Technische Universität Ilmenau

Fakultät für Informatik und Automatisierung

Institut für Biomedizinische Technik und Medizinische Informatik

Studienjahresarbeit

Implementierung eines Strukturoptimierungsalgorithmus' für Neuronale BP-Netze

eingereicht von: **Christian Heller**

Studiennummer: **20887**

Immatr.-Jahr: **1990**

Seminargruppe: **BMT-91**

Studienrichtung: **Biomedizinische Technik und Informatik**

Studiengang: **Elektrotechnik**

Betreuer: **Dipl.-Ing. D. Steuer**

verantwortlicher Hochschullehrer: **Prof. Dr. rer. nat. habil. G. Grießbach**

eingereicht am: **21.06.1996**

Vorwort

Die vorliegende Belegarbeit ist das Ergebnis einer mehrwöchigen Beschäftigung mit dem Aufgabengebiet ‘Neuronale Netzwerke’, welche die programmiertechnische Implementierung eines Algorithmus’ zur Strukturoptimierung in einen gegebenen Netzwerksimulator zum Ziel hatte.

Dazu war, neben der oben genannten, auch eine Einarbeitung in die Grundlagen der Programmierung mit dem Entwicklungssystem ‘Delphi 1.0’ von BORLAND vonnöten.

Der erwähnte Strukturoptimierungsalgorithmus war im Ergebnis einer kurz zuvor abgeschlossenen Diplomarbeit als vorteilhaft befunden und somit zur Integration in das Simulationsprogramm vorgeschlagen worden.

Da davon ausgegangen wird, daß sich der für Strukturoptimierung interessierende Leser mit Neuronalen Netzen bereits auskennt, soll hier nur kurz auf deren Grundlagen eingegangen werden. Größeres Augenmerk wird auf die Darstellung der recherchierten Strukturoptimierungsverfahren gelegt. Ein weiterer Abschnitt verdeutlicht die programmiertechnische Tätigkeit zur Implementierung. Der Funktionsnachweis erfolgte anhand zweier Beispiele, die im letzten Kapitel beschrieben werden.

Inhaltsverzeichnis

1 Neuronale Netze - Sinn, Funktion und Arten

Die Mitte der achtziger Jahre zu neuem Leben erweckten ‘Künstlichen Neuronalen Netze’ (KNN) sind, trotz der mittlerweile klar erkennbaren Beschränkungen, noch lange nicht am Ende ihrer Entwicklung angelangt. Zu ihren heute schon zahlreichen Anwendungsgebieten zählen laut [5] neben allgemeineren wie der Spracherkennung, Steuerung autonomer Fahrzeuge und Wechselkursprognosen vor allem auch solche in der Medizin und Biologie, z.B. die Klassifizierung von Molekülen in ähnlich reagierende Gruppen.

Das ursprünglich verfolgte Ziel der Imitation des menschlichen Gehirns mit seinen 10^{11} Neuronen ist jedoch, nicht zuletzt durch die immer noch unzureichenden Hardwaremöglichkeiten, bis auf unbestimmte Zeit nicht zu erreichen. So beschränkt man sich also zunächst auf die Nachbildung bescheidenerer Nervenstrukturen, beispielsweise von Lebewesen wie Schnecken, Blutegeln oder Tintenfischen mit nur etwa 10000 Neuronen bzw. im Menschen selber auf die Bewältigung von Teilaufgaben wie z.B. die Unterstützung des Gehörs tauber Menschen.

Außerdem wird der Einsatz Künstlicher Neuronaler Netze für Aufgabenstellungen, die ein deterministisches Verhalten benötigen (etwa die Überwachung von Kenndaten in Kernkraftwerken), zunehmend in Frage gestellt. Da die Netze ausschließlich über Beispieldaten lernen, lassen sich über ihr Verhalten bei unbekannten Eingabemustern lediglich statistische Aussagen treffen.

Sinnvoll hingegen erscheint ihr Einsatz für beratende Tätigkeiten, bei denen allerdings stets der Mensch letzter Entscheidungsträger bleibt. So haben Versuche im klinischen Bereich gezeigt, daß Neuronale Netze bei der Diagnose von Krankheiten eine Entlastung für Ärzte darstellen können.

Die momentan beinahe unüberschaubare Zahl verschiedener Arten Neuronaler Netzwerke bringt sowohl Vor- als auch Nachteile mit sich. Zum einen gibt es mittlerweile für viele Probleme spezielle Lösungen; zum anderen wird es dadurch sehr schwer, den Überblick zu behalten. Die Einführung eines allgemeineren Netzwerkmodells könnte hier Abhilfe schaffen, indem die Handhabbarkeit erheblich verbessert würde.

Alle heute existenten Modelle gehen nach [3] zurück auf die Vereinfachungen von McCulloch und Pitts aus dem Jahre 1943. Sie betrachteten ein Neuron als eine Art Addierer mit Schwellwert: „Die Verbindungen (Synapsen) eines Neurons nehmen eine Aktivierung x_i mit einer bestimmten Stärke w_i von anderen Neuronen auf, summieren diese und lassen dann am Ausgang y (Axon) des Neurons eine Aktivität entstehen, sobald die Summe vorher einen Schwellwert T überschritten hat.“

$$z(\vec{w},\vec{x}) = \sum_j w_j x_j = \vec{w}^T \vec{x}$$ **Gleichung 1.1: Aktivitätsfunktion**

$$z(\vec{w},\vec{x}) = \vec{w}^T \vec{x} - T$$ **Gleichung 1.2: Minderung der Aktivität um den Schwellwert**

$$y = S(z)$$ **Gleichung 1.3: Ausgabefunktion**

$$y = f(\vec{x},\vec{w},z,S)$$ **Gleichung 1.4: Transferfunktion =Zusammenfassung der gesamten Reaktion des formalen Neurons als Ergebnis nur einer Funktion**

Diese einfachen Elemente können parallel arbeiten und so ein 'Netzwerk' bilden, dessen Funktion größtenteils durch die Verbindungen zwischen den Neuronen bestimmt wird [8]. Also kann man sagen, daß sowohl Neuronen-Modell als auch die Architektur eines Neuronalen Netzwerkes beschreiben, auf welche Art und Weise dieses Netzwerk seine Eingangs- in Ausgangsgrößen transformiert. Mit anderen Worten erlangt das Netzwerk eine Lösung, indem es seine Eingangs- und Ausgangswerte einander zuordnet, wobei die jeweils zu lösende Zuordnungsaufgabe die Anzahl der Netzwerkein- wie auch -ausgänge bestimmt.

