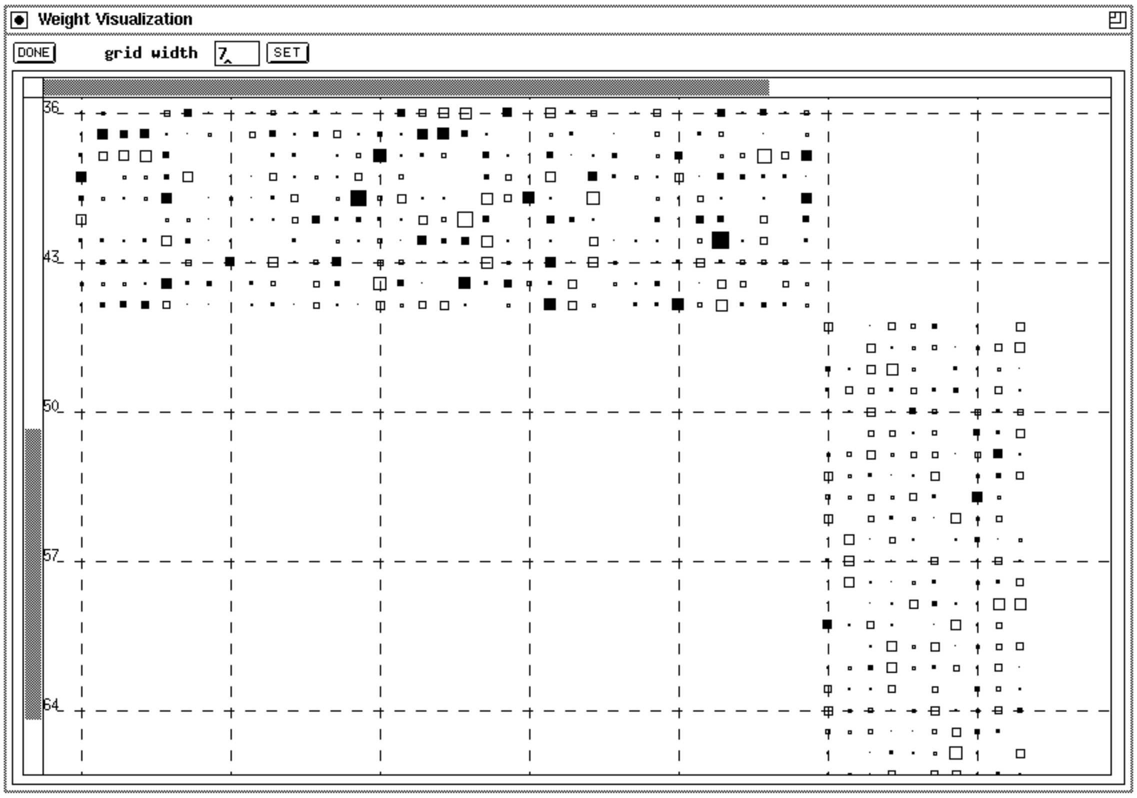

Abbildung 4.9: Ein typisches Hinton–Diagramm

Bei Bildschirmen, die nur die Möglichkeit der Schwarz–Weiß–Darstellung besitzen, ist die einzig mögliche Form der (Gewichts-) Größen Repräsentation die (Quadrat-) Größe. Diese von Hinton [Hinton90] vorgeschlagene Methode der sogenannten *Hinton–Diagramme* erlaubt eine sehr übersichtliche Darstellung vor allem kleinerer Netze.

Um eine genügend feine Rasterung zu ermöglichen, müssen auch sehr große Quadrate zugelassen werden, da kleine Größenunterschiede vom menschlichen Auge nicht gut wahrgenommen werden. Dies führt bereits für mittelgroße Netze zu sehr großen Displays.

Um die Darstellung von negativen Werten zu ermöglichen ist die Verwendung eines Bias für den Wert '0' nicht sinnvoll. Als praktikabel hat sich aber die Verwendung von ausgefüllten Rechtecken für positive, und leeren Rechtecken für negative Werte herausgestellt. Dann erhalten betragsmäßig gleiche Werte auch Quadrate gleicher Größe und die Darstellung eines verschwindenden Wertes (= '0') verschwindet ebenfalls. Für sehr kleine Netze, d.h. Netze mit sehr wenig Verbindungen, wird die maximale Größe eines Quadrates bestimmt als der Quotient aus Fenstergröße und Zellenanzahl. Unterschreitet diese Zahl den Wert 16, so wird 16 als maximale Größe definiert und von Window auf Viewport Darstellung umgeschaltet.

Abbildung 4.9 zeigt ein Gewichtsdisplay mit dem Ausschnitt eines Hinton–Diagramms für das Netz 'letters.net'.

4.2.3 Farbdarstellung

Bei Farbbildschirmen muß nicht mit veränderlichen Quadratgrößen operiert werden. Es können statt dessen farbige Quadrate konstanter Größe verwendet werden, die das darzustellende Gewicht als Farbtönung codieren. Diese von Pinz [PinBis90] eingeführte Darstellung wird als *WV–Diagramm* bezeichnet. Sie erlaubt die übersichtliche Darstellung wesentlich größerer Netze, da pro Verbindung nur ein wesentlich kleineres Flächenstück benötigt wird. Als gute minimale Größe hat sich ein 5x5 Pixel–Quadrat bewährt. In der hier vorliegenden Implementierung wird ein Gewicht von '0' weiß dargestellt. Zunehmend negativere Werte erhalten ein immer tieferes rot, zunehmend positivere Werte erhalten ein immer tieferes grün.

Die WV-Diagramm Darstellung des selben Netzes 'letters.net' findet sich in Abbildung 4.10.

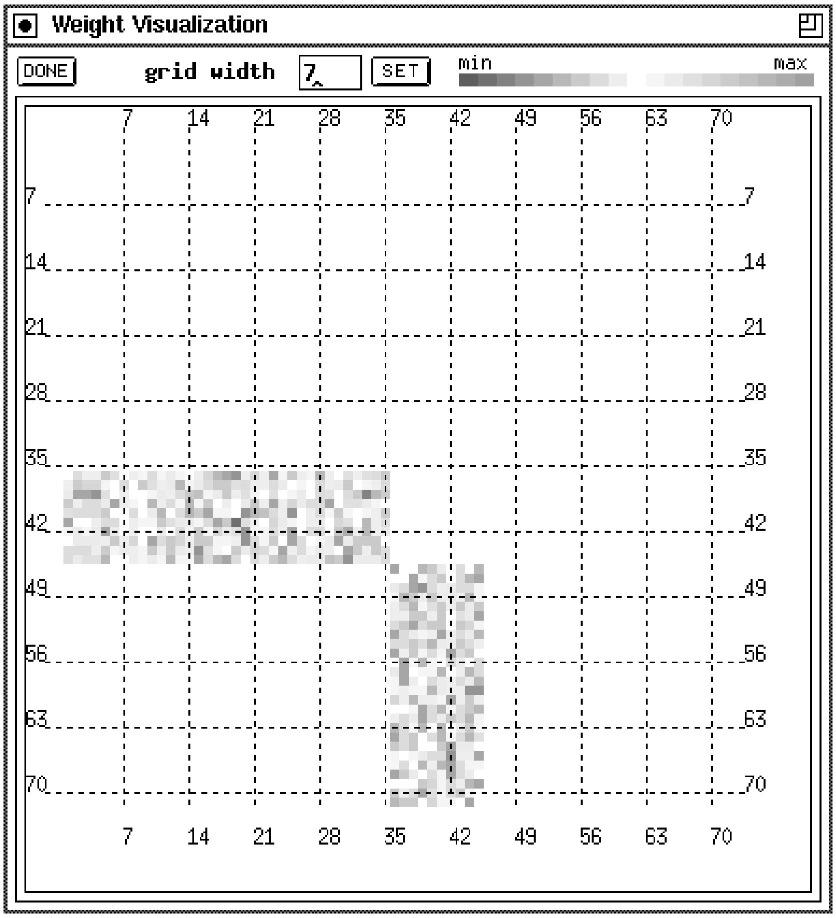

Abbildung 4.10: Ein typisches WV–Diagramm

4.3 Die Inversion eines Netzes

Neben der Frage

> Welcher Klasse ordnet das neuronale Netz das angelegte Muster zu?

ist für die Benutzer neuronaler Netze eine andere Frage oft von viel größerer Bedeutung, nämlich

> Welche Eigenschaften muß ein Muster aufweisen, um von dem neuronalen Netz zu einer bestimmten Klasse zugeordnet zu werden, d.h. um eine bestimmte Ausgabe zu erzeugen?

Um diese Frage beantworten zu können entwickelten J. Kindermann und A. Linden [KinLin90, LinThr90] eine Methode, die das Verhalten des Netzes invertiert. Die „Inversion nichtrekurenter Systeme" ist ein Gradientenverfahren zur Berechnung eines Eingabevektors für nichtzyklische Netzwerke. Obwohl dieses Verfahren für jedes Feed–Forward–Netz benutzt werden kann, erzeugt es nur für bereits trainierte Netze sinnvolle Ausgaben. Die benutze Trainingsmethode spielt dabei aber keine Rolle.

Die Herleitung und Begründung dieses Gradientenverfahrens soll im Folgenden gegeben werden. In Folge der starken Ähnlichkeit dieses Verfahrens mit dem Backpropagation–Algorithmus ähnelt auch diese Herleitung stark der Herleitung der Backpropagation.

4.3.1 Theoretische Grundlagen

Bei der Inversion eines Neuronalen Netzes wird versucht, von einer beliebigen Anfangseingabe ausgehend Schritt für Schritt ein Eingabemuster zu erzeugen, das mit den bestehenden Verbindungen eine gewünschte Ausgabe erzeugt. Um die gesuchte Eingabe zu erreichen, wird die jeweilige Abweichung der vom Netz erzeugten Ausgabe von der gewünschten Zielausgabe als Fehler δ errechnet. Dieser Fehler wird dann benutzt um sich im Eingaberaum net_i auf die Zieleingabe zuzubewegen. Betrag und Richtung dieser Bewegung werden von der Inversionsmethode errechnet.

Bei einstufigen Systemen wird häufig der *Least Mean Square Error* als Fehlermaß verwendet. Der E^{LMS} ist für ein bestimmtes Muster p definiert als

$$E_p^{LMS} = [T_p - O_p]^2 = [T_p - f(\sum_i w_{ij} * o_{pi})]^2$$

und für den globalen Fehler als

$$E^{LMS} = \sum_{p=1}^{n} E_p^{LMS}$$

wobei

w_{ij}	Das Gewicht der Verbindung von Knoten i zu Knoten j
o_{pi}	Die Ausgabe von Knoten i für das Patterns p
T_p	Die Zielausgabe des Patterns p
I_p	Das Eingabemuster des Patterns p
O_p	Das Ausgabemuster des Patterns p
$f()$	Die Aktivierungsfunktion der Knoten der ersten Schicht

Das Trainingsziel muß also sein, E^{LMS}, oder kurz E, zu minimieren, womit sich das Trainingsproblem auf ein Optimierungsproblem reduziert. Ist E nur von einem einzigen Gewicht w abhängig, so ergibt sich eine eindimensionale Fehlerlandschaft wie in Abbildung 4.11.

