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

begonnen am:	1. Juli 1991
beendet am:	15. Februar 1992
CR-Klassifikation:	I.5.1, I.5.3, I.6, C.1.2, C.1.3

Entwicklung eines massiv parallelen Simulatorkerns für neuronale Netze auf der MasPar MP-1

Niels Mache
Diplomarbeit Nr. 845

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

Inhaltsverzeichnis

Einleitung **i**

1 Mehrstufige feedforward-Netzwerke **4**
1.1 Mustererkennung mit feedforward-Netzen 4
1.2 Das Backpropagation-Lernverfahren 5
1.2.1 Die Vorwärtspropagierung 7
1.2.2 Die Rückwärtspropagierung 8
1.2.3 Die Modifikation der synaptischen Gewichte 8
1.2.4 Stochastische Gradientenverfahren 8

2 Die MasPar MP-1 **10**

3 Parallele Algorithmen **13**
3.1 Die maximale Parallelität des Backpropagation Verfahrens 14
3.2 Parallele Algorithmen für Backpropagation auf SIMD-Rechnern im Vergleich 15
3.2.1 Link-Parallelität (Matrizenverfahren) 15
3.2.2 Paralleler Ablauf von Vorwärts- und Rückwärtspropagierung 16
3.2.3 Unit-Parallelität 16
3.2.4 Parallelität des Trainingssatzes 17

4 Backpropagation auf der MasPar MP-1 **20**
4.1 Der parallele Backpropagation-Algorithmus auf der MasPar MP-1 20
4.1.1 Die Vorwärtspropagierung 22
4.1.2 Die Rückwärtspropagierung 24
4.1.3 Die Modifikation der Verbindungsgewichte 27
4.2 Parallele Berechnung verschiedener Trainingssätze 27

5 Aspekte der Implementierung **31**
5.1 Abbildung des Netzwerkes auf die Prozessoren der MasPar 31
5.2 Die Schnittstellen des Simulatorkerns 32
5.2.1 Die erweiterte Anwenderschnittstelle des parallelen Simulatorkerns 32
5.3 Restriktionen der Netzwerktopologie 35
5.4 Erweiterungen des parallelen Simulators 36
5.5 Beschränkungen der MasPar MP-1 37

5.6 Leistungsmessungen 38

A Die Fehlermeldungen des Simulatorkerns 40

Abbildungsverzeichnis

0.1 Das Zusammenwirken der SNNS Komponenten 3

1.1 Dreistufiges feedforward-Netzwerk 6

2.1 Die Architektur der MasPar MP-1 12

3.1 Zuordnung der Neuronen des *6-3-5* Beispielnetzwerkes auf die Prozessorelemente bei paralleler Berechnung aller Units 19

4.1 Zuordnung der Neuronen eines Netzwerkes auf die Prozessoren der MasPar 23
4.2 Vorwärtspropagierung von Layer 0 auf Layer 1 beim *6-3-5* Netzwerk 25
4.3 Vorwärtspropagierung von Layer 1 auf Layer 2 beim *6-3-5* Netzwerk 26
4.4 Rückwärtspropagierung von Layer 2 auf Layer 1 beim *6-3-5* Netzwerk 28
4.5 Änderung der Verbindungsgewichte im Hidden Layer beim *6-3-5* Netzwerk 29
4.6 Änderung der Verbindungsgewichte im Output Layer beim *6-3-5* Netzwerk 30

5.1 Abbildung der Netzwerke auf das Prozessorarray der MasPar für $N_{procs} \leq N_{PExprocs}$ 32
5.2 Abbildung der Netzwerke auf das Prozessorarray der MasPar für $N_{procs} > N_{PExprocs}$ 32
5.3 Die Schnittstellen des Netzwerksimulators 33

Tabellenverzeichnis

5.1 CPS Leistung verschiedener Rechnersysteme 39
5.2 CUPS Leistung verschiedener Rechnersysteme 39

Einleitung

Künstliche Neuronale Netze sind Modelle für informationsverarbeitende Systeme. Diese Systeme setzen sich aus vielen einfachen, parallel arbeitenden Einheiten zusammen, die Informationen über ein Netz von Verbindungen austauschen. Sie sind interessant, weil sie die Struktur biologischer Nervensysteme nachbilden und in Bezug auf Lernfähigkeit und Parallelität anderen Ansätzen der KI überlegen sind.
Am IPVR[1] wurde bereits ein Simulator für neuronale Netze, der SNNS[2], entwickelt ([Som90] und [Mac90]). Dieser Netzwerksimulator ist auf verschiedenen Rechnern mit dem UNIX-Betriebsystem lauffähig und besteht aus einer graphischen Benutzeroberfläche, einem Simulatorkern und einem Netzwerkcompiler (Abb. 0.1).

Das Ziel dieser Arbeit ist der Entwurf und die Realisierung eines massiv parallelen Simulators zur Simulation von neuronalen Netzen auf der MasPar MP-1216, einem am IPVR verfügbaren SIMD-Rechner mit über 16.000 Prozessoren. Der hier vorgestellte parallele Netzwerksimulator ermöglicht die Simulation und das Training von mehrstufigen feedforward-Netzen (multilayer perceptrons) mit teilweise bzw. vollständig verbundenen Ebenen. Bei der Konzeption des Simulatorkerns wurde die spätere Erweiterung des parallelen Simulators auf andere Netzwerktopologien besonders berücksichtigt. Eine wesentliche Eigenschaft des parallelen Netzwerksimulators ist die Möglichkeit der Interaktion durch den Anwender (über die graphische Benutzeroberfläche des SNNS) sowie die (Abwärts-) Kompatibilität der Funktionsschnittstellen mit denen des sequentiellen Äquivalents. Die Leistung des parallelen Simulators beträgt auf der MasPar MP-1216 bis zu 67 Millionen Gewichtsänderungen pro Sekunde (CUPS) bzw. 176 Millionen Verbindungen pro Sekunde (CPS). Das NETtalk Netzwerk aus [SR86] mit 203 Eingangsneuronen, 60 Neuronen in der verdeckten Schicht und 26 Neuronen in der Ausgabeschicht wird mit 98 Millionen CPS und 41 Millionen CUPS simuliert. Die Anzahl der Gewichtsänderungen pro Sekunde (CUPS) berücksichtigt die Zeit, die zur Propagierung der Eingabemuster durch das Netzwerk, die Berechnung der Fehlerwerte und die Änderung der synaptischen Gewichte nach der Abarbeitung des Trainingssatzes benötigt wird.

[1] Institut für Parallele und Verteilte HöchstleistungsRechner
[2] Stuttgarter Neuronale Netze Simulator

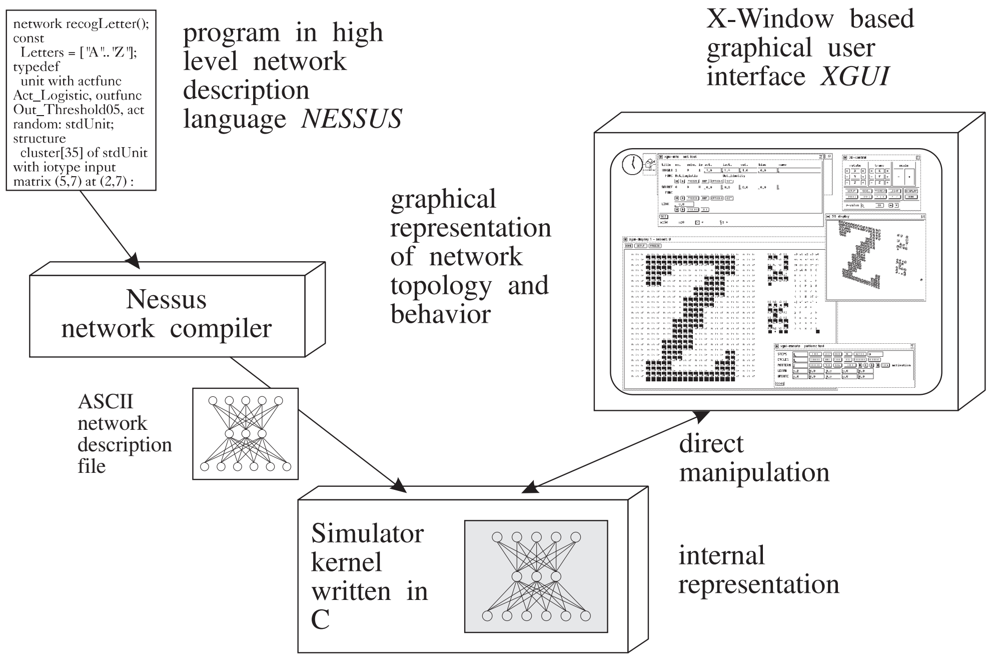

Abbildung 0.1: Die Struktur des SNNS besteht aus drei Komponenten: dem Simulatorkern, der graphischen Benutzeroberfläche und dem Netzwerkcompiler.