Zu den neuronalen Modellen, die, aufbauend auf der biokybernetischen Überlegung von McCulloch und Pitts, in den letzten Jahrzehnten entwickelt wurden, zählen das 'Perceptron', das 'ADALINE' (Adaptive Linear Element), Assoziative Lernregeln wie 'Hebb'-, 'Instar'- und 'Outstar-Regel', 'Competitive'-, 'Feature Map'- und 'ART' (Adaptive Resonance Theory)- Netzwerke. Für nähergehende Erläuterungen sei auf [8] u.a. Quellen verwiesen. Ein Überblick ist im Anhang A zu ersehen.

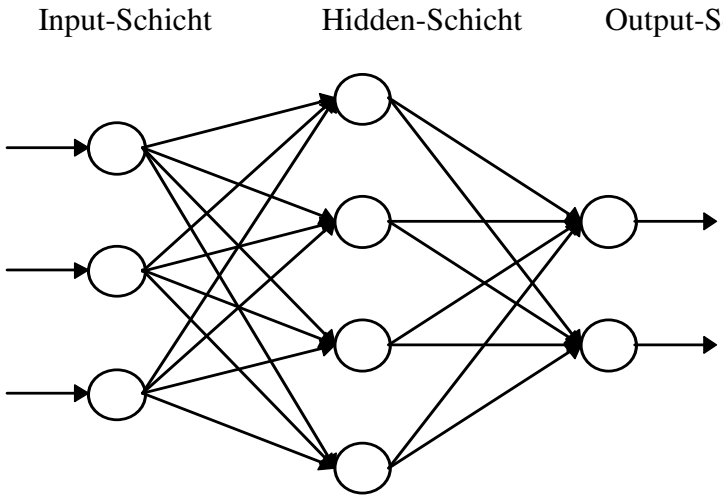

Abbildung 1.1: Struktur eines Backpropagation-Netzes

Der für die vorliegende Arbeit relevante Algorithmus nennt sich ‘Backpropagation’ (BP) und entstand durch die Verallgemeinerung der ‘Widrow-Hoff-Lernregel’ (ADALINE) auf vielschichtige Netzwerke und nichtlineare, differenzierbare Übertragungsfunktionen. Er wird in ca. 80 bis 90 % aller praktischen Applikationen verwandt und kann ‘Multilayered Feedforward Networks’ mit differenzierbaren Übertragungsfunktionen trainieren, um erstens Funktionen zu approximieren, zweitens spezifische Ein- und Ausgangsvektoren zu assoziieren oder drittens Eingangsvektoren in einer gegebenen Form zu klassifizieren.

Die Lernregel wird dazu genutzt, die Gewichte und den Schwellwert (Bias) der Neuronen so einzustellen, daß die quadratische Fehlersumme des Netzwerkes minimiert wird (Prozedur mit abfallendem Gradienten). Dazu wird nach dem Durchlaufen (Propagation) des Eingabesignals $\vec{x}$ durch die Netzschichten der Fehler $\delta = \vec{y}(\vec{x}) - \vec{L}(\vec{x})$ des Ausgangssignals $\vec{y}$ bezüglich der gewünschten Ausgabe $\vec{L}$ durch alle Schichten zurückgeführt (‘Error Back-Propagation’). Die darauf folgende Gewichtsänderung errechnet sich nach der ‘Delta-Regel’: $\Delta w_{ij}(\vec{x}) = \gamma \delta_i x_j$, wobei γ die konstante Lernrate darstellt.

Trainierte Backpropagation-Netzwerke tendieren dazu, richtige Antworten zu geben, auch wenn sie mit Eingängen gespeist werden, welche ihnen völlig unbekannt sind. Normalerweise antwortet das Netzwerk in einem solchen Fall mit einem Ausgangswert, dessen zugehöriger (vorher erlernter) Eingang dem gerade präsentierten, unbekannten Eingangswert am ähnlichsten ist. Diese Eigenschaft der Verallgemeinerung macht es möglich, ein Netzwerk auf repräsentative Eingangs-/Zielpaare zu trainieren und gute Ergebnisse für neue Eingangswerte zu erhalten, ohne das Netzwerk für diese unbekannten Eingangs-/Ausgangspaare neu trainieren zu müssen.

Das Backpropagation-Training kann zu einem lokalen, statt globalen Minimum führen. Das gefundene lokale Minimum kann durchaus zufriedenstellend sein- wenn nicht, so könnte ein Netzwerk mit mehr Schichten und Neuronen besser arbeiten. Doch selbst dann ist die Zahl der notwendigen Neuronen oder Schichten nur schwer zu bestimmen.

Die Architektur eines Backpropagation-Netzwerkes ist nicht vollständig abhängig vom zu lösenden Problem. Lediglich die Anzahl der Netzwerkeingänge und die Zahl der Neuronen in der Ausgangsschicht werden vom Problem bestimmt. Wie viele und wie groß die Schichten zwischen Netzwerkein- und -ausgängen sind, hängt allein vom Designer ab.

Es ist erwiesen, daß ein zweischichtiges, sigmoid-lineares Netzwerk jede funktionelle Beziehung zwischen Ein- und Ausgang darstellen kann, insofern die sigmoide Schicht genügend Neuronen hat. Je mehr Neuronen sich in zwischengelagerten Schichten befinden, desto mehr Freiheit hat das Netzwerk (z.B. eine größere Zahl zu optimierender Variablen). Die zusätzlichen Neuronen vergrößern die Chance, daß selbst ein lokales Minimum nur einen kleinen Fehler liefern wird.

Implementiert man ein sogenanntes ‘Momentum’ in das Backpropagation-Netzwerk, so verringert das die Wahrscheinlichkeit, daß ein Netzwerk in einem flachen, lokalen Minimum der Fehleroberfläche hängenbleibt und somit auch die Trainingszeiten. Ein zu großer Wert für die Lernrate resultiert in instabilem Lernen. Ein zu kleiner Wert verursacht zu lange Trainingszeiten. Eine adaptive Lernrate verringert die Trainingszeit, indem sie die Lernrate ausreichend hoch hält, während die Stabilität gewahrt bleibt [8].

2 Optimierungsverfahren

2.1 Sinn und Zweck

Seit erwiesen ist, daß für viele, mit Neuronalen Netzen lösbare Aufgaben eine optimale Struktur ermittelt werden kann- als typisches Beispiel sei hier auf das nicht linear separierbare 'XOR-Problem' mit zwei Schichten (ohne Eingang) verwiesen, gehen die Bestrebungen zunehmend dahin, diese Suche einer Idealstruktur zu automatisieren.