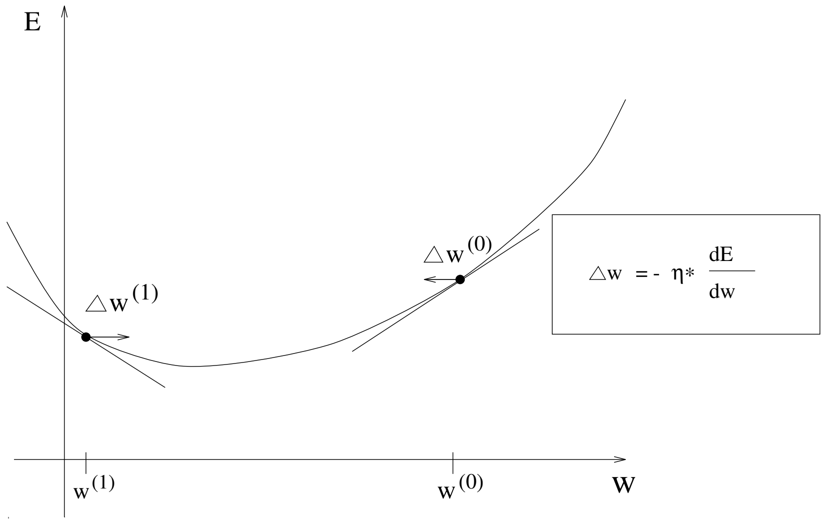

Abbildung 4.11: Eine eindimensionale Fehlerkurve

Aus dieser Abbildung wird auch deutlich, daß die Kenntnis der Ableitung ein iteratives Verfahren zur Fehlerminimierung ermöglicht. Ist nämlich $E^{'}(w^{(0)})$ positiv, so ist E an der Stelle $w^{(0)}$ ansteigend. Daraus folgt aber, daß eine Verringerung von $w^{(0)}$ eine Veringerung von $E(w^{(0)})$ zur Konsequenz hat. Umgekehrt folgt aus einer Vergrößerung von $w^{(1)}$ bei negativem $E^{'}(w^{(1)})$ ebenfalls eine Verringerung von $E(w^{(1)})$.

Abbildung 4.12 veranschaulicht dasselbe Verfahren nochmals für zwei Dimensionen / Gewichte. Die Minimierung von E wird durch iterative Anpassung von W mittels

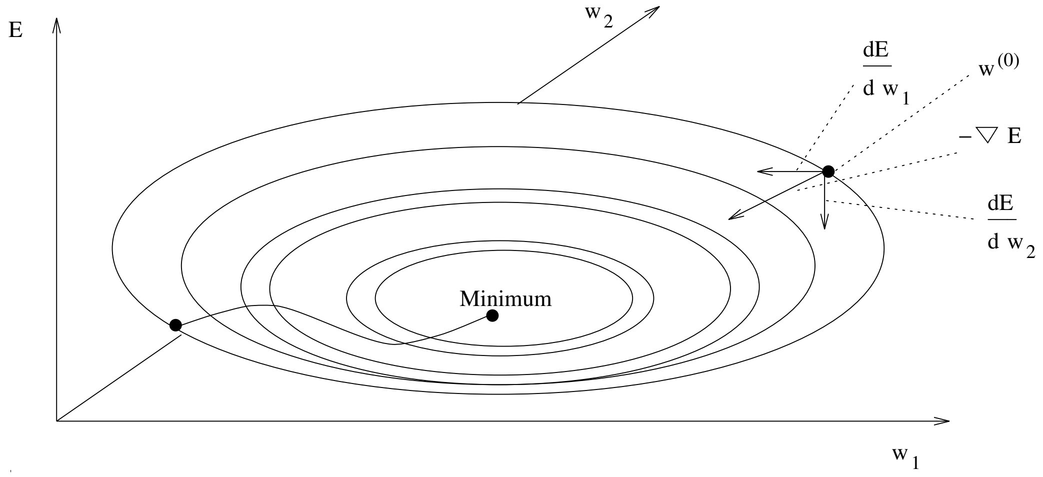

Abbildung 4.12: Eine zweidimensionale Fehlerfläche

$$\triangle W = -\eta * \frac{dE}{dW}$$

erreicht. Nach dem Start an einem beliebigen Initialwert $w^{(0)}$ erreicht man bei genügend kleiner Schrittweite η über die Wertfolge

$$w^{(0)}, w^{(1)}, w^{(2)}, \cdots$$

das Fehlerminimum beliebig genau.

Da E im allgemeinen eine Funktion mehrerer Veränderlicher ist, wird für jedes Gewicht die partielle Ableitung benötigt. Es gilt daher die neue Modifikationsregel

$$\triangle w_{ij} = -\eta * \frac{\partial E}{\partial w_{ij}}$$

Der Gradient von w ist dann definiert als vektorielle Zusammenfassung aller partiellen Ableitungen

$$\nabla_w E = \left(\frac{\partial E}{\partial w_{i1}}, \frac{\partial E}{\partial w_{i2}}, \frac{\partial E}{\partial w_{i3}}, \cdots \right)$$

und die neue Modifikationsregel kann geschrieben werden als

$$\triangle W = -\eta * \nabla_w E$$

Als Fehlersignal der Unit j wird dann der Term

$$\delta_{pj} = -\frac{\partial E}{\partial net_{pj}} = -\frac{\partial E}{\partial a_{pj}} * \frac{\partial a_{pj}}{\partial net_{pj}}$$

definiert, wobei net_{pj} die Netzeingabe der Unit j für das Pattern p ist. Definiert ist sie als

$$net_{pj} = \sum_{i \in P(j)} w_{ij} * a_i$$

mit

P(j)	Die Menge der Vorgänger von Unit j
a_i	Die Aktivierung der Unit i

Wird eine logistische Abbildungsfunktion von a_i auf net_i verwendet, wie in Kapitel 4.3.2 beschrieben, dann ergeben sich die zwei partiellen Ableitungen im Fehlersignal der Unit i zu

$$\frac{\partial E_p}{\partial a_{pi}} = \begin{cases} -2 * (T_{pi} - a_{pi}) & \text{falls } i \text{ eine Ausgabezelle ist} \\ -\sum_{k \in S(i)} \delta_{pk} * w_{ik} & \text{sonst} \end{cases}$$

und

$$\frac{\partial a_{pi}}{\partial net_{pi}} = a_{pi} * (1 - a_{pi})$$

Das Fehlersignal kann berechnet werden als

$$\delta_{pi} = a_{pi} * (1 - a_{pi}) * \sum_{k \in S(i)} \delta_{pk} * w_{ik}$$

und für den Anpassungswert der Unit–Aktivierung folgt

$$\triangle net_{pi} = \eta * \delta_{pi} \quad \text{bzw.} \quad net_{pi} = net_{pi} + \eta * \delta_{pi}$$

wobei der Faktor 2 aus der Ableitung $\frac{\partial E_p}{\partial a_{pi}}$ der Einfachheit halber mit der Schrittweite η verrrechnet wird.

4.3.2 Das Verfahren

Für die Gradientenbestimmung wird zuerst ein beliebiges Muster an den Eingabezellen angelegt. Dieses Muster (Eingangsvektor $I^{(0)}$) wird vom Netz verarbeitet und erzeugt eine erste Ausgabe (Ausgabevektor $O^{(0)}$), bedingt durch die interne Struktur des Netzes und die Gewichte der Verbindungen (funktionale Ausprägung f_w). Es gilt also

$$O^{(0)} = f_w(I^{(0)})$$

Der Ausgabevektor $O^{(0)}$ wird nun mit dem Zielvektor T verglichen, der die gewünschte Ausgabe darstellt. Als Maß für die Qualität der erzeugten Ausgabe kann folgendes Fehlermaß dienen:

$$E = (T - f_w(I(t)))^2$$

Die Abweichungen der einzelnen Aktivierungen der Ausgabeknoten werden als Fehlersignale nun im Netz zurückpropagiert (siehe Abbildung 4.13).

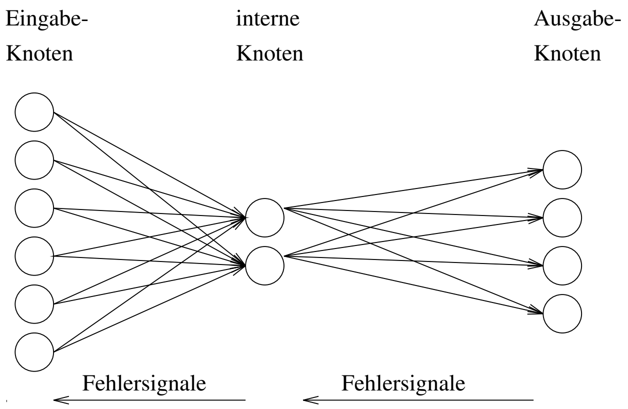

Abbildung 4.13: Zurückpropagieren der Fehlersignale im Netz

Dieses Zurückpropagieren entspricht exakt demselben Vorgang im Backpropagation Trainingsverfahren. Mit dem Unterschied allerdings, daß hier keinerlei Gewichte verändert werden und das Fehlersignal bis an die Eingabeunits weitergegeben wird. Es gilt also:

$$\delta_{i^0} = \begin{cases} f'_{out\ i}(f_w(I^{(0)}) - T_i) & \text{falls } i \text{ eine Output Unit ist} \\ f'_{out\ i}(\sum_k \delta_{k^{(0)}} * w_{ik}) & \text{sonst} \end{cases}$$

wobei:

δ_{i^0}	der Fehler von Unit i im (0)ten Schritt
$f_w(I^{(0)})$	die durch Anlegen von $I^{(0)}$ erzeugte Ausgabe
T_i	Zielausgabe für Unit i
w_{ik}	Verbindungsgewicht zwischen Unit i und Nachfolger k
k	Index für alle Nachfolgeunits von Unit i
$f'_{out\ i}$	Ableitung der Ausgabefunktion für Unit i

und

$w_{ik}^{(1)} = w_{ik}^{(0)}$

Wenn die Fehlersignale, als durch die Verbindungsstärken gewichtete Fehlerinformation der Ausgabeunits, an den Eingabeunits errechnet worden sind, entspricht dieser Fehlervektor einem Gradienten im Eingaberaum, d.h. er gibt die Richtung an, in der sich die Eingabe ändern muß um den Fehler der Ausgabe zu minimieren. Wird dieser Fehlervektor $F^{(0)}$ mit dem Eingabevektor $I^{(0)}$ verrechnet, ergibt sich ein neuer, fehlerminimierter Eingabevektor $I^{(1)}$, der dem Netz eingegeben wird und einen neuen Ausgabevektor $O^{(1)}$ erzeugt. Dabei wird $I^{(1)}$ errechnet als:

$$I^{(1)} = I^{(0)} + \eta * \delta_{i^{(0)}}$$

wobei:

η	Die Schrittweite im Eingaberaum (Lernfaktor)
δ_{i^0}	Das Fehlersignal wie oben

Dabei kann es aber vorkommen, daß die Aktivierungen der Eingabeunits aus dem vorgesehenen Intervall $[0, 1]$ gedrängt werden. Dies würde zwar den Netzwerkfehler minimieren, soll aber vermieden werden, um eine vernünftige Aktivierungssemantik zu erhalten (siehe dazu [Anders90, Hinton89, Linden90]). Deswegen findet die Aktualisierung des Eingabevektors nicht direkt, sondern über den Umweg einer Pseudonetzeingabe net_i statt. Diese ist definiert als:

$$net_i = -\ln(\frac{1}{a_i} - 1)$$

Um die Umkehrfunktion eindeutig definiert zu halten, muß für die Zellaktivierung statt dem Intervall $[0, 1]$ das Intervall $(0, 1)$ vorgeschrieben werden, was die Semantik allerdings praktisch nicht verändert [1]. Erreicht das Fehlersignal beim Zurückpropagieren die Eingabezellen, wird nun statt a_i die Pseudonetzeingabe net_i mit

$$net_i = net_i + \eta * \delta_i$$

verändert. Dann wird die Unitaktivierung aktualisiert mit

$$a_i = \frac{1}{1 + e^{-net_i}}$$

Hierdurch ist gewährleistet, daß a_i immer in dem Intervall $(0, 1)$ liegt.

Iteratives Wiederholen dieses Verfahrens führt zu einer Sequenz von Eingabevektoren $I^{(0)}, I^{(1)}, I^{(2)}, \ldots,$ die bei geeigneter Schrittweite η zu einem $I^{(t)}$ konvergiert, bei dem

[1] Intern wird statt 0 0.0001, und statt 1 0.9999 verwendet

der erzeugte Fehlervektor kleiner als ein vorgegebenes Maß wird, und somit die gewünschte Ausgabe optimal erzeugt.

Dieses Verfahren ist keineswegs eindeutig und kann bei verschiedenen Schrittweiten η oder verschiedenen Startvektoren $I^{(0)}$ durchaus verschiedene Vektoren $I^{(t)}$ erzeugen. Dies ist dadurch zu erklären, daß die Fläche im Eingaberaum durchaus mehrere lokale Minima besitzen kann.

Die Fülle von möglichen Zieleingaben $I^{(t)}$ kann durch eine zusätzliche Bedingung für die Eingabeknoten eingeschränkt werden. Diese Bedingung kann den Eingabeunits vorschreiben möglichst nahe an einem bestimmten Wert zu liegen. So kann man festlegen, daß möglichst alle Units möglichst aktiv ($a_i \rightarrow 1$), möglichst alle Units möglichst *nicht* aktiv ($a_i \rightarrow 0$), oder möglichst indifferent ($a_i \rightarrow 0.5$) sein sollen.

Allgemein legt diese zusätzliche Bedingung fest, daß der gesuchte $I^{(t)}$ einerseits $f_w(I(t))$ möglichst gut an T approximiert, und andererseits selber nahe einem Eingabezielvektor I_T liegt. Analytisch kann dies in einem erweiterten Fehlermaß

$$E^{'} = (T - f_w(I(t)))^2 \; + \; k * (I_T - I(t))^2$$

erfasst werden, wobei k angibt, wie stark sich die zusätzliche Bedingung (2. Approximation) auswirken soll.

4.3.3 Das Inversions Display

Abbildung 4.14 zeigt ein typisches Inversions–Display.

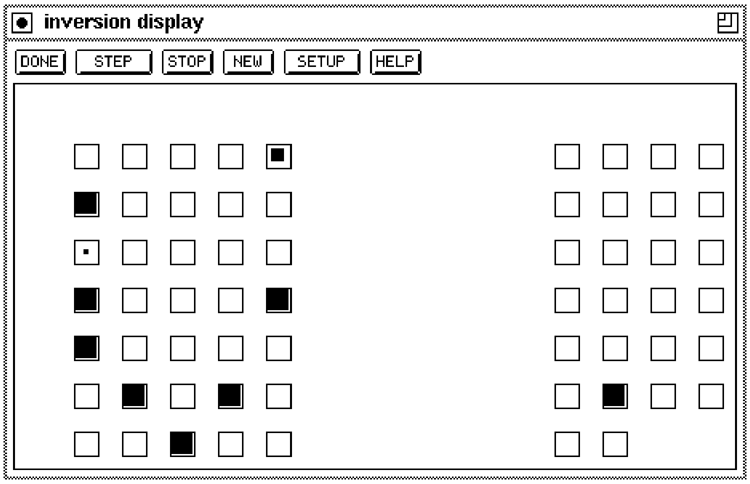

Abbildung 4.14: Ein typisches Inversions–Display

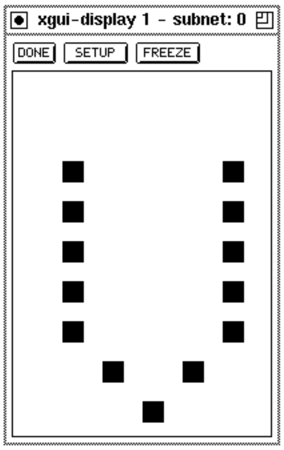

Abbildung 4.15: Das Muster des Buchstabens V

Das dargestellte Muster zeigt die Units die minimal aktiviert sein müssen (Zusatzbedingung $a_i \to 0$) damit das trainierte Netz 'letters.net' den Buchstaben 'V' erkennt. Das Verfahren terminierte nach 52 Iterationen bei einer zulässigen Abweichung der Aktivierung der Ausgabeunits vom Zielvektor um 0.1, einer Lernrate von 2.0 und einer Gewichtung der 2. Approximation von 0.5. Das vorhergehende Training des Netzes wurde nach 1000 Zyklen Backpropagation mit einem Fehler von 0.06622 und einer Standardabweichung des Fehlers von 0.0522 beendet.

Das zugehörige Original–Pattern für den Buchstaben 'V' ist in Abbildung 4.15 dargestellt.

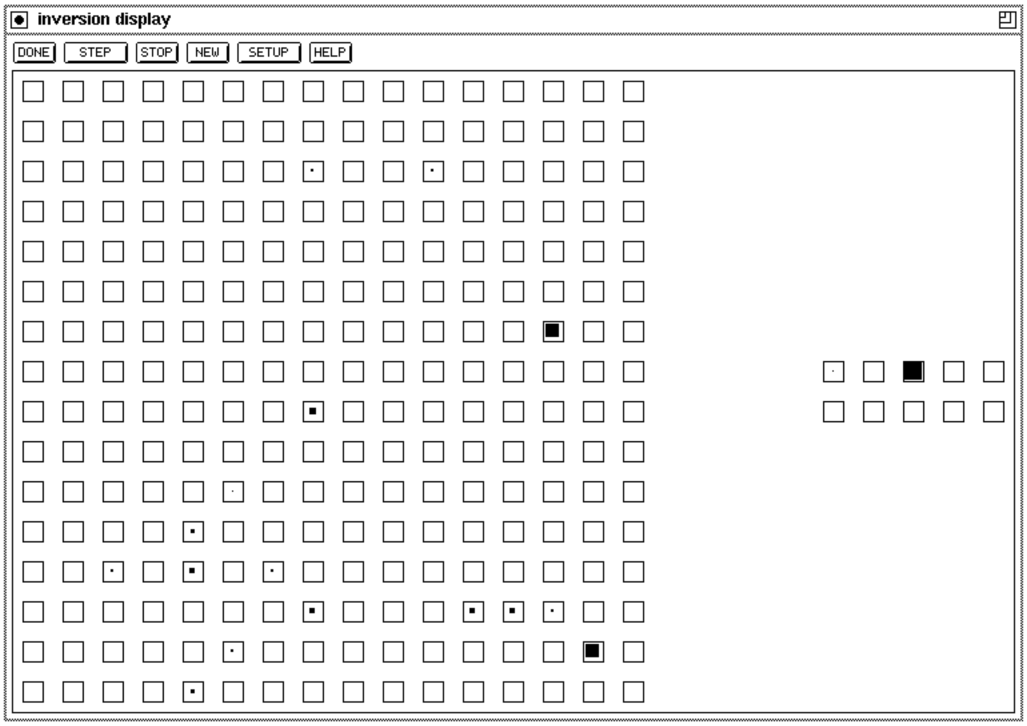

Abbildung 4.16: Das Inversionsergebnis der Ziffer 2 für minimale Aktivierung

An einem etwas größeren Netz kann man gut den Einfluß erkennen, den die Wahl der Gewichtung der Eingabeapproximation ausübt. Die Abbildungen 4.16 und 4.17 stellen das Inversionsergebnis für die Ziffer 2 des Netzes 'big_digits.net' dar. Im ersten Fall wurde eine Eingabeaktivierung von 0.0 gewählt. Das Verfahren konvergierte nach 233 Zyklen mit einer Schrittweite von 2.0.

Im zweiten Fall betrug die Eingabeaktivierung 1.0. Das Verfahren konvergierte hier mit einer Schrittweite von 5.0 erst nach 619 Zyklen.

Das Orginalmuster der Ziffer 2 ist in Abbildung 4.18 dargestellt.