Kapitel 1

Mehrstufige feedforward-Netzwerke

Mehrstufige feedforward-Netzwerke sind künstliche neuronale Netze in denen die Neuronen, in Schichten angeordnet, die Information ausgehend von den Eingabeneuronen zu den Ausgabeneuronen übertragen. Die Abbildung 1.1 zeigt ein feedforward-Netz mit voller Vernetzung der jeweils benachbarten Schichten. Die Informationsausbreitung läuft in feedforward-Netzen nur unidirektional von den Eingabe- zu den Ausgabeneuronen über die Ebenen des Netzwerkes hinweg. Diese Netze sind daher nichtrekurrent, d.h. es gibt keine Verbindungen, die von einer höheren Schicht von Neuronen zu einer tieferliegenden Schicht gerichtet sind. Neuronen in feedforward-Netzen mit mehreren Schichten (multilayer perceptrons), die weder der Eingabe- noch der Ausgabeschicht angehören, nennt man *Hidden Units*. Der folgende Abschnitt erläutert die Grundlagen der feedforward-Netzwerke sowie der Lernverfahren, die für das Verständnis des parallelen Simulators von Bedeutung sind. Eine weitergehende Einführung und umfassende formale Beschreibung zu neuronalen Netzen findet man in [RM86] und [RMS90].

1.1 Mustererkennung mit feedforward-Netzen

Die Mustererkennung ist eines der Hauptanwendungsgebiete für neuronale Netze. Ein mustererkennendes System ist in der Lage, zu einem vorgegebenen Muster die Angehörigkeitsklasse zu bestimmen, d.h. diejenige Klasse zu bestimmen, der dieses gegebene Muster angehört. Feedforward-Netzwerke sind für viele Anwendungen, die Musterklassifikationen erfordern, recht gut geeignet und gehören in der Praxis zu den meistverwendeten Netzwerken. Am IPVR wurden bereits einige Arbeiten aus dem Bereich der Mustererkennung, wie zum Beispiel die Erkennung von hand- und maschinengeschriebenen Zeichen und die translations- und rotationsinvariante Klassifikation von Werkstücken, mit dem SNNS erfolgreich durchgeführt.

Damit ein feedforward-Netzwerk Muster klassifizieren kann, muß es vorher trainiert werden. Das Training dieser Netze geschieht mit Hilfe von Beispielmustern aus einer Trainingsmenge, zu denen im Laufe des Lernvorganges eine Klassenstatistik bestimmt wird. Aufgrund dieser Klassenstatistik sollte das Netz nach erfolgreichem Training in der Lage sein, auch für unbekannte Muster die Angehörigkeitsklasse richtig zu bestimmen. Beim Backpropagation Lernverfahren setzt sich der eigentliche Lernvorgang aus dem normalen Arbeitsvorgang, d.h. dem Vorgang der Musterklassifikation, und einer Trainings- oder Lernphase zusammen:

1. Bei der Vorwärtspropagierung (*forward propagation*) werden die Eingabeneuronen des Netzes mit einem Eingabemuster aus der Trainingsmenge versorgt. Für diese Eingabemuster berechnen alle Neuronen (Units) des Netzes ihre Aktivierung. Enthält das Netzwerk n verdeckte Schichten (Hidden Layers) von Neuronen, erfolgt die Berechnung in $(n+1)$ sequentiellen Schritten.

2. In der nächsten Phase (Rückwärtspropagierung) vergleicht jede Ausgangsunit ihren Ausgabewert mit dem zugehörigen Ausgabemuster der Trainingsmenge. Die Differenz der beiden Werte stellt den Fehler des Netzwerkes für dieses bestimmte Muster dar. Die Neuronen der tieferliegenden Schichten berechnen ihren Fehlerwert dadurch, daß sie die Fehler der ihnen nachfolgenden Units, mit ihren Verbindungsstärken gewichtet, aufsummieren. Da hier der Informationsfluß rückwärts von den Ausgabe- zu den Eingabeunits erfolgt, nennt man diesen Schritt *backward propagation*. Je nach verwendetem Lernverfahren werden diese Fehlerwerte für ein oder mehrere Trainingsmuster aufsummiert. Auch dieser Vorgang erfordert $(n+1)$ Schritte.

3. In der letzten Phase werden die Verbindungsgewichte entsprechend den Fehlerwerten der einzelnen Neuronen modifiziert [1]. Dieser Vorgang heißt *weight update* und kann für alle Verbindungen gleichzeitig in einem einzigen Zeitschritt erfolgen.

Der Lernvorgang wird solange wiederholt bis der Gesamtfehler des Netzwerkes, d.h. die Summe der Einzelfehler, die das Netz bei den Mustern der Trainingsmenge erzeugt, einen vorgegebenen Wert unterschreitet. Die Brauchbarkeit eines Netzwerkes richtet sich danach, inwieweit das Netz unbekannte Muster, also Muster, die beim Lernvorgang nicht in der Trainingsmenge enthalten waren, richtig klassifizieren kann. Erfüllt ein Netzwerk nicht die Anforderung nach Generalisierung der Musterklassen, so müssen beispielsweise das Lernverfahren bzw. dessen Parameter oder auch die Topologie des Netzwerkes modifiziert werden.

1.2 Das Backpropagation-Lernverfahren

Für Multilayer Perceptrons ist eine Vielzahl von effizienten Lernverfahren bekannt. Zu diesen gehören insbesondere die verschiedenen Varianten des Backpropagation-Verfahrens. Der Backpropagation-Algorithmus (Werbos 1974,

[1] Die Änderung der Verbindungsgewichte beeinflussen i.a. noch andere Parameter wie z.B. Lernfaktoren und den Ausgabewert der Neuronen in den tieferliegenden Schichten.

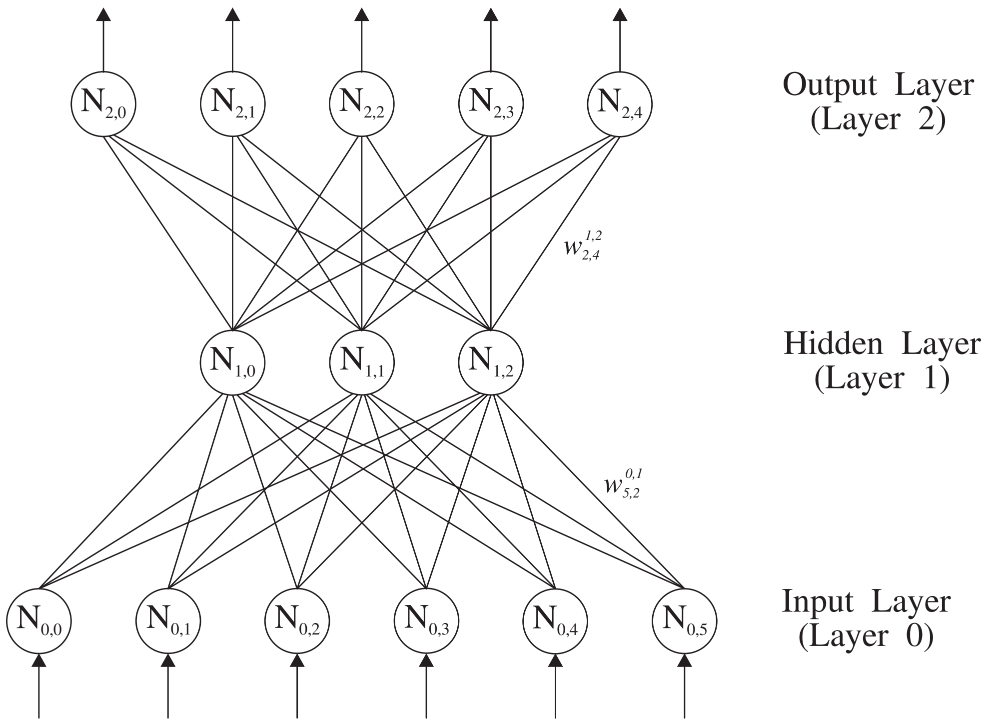

Abbildung 1.1: Dreistufiges *6-3-5* feedforward-Netzwerk (6 Input-, 3 Hidden-, und 5 Output-Units) mit totaler Vernetzung der jeweils benachbarten Ebenen. Das Verbindungsgewicht $w_{i,j}^{l,m}$ bezeichnet das synaptische Gewicht der (gerichteten) Verbindung vom Neuron i in der Schicht l zu dem Neuron j der Schicht m. Für die feedforward-Netzwerke gilt daher $l < m$. $N_{k,l}$ bezeichnet das l-te Neuron in der k-ten Schicht.

Rumelhart und Williams 1986) minimiert den Fehler, d.h. die Differenz zwischen den Ausgabewerten der Ausgangsneuronen und den zugehörigen Ausgabemustern der Trainingsmenge, eines mehrstufigen feedforward-Netzwerkes durch die Änderung der Verbindungsgewichte der Neuronen. Im folgenden Abschnitt werden die drei Phasen des Backpropagation-Lernverfahrens formal beschrieben.