Doch was bedeutet der Begriff 'Optimale Struktur' in diesem Zusammenhang? Wie aus Kapitel 1 bekannt sein dürfte, kann durch die Erhöhung der Anzahl der Neuronen bzw. Schichten z.B. vermieden werden, daß ein Backpropagation-Netzwerk ein lokales, statt globales Minimum als Ergebnis liefert. Allerdings können sich die so erworbenen Vorteile schnell umkehren in einen Nachteil, genannt ‚Overfitting-Phänomen'. Dieses Überladen mit Neuronen führt letztendlich zu einem schlecht verallgemeinernden Netzwerk.

Deswegen verwendet man 'Pruning Methods' [*], um die Netzwerkstruktur zu vereinfachen und das Lernen zu beschleunigen. Dazu merken Ch. Jutten und O. Fambon [6] an:

> „While constructive methods try to build up a minimal model by adding successive parameters (or units), the pruning approach starts off with an already built model, and tries to extract its essence by pruning useless parameters."

2.2 Konstruktive Methoden

Wie in obigem Zitat genannt, versucht diese Gruppe von Methoden, eine minimale Netzkonfiguration zu erstellen, indem Neuronen sukzessive hinzuaddiert werden. Sie zählen nicht zu den eigentlichen Pruning-Methoden, da sie schon während des Konfigurierens aktiv sind. Als Vertreter wurden hier lediglich zwei Methoden aus [12] recherchiert.

Die 'Variation der Hidden-Units' nach Hirose erhöht die Anzahl der Neuronen, wenn der totale Fehler nach 100 Lernschritten nicht mehr als 1 % zum letzten Schritt gefallen ist. Dagegen werden, wenn das Netz konvergiert, Hidden-Elemente gelöscht und das Netz neu belehrt, bis schließlich keine Konvergenz mehr gegeben ist. Die letzte, funktionierende Konfiguration gilt dann als optimale Struktur des Netzes.

Das zweite Verfahren 'Divide and Conquer Network' (DCN) von S. Romanjuk könnte man frei übersetzen mit ‚Abspalten-und-Hinzunehmen' ...von Mustern/ Neuronen. Hierbei werden durch die Conquer-Phase ein erstes und innerhalb der Divide-Phase weitere Neuronen in ein-und-dieselbe Schicht eingefügt. Klassifizierte Muster werden währenddessen aus dem Set entfernt und die korrespondierenden Gewichte vor dem nächsten Anlernen des Netzes eingefroren, was zu einer schnelleren Konvergenz führt. Wird nach einem Lernschritt kein Muster klassifiziert, schaltet der Algorithmus wieder in die Conquer-Phase, welche ein neues Neuron in eine andere Schicht einfügt usw., bis schließlich alle Muster klassifiziert sind oder die maximale Anzahl von Zellen erreicht ist.

[*] Ihrer Bedeutung nach in etwa gleichzusetzen mit dem Begriff 'Beschneidungsmethoden'

2.3 Pruning

Man unterscheidet nach [6] mehrere Klassen von Pruning-Verfahren, z.B. jene während des Lernens und solche nach dem Lernen. Erstere nutzen ein spezielles Lernverfahren, um die Komplexität des Netzwerkes schon während des Lernprozesses zu reduzieren. Die zweite Klasse zielt ab auf die Entfernung unnützer Parameter, nachdem die Lernphase eine gewisse Konvergenz erreicht hat. Beide Arten werden als ‘indirekt’ bezeichnet, da sie lediglich die Verbindungen zwischen den Neuronen, d.h. deren Gewichte, verändern.

Das Entfernen der Neuronen selbst steht im Mittelpunkt der ‘Direkten Verfahren’. Zu ihnen zählt auch der Algorithmus nach Trautwein- hier ‘Weight and Power’ genannt-, welcher in Kapitel 2.3.2 näher behandelt werden wird.

Andere Gruppen von Pruning-Verfahren spielen (noch) eine eher untergeordnete Rolle und sollen in einem gemeinsamen Abschnitt nur kurz erwähnt werden.

Für einen zusammenfassenden Überblick sei auf Anhang B verwiesen.

2.3.1 Indirektes Pruning

2.3.1.1 During-learning Pruning (on-line)

Der Ausdünnungsprozeß erfolgt im Fall des von G.E. Hinton beschriebenen ‘Weight Decay’/ ’Weight Elimination’ durch das Reduzieren und evtl. Löschen betragsmäßig großer Gewichte mittels eines zusätzlichen Terms in der Fehlerfunktion.

Demgegenüber werden beim Algorithmus ‘Backpropagation with Mortality’ nach P. Kerlirzin und F. Vallet die Gewichte so gesetzt, daß die durch sie verschlüsselte Information gleichmäßiger verteilt ist. Das geschieht durch die zufällige Auswahl nicht nur der Muster, sondern auch der Netzkonfiguration beim Anlernen und bewirkt, daß später beliebige Gewichte eliminiert werden können, ohne daß die Leistung nennenswert sänke.

2.3.1.2 Post-learning Pruning (off-line)

Eines der ersten Beschneidungsverfahren war das ‘Optimal Brain Damage’ (OBD) nach Le Cun, bei welchem die „Gewichte gelöscht werden, die beim Nullsetzen im Analyseprozeß (der auf der Taylorreihenentwicklung basiert) die geringste Änderung in der Fehlerfunktion hervorrufen“ [12].

Aus seiner Verfeinerung entwickelte Hassibi das Verfahren des ‘Optimal Brain Surgeon’ (OBS), wo zur Taylorreihenentwicklung, statt einer approximierten, die vollständige Hesse-Matrix verwendet wird und dadurch mehr Gewichte entfernt werden können.

Die ‘Statistical Stepwise Method’ ähnelt den oben genannten Verfahren hinsichtlich ihres Beschneidungskriteriums, wonach nur Gewichte eliminiert werden, die statistisch nicht signifikant sind. Außerdem gilt als Voraussetzung, daß das resultierende Netz, beurteilt nach ‘Akaike’s B Information Criterion’ (BIC), besser als das vorhergehende ist. Mit ‘statistisch nicht signifikant’ könnten z.B. die betragsmäßig kleinsten Gewichte gemeint sein.