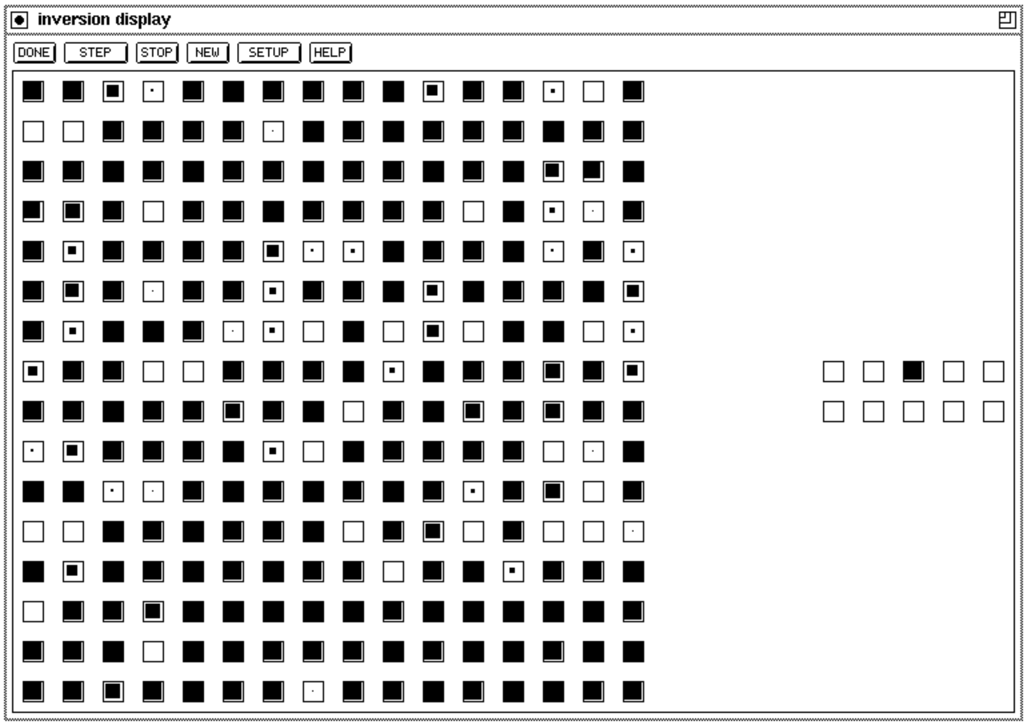

Abbildung 4.17: Das Inversionsergebnis der Ziffer 2 für maximale Aktivierung

Das Display besteht prinzipiell aus zwei Teilen. Während der größere, untere Teil eine Skizze des Netzes beinhaltet, in der die Ein– und Ausgabe–Units mit ihren Aktivierungen dargestellt sind, besteht der obere Teil des Displays aus einer Reihe von Bedienknöpfen.

Diese und ihre Funktionen sind im einzelnen:

DONE Zum Beenden des Inversionsalgorithmus und Schließen des Displays.

STEP Zum Starten / Fortsetzen des Algorithmus. Das Programm iteriert dann so lange, bis das Verfahren konvergiert hat (d.h. keine Unit mit einer unzulässig

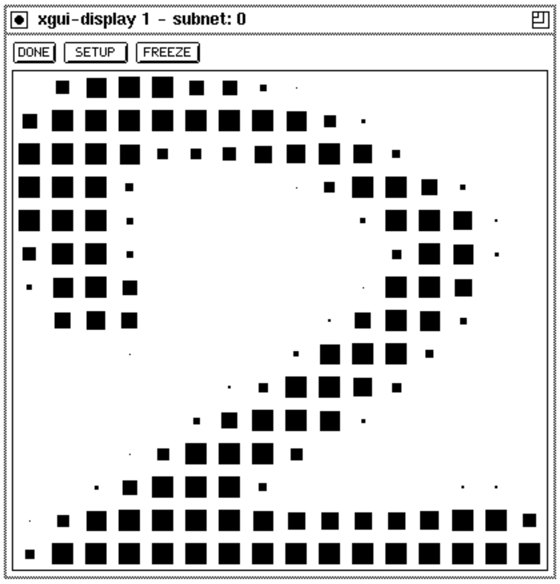

Abbildung 4.18: Das Muster der Ziffer 2

Abbildung 4.19: Die Bedienknöpfe des Inversionsdisplays

hohen Abweichung in der Aktivierung mehr existiert), oder es mit dem **STOP** Button unterbrochen wird.

Die theoretische Bedingung für Konvergenz, nämlich

$$E^{'} = 0$$

kann schon aus Gründen der Rechengenauigkeit nicht realisiert werden. Statt dessen wird das Konvergenzkriterium als erfüllt betrachtet, wenn

$$E^{'} \leq \delta_{max}$$

wobei δ_{max} ein beliebiger, vom Benutzer definierbarer, Wert ist, der den maximalen Abstand der Ausgabe einer Unit ihrer Zielausgabe angibt. Das Verfahren wird deshalb beendet, wenn die Ausgaben o aller Ausgabe–Units folgende Bedingung erfüllen

$$-\delta_{max} \leq o \leq \delta_{max}$$

Die Variable δ_{max} kann vom Benutzer im Setup Window unter `delta_max` gesetzt werden (siehe unten).

Eine ursprünglich geplante, automatische Erkennung von Endlosschleifen konnte nicht verwirklicht werden. Endlosschleifen treten auf, wenn der Fehler des Netzwerkes nach anfänglichem Sinken, durch eine zu groß gewählte Schrittweite plötzlich wieder ansteigt, bevor er sich erneut verringert. Probeläufe des Programms haben ergeben, daß diese Schleifen zu jeder Zeit auftreten können, also sowohl solange der Fehler noch relativ groß ist, als auch wenn das Verfahren bereits beinahe konvergiert hat. Es kann vorkommen, daß ein Durchgang konvergiert, obwohl in seinem Verlauf ein sehr großer Sprung zu einem größeren Netzwerkfehler hin stattfindet. Ebenso kann ein Durchlauf, dessen Fehler nur minimal expandiert, trotzdem unter Umständen nicht konvergieren. Auch ein endloses Schwingen, bei dem ein bestimmter Fehlerwert nie zweimal erreicht wird, wurde beobachtet. Dieses alles macht eine automatische Erkennung unmöglich und legt die Verantwortung über einen Abbruch ausschließlich dem Benutzer auf.

STOP Zum Unterbrechen des Algorithmus. Dies kann aus mehreren Gründen notwendig bzw. vorteilhaft sein. Während des Inversionslaufes findet keine automatische Bildschirmauffrischung statt. Möchte der Benutzer also einen Eindruck vom momentanen Stand der Dinge, muß er das Programm mit **STOP** unterbrechen. Es wird dann der augenblickliche Zustand der Ein– und Ausgabeunits dargestellt. Ein Anklicken des **STEP** Buttons setzt dann das Programm an der zuvor verlassenen Stelle fort. Während der Unterbrechung kann der Benutzer aber auch Variablen ändern oder das Programm in den Ausgangszustand zurückbringen. Eine Unterbrechung mit dem **STOP** Button ist auch angezeigt, wenn die Entwicklung des Netzwerkfehlers, die immer im Orginalfenster angezeigt wird, darauf hindeutet, daß sich der Algorithmus in einer Endlosschleife befindet (siehe oben).

NEW Setzt das Programm in den definierten Ausgangszustand zurück.

SETUP Öffnet ein Popup Window zum Einstellen von allen an der Inversion beteiligten Variablen. Diese sind im einzelnen:

eta	Die Schrittweite beim Ändern der Aktivierungen. Entspricht dem Lernfaktor.
delta_max	Die maximale Abweichung der Aktivierung einer Ausgabe–Unit, ab der die Unit als fehlerhaft geführt wird.
Input pattern	Anfangsaktivierung aller Eingabe–Units.
2nd approx ratio	Gewichtung der Nähe zum Eingabezielvektor.

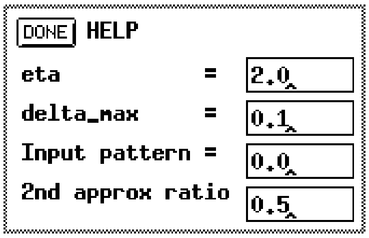

Abbildung 4.20: Popup zum Setzen der Inversionsvariablen

Eine kurze Beschreibung dieser Variablen kann vom Benutzer in einem Hilfefenster (Abbildung 4.21) angefordert werden. Eine detailierte Beschreibung dieser Variablen findet sich in Abschnitt 4.3.2, eine Darstellung des Setup Fensters in Abbildung 4.20.

An dieser Stelle sei angemerkt, daß der Algorithmus sehr instabil ist. Ein Durchgang kann konvergieren, während ein anderer Durchgang mit leicht veränderten Variablen zu einer Endlosschleife führt. Die standardmässig eingestellten Werte haben sich als allgemein relativ günstig erwiesen und sollten deshalb nur mit Vorsicht geändert werden. Der Benutzer sollte auch damit rechnen, bei einem neuen Netzwerk eventuell mehrere verschiedene Variablenwerte testen zu müssen, bevor der Algorithmus konvergiert und ein zufriedenstellendes Ergebnis liefert.

Im allgemeinen ist die Konvergenz um so wahrscheinlicher, je besser das Netz vorher trainiert wurde. Die Standardabweichung des Trainingsfehlers sollte dabei kleiner als 0.05 geworden sein.

HELP Öffnet das in Abbildung 4.22 dargestellte Hilfefenster, welches die Handhabung des Inversionsdisplays kurz erklärt.

```
popupWin
DONE
eta           : Learning parameter; should range from 1.0 to 10.0
delta_max     : Maximum allowed error for an output unit
Input  pattern: Initial activation for all input units
2nd approx    : Degree to which the target input pattern
                is dependent on the initial input pattern
                good values range from 0.2 to 0.8
```

Abbildung 4.21: Hilfefenster zur Einstellung der Inversionsvariablen

```
DONE
Start/continue inversion with 'STEP'
Halt inversion run anytime with 'STOP'
Reset inversion with 'NEW'
Change parameters with 'SETUP'
After SETUP continue with 'STEP',
  or restart with 'NEW','STEP'
HINT: If inversion is performed on an untrained net,
      or on a trained net with an average error of more than 0.05,
      the algorithm is very likely not to converge !!
```

Abbildung 4.22: Hilfe Fenster für das Inversionsdisplay

Die Darstellung des Netzes im unteren Teil des Displays richtet sich nach den Einstellungen der Parameter des zuletzt geöffneten 2D–Displays. Die Lage, Farbcodierung und Größe der Units wird von diesem Displaypointer gelesen. Die Größe und Form des Displays passt sich automatisch der Lage der darzustellenden Units an.

Ein Testen der Gültigkeit des errechneten Eingabemusters kann auf die nachfolgend erklärte Art und Weise sehr einfach durchgeführt werden:

Im regulären 2D–Display werden alle Eingabe–Units mit der Maus selektiert und sofort wieder deselektiert. Dies überträgt das Eingabemuster. Daraufhin werden alle Ausgabe–Units selektiert, mit der Eingabesequenz U S A (Unit Set Activation) auf Null gesetzt, und wieder deselektiert. Durch ein Anklicken des **NEW** Buttons in der 'Pattern' Zeile des Remote Panels wird dieses Muster intern gespeichert und kann über den **TEST** Button in der 'Cycles' Zeile des Remote Panels abgeprüft werden.

Soll mit dem geladenen / definierten Satz von Mustern nachfolgend noch ein Training stattfinden, so empfiehlt es sich, das soeben erzeugte Muster wieder zu löschen. Andernfalls würde das Training des Netzes verfälscht werden.