1.2.1 Die Vorwärtspropagierung

In der ersten Phase des Backpropagation-Verfahrens werden zu einem vorgegebenen Eingabemuster die Aktivitäten bzw. Ausgabewerte der Neuronen des Netzwerkes bestimmt. Für die Aktivierung a_i^k des Neurons i in der k-ten Schicht eines vollständig verbundenen Netzwerkes gilt:

$$a_i^k = \sigma \left(\sum_{l=0}^{k-1} \sum_{j=0}^{N_{nodes}(l)-1} w_{j,i}^{l,k} \, o_j^l \right) .$$

Hierbei ist $w_{j,i}^{l,k}$ das synaptische Gewicht der Verbindung des Neurons j aus der Schicht l mit dem Neuron i in der Schicht k, $N_{nodes}(l)$ steht für die Anzahl der Neuronen die der Schicht l angehören. Den Ausgabewert des Neurons j aus der (tieferliegenden) Schicht l des Netzwerkes bezeichnet o_j^l. Die Ausgabewerte der Eingangsneuronen (Neuronen aus der Schicht 0) sind genau die Werte des vorgegebenen Eingabemusters, d.h. für die Eingangsneuronen gilt $o_j^0 = I_j$. Die Aktivierungsfunktion $\sigma(x)$ ist hier die *sigmoide* Funktion $\sigma(x) = \frac{1}{1+e^{-x}}$, die den Zusammenhang zwischen der synaptischen Eingabe des Neurons und dessen Aktivierung beschreibt. Die monoton steigende *sigmoide* Funktion bildet den Wertebereich $x \in [-\infty, +\infty]$ auf den Funktionsbereich $\sigma(x) \in [0,1]$ ab.

Beim Backpropagation-Algorithmus nach [RHW86] ist der Ausgabewert o_i^k identisch mit der Aktivierung des Neurons, d.h. $o_i^k = a_i^k$. Die Ausgabewerte der Neuronen liegen daher im Intervall $[0,1]$, das bedeutet daß die Werte der Ausgabemuster ebenfalls im Bereich $[0,1]$ liegen sollten, da ansonsten die Minimierung des Gesamtfehlers des Netzwerkes gar nicht oder nur sehr schlecht möglich ist. Ein Maß dafür wie gut das Netzwerk die Eingabemuster „erkennt", d.h die Eingabemuster auf die Ausgabemuster abbildet, ist der Gesamtfehler E. Der Fehler E des Netzwerkes ist die (quadratische) Differenz zwischen den Ausgabewerten $o_i^{N_{layers}-1}(k)$ der Ausgangsneuronen nach der Propagierung des k-ten Eingabemusters und dem Zielwert $T_i(k)$ des Ausgabemusters aufsummiert über alle Musterpaare $k = 0, \ldots, (N_{patterns} - 1)$. Formal:

$$E = \frac{1}{2} \sum_{k=0}^{N_{patterns}-1} \sum_{i=0}^{N_{nodes}(N_{layers}-1)-1} \left(T_i(k) - o_i^{N_{layers}-1} \right)^2 .$$

Durch die schrittweise Änderung der synaptischen Gewichte der Verbindungen zwischen den Neuronen des Netzwerkes kann der Fehler E minimiert werden. Dazu werden die Fehlerwerte der Neuronen von der Ausgabeschicht zur Eingabeschicht zurückpropagiert.

1.2.2 Die Rückwärtspropagierung

Die Minimierung des Gesamtfehlers E des Netzwerkes geschieht beim Backpropagation-Verfahren durch die näherungsweise Durchführung des Gradientenabstiegs für E. Beim echten Gradientenverfahren werden die synaptischen Gewichte $w_{i,j}^{l,m}$ für jede Präsentation des gesamten Trainingssatzes um den Betrag $\Delta w_{i,j}^{l,m}$ mit

$$\Delta w_{i,j}^{l,m} = -\eta \frac{\partial E}{\partial w_{i,j}^{l,m}}$$

schrittweise geändert. Für ein hinreichend kleines $\eta > 0$, den *Lernfaktor*, bewegt man sich mit der Änderung der synaptischen Gewichte in Richtung des steilsten Abfalls von E.

Der Fehler $\delta_i^l(k)$, den das Neuron i aus der Schicht l des Netzwerkes bei der Propagierung des k-ten Musters erzeugt, kann wie folgt berechnet werden:

$$\delta_i^l(k) = \begin{cases} \sigma'(a_i^l)\ (T_i(k) - o_i^l) & \text{falls } l = N_{layers} - 1, \\ \sigma'(a_i^l) \sum_{m=l+1}^{N_{layers}-1} \sum_{j=0}^{N_{nodes}(m)-1} \delta_j^m(k)\, w_{i,j}^{l,m} & \text{sonst.} \end{cases}$$

Dabei ist $\sigma'(x) = x \cdot (1 - x)$ die Ableitung der *sigmoiden* Funktion.

1.2.3 Die Modifikation der synaptischen Gewichte

Nach der Propagierung aller Eingabemuster des Trainingssatzes und der Berechnung der Fehlerwerte $\delta_i^l(k)$ für alle Neuronen und Trainingsmuster werden die Verbindungsgewichte $w_{i,j}^{l,m}$ der Neuronen geändert. Die Vorschrift zur Änderung $\Delta w_{i,j}^{l,m}$ des synaptischen Gewichts der Verbindung zwischen dem Neuron i aus der Schicht l und dem Neuron j der Schicht m des Netzwerkes ist:

$$\Delta w_{i,j}^{l,m} = \eta \sum_{k=0}^{N_{patterns}-1} \delta_j^m(k)\, o_i^l(k).$$

1.2.4 Stochastische Gradientenverfahren

Das stochastische Gradientenverfahren ist eine Vereinfachung des echten Gradientenverfahrens. Der Unterschied zum echten Gradientenverfahren besteht darin, daß die synaptischen Gewichte nicht entsprechend dem Gesamtfehler des Netzwerkes, sondern nur entsprechend dem Fehler des jeweils aktuellen Musterpaares modifiziert werden. Die Verbindungsgewichte werden deshalb gleich nach der Präsentation eines Musters geändert und nicht wie beim echten Gradientenverfahren nach der Präsentation der gesamten Trainingsmenge. Lernalgorithmen, die auf dem stochastischen Gradientenverfahren basieren, lassen sich auf massiv parallelen Rechnersystemen aufgrund des hohen Kommunikationsaufwands jedoch nicht sehr effizient realisieren (siehe auch Kapitel 3.2.4).

Für die Änderung $\Delta w_{i,j}^{l,m}$ des Verbindungsgewichts $w_{i,j}^{l,m}$ beim stochastischen Gradientenverfahren gilt:

$$\Delta w_{i,j}^{l,m} = \eta\, \delta_j^m(k)\, o_i^l(k).$$

Im Gegensatz zum echten Gradientenverfahren wird die Änderung $\Delta w_{i,j}^{l,m}$ des Verbindungsgewichts $w_{i,j}^{l,m}$ nicht über alle Muster der Trainingsmenge aufsummiert, sondern gleich nach der Berechnung des Fehlers $\delta_j^m(k)$ des Neurons zum Verbindungsgewicht $w_{i,j}^{l,m}$ aufaddiert.

Kapitel 2

Die MasPar MP-1

Die MasPar MP-1 ist ein nach dem *SIMD* (Single Instruction, Multiple Data) Prinzip arbeitendes, massiv paralleles Computersystem. Die herausragenden Eigenschaften von parallelen Rechnersystemen, die auf dem *SIMD*-Prinzip basieren, sind die Skalierbarkeit in Bezug auf die Anzahl der Prozessoren und die außergewöhnlich hohe Gesamtleistung dieser Systeme. Die Struktur der MasPar MP-1 und das Zusammenspiel der einzelnen Komponenten ist in Abbildung 2.1 dargestellt. Die MasPar MP-1 setzt sich aus den folgenden Komponenten zusammen:

- Das **Array der Prozessorelemente (PE Array)**.
  Das PE Array ist die arithmetische und logische Verarbeitungseinheit der Maschine, hier werden die Operationen (single instruktion) parallel auf den Daten (multiple data) ausgeführt.

- Das **Steuerwerk der Prozessorelemente**, die **Array Control Unit (ACU)**.
  Die Array Control Unit erfüllt zwei Funktionen : zum einen steuert die ACU die Programmausführung aller Prozessorelemente, indem sie die Instruktionen der Prozessorelemente generiert und an das PE Array weiterleitet, zum anderen kann die ACU unabhängig vom PE Array in sequentieller Art und Weise das Anwendungsprogramm ausführen.

- Die **Kommunikationsmechanismen**.
  Für die Kommunikation zwischen den Prozessorelementen des PE Arrays bzw. dem PE Array und der ACU gibt es mehrere Kommunikationskanäle:

  - Das X-Netzwerk zur Kommunikation zwischen benachbarten Prozessorelementen. Das 2-dimensionale X-Netzwerk ermöglicht die schnelle Kommunikation der Prozessoren im PE Array mit den jeweils acht benachbarten Prozessorelementen.

  - Der globale Router zur Kommunikation zwischen beliebigen Prozessorelementen. Der Router ist ein 3-stufiger Kreuzschienenverteiler und erlaubt bis zu 1024 gleichzeitige Kommunikationsverbindungen (bei einer MP-1 mit 16K Prozessoren).