2.3.2 Direktes Pruning (u.a. Algorithmus nach Trautwein)

Betrachtet man ein Multi-Layer-Perceptron als Anordnung von Schichten, die ein Set von Eingangsmustern auf Ausgangsmuster projizieren, wobei das erste Set von Eingangsmustern als Lernbasis gilt, so kann man nach [6] durch Betrachtung der Beziehungen zwischen Ein- und Ausgang folgende Strategien zum ‘Pruning inside the MLP’ aufstellen:

> „1. Eine Unit mit stabilem Ausgang für jedes Eingangsmuster kann gelöscht werden,
>
> 2. Wenn zwei Units zum Eingangs-Set identische oder invertierte Ausgänge haben, kann eine davon gelöscht werden,
>
> 3. Wenn unterschiedliche Muster zwischen Ein- und Ausgang gleich bleiben, kann die komplette Schicht gelöscht werden,
>
> 4. Units, die nicht essentiell für die Generierung von Ausgangsmustern sind, können gelöscht werden.“

Etwas anders verfährt man beim ‘Local LS’, vorgeschlagen von Pelillo und Fanelli. Hier wird der Einfluß einer Unit unterdrückt und die Ausgangsgewichte der verbleibenden Units der Schicht aktualisiert, um die internen Eingänge zur nächsten Schicht nicht zu verändern. Danach wird m.H. einer iterativen Prozedur ein Gleichungssystem gelöst, in dessen Ergebnis der Einfluß der verbliebenen Units minimiert wird. Eliminiert werden, nach einer vorgeschlagenen Faustregel, zuerst die Units mit dem geringsten Einfluß.

Als weiteres direktes Verfahren sei das der ‘Skelettierung’ genannt. Hier wird jedem Neuron ein Bewertungsfaktor (Differenz zwischen den Fehlern des Netzes mit und ohne das betreffende Neuron) zugeordnet, der Aussagen über den Einfluß des Neurons macht. Je nach festgelegter Relevanzgrenze können dann unwichtige Neuronen gelöscht werden.

Schließlich reiht sich hier auch das von M. Trautwein vorgeschlagene Verfahren zur Verknüpfung von Gewichten und Leistungen (‘Weight and Power’), nachzulesen in [12], ein. Im Verlauf seiner Untersuchungen stellte er fest, daß: „Neuronen mit kleinen Eingangsgewichten ebenfalls kleine Ausgangsgewichte besitzen“ und so „nach beiden Seiten wenig Einfluß haben. ... Die Inputseite der Hiddenschicht wird als bestimmende Seite definiert, da sie in Richtung des normalen Datenflusses zuerst durchlaufen wird und außerdem (i.d.R.) mehr Gewichtsverbindungen zur Hiddenschicht aufweist.“

Mit diesem Wissen kann man nun die Summe aller Eingangsgewichte zur jeweiligen Hiddenschicht bilden und ebenso die Summe der Leistungen der Neuronen dieser Schicht. Anschließend relativiert man die Einzelwerte für Gewicht und Leistung, indem man sie auf die Summen bezieht; so können die Beträge direkt verglichen werden.

Die relativierten Gewichtswerte summiert man erneut auf, diesmal jedoch getrennt für jedes Neuron. Multipliziert man nun die relativierte Leistung des Neurons, die zugleich ein Maß für dessen Aktivität ist, mit der Summe seiner Eingangsgewichte, so erhält man als Ergebnis eine Größe, die über die Relevanz der jeweiligen Unit gute Aussagen liefert.

Jetzt bildet man noch den Mittelwert über alle ‘Relevanzgrößen’ der Hiddenschicht und bezieht die Einzelwerte auf ihn. Ergebnis ist die Relevanz der einzelnen Neuronen als prozentualer Anteil der gesamten Hiddenschicht. Nach einer beliebig festzulegenden Prozentgrenze können nun Hidden-Units eliminiert werden.

2.3.3 Andere Verfahren

Für die innerhalb des Gebietes Künstliche Intelligenz (KI) schon früher entwickelten ‘Baumstrukturen’ existieren etliche, brauchbare Pruning-Methoden. Doch, obwohl die Strukturen leicht auf Neuronale Netze abzubilden sind, bereiten die Pruning-Prozeduren Probleme, da sie sich eben auf die Struktur (Klassifizierung) des jeweiligen Baumes beziehen und nicht für andere zu gebrauchen sind.

Ebenfalls erwähnt werden in [6] ‘Genetische Algorithmen’, die aber eher als konstruktive Methoden zum Designen einer Neuronalen Netzwerk-Struktur, als zum Pruning zu gebrauchen sind. Neben den geforderten, zu großen Rechenkapazitäten, scheinen sie außerdem (noch) nicht effizient genug zu arbeiten.

3 Implementierung

3.1 Der Netzwerk-Simulator

Das gegebene Programm ‘NEURONAL’ wurde ursprünglich unter ‘TURBO PASCAL for Windows’ entwickelt und später nach ‘DELPHI 1.0’ (beide von Borland) portiert. Es dient der Simulation Neuronaler Netzwerke mit bis zu 5 Hiddenschichten (ohne Ein- und Ausgangsschicht) und je 50 Neuronen.

Es wurden verschiedene Lernverfahren, als wichtigstes Backpropagation, implementiert. Netzkonfiguration und Lernparameter werden veranschaulicht durch zwei Fenster- ‚Netzstruktur’ und ‚Statistik’. Während des Anlernens des Netzwerkes mit vorher eingegebenen Mustern öffnen sich zwei weitere Fenster zur Visualisierung der Gewichtsbeträge und des Fehlerverlaufes. Weitere statistische Berechnungen sowie eine Datenbankanbindung sind zu finden unter dem Menüpunkt ‚Tools’.

Abbildung 3.1: Der Netzwerksimulator 'NEURONAL'

3.2 Menüpunkt ‘Strukturoptimierung’

Zur Implementierung des Strukturoptimierungsverfahrens wurde ein neuer Menüpunkt ‘Strukturoptimierung’ erstellt. In Anlehnung an die für Kapitel 2 erarbeitete Gliederung ist der neue Algorithmus im Untermenüpunkt ‘Direkt (Neuronen)’ zu finden. Hier könnten also auch etwaige, zukünftige Pruning-Verfahren in weitere Untermenüs eingefügt werden. Zusätzlich wurde ein Speed-Button zum Aufruf des Mustereditors ergänzt und ein neuer für den Strukturoptimierungsalgorithmus kreiert.

Bei der Anwahl öffnet sich ein Formular, auf welchem linkerhand die aktuelle Anzahl der Neuronen je Schicht zu ersehen ist. Die vier zur Auswahl stehenden Schalter sind- dank dem Hilfsmittel der ‚Hints’- selbsterklärend.