Kapitel 5

Implementierung

5.1 Module

Die im Rahmen dieser Diplomarbeit neu entwickelte Software ist auf vier Module verteilt. Soweit Änderungen in anderen Modulen notwendig waren, ist dies bei den zugehörigen neuen Modulen vermerkt. Die Module im Einzelnen sind:

5.1.1 ui_funcdispl.c

Dieses Modul enthält alle zur Darstellung der Unit–Funktionen notwendigen Funktionen. Als Include–Files werden außer *ui_funcdispl.h* noch eine Reihe bereits seither benutzter '.h' Files von der Oberfläche und von X-Windows benötigt. Die hier definierte Funktion *ui_createMyScreen* wird auch vom Modul *ui_inversion.c* genutzt. Aufgerufen wird dieses Modul von der Funktion *ui_xCreateInfoPanel* im Modul *ui_info.c*. Zu diesem Zweck wurden dort vier neue Menuepunkte und Callback Funktionen definiert, sowie alle Funktionen dieses Moduls als 'extern' deklariert. *ui_funcdispl.c* ist ein Modul der Oberfläche XGUI.

5.1.2 ui_displweights.c

Das Modul *ui_displweights.c* verwirklicht die neue Funktionalität zur Darstellung der Verbindungsgewichte des Netzwerks. Es beinhaltet sowohl die Funktionen für Schwarz–Weiß– als auch für Farbdarstellung. Zum Aufruf dieses Moduls wurde in der Funktion *ui_xCreatePanel* des Moduls *ui_main.c* ein Callback–Aufruf und in der Funktion *ui_netUpdateWhole* des Moduls *ui_netUpdate.c* ein Unterprogrammaufruf eingefügt. Auch dieses Modul ist ein Teil der Oberfläche XGUI.

5.1.3 ui_inversion.c

ui_inversion.c beinhaltet die Funktionen zur Verwaltung und Darstellung des Inversionsalgorithmus. Dieses Modul ruft das Modul *krui_inversion.c* des Kerns zur Durchführung der

Inversion auf und ist deshalb ein Teil der Kern–Oberflächen–Schnittstelle. Aufgerufen wird es ebenfalls über einen Callback in der Funktion *ui_xCreatePanel*.

5.1.4 krui_inversion.c

Dieses Modul beinhaltet die drei Funktionen *krui_initInversion*, *krui_inv_forwardPass* und *krui_inv_backwardPass*, die den eigentlichen Inversionsalgorithmus implementieren. Dieses Modul ist als einziges der hier neu Entwickelten ein Teil des SNNS Kerns. Es ist daher nicht in die Oberflächen–Bibliothek *libxgui.a* eingebunden, sondern in die Kern–Bibliothek *libkernel.a*. Ein Aufruf dieser Funktion ohne die Verwendung der Oberfläche XGUI ist möglich. Die resultierenden Aktivierungswerte der Ein– und Ausgabe–Units können dann im Feld `topo_ptr` abgerufen werden. Dieses Feld wurde bereits für die Implementierung vieler Lernverfahren verwendet.

5.2 Funktionen

5.2.1 Funktionen des Moduls ui_funcdispl.c

ui_functionPopup

Hauptprogramm für die Darstellung der Unitfunktionen. Prüft, ob eine Darstellung möglich ist, stellt fest, welche Funktion gezeichnet werden soll und baut die Widget–Hierarchie auf.

ui_createMyScreen

Setzt alle zum Erstellen eines Widgets notwendigen Variablen mittels `XtSetArg()` und erzeugt es dann mit `XtCreateManagedWidget`.

ui_drawFunction

Zeichenroutine für das Modul. Hier werden die vier möglichen Fälle (Target oder Source Unit, Aktivierungs– oder Ausgabefunktion) unterschieden und die entsprechenden Funktionen zur Punktegenerierung gerufen. Die Funktion zeichnet sowohl den Funktionsgraphen als auch ein Achsenkreuz mit Skalierung.

local_kill

Callbackfunktion für den **DONE** Button. Schließt das zugehörige Fenster einer Unitfunktion.

ui_setXY

Callbackfunktion für den **SET** Button des Unit–Function–Displays. Liest die Werte der Dialogfenster für die Darstellungsgrenzen in globale Variablen ein.

5.2.2 Funktionen des Moduls ui_displweights.c

ui_displWeights

Hauptprogramm für die Darstellung der Verbindungsgewichte. Prüft, ob eine Darstellung möglich ist, setzt die Größe des Fensters und der Quadrate, baut die Widget–Hierarchie auf und definiert die notwendigen Callbacks. Auch der Bildschirmtyp wird festgestellt und dann automatisch die Darstellung in Hinton–Diagrammen (schwarz-weiß) oder WV-Diagrammen (Farbe) angewählt.

ui_displWeightsBW

Darstellungsfunktion für Hinton–Diagramme auf Schwarz–Weiß–Bildschirmen. Die Funktion zeichnet ein Hilfsgitter inclusive der Beschriftung, bevor die Verbindungsgewichte als Quadrate an den Gittermaschen gezeichnet werden.

ui_displWeightsCOL

Analog zur Funktion *ui_displWeightsBW*, aber für die Darstellung auf Farbbildschirmen mittels WV–Diagrammen. Zusätzlich wird hier auch noch die notwendige Farbpalette für die Gewichtsrepräsentation erzeugt. Um das Zeichnen der Gewichte zu beschleunigen werden 20 Graphikkontexte definiert. Jeder Kontext erhält eine andere Vordergrundfarbe. Beim Zeichnen der Quadrate wird dann der jeweils zum Gewicht passende Kontext verwendet. Dieses Verfahren ist wesentlich schneller als das Neusetzen des Vordergrundes.

ui_displWeightsFromUpdate

Funktion zur Vorverarbeitung bei Aufruf durch die Funktion *ui_netUpdateWhole*. Es wird geprüft ob ein Update erfolgen muß, d.h. ob das Fenster überhaupt geöffnet ist, ob sich das Netz seit dem letzten Aufruf von *ui_displWeights* geändert hat und ob die Funktion *ui_displWeightsBW* oder *ui_displWeightsCOL* gerufen werden muß.

ui_uncreate

Callback–Funktion für den **DONE** Button. Schließt das Fenster.

5.2.3 Funktionen des Moduls ui_inversion.c

ui_inversion

Hauptprogramm für den gesammten Inversionsalgorithmus. Prüft, ob ein Netz geladen ist, genügend Speicherplatz vorhanden ist, stellt die aktuelle Pixelgröße für die Unit–Quadrate fest und definiert alle notwendigen Widgets und Callbacks.

ui_invSetup

Baut das Setup–Panel für die Inversionsvariablen auf und definiert die dafür notwendigen Widgets und Callbacks.

ui_popupDone

Callback–Funktion für den **DONE** Button. Schließt das Setup–Fenster.

ui_helpDone

Callback–Funktion für den **DONE** Button. Schließt das Hilfe–Fenster.

ui_invHelp

Hilfefunktion für die Inversion allgemein und das Setup–Panel speziell. Baut ein Fenster auf und stellt darin kurz die wichtigsten Informationen zum jeweiligen Kontext zur Verfügung.

ui_set_IO_units

Initialisiert die Ein– und Ausgabe–Units. Dazu werden zwei doppelt verkettete Listen angelegt, eine für die Eingabe–Units die andere für die Ausgabe–Units. Die Eingabe–Units werden mit dem im Setup–Panel definierten Wert `INPUT_PAT` als Aktivierung und der Pseudonetzeingabe initialisiert (zur Berechnung der Pseudonetzeingabe siehe Seite 28). Die Ausgabe–Units erhalten den Wert 0.0.

ui_closeDisplay

Callback–Funktion für den **DONE** Button. Schließt das Inversions–Display und löscht die in `ui_set_IO_units` angelegten Listen.

ui_inv_new

Setzt das Netz in den definierten Ausgangszustand zurück. Iteration counter und Aktivierungen der Units werden zurückgesetzt. Abschließend wird das Netz neu gezeichnet.

ui_invEvent

Der Eventhandler für die Inversion. Diese Funktion verhindert, daß das Inversions–Display bei einem Expose Event mehrmals gezeichnet wird, was bei großen Netzen zu einer deutlichen Verlangsamung führen würde.

ui_drawInput

Zeichnet die Aktivierungen der Eingabe– ***und Ausgabe***–Units. Farb– oder Monochrom–Bildschirme werden erkannt und entsprechend behandelt. Die Darstellung der Aktivierung der Units ist analog zu der im normalen Display Window.

ui_start_inversion

Die Callback–Funktion für den **STEP** Button. Sie startet die Funktion *ui_do_inversion* als eine sogenannte Working Procedure, die im Hintergrund weiterläuft, während der Benutzer mit dem Programm *snns* andere Aufgaben verfolgen kann.

ui_stop_inversion

Löscht die von *ui_start_inversion* angelegte Working Procedure, d.h. unterbricht den Inversionslauf. Die Funktion zeichnet das Netz neu und gibt eine Statusmeldung auf das Shell Window aus. Sie ist die Callback–Funktion für den **STOP** Button.

ui_do_inversion

Monitorprogramm für die Inversion. Diese Funktion prüft, welche Ausgabe–Units als Zielausgaben selektiert wurden und ruft die Kern–Funktionen des Inversionslaufes auf. Sie gibt neben Fehlermeldungen auch fortwährend Statusmeldungen über den momentanen Zustand der Inversion am Shell Window aus.

5.2.4 Funktionen des Moduls krui_inversion.c

krui_initInversion

Initialisiert die Inversionsmethode. Baut eine Liste von topologisch sortierten Units auf und prüft dabei, ob ein Inversionslauf mit dem aktuellen Netz überhaupt möglich ist.

krui_inv_forwardPass

Propagiert die Aktivierungen von den Eingabeunits zu den Ausgabeunits. Diese Funktion entspricht nahezu vollständig der Funktion zur Vorwärtspropagierung beim Lernverfahren Backpropagation.

krui_inv_backwardPass

Führt die eigentliche Netzinversion durch. Die Fehlerwerte der Ausgabeunits werden von dieser Funktion an die Eingabeunits zurückgereicht. Die Verbindungsgewichte des Netzes bestimmen den Einfluß der einzelnen Fehlerwerte an der nächsthöheren Schicht, werden selbst aber nicht verändert. Abschließend wird noch die Aktivierung und die Pseudo–Aktivierung der Eingabeunits berechnet (siehe Abschnitt 4.3.2).

5.3 Aufrufstruktur

Alle neu implementierten Module sind über Pull–Up–Menüs des Info–Panels ansprechbar. Der Aufruf erfolgt dann durch Callback Funktionen. Die Aufrufe von X-Window und X-Intrinsic Standardfunktionen werden im Folgenden nicht erwähnt.