  - Zwei Bussysteme zur Kommunikation zwischen der ACU und dem PE Array. Über ein gemeinsames Bussystem steuert die ACU die Operationen der Prozessorelemente und transferiert Daten zu und von den Prozessorelementen des PE Arrays. Ein zweites Bussystem dient zur Übertragung von Statussignalen der Prozessorelemente zur ACU.

- Das **UNIX Subsystem (USS)**.
  Auf dem UNIX Subsystem stehen die Dienstleistungsprogramme wie zum Beispiel Compiler, Debugger und Netzwerkdienste zur Verfügung. Der Datenaustausch zwischen herkömmlichen (sequentiellen) Programmen und der Array Control Unit der MasPar geschieht über den VME-Bus des UNIX Subsystems.

- Das **I/O Subsystem**.
  Das I/O Subsystem unterstützt den Hochgeschwindigkeits-Datentransfer zwischen dem Array der Prozessorelemente und den Ein- und Ausgabegeräten bzw. Massenspeichern.

Die Programmierung der MasPar MP-1 kann wahlweise in der Programmiersprache MPL (MasPar Parallel Application Language) oder in Fortran erfolgen. Für die Programmierung des parallelen Simulators für neuronale Netzwerke wurde die Programmiersprache MPL gewählt. Die Sprache MPL basiert auf der Programmiersprache C und ermöglicht die effiziente und flexible Implementierung der parallelen Algorithmen. Programme für die MasPar MP-1 werden üblicherweise in einen parallelen und einem sequentiellen Teil aufgeteilt. Der parallele, in MPL implementierte Teil wird auf der ACU der MasPar ausgeführt, den sequentiellen Teil des Programms führt das UNIX Subsystem aus. Die sequentiellen und parallelen Programmteile werden unabhängig voneinander abgearbeitet und können bei Bedarf synchronisiert werden. Der Datenaustausch zwischen der ACU bzw. dem PE Array der MasPar und dem Subsystem geschieht mittels Blocktransferfunktionen, die aus dem sequentiellem und parallelen Teil des Programms heraus initiiert werden können.

Die Leistung der MasPar MP-1 ist, wie die aller Parallelrechner, sehr von dem Kommunikationsaufwand der verwendeten Algorithmen abhängig. Erfahrungsgemäß kann die Spitzenleistung eines Parallelrechners um Größenordnungen höher liegen als die effektive Verarbeitungsgeschwindigkeit der zur Anwendung kommenden Algorithmen. So gibt die MasPar Computer Corporation eine Gleitkomma-Spitzenleistung von 1200 MFLOPS [1] bei der MP-1 mit 16K Prozessoren an. Leider ist dieser Wert für praktische Anwendungen wenig aussagekräftig, denn schon die für die Bestimmung der Gleitkommaleistung günstigsten Algorithmen, die keine Kommunikation zwischen den Prozessorelementen erfordern, z.B. die Berechnung des Vektor-Punktprodukts, lassen die Leistung auf etwa 800 MFLOPS absinken. Der Kommunikationsaufwand zwischen der ACU bzw. dem PE-Array und dem Subsystem sollte ebenfalls nicht vernachlässigt werden. Ist ein Datenaustausch zwischen dem Subsystem und der ACU bzw.

[1] Die Spitzenleistung von 1200 MFLOPS kann durch Registeroperationen bzw. durch die Ausnutzung der gleichzeitig ausführbaren Load & Store Operationen erreicht werden.

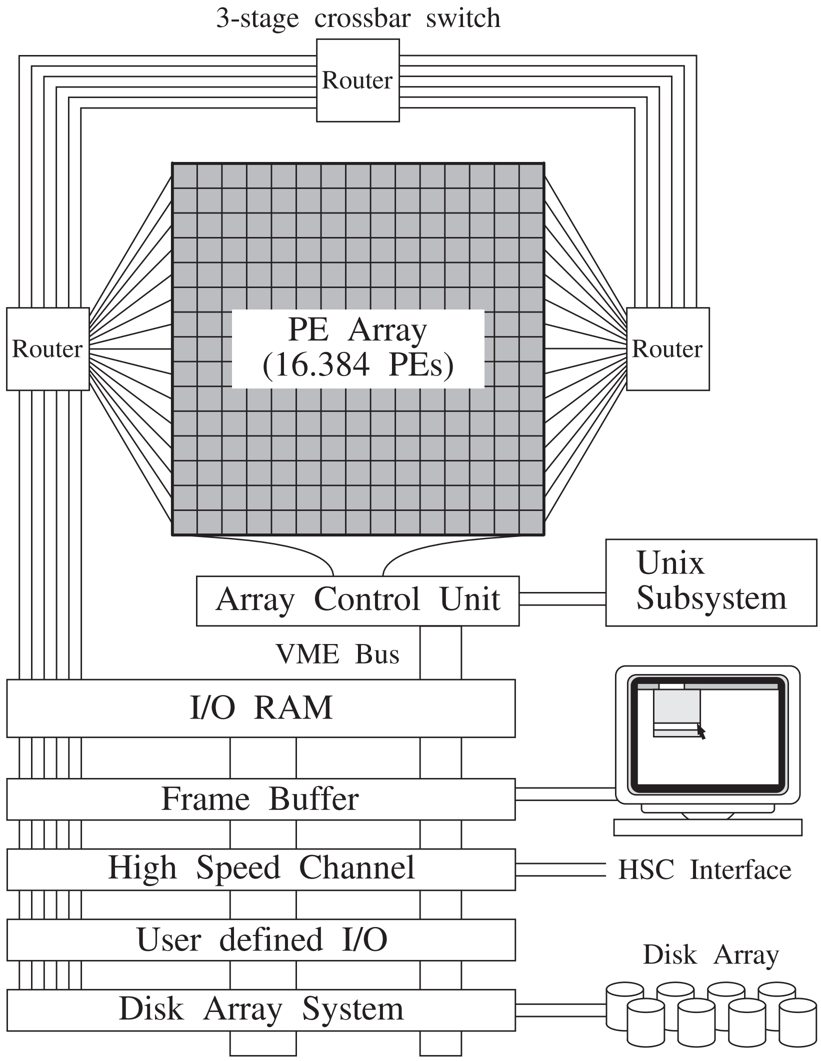

Abbildung 2.1: Die Architektur der MasPar MP-1

dem PE-Array in größerem Umfang notwendig, kann die Kommunikationszeit zwischen den beiden Systemen unter Umständen zum bestimmenden Faktor für die Verarbeitungsgeschwindigkeit des Rechnersystems werden.

Kapitel 3

Parallele Algorithmen

Der Vergleich von natürlichen Nervensystemen mit der heutigen Computerhardware führt auf fundamentale Unterschiede der zugrundeliegenden Funktionsprinzipien: Die Leistungsfähigkeit der heutzutage verwendeten Computer basiert auf der streng sequentiellen Informationsverarbeitung durch einen Prozessor oder einige wenige Prozessoren. Diese Prozessoren, basierend auf der *von-Neumann-Architektur*, verarbeiten Daten und Programme streng sequentiell in einzelnen, elementaren Schritten. Im Gegensatz dazu verarbeiten biologische Nervensysteme Information hochgradig parallel und verteilt. Die Reaktionsgeschwindigkeit einzelner Neuronen ist im Vergleich mit den Prozessoren moderner Computersysteme zwar sehr langsam, dennoch wird dieser Nachteil durch die große Anzahl von Neuronen und die massiv parallele Struktur mehr als ausgeglichen. Neben diesen strukturellen Unterschieden in der Arbeitsweise von Computern und Nervensystemen haben vor allem die kognitiven Leistungen, die Lernfähigkeit und nicht zuletzt die Fehlertoleranz biologischer Systeme zur Entwicklung konnektionistischer Modelle geführt.

Die Simulation und speziell das Training künstlicher neuronaler Netzwerke erfordert eine große Anzahl von Rechenoperationen [1]. Zwar ist die Berechnung von größeren Netzwerken durch „sequentiell" arbeitende Supercomputer [2] möglich, aber diese Vorgehensweise widerspricht nicht nur der parallelen Struktur neuronaler Netzwerke, sondern ist der Simulation mit massiv paralleler Computerhardware auch in Punkto Leistung und Kosteneffizienz unterlegen. Auf der anderen Seite bringt die Anwendung von parallel arbeitenden Computersystemen auch Nachteile mit sich: der höhere Aufwand für den Entwurf paralleler Algorithmen und besonders die Abhängigkeit der Algorithmen von der verwendeten Computerhardware haben eine gewisse Unflexibilität der Verfahren in Bezug auf Modifikationen an Hard- und Software zur Folge.

[1] Für die Berechnung (Backpropagation-Algorithmus aus [RHW86]) eines einzigen Trainingsdurchlaufes mit 1000 Mustern in einem schichtweise vollvernetzten *1000-500-1000* feedforward-Netz sind mindestens 6 Milliarden Rechenoperationen (Addition/Multiplikation) erforderlich.

[2] Heutige Supercomputer (Cray, Convex, NEC SX etc.) verwenden wenige (meist unter 10) aber leistungsfähige Prozessoren, die den Datenstrom im Pipelinebetrieb bearbeiten.