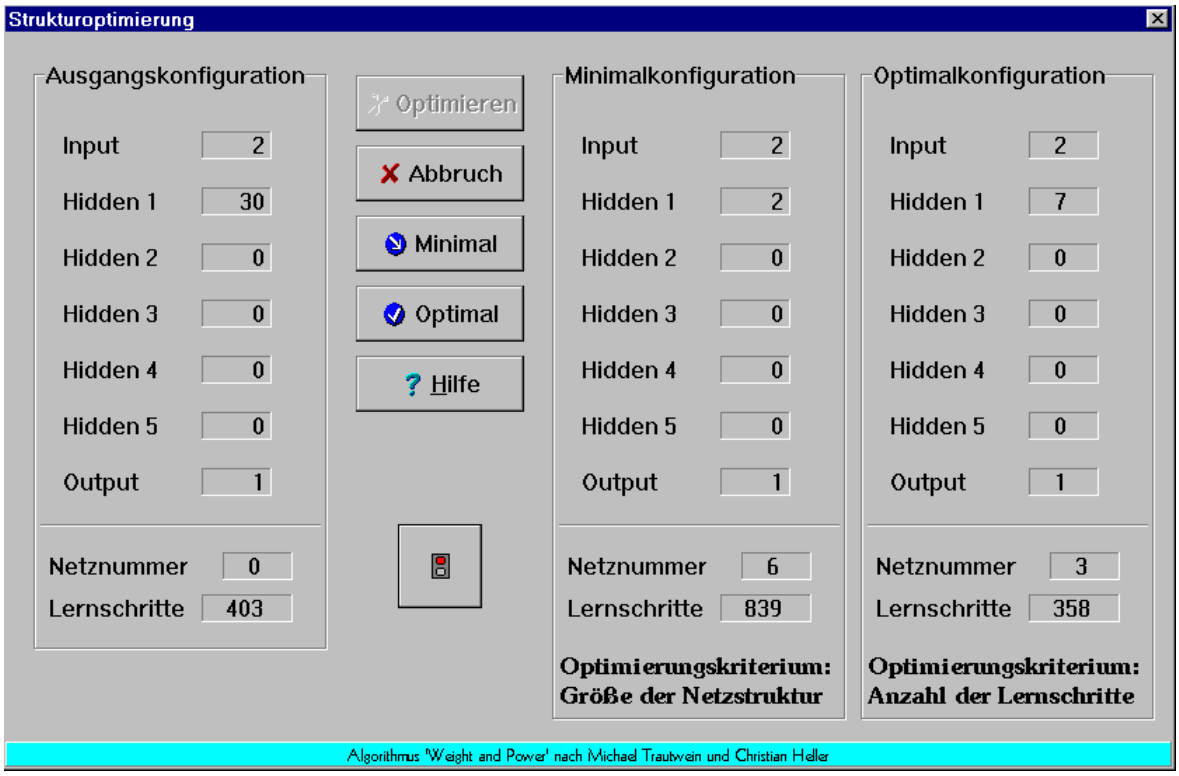

Abbildung 3.2: Formular 'Strukturoptimierung'

Leitet man eine Optimierung ein, so löscht der Algorithmus alle Neuronen geringer Relevanz, setzt die Gewichte zurück und belehrt das Netz, bis die Fehlergrenze unterschritten oder die zu Beginn eingestellte Lernschrittanzahl erreicht ist. Nun optimiert er wieder die entstandene Struktur und der Zyklus beginnt von neuem. Stellt der Algorithmus fest, daß sich die Struktur nach einem Optimierungsschritt nicht mehr ändert, so beendet er den Zyklus. Ausgegeben werden dann auf der rechten Seite zum einen die Minimalkonfiguration bzgl. der Netzstruktur und zum anderen die Konfiguration mit der geringsten Lernschrittanzahl.

Bei Betätigen des Schalters ‚Minimal' wird die Minimalkonfiguration beibehalten und steht für neue Anlernversuche zur Verfügung. Verworfen wird sie hingegen, wenn der Schalter ‚Abbruch' gewählt wird. Hierbei stellt das Netzwerk seine Ausgangskonfiguration von vor dem Aufruf der Optimierung wieder her. Mit dem Schalter 'Optimal' wurde eine weitere Optimierungsmöglichkeit geschaffen. Seine Anwahl bewirkt, daß aus allen optimierten Netzen dasjenige mit der geringsten Lernschrittanzahl errechnet und seine Struktur hergestellt wird, bevor sich das Formular schließt.

3.3 Programmiertechnische Gestaltung

Die Unit ‘ChStrOpt’ enthält im ‘Interface’-Teil, neben der üblichen Aufzählung der benutzten Units und der Definition der Klasse für das Formular, lediglich die Variable ‘StruktOpt’ vom Typ ‘TStruktOpt’, also eine Instanz der oben definierten Klasse.

In der ‘Uses’-Deklaration des ‘Implementation’-Teils steht, um zirkuläre Referenzen zu vermeiden, die benötigte Hauptunit ‘MtNetz’. Dann folgen die nur innerhalb der Unit ‘ChStrOpt’ geltenden Variablen und schließlich 12 Prozeduren, welche den Programmablauf bestimmen. Sechs von ihnen werden als Ereignis ‘OnClick’ beim Betätigen eines der sechs im Formular angeordneten Schalter ausgeführt. Die anderen sind lediglich Hilfsprozeduren. Auf alle soll im folgenden stichpunktartig eingegangen werden:

ErmAnzNeurJeSchicht: Dient der Ermittlung der aktuellen Neuronenanzahl je Netzwerkschicht.

TStruktOpt.FormCreate(Sender:TObject): Schreibt beim Erstellen des Formulars die aktuelle Neuronenanzahl je Schicht auf die Panels der linken Seite und aktiviert/ deaktiviert die Schalter.

SpeichCfg(nameFileCfg:string): Speichert die aktuelle Netzkonfiguration in temporären Files.

LadenCfg(nameFileCfg:string): Lädt die Konfiguration des durch den Namen ‘nameFileCfg’ angegebenen Files, nachdem die alte gelöscht wurde.

PiYield: Dient dem Abfangen und Umleiten der Botschaft für Abbruch während des Optimierens.

ProzedurAnlernen: Wurde modifiziert aus MtNetz übernommen; Prozedur zum Anlernen des Netzes.

TStruktOpt.BitBtn1Click(Sender:TObject): Eigentlicher Algorithmus zur Strukturoptimierung nach der unter 2.3.3 und 3.2 beschriebenen Vorgehensweise.