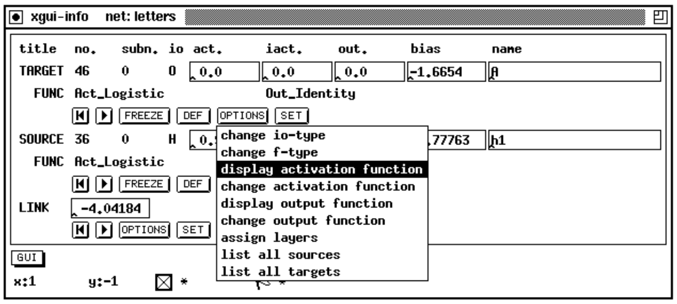

Abbildung 5.1: Aufruf zur Darstellung der Aktivierungsfunktion der Target–Unit

Die Darstellung der Unit Funktionen wird durch das Anklicken des entsprechenden Menüpunktes unter dem **OPTION** Button der Target– oder Source–Unit Zeile aktiviert.

Dies ist die interne Aufrufstruktur für die Darstellung der Unit–Funktionen

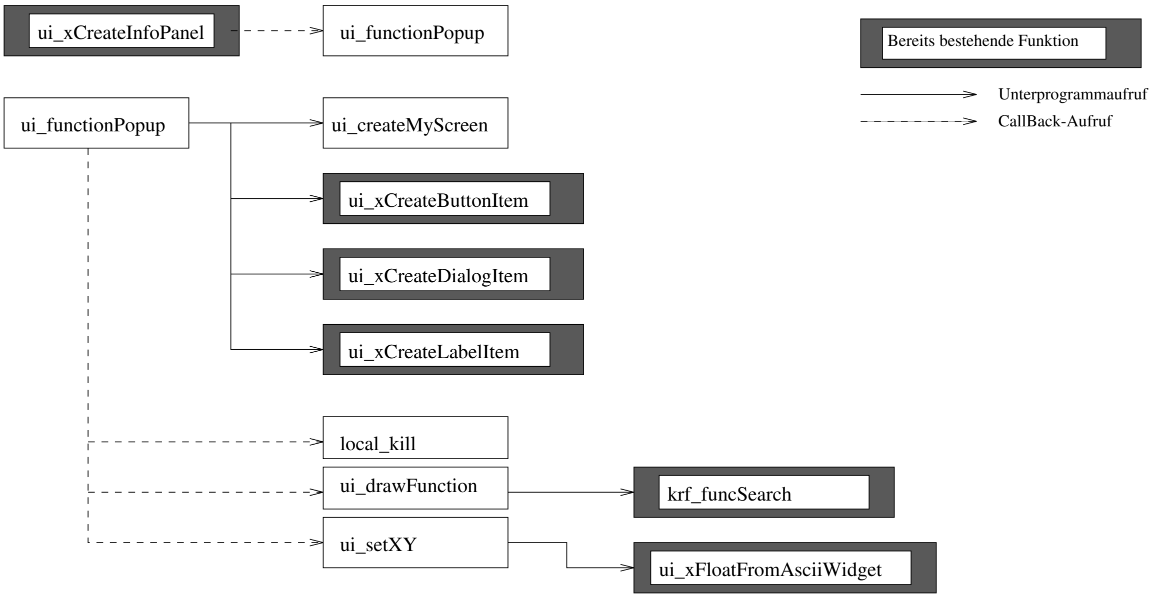

Abbildung 5.2: Aufrufstruktur für das Modul ui_funcdispl.c

Um das Modul zur Visualisierung der Gewichte aufzurufen, klickt man den Menüpunkt "Weights" am **XGUI** Button des Info–Panels.

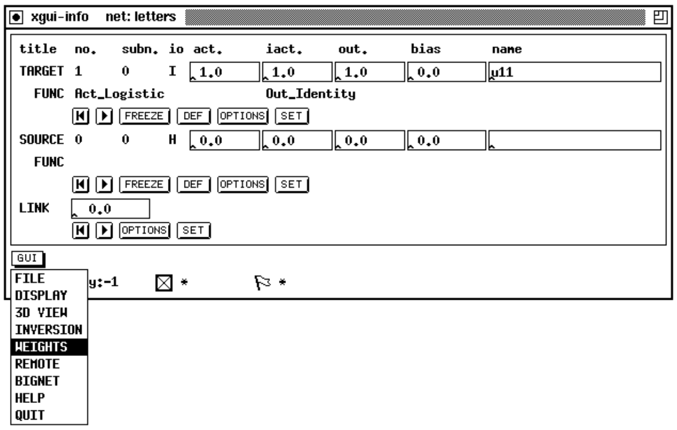

Abbildung 5.3: Aufruf zur Darstellung der Gewichte des Netzes

Die zugehörige interne Aufrufstruktur ist in Abbildung 5.4 dargestellt.

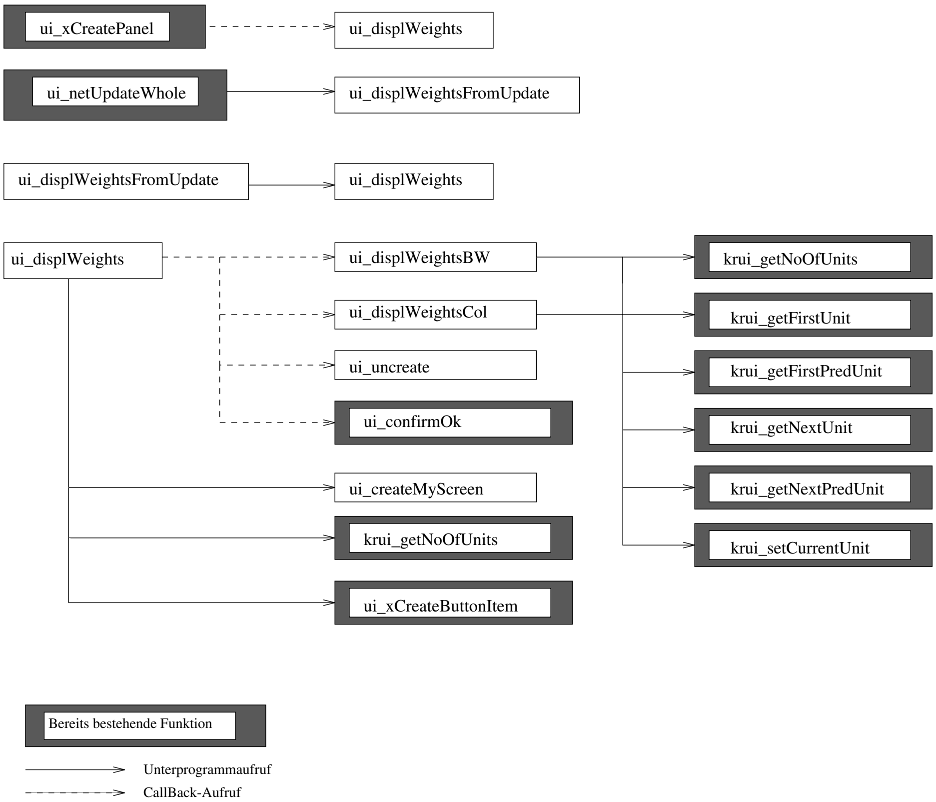

Abbildung 5.4: Aufrufstruktur für das Modul ui_displweights.c

Wie auch bei den anderen Darstellungen der internen Aufrufstrukturen repräsentieren hier durchgezogene Pfeile die Unterprogrammaufrufe. Gestrichelte Pfeile stellen Aufrufe über Callbacks oder Eventhandler dar. Dunkel umrandete Kästchen sind Files die bereits implementiert waren und nicht im Rahmen dieser Diplomarbeit erstellt worden sind.

Für das Modul zur Netzinversion gilt analog:

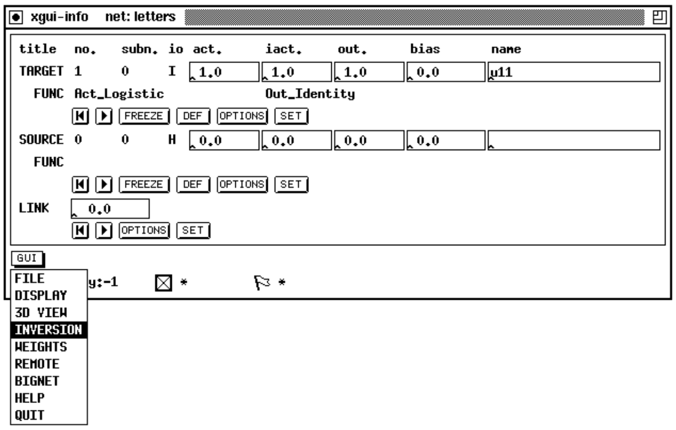

Abbildung 5.5: Aufruf zur Inversion des Netzes

mit folgender Aufrufstruktur:

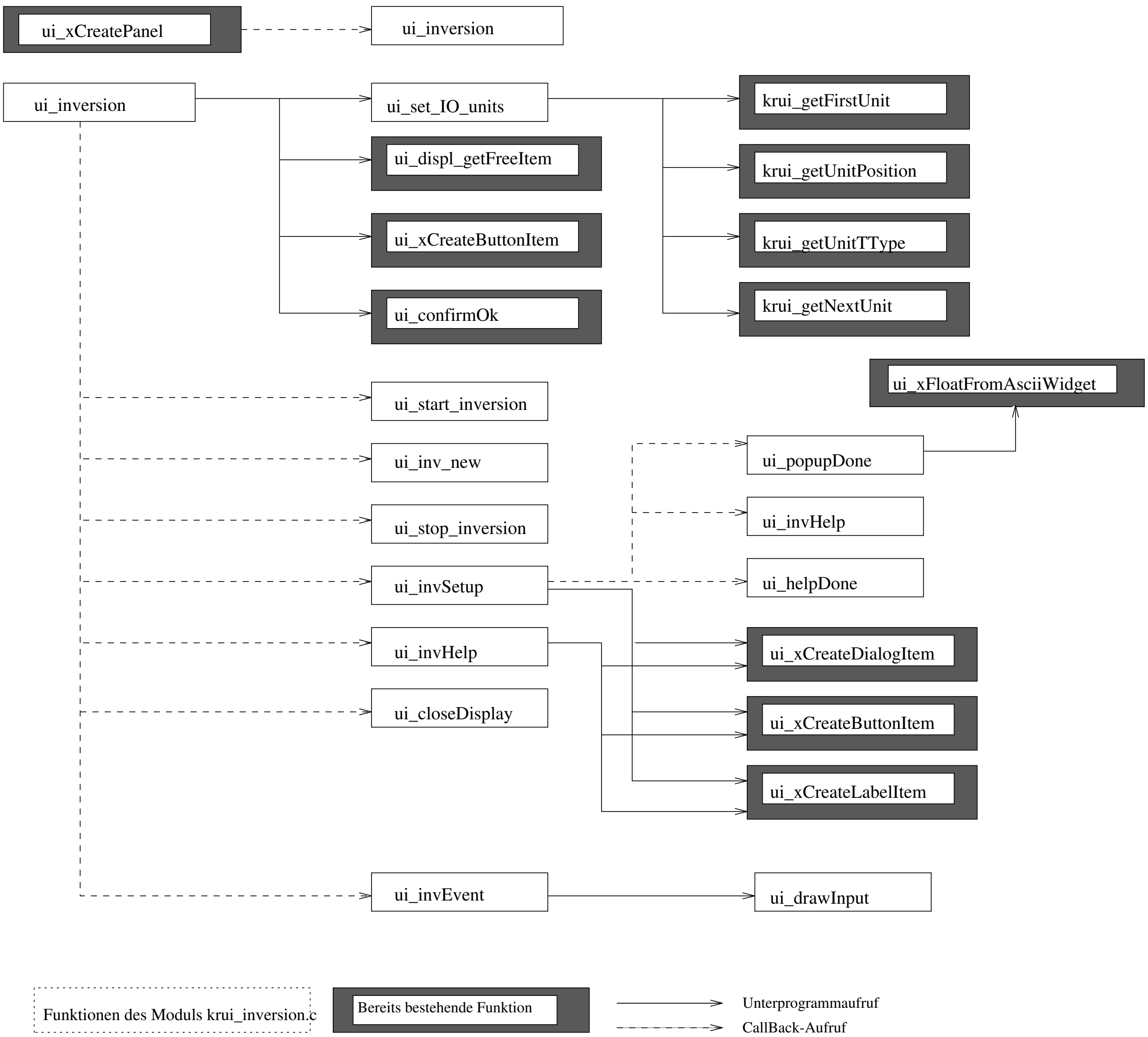

Abbildung 5.6: Aufrufstruktur für die Module ui_inversion.c und krui_inversion.c Teil I