3.1 Die maximale Parallelität des Backpropagation Verfahrens

Die Lernphase des Backpropagation-Verfahrens setzt sich, wie bereits erwähnt, aus der Vorwärts- und Rückwärtspropagierungsphase und der Änderung der Verbindungsgewichte zusammen. Bei der Vorwärtspropagierung (*forward propagation*) berechnet jede Unit für ein vorgegebenes Eingabemuster ihre Aktivierung und die Aktivierungsfunktion. Diese Phase läßt drei Aspekte der Parallelität erkennen:

1. Unit-Parallelität:
   Die Berechnung der Aktivierungen sowie der Aktivierungsfunktionen kann für alle **Units** parallel erfolgen. Enthält das Netzwerk n Ebenen von Gewichten, so werden die Ausgangswerte der Units wie in einer Pipeline mit n Stufen berechnet. Nach einer Einlaufphase von n Takten stehen daher mit jedem Takt die Ausgangswerte der Output-Units zur Verfügung [3].

2. Link-Parallelität:
   Die Berechnung der Aktivierungen der Units kann für alle (Eingangs-) **Verbindungen** der jeweiligen Unit parallel erfolgen.

3. Trainingsmuster-Parallelität:
   Die Berechnung aller Aktivierungen und Aktivierungsfunktionen kann für den gesamten **Trainingssatz** parallel erfolgen.

Bei der Rückwärtspropagierung (*backward propagation*) berechnet jede Unit zuerst ihren Fehlerwert, indem sie die Fehler der ihr nachfolgenden Units gewichtet aufsummiert (*error propagation*). Die der Rückwärtspropagierung inhärente Parallelität ist mit der Parallelität der Vorwärtspropagierung vergleichbar. Der wesentliche Unterschied zur Vorwärtspropagierung liegt darin, daß zur Rückwärtspropagierung des Fehlers auf die Ebene l des Netzwerkes die Multiplikation des Fehlervektors $\vec{\delta_m}$ der Nachfolger-Units in der Schicht m mit der **transformierten** Gewichtsmatrix $(W_l^m)^T$ der Verbindungsgewichte der Schicht l mit der Schicht m erforderlich ist. Formal:

$$\vec{\delta}_l = \sum_{k=l+1}^{N_{layers}-1} (W_l^k)^T \cdot \vec{\delta}_k,$$

dabei ist N_{layers} die Anzahl der Ebenen des Netzwerkes. Die Berechnung der Fehlerwerte bei der Rückwärtspropagierung kann ebenso wie die Bestimmung der Aktivierungen in der Arbeitsphase parallel für alle Units, Links und Trainingsmuster erfolgen.

Der letzte Schritt ist die Justierung der Verbindungsgewichte (*weight update*), die ebenfalls für alle Units, Links und Trainingsmuster parallel vorgenommen werden kann, da die jeweiligen Änderungen der Verbindungsgewichte voneinander unabhängig sind. Die Rückwärtspropagierung und die Modifikation der

[3] Wird die Aktivierung und die Ausgabe (d.h. das Ergebnis der Aktivierungsfunktion) *gleichzeitig* berechnet, werden Pipeline und Einlaufphase doppelt so lang.

Verbindungsgewichte kann sogar gleichzeitig erfolgen, falls die Berechnung der Fehlerwerte der Neuronen mit der Änderung der Verbindungsgewichte synchronisiert wird bzw. wenn während der Berechnung der Fehlerwerte der Neuronen in der Schicht $n-1$ die Änderung der Verbindungsgewichte der Neuronen in der Schicht n erfolgt.

Betrachtet man die Vorwärts- und Rückwärtspropagierung nicht getrennt, sondern als einen einzigen Vorgang, so erkennt man eine weitere Parallelität des Backpropagation-Verfahrens:

- die Vorwärts- und Rückwärtspropagierung sowie die Justierung der Verbindungsgewichte kann unter bestimmten Voraussetzungen [4] parallel ablaufen. Der gleichzeitige Ablauf der drei Phasen funktioniert in einem Netz mit drei Schichten folgendermaßen: während in der ersten Schicht die Aktivierungen der Hidden-Units für das Eingabemuster n berechnet werden, läuft gleichzeitig die Berechnung der Fehlerwerte der Hidden-Units für das Ausgabemuster (*n-1*), sowie die Modifikation der Verbindungsgewichte vom Hidden Layer zum Output Layer.

  Da der parallele Ablauf der Arbeits- und Lernphase für SIMD-Rechner jedoch keine echten Vorteile bringt, wird hier auf dieses Verfahren nicht näher eingegangen.

3.2 Parallele Algorithmen für Backpropagation auf SIMD-Rechnern im Vergleich

Bei der Entwicklung effizienter, paralleler Algorithmen spielen zwei durch die Hardware des Parallelrechners vorgegebe Parameter, eine entscheidende Rolle:

1. die Rechnerarchitektur hinsichtlich der Befehls- und Datenströme und
2. die Topologie der Kommunikationsstruktur(en) des Parallelrechners.

Aus der Literatur sind eine Reihe von parallelen Algorithmen zur Simulation von neuronalen Netzen auf SIMD-Rechnern bekannt. In [Sin90] werden beispielsweise fünf parallele Verfahren für die Connection Machine diskutiert. Im folgenden Abschnitt werden die in Kapitel 3.1 erläuterten, grundlegenden Parallelisierungsmöglichkeiten des Backpropagation-Verfahrens auf ihre Eignung in Bezug auf die MasPar MP-1 untersucht.

3.2.1 Link-Parallelität (Matrizenverfahren)

Für massiv parallele SIMD-Rechner mit einem 2-dimensionalen gitterförmigen Kommunikationsnetz, wie z.B. die feinstrukturierte MasPar MP-1, scheint auf den ersten Blick die Parallelisierung der Verbindungen, d.h. die direkte Abbildung der Verbindungsgewichte in Form einer Matrix auf die einzelnen Prozessoren, gut geeignet. Diese Vorgehensweise hat jedoch einige schwerwiegende

[4] Für den parallelen Ablauf der Arbeits- und Lernphase müssen Einschränkungen beim Pipelining der Aktivierungen der Units über die Schichten des Netzwerkes hinweg gemacht werden.

Mängel: das größte Problem ist die schlechte Auslastung der Prozessorelemente, die dadurch zustandekommt, daß es für den allgemeinen Fall eines mehrschichtigen Netzwerkes (z.B. ein k-l-m-Netz) nicht möglich ist, die Verbindungsgewichte $w_{i,j}$ der Gewichtsmatrizen W gleichmäßig auf alle zur Verfügung stehenden Prozessoren zu verteilen. Die schlechte Verteilung der Gewichtsmatrizen auf ein $k \times k$ Gitter aus Prozessorelementen kommt aufgrund der unterschiedlichen Dimension dieser Matrizen zustande. Hinzu kommt, daß bei der Berechnung der Aktivierungsfunktion der größte Teil der Prozessorelemente untätig ist. Auch die Simulation von Netzwerken mit spärlicher Vernetzung hat dünn besetzte Gewichtsmatrizen zur Folge und verschwendet damit Rechenkapazität und Speicher. Der Kommunikationsaufwand zwischen den Prozessoren der MasPar wird jedoch bei geeigneter Implementierung relativ gering sein [5].

3.2.2 Paralleler Ablauf von Vorwärts- und Rückwärtspropagierung

Der gleichzeitige Ablauf der Vorwärts- und Rückwärtspropagierung bzw. der Änderung der Verbindungsgewichte läßt sich auf SIMD-Rechnern nicht vorteilhaft realisieren. Einer der Gründe liegt in der unterschiedlichen Laufzeit der drei Phasen (die Justierung der Verbindungsgewichte benötigt beispielsweise eine Rechenoperation mehr als die Vorwärtspropagierung). Diese Laufzeitdifferenz hat eine schlechte Auslastung der Prozessorelemente zur Folge.

3.2.3 Unit-Parallelität

Die Unit-Parallelität kann bei geeigneter Speicherung der Verbindungsgewichte innerhalb der Prozessoren gut ausgenutzt werden. Hierbei repräsentiert jeder einzelne Prozessor genau eine Hidden- bzw. Output-Unit und speichert lokal die Gewichte der Eingangsverbindungen dieser Unit. Die Prozessoren jeweils einer Netzwerkschicht werden hierbei in einem Ring angeordnet und jeder Prozessor verfügt über:

- eine „Ablage“ und einen Akkumulator zur Speicherung von Zwischenergebnissen,
- eine bei vollverbundenen Netzwerken unidirektionale, ansonsten bidirektionale Kommunikationsverbindung mit dem linken Nachbarprozessor sowie
- eine bidirektionale Verbindung mit genau dem Prozessor, der die Unit in der nachfolgenden Schicht repräsentiert (diese Verbindung ist nur bei Prozessorelementen, die Hidden-Units darstellen, erforderlich).

[5] Nach [RCB88] gilt für den Kommunikationsaufwand der Connection Machine jedoch genau das Gegenteil. Dieser Unterschied ist wohl auf das Fehlen der für „broadcasting“ Kommunikation im Prozessornetz wichtigen Pipelinekommunikation bei der Connection Machine zurückzuführen.