TStruktOpt.BitBtn4Click(Sender:TObject): Mit dem Schalter ‘Abbruch’ setzt man diese Prozedur in Gang, welche die Originalkonfiguration von vor dem Beginn des Optimierens wiederherstellt.

TStruktOpt.BitBtn5Click(Sender:TObject): Stellt die Konfiguration mit der kleinsten Netzwerkstruktur her und schließt das Formular. Wird aufgerufen durch den Schalter ‘Minimal’.

TStruktOpt.BitBtn2Click(Sender:TObject): Beim Betätigen des Schalters ‘Optimal’ wird die optimale Konfiguration (bzgl. der Anzahl der Lernschritte) hergestellt, nachdem die alte gelöscht wurde.

TStruktOpt.BitBtn3Click(Sender:TObject): Zum Einfügen einer Anwender-Hilfe gedacht.

TStruktOpt.SpeedButton1Click(Sender:TObject): Setzt die Variable ‘Abbruch’ auf den Wert ‘true’, um einen Abbruch des Optimiervorganges zu ermöglichen.

4 Beispiele

Als gegebenes Muster wurde, aufgrund des hohen Bekanntheitsgrades, das nicht linear separierbare ‘XOR-Problem’ * gewählt, für das eine Mindestkonfiguration von 2:2:1 gefordert wird. Tabellarisch dargestellt hat es folgende Form:

Eingang 1	Eingang 2	Ausgang
0	0	0
0	1	1
1	0	1
1	1	0

Tabelle 4.1: XOR-Problem

Da der Algorithmus so erweitert wurde, daß er nicht nur, wie in [12] vorgeschlagen, Konfigurationen mit einer, sondern auch mit bis zu fünf Hiddenschichten bearbeiten kann, sollen die folgenden Beispiele entsprechend eingeteilt werden.

4.1 Eine Hiddenschicht

Die Ergebnisse des ersten Versuches sind in der folgenden Tabelle dargestellt:

Nr.	Netzkonfig. minimal			Netzkonfig. optimal		
	Struktur	Netznr.	Schritte	Struktur	Netznr.	Schritte
1	2-2-1	5	689	2-19-1	1	365
2	2-2-1	5	585	2-19-1	1	344
3	2-2-1	6	1514	2-16-1	1	330
4	2-2-1	5	558	2-30-1	0	354
5	2-2-1	5	1073	2-17-1	1	349
6	2-2-1	4	657	2-30-1	0	346
7	2-2-1	6	14999	2-30-1	0	307
8	2-1-1	6	14999	2-30-1	0	305
9	2-1-1	6	14999	2-30-1	0	317
10	2-2-1	5	573	2-19-1	1	321
∅	2-1,8-1	-	5064,6	2-24-1	-	333,8

Tabelle 4.2: Optimierung mit einer Hiddenschicht

Es wurde eine Konfiguration mit zwei Input-, 30 Hidden- und einem Output-Neuron und folgenden Parametern untersucht:

* auch bekannt als ‘Antivalenz’ bzw. ‘Exklusives ODER’

Lernalgorithmus: Backpropagation
Lernrate: 0,70
Momentum: 0,90
Abbruchfehler: 0,01
Lernverfahren: epochal

4.2 Mehrere Hiddenschichten

Das zweite Beispiel hatte die folgende Struktur:

Inputschicht: 2 Neuronen
Hiddenschicht 1: 10 Neuronen
Hiddenschicht 2: 6 Neuronen
Hiddenschicht 3: 8 Neuronen
Outputschicht: 1 Neuron

Die gewählten Parameter lauten:

Lernalgorithmus: Backpropagation
Lernrate: 0,3
Momentum: 0,0
Abbruchfehler: 0,01
Lernverfahren: epochal
Adaptionskonstante: 1/n mit n=65 (variabler Wert zur adaptiven Glättung der Gewichtsänderung)

Die erhaltenen Ergebnisse sind in folgender Tabelle zusammengefaßt:

Nr.	Netzkonfig. minimal			Netzkonfig. optimal		
	Struktur	Netznr.	Schritte	Struktur	Netznr.	Schritte
1	2-1-1-1-1	7	14999	2-10-6-8-1	0	470
2	2-1-1-1-1	7	14999	2-10-6-8-1	0	489
3	2-1-1-1-1	6	14999	2-10-6-8-1	0	524
4	2-1-1-2-1	5	14999	2-10-6-8-1	0	632
5	2-1-1-2-1	4	14999	2-10-6-8-1	0	616
6	2-1-1-1-1	5	14999	2-10-6-8-1	0	621
7	2-1-1-2-1	6	14999	2-10-6-8-1	0	737
8	2-1-1-2-1	4	14999	2-10-6-8-1	0	655
9	2-1-1-2-1	5	14999	2-10-6-8-1	0	775
10	2-1-1-2-1	5	14999	2-10-6-8-1	0	980
∅	2-1-1-1,6-1	-	14999	2-10-6-8-1	-	649,9

Tabelle 4.3: Optimierung mit drei Hiddenschichten

4.3 Auswertung

Wie schon in [12] ausgeführt, konnte auch hier festgestellt werden, daß der Algorithmus ‘Weight and Power’ durch das direkte Entfernen von Neuronen (und nicht nur Gewichten) schnell arbeitet. Die Effizienz wird zusätzlich gesteigert durch das Entfernen mehrerer Neuronen in einem einzigen Optimierungsschritt, was durch die prozentuale Relevanzgrenze ermöglicht wird.

Hinsichtlich der optimierten Strukturen kann, betrachtet man das erste Beispielnetz, festgestellt werden, daß in 80% der Versuche die für das XOR-Problem als ausreichend definierte Struktur von 2:2:1 errechnet wurde. Nur zwei Durchläufe lieferten 2:1:1-Netze als Ergebnis, welche für das gegebene Problem zu klein waren und folglich auch keine Konvergenz fanden. Dieses Resultat bestätigt die sehr gute Urteilsfähigkeit des Strukturoptimierungsverfahrens, das durch die Verknüpfung der Eingangsgewichte mit der Leistung der Neuronen ein optimales Maß zur Bestimmung der Relevanz der Neuronen zur Anwendung bringt.

Obwohl der Algorithmus Netze mit bis zu fünf Hiddenschichten bearbeitet, könnte er noch verbessert werden insofern, als auch das Eliminieren der Schichten ermöglicht würde. Bisher endet die Optimierung an dem Punkt, wo eine minimale Neuronenanzahl in der Schicht erreicht ist. Das heißt, betrachtet man das zweite Beispiel, konkret, daß mit dem vorliegenden Algorithmus die optimale Konfiguration für z.B. das XOR-Problem (2:2:1) nicht erreicht werden kann, solange die Ausgangskonfiguration mehr als drei Schichten enthält.