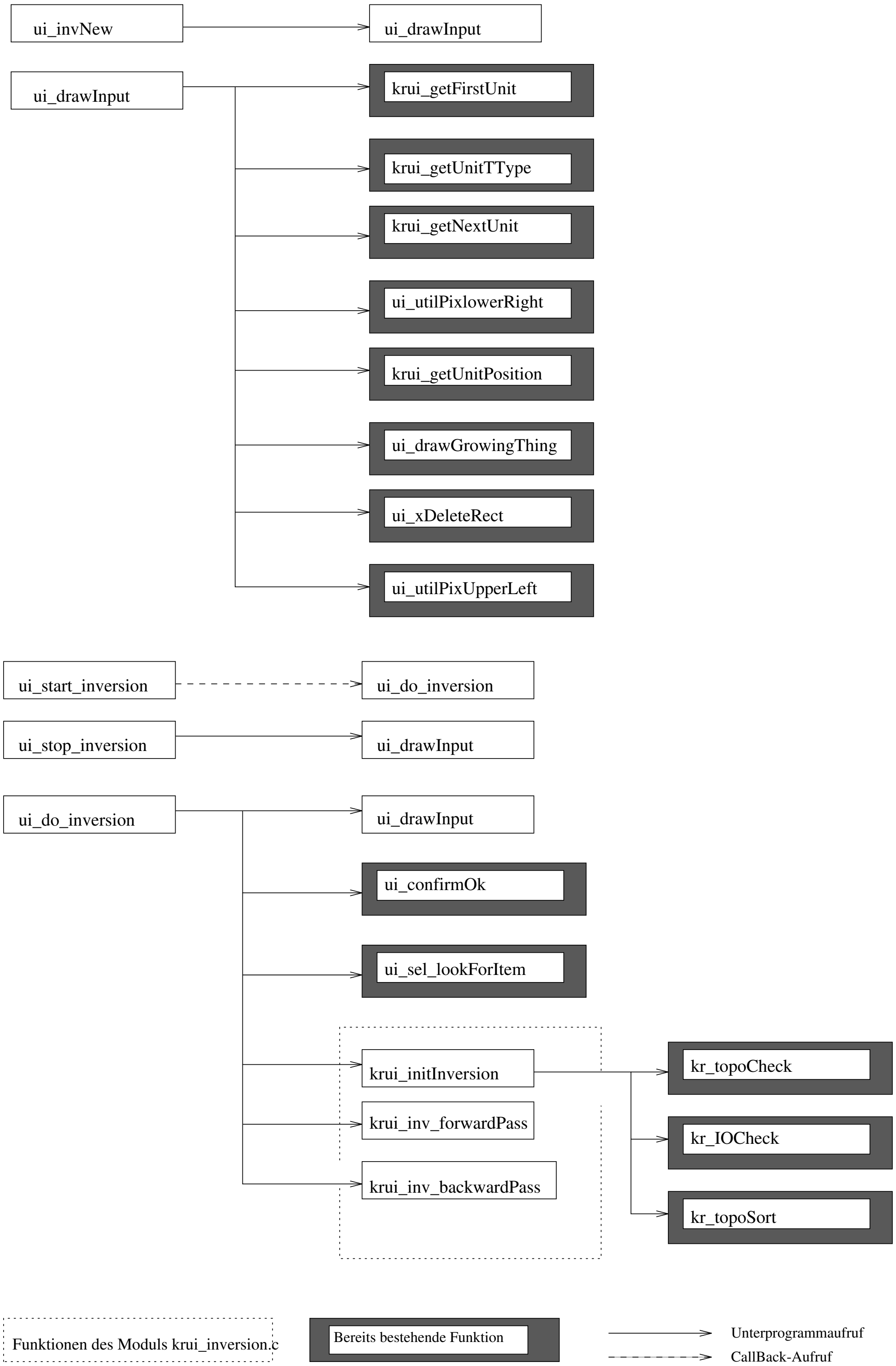

Abbildung 5.7: Aufrufstruktur für die Module ui_inversion.c und krui_inversion.c Teil II

5.4 Globale Variablen und Datenstrukturen

Window FUNC_GRAPH_window[6]
Die Fenster für die Darstellung von Unitfunktionen. Die Felder repräsentieren im einzelnen:

FUNC_GRAPH_window[1]	Fenster für Target Unit Ausgabefunktion
FUNC_GRAPH_window[2]	Fenster für Target Unit Aktivierungsfunktion
FUNC_GRAPH_window[4]	Fenster für Source Unit Ausgabefunktion
FUNC_GRAPH_window[5]	Fenster für Source Unit Aktivierungsfunktion

Das Feld 3 war für die Darstellung der Sitefunktion vorgesehen und wird z.Z. nicht verwendet.

Window WEIGHT_GRAPH_win
Das Fenster für die Darstellung der Gewichte.

GC FUNC_GRAPH_gc[6]
Die zu den FUNC_GRAPH_window–Fenstern gehörenden Graphikkontexte.

GC WEIGHT_GRAPH_gc[21]
Die zum WEIGHT_GRAPH_win–Fenster gehörenden Graphikkontexte. Bei Farbbildschirmen ist in jedem GC eine andere Vordergrundfarbe definiert. Die verschiedenen Gewichte werden alle mit dem gleichen Befehl als ausgefüllte Rechtecke gezeichnet, und der jeweils für das Gewicht passende Kontext (Vordergrundfarbe) verwendet. Die verschiedenen Farben werden in WEIGHT_GRAPH_color_def[21] gehalten. Bei Schwarz–Weiß–Bildschirmen wird nur das erste Feld belegt.

int FUNC_GRAPH_screen
Screen–Variable zur Darstellung der Unit–Funktionen.

int WEIGHT_GRAPH_screen
Screen–Variable zur Darstellung der Gewichte.

int INV_screen
Screen–Variable für die Netzinversion.

Display *INV_display
Display Struktur für die Netzinversion.

XColor WEIGHT_GRAPH_color_def[21]
Farbdefinition für die Gewichte. Die Farbverteilung ist wie folgt:

WEIGHT_GRAPH_color_def[0]	sehr dunkles rot
⋮	
WEIGHT_GRAPH_color_def[9]	sehr helles rot
WEIGHT_GRAPH_color_def[10]	weiss
WEIGHT_GRAPH_color_def[11]	sehr helles grün
⋮	
WEIGHT_GRAPH_color_def[19]	sehr dunkles grün
WEIGHT_GRAPH_color_def[20]	schwarz zur Beschriftung

Diese Verteilung gilt natürlich nur für Farbbildschirme. Bei Schwarz–Weiß–Darstellung ist das erste Feld mit schwarz, das zweite mit weiß belegt und der Rest unbelegt.

int FUNC_GRAPH_created[6]
Ein Flag, das anzeigt, welches Fenster für die Darstellung von Unit–Funktionen bereits einmal geöffnet war. Ist ein Feld nicht gesetzt, dann wurde das entsprechende Fenster noch nicht geöffnet bzw. bereits wieder geschlossen.

int WEIGHTS_WINDOW_created
Ein Flag das anzeigt, daß das Fenster für die Darstellung der Gewichte geöffnet ist, d.h. bei einem Update berücksichtigt werden muß.

int FUNC_GRAPH_case
Variable zum Indizieren des Arrays FUNC_GRAPH_created[6].

int FUNC_GRAPH_win_ex[6]
Dieses Flag zeigt an, welches Fenster für die Darstellung von Unit–Funktionen **zur Zeit** geöffnet ist, d.h. bei einem Update berücksichtigt werden muß.

Widget FUNC_GRAPH_x_minW[6], FUNC_GRAPH_x_maxW[6],
Widget FUNC_GRAPH_y_minW[6], FUNC_GRAPH_y_maxW[6]
Hier handelt es sich um Dialog–Widgets, über die der Darstellungsbereich der Unit–Funktionen eingelesen wird.

Widget INV_inPatW, INV_etaW, INV_delta_maxW, INV_ratioW
Dialog–Widgets, über die die notwendigen Variablen zur Netzinversion gelesen werden.

Widget WEIGHT_GRAPH_displaywidget
Widget WEIGHT_GRAPH_mainwidget
Das Root– und das Zeichen–Widget für die Darstellung der Gewichte.

float FUNC_GRAPH_x_min[6], FUNC_GRAPH_x_max[6],
float FUNC_GRAPH_y_min[6], FUNC_GRAPH_y_max[6]
Die Werte, die die von den zugehörigen Widgets mit gleichem Namenstamm eingelesenen Darstellungsgrenzen speichern.

float INV_input_pat, INV_ratio, INV_delta_max, INV_eta
Die Werte, die die von den zugehörigen Widgets mit gleichem Namenstamm eingelesenen Inversionsparameter speichern.

int WEIGHT_squaresize, WEIGHT_windowsize, WEIGHT_viewsize
Diese Variablen halten die aktuelle Normgröße der Units, die Größe des Fensters, und die Größe des Viewports für die Darstellung der Gewichte fest.

```
typedef struct INV_UnitList{
    int no;
    FlintType act;
    FlintType i_act;
    FlintType im_act;
    struct PosType gridPos;
    struct INV_UnitList *next;
    struct INV_UnitList *prev;
};
struct INV_UnitList *INV_inputs,*INV_outputs;
```

UnitList ist eine Struktur zur Verwaltung der an der Netzinversion beteiligten Units, d.h. Ein– und Ausgabe–Units. *INV_inputs zeigt dabei auf die erste Eingabe–Unit einer doppelt verketteten Liste, *INV_output auf die erste Ausgabe–Unit einer ebenfalls doppelt verketteten Liste. Die einzelnen Felder haben die Bedeutung:

no	Nummer der Unit
act	Tatsächliche Aktivierung der Unit
i_act	Anfangsaktivierung der Unit
im_act	Pseudo–Aktivierung der Unit
gridPos	Position in Netzkoordinaten
next	Nachfolger in der Liste
prev	Vorgänger in der Liste

int INV_created, INV_new, INV_running,
int INV_cycle, INV_units
Eine Reihe von Zustandsflags und –variablen, die den momentanen Zustand der Netzinversion beschreiben.