Gegebenenfalls werden zusätzliche „Ablagen" am rechten Ende der Kette von Prozessoren eingefügt [6]. Die Abbildung 3.1 zeigt die Zuordnung der Neuronen des *6-3-5* Beispielnetzwerkes auf die einzelnen Prozessoren des Parallelrechners. Man beachte, daß die Eingabeneuronen des Netzwerkes nicht auf die Prozessorelemente abgebildet werden.

Die Anordnung der Gewichte im lokalen Speicher des Prozessors ist im Nachbarprozessor jeweils um eine Stelle verschoben [7], d.h. die Verbindungsgewichte werden mit dem Skew-Faktor 1 gespeichert. Abb. 4.2 zeigt die Arbeitsphase der ersten Netzwerkschicht. Auf eine ausführliche Erläuterung des Verfahrens wird an dieser Stelle verzichtet, da der später detalliert beschriebene und auf der MasPar implementierte Algorithmus in ganz ähnlicher Weise funktioniert. Statt dessen sollen hier die Vor- und Nachteile dieses Verfahrens genannt werden. Zu den Vorteilen zählt offensichtlich der moderate Kommunikationsaufwand, weniger offensichtlich hingegen ist die vorteilhafte Speicherung der Verbindungsgewichte, die auch während der Lernphase beibehalten werden kann, sodaß der Austausch der Gewichte zwischen den Prozessoren nicht notwendig ist. Ein weiterer Vorteil ist, daß sich dieser Algorithmus auch für Netzwerke mit spärlicher Vernetzung eignet [8]. Zu den Nachteilen gehört zweifellos die direkte Abbildung des Netzwerkes auf die einzelnen Prozessoren, wodurch die Auslastung aller verfügbaren Prozessorelemente [9] im allgemeinen nur sehr schlecht realisiert werden kann. Ein weiterer Nachteil kann durch den begrenzten (lokalen) Speicher der Prozessorelemente entstehen und limitiert die maximale Anzahl der (Eingangs-) Verbindungen einer Unit.

3.2.4 Parallelität des Trainingssatzes

Die ausschließliche Realisierung der Trainingssatz-Parallelität bedeutet, daß jeder Prozessor eine komplette Kopie des Netzwerkes speichert und auf diese Kopie jeweils ein, von Prozessor zu Prozessor unterschiedliches, Musterpaar anwendet. Das Verfahren kann aber in dieser Form auf massiv parallelen Rechnersystemen praktisch nicht angewendet werden, da die Maximalgröße des auf einem einzelnen Prozessor speicherbaren Netzwerkes zur Gesamtanzahl der Prozessoren, und damit zum Umfang des Trainingssatzes, in einem schlechten Verhältnis steht (entweder ist das Netzwerk für die Speicherung in den Prozessoren zu groß,

[6] Sei i die Anzahl der Units in der Schicht n des Netzwerkes und j die Anzahl der Units in der Schicht $n-1$. Die Anzahl der „Ablagen" in dem Ring von Prozessoren, die die Schicht n repräsentieren, beträgt j falls $i \leq j$, i falls $i \bmod j \equiv 0$ und $j \cdot (int(i/j)+1)$ sonst. Das heißt die Anzahl der „Ablagen" ist immer ein Vielfaches der Anzahl der Units aus der Vorgängerschicht des Netzwerkes.

[7] Bei einer totalen Vernetzung der Units mit ihren Vorgängerunits.

[8] Bei der Darstellung von Netzwerken mit spärlicher Vernetzung werden im Gegensatz zum Matrizenverfahren nur die Verbindungsgewichte der tatsächlich vorhandenen Verbindungen gespeichert. Die Vernetzung muß allerdings regelmäßig und zyklisch sein, irreguläre Vernetzungen sind nicht effizient realisierbar. Dies dürfte aber keine wesentliche Einschränkung darstellen.

[9] Massiv parallele Computersysteme verfügen über mehr als 1000 Prozessoren, die MasPar MP-1216 des IPVR über 16.384. Dies bedeutet in Bezug auf die Simulation von neuronalen Netzwerken, daß die Anzahl der Prozessoren im allgemeinen deutlich größer als die Anzahl der Neuronen des zu simulierenden Netzwerkes ist.

oder der Trainingssatz ist für die Auslastung aller verfügbaren Prozessoren zu klein). Die Nachteile der für dieses Verfahren ungünstigen Konstellation aus Speichergröße und Prozessoranzahl lassen sich etwas mindern, indem man die Verbindungsgewichte des Netzwerkes nicht in den Prozessorelementen, sondern im Front-End Prozessor speichert.

Es gibt jedoch eine weitere, prinzipielle Einschränkung. Um die Parallelität des Trainingssatzes effizient auszunutzen, sollte die Modifikation der Verbindungsgewichte erst **nach** der Abarbeitung des Trainingssatzes erfolgen. Diese sogenannten *batch*-Lernverfahren sind hier realisierbar. Lernverfahren, die auf dem stochastischen Gradientenverfahren basieren, d.h. Lernverfahren, die nach jedem Trainingsmuster die Verbindungsgewichte modifizieren, sind mit diesem Algorithmus nicht sehr effizient realisierbar, da für die Modifikation der Verbindungsgewichte die Summe der Fehlerwerte über alle Netzwerke gebildet werden muß.
Leistungsfähige parallele Backpropagation-Verfahren lassen sich erst durch die gemeinsame Ausschöpfung der Parallelität bei der Berechnung der Units und des Trainingssatzes erzielen. Im folgenden Kapitel wird der in dieser Arbeit auf der MasPar implementierte Algorithmus ausführlich beschrieben.

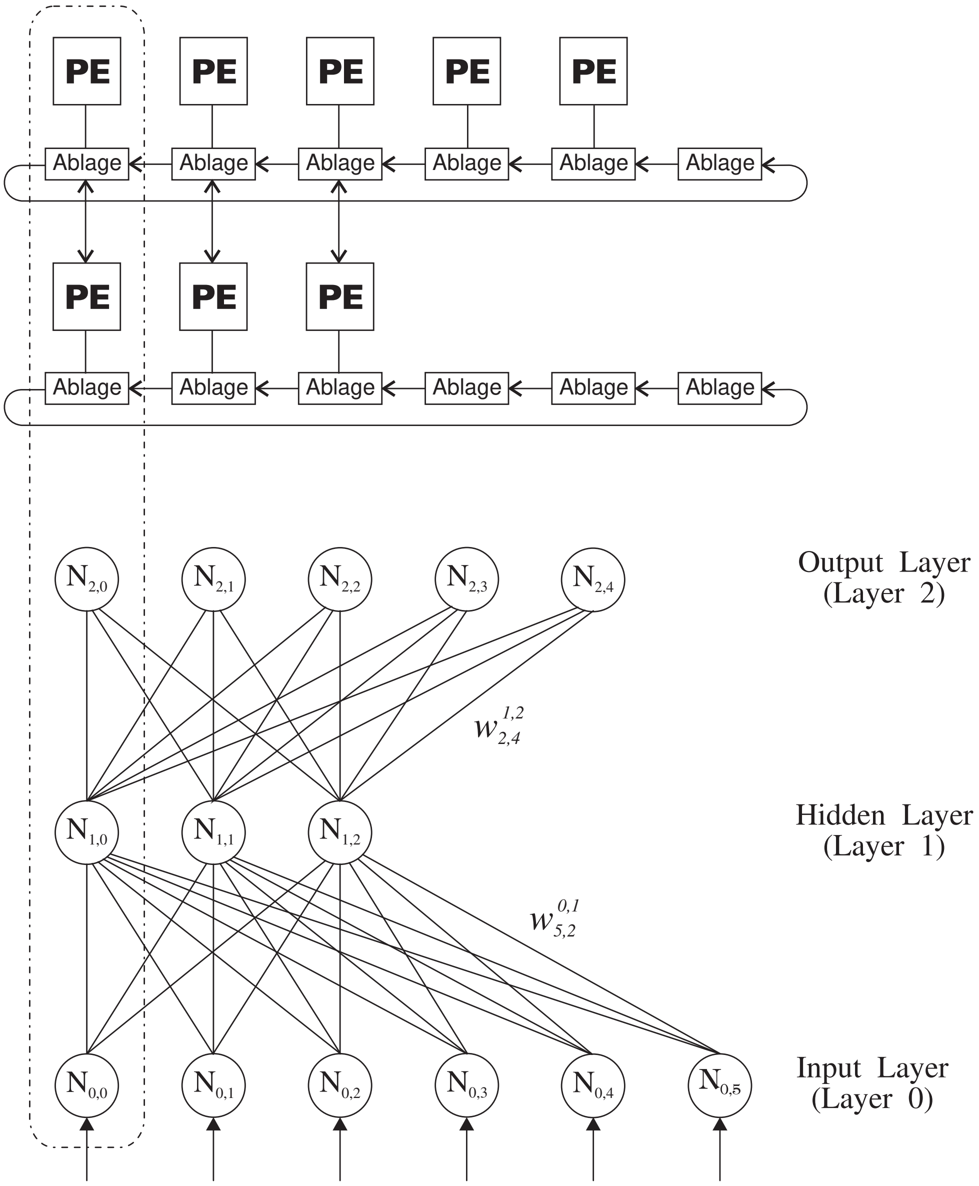

Abbildung 3.1: Mit Ausnahme der Eingabeneuronen wird jedes Neuron des feedforward-Netzwerkes auf ein Prozessorelement abgebildet. Die Anzahl der benötigten Prozessoren ist gleich der Gesamtanzahl der Neuronen des Netzwerkes ohne die Eingabeneuronen. Sei i die Anzahl der Units in der Schicht n des Netzwerkes und j die Anzahl der Units in der Schicht $n-1$. Die Anzahl der „Ablagen“ in dem Ring von Prozessoren, die die Schicht n repräsentieren, beträgt j falls $i \leq j$, i falls $i \bmod j \equiv 0$ und $j \cdot (int(i/j)+1)$ sonst.