Außerdem muß erwähnt werden, daß, neben der Größe der Struktur, auch die Nichtkonvergenz beschnittener Netze, welche bis zur vorher eingegebenen, maximalen Lernschrittanzahl durchlaufen, zu einer immensen Verlängerung der Optimierung führen kann. Dieser Geschwindigkeitsverlust durch die mit abnehmender Neuronenzahl zunehmende Unfähigkeit der Netze zum Konvergieren wird zum Teil dadurch ausgeglichen, daß die Berechnungen für kleinere Strukturen wesentlich schneller ablaufen.

Zu den Ergebnissen der nach der Lernschrittanzahl optimierten Netze muß gesagt werden, daß sie nicht befriedigen konnten. Fragestellung hierbei war, ob auch andere Kriterien als die minimale Struktur zur Auswahl eines optimalen Netzes geeignet sind. Während durch den Algorithmus ‘Weight and Power’ die minimalste Konfiguration gesucht wird, die gerade noch imstande ist, ein gegebenes Problem zu lösen, sollte m.H. der Lernschritte als Kriterium ein Netz mit möglichst geringer Anlernzeit gefunden werden. Als Optimalnetz ergab sich in beiden Beispielen entweder die Ausgangskonfiguration oder die erste optimierte Struktur, die aber nicht unbedingt die schnellsten waren, was die Dauer des Anlernvorganges betrifft. Es stellt sich die Frage, ob die Anzahl der Lernschritte als Kriterium ausreicht. Die tatsächlich benötigte Zeit zum Anlernen kann aus den Lernschritten als Maß nicht bestimmt werden; notwendig ist die Einbeziehung weiterer Größen, die den Rechenaufwand eines Netzes repräsentieren. In wie weit hier u.U. die simple Messung der benötigten Zeit weiterhelfen kann, bleibt zu untersuchen. Fest steht, daß von vornherein einige Probleme, wie z.B. unterschiedliche Resultate- je nach Rechnerauslastung-, zu erwarten sind.