INV_created	1, wenn das Inversionsfenster geöffnet ist
INV_new	1, wenn die Inversion neu gestartet werden soll
INV_running	1, solange die Inversion läuft
INV_cycle	Anzahl der bereits durchlaufenen Zyklen
INV_units	Anzahl der Ausgabe–Units, die noch einen unzulässig großen Fehler aufweisen.

Kapitel 6

Zusammenfassung

6.1 Die Visualisierungstools

Der Stuttgarter Neuronale Netze Simulator SNNS ist ein leistungsfähiger Simulator zur Erforschung konnektionistischer Systeme. Im Rahmen dieser Diplomarbeit wurden Hilfsmittel für diesen Simulator entwickelt, die es ermöglichen, den inneren Zustand des Netzes graphisch darzustellen. Alle Bestandteile des Netzes, sowohl Units und Links, als auch das Netz an sich können jetzt während des Lernvorgangs beobachtet werden. Damit kann jetzt besser untersucht werden, warum sich ein Netz so und nicht anders verhält, wenn es eine Eingabe erhält.

Zur Beobachtung einer Unit stehen zwei Displays zur Verfügung. Je eines zur Darstellung der Aktivierungs– und der Ausgabefunktion.

Zur Darstellung der Verbindungsgewichte wurde ein weiteres Display entwickelt. Hier werden für Schwarz–Weiß– und Farbbildschirme automatisch die passenden Diagramme, Hinton– oder WV–Diagramme, generiert.

Die implementierte Inversionsmethode bietet zuletzt die Möglichkeit, die Strukturen des Netzwerks als Ganzes zu betrachten und Hinweise auf die Abhängigkeit der Ausgabe von bestimmten Eingangsgrößen zu erhalten.

6.2 Aufgetretene Probleme

Unerwartet schwierige Probleme gab es bei der Darstellung der Unit–Funktionen. Die Art der Einbindung der Site– und Aktivierungsfunktionen in den Kern lies einen einfachen Aufruf mit einem Eingangswert als Argument und Rückgabe einer Aktivierung nicht zu. Da eine vollständige Änderung des Kerns für die Visualisierung nicht in Frage kam, wurde auf die Darstellung der Site–Funktionen verzichtet.

Die Darstellung der Aktivierungsfunktionen konnte durch Festhalten aller beteiligten Verbindungsgewichte doch noch realisert werden. Dazu werden nun die Gewichte aller Eingangsverbindungen einer bestimmten Unit in einer verketteten Liste gespeichert. Die Gewichte werden dann verändert und die Unit der Aktivierungsfunktion als Parameter übergeben. Nach dem Zeichnen werden die Verbindungen dann wieder mit den gespeicherten Gewichten versehen, so daß keine dauerhafte Änderung des Netzes vorgenommen wird.

Auch die Implementierung der Inversionsmethode gestaltete sich schwierig. Die Verwendung von Strukturen des Kerns mußte so durchgeführt werden, daß eine Änderung der definierten Muster nicht stattfand. Eine spätere Verwendung von ihnen, zum Beispiel bei einem erneuten Training, hätte sonst das Systemverhalten geändert. Dieses Ziel wurde durch ein Kopieren aller Informationen der Units in eine verkettete Liste erreicht.

6.3 Ausblick

Ein weiterer Ausbau von Visualisierungstools erscheint möglich und wünschenswert. So ist eine Ausgabe der Fehlersignale als Funktionsgraph denkbar, die den Lernerfolg des Netzes verdeutlichen würde. Eine weitere Möglichkeit ist die Ausgabe der Vorgänger– und Nachfolger–Units der aktuellen Unit nicht nur alphanumerisch, sondern auch graphisch in einem Display. Auch ein weiterer Ausbau der hier entwickelten Module ist geplant. So soll der Benutzer bei der Darstellung der Netzwerksgewichte die Möglichkeit erhalten die Dichte des gezeichneten Gitters und die maximale Größe der Quadrate selbst festzulegen. Auch soll ein Anklicken der Quadrate in diesem Display ein Popup Window öffnen, welches Source und Target Unit der Verbindung sowie ihr exaktes Gewicht darstellt. Eine Weiterentwicklung des Inversionsalgorithmus ist ebenfalls geplant. Diese soll nicht nur eine einheitliche Erstaktivierung der Eingabezellen erlauben, sondern auch das Setzen eines Initialmusters in der Eingabeschicht ermöglichen.

Literaturverzeichnis

[Anders90] K.H. Anders. *Backpropagation.* Hauptseminar am Institut für Informatik, Universität Stuttgart, 1990.

[Banaha84] M. Banaham, A.Rutter. *UNIX.* Hanser Verlag 1984.

[BisPin91] H. Bischof, A. Pinz. *Visualization Methods for Neural Networks.* in: H. Ziegeler (ed.) *Konnektionismus: Beiträge aus Theorie und Praxis.* Schriftenreihe der Österreichischen Gesellschaft für Artifical Intelligence, Bd 3, pp. 1-15, 1991.

[BisPin90] H. Bischof, A. Pinz. *Verwendung von neuronalen Netzwerken zur Klassifikation natürlicher Objekte am Beispiel der Baumerkennung aus Farb-Infrarot-Luftbildern.* in: Dorffner (ed.) *Konnektionismus in Artificial Intelligence und Kognitionsforschung.* Informatik Fachberichte, Springer Verlag pp. 112-120, 1990.

[HechtN90] R. Hecht-Nielsen. *Neurocomputing.* Addison-Wesley 1990.

[Hinton89] G. Hinton. *Connectionist Learning Procedures.* Artificial Intelligence, Vol. 40, pp. 185-234, 1989.

[Hinton90] G. Hinton. *Deterministic Boltzman Learning in Networks with asymetric Connectivity.* Proceedings of the 1990 Summer School pp. 3-9, 1990.

[Huebne92] R. Hübner. *3D–Visualisierung der Topologie und der Aktivität neuronaler Netze.* Diplomarbeit 846, IPVR, Universität Stuttgart, 1992.

[Jones89] O. Jones. *Introduction to the X Window System.* Prentice Hall, 1989.

[KerRit88] D. Kernighan, D. Ritchie. *The C Programming Language.* Prentice Hall, 1988.

[KinLin90] J. Kindermann, A. Linden. *Inversion of Neural Networks by Gradient Descent.* Parallel Computing, Vol. 14, pp 277-286, 1990.

[Korb89] T. Korb. *Entwurf und Implementierung einer deklarativen Sprache zur Beschreibung neuronaler Netze.* Studienarbeit 789, IPVR, Universität Stuttgart, 1989.

[KoZeBa90] T. Korb, A. Zell, R. Bayer. *A Neural Network Simulation Environment.* Applications of Neural Networks Conf., Proc. SPIE's 1990 Aerospace Sensing Intl. Symposium, Vol. No. 1294, 16-20, Orlando Florida pp. 535-544. 1990.

[Kopka91] H. Kopka. LaTeX *Eine Einführung.* Addison-Wesley 1991.

[LinThr90] A. Linden, S. Thrun. *Inversion in Time.* Neural Networks Eurasip Workshop Proceedings pp. 130-141, 1990.

[Linden90] A. Linden. *Untersuchung von Backpropagation in konnektionistischen Systemen.* Diplomarbeit im Fach Informatik, Rheinische Friedrich–Wilhelms–Universität Bonn, 1990.

[Mache90] N. Mache. *Entwurf und Realisierung eines effizienten Simulatorkerns für neuronale Netze.* Studienarbeit 895, IPVR, Universität Stuttgart, 1990.

[MIT85] MIT. *X Toolkit Intrinsics – C Language Interface.* Technical Report, MIT X Consortium, 1985. X Window System – X Version 11, Release 3.

[OReil90a] O'Reilly & Associates. *X Toolkit Intrinsic Programming Manual.* Prentice Hall, 1990.

[OReil90b] O'Reilly & Associates. *X Toolkit Intrinsic Reference Manual.* Prentice Hall, 1990.

[RumMcC] D. Rummelhart, J. McClelland. *Parallel Distributed Processing.* Volume 1, MIT Press, 1986.

[Peters89] C. Peterson. *Athena Widget Set – C Language Interface.* Technical Report, MIT X Consortium, 1989. X Windows System - X Version 11, Release 4.

[PinBis90] A. Pinz, H. Bischof. *The IVF Weight-Visualization Diagramms (WV-Diagramms).* IVF report (Institute of surveying and remote sensing Univ. of Bodenkultur, Vienna, 1990.

[Sommer89] T. Sommer. *Entwurf und Realisierung einer graphischen Benutzeroberfläche für einen Simulator konnektionistischer Netzwerke.* Studienarbeit 746, Institut für Informatik, Universität Stuttgart 1989.

[Zell91] A. Zell. *Neuronale Netze.* Vorlesungs Skriptum im Sommersemester 1991 IPVR, Fakultät Informatik, Universität Stuttgart 1991

[ZellEA91a] A. Zell, T. Korb, N. Mache, T. Sommer. *SNNS Benutzerhandbuch.* Bericht Nr. 1/91 IPVR, Fakultät Informatik, Universität Stuttgart 1991.

[ZellEA91b] A. Zell, T. Korb, N. Mache, T. Sommer. *Nessus-Handbuch.* Bericht Nr. 3/91 IPVR, Fakultät Informatik, Universität Stuttgart 1991.

[ZellEA90] A. Zell, N. Mache, T. Sommer. *Recent Developments of the SNNS Neuronal Network Simulator.* Applications of Neural Networks Conf., Proc. SPIE's 1990 Aerospace Sensing Intl. Symposium, Vol. No. 1294, 16-20, Orlando Florida pp. 535-544, 1990.

[ZellEA91c] A. Zell, N. Mache, T. Sommer, T. Korb. *The SNNS Neural Network Simulator.* GWAI-91, 15. Fachtagung für künstliche Intelligenz, Bonn, Informatik-Fachberichte 285, pp. 254-263, Springer Verlag, 1991.

[ZellEA91d] A. Zell, N. Mache, T. Korb. *The SNNS Neural Network Simulator*. 13. DAGM Symposium, München, Informatik-Fachberichte 290, pp. 454-461, Springer Verlag, 1991.