Kapitel 4

Backpropagation auf der MasPar MP-1

Leistungsfähige Backpropagation-Algorithmen für die Simulation und das Training von neuronalen Netzen auf der MasPar erhält man durch die Kombination der parallelen Unit- und Trainingssatzalgorithmen. Unter bestimmten Voraussetzungen kann aber auch zusätzlich das Matrizenverfahren zur parallelen Berechnung der Verbindungen einer Unit zur Anwendung gelangen [1]. Ähnliche Verfahren wurden bereits auf der Connection Machine und einem speziell hierfür entwickelten Multiprozessorsystem realisiert ([Zha90] bzw. [KYIA90]).

4.1 Der parallele Backpropagation-Algorithmus auf der MasPar MP-1

Der Backpropagation-Algorithmus für die MasPar ist in seiner Funktion dem Algorithmus zur parallelen Berechnung der Units aus Kapitel 3.2.3 recht ähnlich. Die beiden Algorithmen unterscheiden sich im wesentlichen in zwei Punkten: zum einen werden bei dem hier zur Anwendung gelangten Verfahren die einzelnen Neuronen des Netzwerkes nicht direkt auf die Prozessoren abgebildet, sondern die Prozessoren berechnen jeweils eine ganze „Spalte“ von Neuronen des Netzwerkes, der zweite Unterschied ermöglicht die parallele Berechnung der Trainingssätze.

Beim parallelen Backpropagation-Verfahren auf der MasPar repräsentiert ein Prozessorelement alle Hidden und Output Units in einer „Spalte“ des Netzwerkes (siehe Abb. 4.1), d.h. die Darstellung des Netzwerkes durch die Prozessoren ist „1-dimensional“ und die Berechnung der Units erfolgt nur für jeweils eine Schicht des Netzwerkes parallel. Die Abbildung von mehreren Netzwerken im Prozessorarray zur parallelen Berechnung der Trainingssätze ist, durch die im Vergleich zum 2-dimensionalen Prozessorarray um eine Dimension niedrigere Netzwerkdarstellung durch die Prozessoren, besonders gut möglich.

[1] Die Parallelisierung der Verbindungsgewichte bietet sich an, falls der lokale Speicher der Prozessoren zu klein zur Speicherung der Gewichte ist. In diesem Fall werden die Verbindungsgewichte, die sonst in einem einzigen Prozessor gespeichert würden, auf mehrere Prozessoren verteilt.

Zur Darstellung eines Netzwerkes wird, mit einer Ausnahme, genau die Anzahl der Prozessoren benötigt, die der maximalen Anzahl der Neuronen in einer Schicht, mit Ausnahme der Eingabeschicht, entspricht, d.h. die „Breite“ des Netzwerkes bestimmt die Menge der Prozessoren. Falls die Eingabeschicht des Netzwerkes genau die Schicht ist, die am meisten Neuronen enthält, übernimmt ein zusätzlicher Prozessor die Speicherung der Eingabemuster. Die Anzahl der benötigten Prozessoren N_{procs} ist bei gegebener Anzahl $N_{nodes}(k)$ der Neuronen in der Ebene k eines Netzwerkes mit N_{layers}-Ebenen für $k = 0, \ldots, N_{layers} - 1$:

$$N_{procs} = \begin{cases} \max N_{nodes}(k) & \text{falls } N_{nodes}(0) \leq \max N_{nodes}(k) \\ & \text{für } k = 1, \ldots, N_{layers} - 1, \\ \max N_{nodes}(k) + 1 & \text{sonst.} \end{cases}$$

Jeder Prozessor speichert lokal die synaptischen Gewichte der Eingangsverbindungen der Neuronen, die einer „Spalte“ des Netzwerkes entsprechen. Wie beim Algorithmus aus Kapitel 3.2.3 sind die Prozessoren in einem Ring angeordnet. Die Anzahl der Prozessorelemente im Ring (der „Durchmesser“ des Rings) sollte jedoch dynamisch konfigurierbar sein, da sich die Anzahl der aktiven Prozessoren bei der Berechnung der einzelnen Schichten des Netzwerkes im allgemeinen ändert [2]. Die einzelnen Prozessoren im Ring verfügen jeweils über

- eine „Ablage“ und einen Akkumulator zur Speicherung von Zwischenergebnissen und
- eine bei vollverbundenen Netzwerken unidirektionale, ansonsten bidirektionale Kommunikationsverbindung mit dem linkem Nachbarprozessor.

Während der Vorwärts- und Rückwärtspropagierung werden die in den „Ablagen“ gespeicherten Zwischenwerte nach jedem Berechnungsschritt zyklisch um eine Stelle nach links verschoben. Die Anzahl der „Ablagen“, die zur Berechnung einer Schicht des Netzwerkes benötigt werden, ist so groß wie das Maximum aus der Anzahl der Neuronen in der Vorgänger- und in der zu berechnenden Schicht. Ein Sonderfall tritt ein, wenn die Eingabeschicht mehr Neuronen enthält als die übrigen Schichten des Netzwerkes, d.h. wenn die Anzahl der benötigten „Ablagen“ größer ist als die Anzahl der zur Darstellung des Netzwerkes verwendeten Prozessorelemente. In diesem Fall speichert bei der Vorwärtspropagierung ein zusätzlicher Prozessor das aktuelle Eingabemuster der Eingabeschicht und übernimmt damit die Funktion der fehlenden „Ablagen“. Die Anzahl der „Ablagen“ $N_{tray}(k)$, die zur Berechnung der ersten Schicht des Netzwerkes benötigt werden, ist wie folgt definiert:

$$N_{tray}(1) = \begin{cases} N_{nodes}(1) + 1 \equiv N_{procs} & \text{falls } N_{nodes}(0) > \max N_{nodes}(k) \\ & \text{für alle } k \in [0, N_{layers} - 1], \\ \max\left(N_{nodes}(0), N_{nodes}(1)\right) & \text{sonst.} \end{cases}$$

Für alle anderen Schichten $k = 2, \ldots, N_{layers} - 1$ des Netzwerkes gilt:

$$N_{tray}(k) = \begin{cases} N_{nodes}(k-1) & \text{falls } N_{nodes}(k-1) > N_{nodes}(k), \\ N_{nodes}(k) & \text{sonst.} \end{cases}$$

[2] Die Anzahl der aktiven Prozessoren bei der Berechnung der Schicht k des Netzwerkes ist $N_{nodes}(k)$ für $k = 1, \ldots, N_{layers} - 1$.

Die Abbildung 4.2 stellt die Vorwärtspropagierung in der ersten Netzwerkschicht schematisch dar. Die Anordnung der Gewichte im lokalen Speicher der Prozessoren ist nicht immer um eine Stelle verschoben, sondern hängt davon ab, ob die Anzahl der Neuronen in der zu berechnenden Schicht größer als in der Vorgängerschicht ist oder nicht. Für den Fall, daß die Anzahl der Neuronen in der Schicht $k-1$ größer oder gleich der Anzahl der Neuronen in der Schicht k ist, d.h. wenn gilt $N_{nodes}(k-1) >= N_{nodes}(k)$, ist die Anordnung der Gewichte im lokalen Speicher der Prozessoren im Vergleich zu der Anordnung im linken Nachbarprozessor um jeweils eine Stelle nach „unten“ verschoben [3], d.h. die Verbindungsgewichte werden mit dem Skew-Faktor 1 gespeichert.

Ist die Anzahl der Neuronen in der Vorgängerschicht $k-1$ jedoch kleiner als die Anzahl der Neuronen in der Schicht k ($N_{nodes}(k-1) < N_{nodes}(k)$), so bedeutet dies, daß zur Berechnung der Schicht k mehr „Ablagen“ (und Prozessorelemente) verwendet werden als es Neuronen in der Vorgängerschicht $k-1$ gibt. Die „Ablagen“ der überzähligen Prozessoren werden dann mit einem besonderen Floating-Point Wert (*Not A Number* oder *Maxfloat*) gefüllt. Dieser spezielle Gleitkommawert signalisiert, daß das Ergebnis einer Gleitkommaberechnung nicht im darstellbaren Wertebereich liegt und führt normalerweise zur Programmunterbrechung. Hier dient dieser Wert dazu, die Prozessorelemente zyklisch an- und auszuschalten: ein Prozessorelement beteiligt sich nur dann an einer Berechnung, wenn seine „Ablage“ nicht den speziellen Gleitkommawert enthält. In den Abbildungen ist dieser Wert durch einen Strich in der Ablage dargestellt.