5 Anhang

5.1 Neuronale Netzwerke

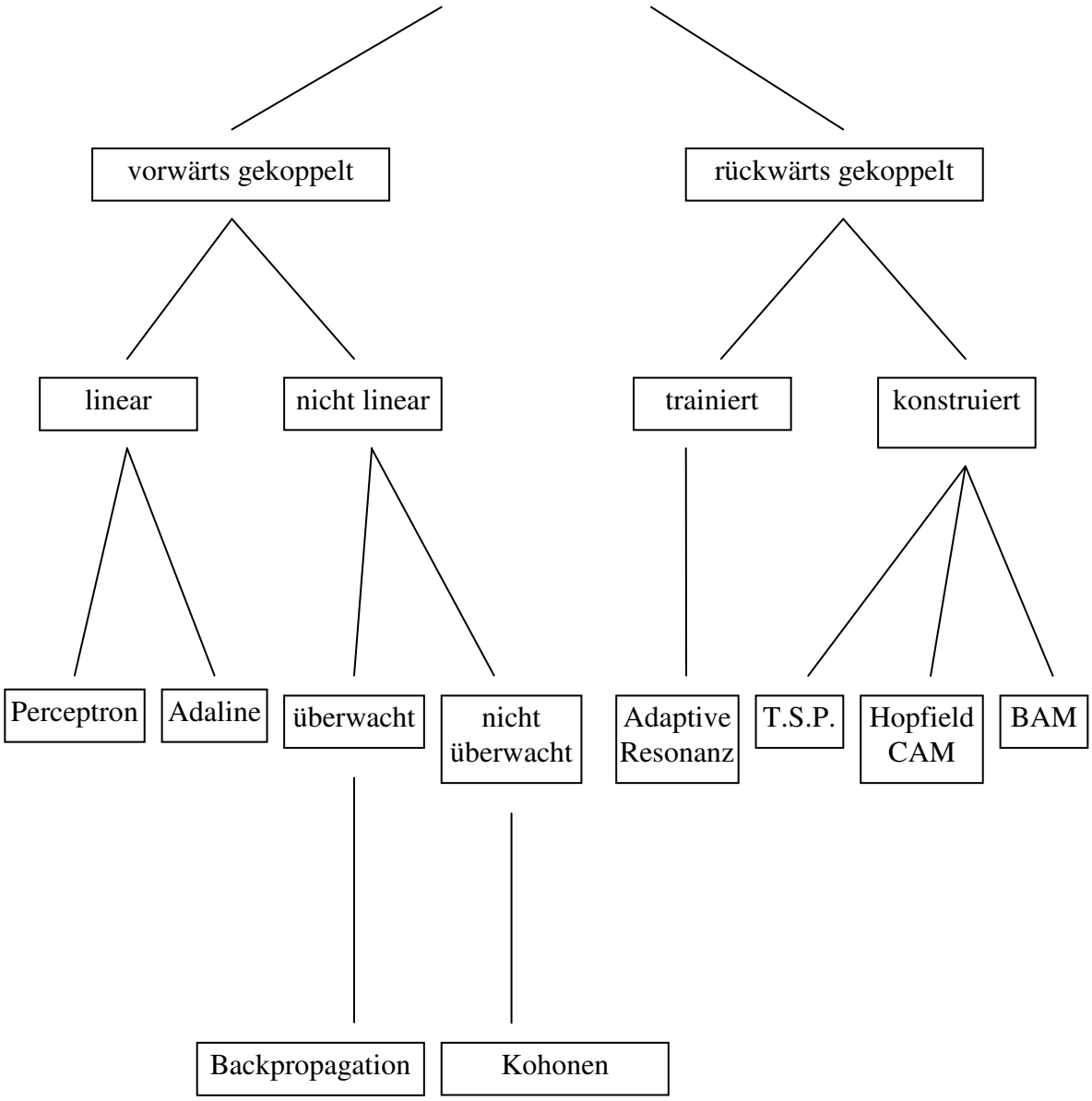

Abbildung 5.1: Arten Neuronaler Netzwerke

5.2 Optimierungsverfahren

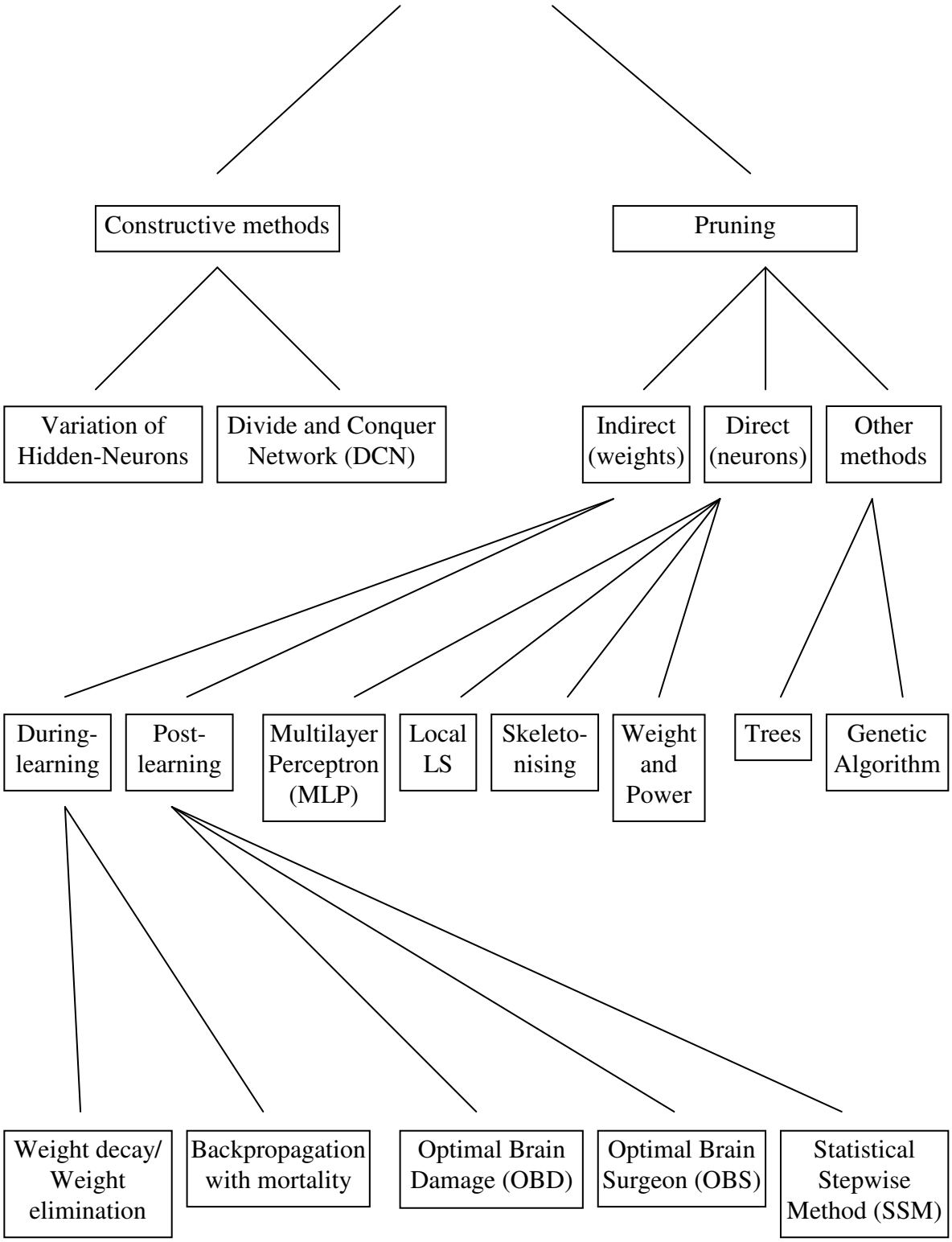

Abbildung 5.2: Methoden zur Strukturoptimierung

6 Verzeichnisse

6.1 Literaturverzeichnis

[1] Böhme, H.-J. *Künstliche Neuronale Netzwerke*. Arbeitsblätter zur Vorlesung. TU Ilmenau. Wintersemester 1994/95.

[2] *Borland Delphi: Das Buch*. [u.a. CD-ROM mit Delphi-Manuals und viel Zubehör] Hg. Borland GmbH Deutschland. München: Tewi, 1995.

[3] Brause, Rüdiger. Neuronale Netze: Eine Einführung in die Neuroinformatik. Stuttgart: Teubner, 1991.

[4] *Delphi Benutzerhandbuch*. Hg. Borland GmbH. Langen, 1995.

[5] Frey, Hans Georg und Justus Schach. „Modern geknotet: Neuro-Netze: Strukturen, Typen, Anwendungen". *c't*. Heft 2 (Februar 1996): 256-264.

[6] Jutten, Christian und Olivier Fambon. *Pruning Methods: A Review*. [Quellzeitschrift lag vor im Bereich Neuroinformatik, Paterre, BMTI-Gebäude, TU Ilmenau]

[7] Krol, Rüdiger. Implementation eines Neuronalen Backpropagation-Netzes und Erweiterung zur Beschleunigung des Lernprozesses und dessen Stabilisierung. Studienjahresarbeit. TU Ilmenau, 1995.

[8] *MATLAB: Neural Network Toolbox-User's Guide*. Hg. The Math Works, Inc. Natick, June 1992.

[9] Poenicke, Klaus. „Wie verfaßt man wissenschaftliche Arbeiten?: Ein Leitfaden vom 1. Studiensemester bis zur Promotion". *Die Duden-Taschenbücher*. Bd. 21. 2., neu bearb. Aufl. Mannheim, Wien, Zürich: Dudenverl., 1988.

[10] Ritter, Helge, Klaus Schulten und Thomas Martinetz. *Neuronale Netze: Eine Einführung in die Neuroinformatik selbstorganisierender Netzwerke*. Bonn; München; Reading, Mass. [u.a.]: Addison-Wesley, 1991.

[11] Schäpers, Arne. „Ableitungen und Abwege: Listen in Delphi". *c't*. Heft 5 (Mai 1996): 292-297.

[12] Trautwein, Michael. Anbindung einer Datenbank an einen Netzwerk-Simulator unter Windows und Schaffung von Tools zur Auswertung der Lerndaten. Diplomarbeit. TU Ilmenau, 14.11.1995.

[13] Zell, Andreas. *Simulation Neuronaler Netzwerke*. 1. Aufl. Addison-Wesley (D) GmbH, 1994.

6.2 Abbildungsverzeichnis

6.3 Gleichungsverzeichnis

6.4 Tabellenverzeichnis

Erklärung

Hiermit versichere ich, die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Hilfsmittel geschrieben zu haben.

Stützerbach, den 21.06.1996