Die Verbindungsgewichte der Eingangsverbindungen werden für die Prozessoren $1, \ldots, N_{nodes}(k-1)$ mit dem Skew-Faktor 1 gespeichert. Die übrigen an der Berechnung beteiligten Prozessoren $N_{nodes}(k-1)+1, \ldots, N_{nodes}(k)$ speichern die Verbindungsgewichte in der normalen Reihenfolge.

Durch die verschobene Speicherung der Verbindungsgewichte ist die Berechnung von Wx in der Arbeitsphase und die Berechnung von $W^T x$ bei der Fehlerpropagierung ohne den Austausch der Verbindungsgewichte möglich. Die Operationen Wx bzw. $W^T x$ werden am Beispiel des 3-stufigen Netzwerkes aus Abb. 1.1 erläutert.

Die Funktion des parallelen Backpropagation-Verfahrens wird in den folgenden Kapiteln für schichtweise vollvernetzte feedforward-Netzwerke und das Lernverfahren aus [RHW86] (mit *sigmoider* Transferfunktion) erklärt.

4.1.1 Die Vorwärtspropagierung

Die Berechnung der Aktivierung der Units, d.h. die Berechnung von Wx (W ist die Gewichtsmatrix und x der Eingabevektor) in der Arbeitsphase, geschieht folgendermaßen:

Zuerst werden die Eingabewerte x_j^k in den „Ablagen“ der Units gespeichert. Die Eingabewerte x_j^0, d.h die Eingabewerte der Hidden Units aus Layer 1, sind die Elemente des Eingabemusters n, d.h. $x_j^0 = I_j(n)$. Ansonsten stehen die Eingabewerte x_j^k für die Ausgabewerte o_j^k der Units.

[3] Bei einer totalen Vernetzung der Units mit ihren Vorgängerunits.

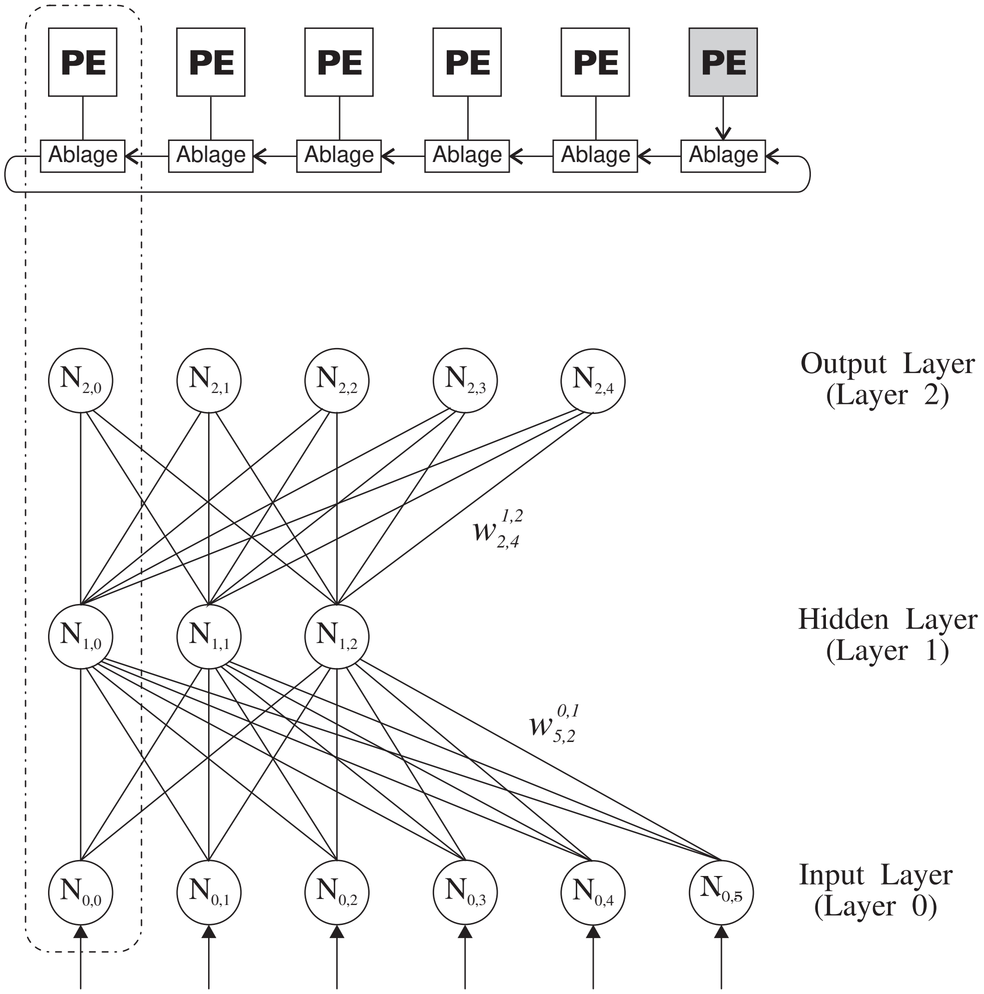

Abbildung 4.1: Die Neuronen und die Verbindungen eines Multilayer Perceptrons werden spaltenweise auf die Prozessoren der MasPar abgebildet. Die Anzahl der benötigten Prozessoren ist gleich der maximalen Anzahl der Neuronen in einer Schicht mit Ausnahme der Eingabeschicht. Falls die Anzahl der Eingabeneuronen größer ist als die Anzahl der Neuronen in den anderen Schichten des Netzwerkes, d.h. wenn $N_{nodes}(0) > \max N_{nodes}(k)$ für $k = 1, \ldots, N_{layers} - 1$, speichert ein weiterer Prozessor das Eingabemuster während der Vorwärtspropagierung von Schicht 0 auf Schicht 1 (in der Abbildung das Prozessorelement ganz rechts).

Während der Berechnung werden die in den „Ablagen" gespeicherten Eingabewerte nach links rotiert. Jeder Prozessor i addiert bei jedem Berechnungsschritt den Wert $w_{j,i}^{k-1,k}\, x_j^{k-1}$ zum Inhalt seines Akkumulators, der mit 0 initialisiert wurde, hinzu. Diese Operation, bestehend aus einer Multiplikation, einer Addition und einer Schiebeoperation, wird $\max(N_{nodes}(k-1), N_{nodes}(k))$-mal wiederholt. Die Berechnung der Netzeingabe

$$net_i^k = \sum_{j=0}^{N_{nodes}(k-1)-1} w_{j,i}^{k-1,k}\, x_j^{k-1}$$

des i-ten Neurons in der k-ten Schicht des Netzwerkes dauert somit nur $\max(N_{nodes}(k-1), N_{nodes}(k))$ Schritte, d.h. die Zeit zur Berechnung der Aktivierung des gesamten Netzwerkes ist, bei ausreichend vielen Prozessoren, proportional zur Anzahl der Neuronen des Netzwerkes [4].

Nach Bestimmung der Netzeingabe net_i^k der Neuronen erfolgt die Berechnung der Aktivierungs- und Ausgangswerte a_i^k und o_i^k der Neuronen durch die Auswertung der *sigmoiden* Transferfunktion [5]. Der Ausgabewert $o_i^k = a_i^k = \sigma(net_i^k + \theta_i^k)$ ist hier identisch mit der Aktivierung a_i^k des Neurons. θ_i^k bezeichnet den Schwellenwert oder *Offset* des Neurons.

Diese Berechnungsschritte werden für die Schichten $1, \ldots, (N_{layers}-1)$ des Netzwerkes wiederholt. Nach $(N_{layers}-1)$ Durchläufen stehen die Ausgangswerte der Ausgangsunits fest. Die Berechnung der Netzeingabe der Ausgabeneuronen ist in Abbildung 4.3 dargestellt.

4.1.2 Die Rückwärtspropagierung

Bei der Propagierung des Fehlers werden zuerst die Fehlerwerte der Neuronen, beginnend mit der Ausgabeschicht, bestimmt. Danach erfolgt die Berechnung der Korrekturwerte für die Verbindungsgewichte und Schwellenwerte.

Die Berechnung der Fehlerwerte der Ausgabeneuronen gestaltet sich recht einfach, da sich der Fehler des Neurons i direkt aus der Differenz zwischen dem Zielwert des Ausgabemusters und dem Ausgangswert des Neurons ergibt:

$$\delta_i^{N_{layers}-1} = o_i^{N_{layers}-1} \cdot (1 - o_i^{N_{layers}-1}) \cdot (T_i(n) - o_i^{N_{layers}-1}).$$

$T_i(n)$ ist der Zielwert des Neurons i beim Muster n des Trainingssatzes.

Für die Bestimmung des Fehlers der Neuronen in den verdeckten Schichten $k = 1, \ldots, (N_{layers}-2)$ des Netzwerkes, wird der Fehlerwert der nachfolgenden Neuronen der Schicht $(k+1)$ gewichtet aufsummiert. Es gilt:

$$\delta_i^k = o_i^k \cdot (1 - o_i^k) \cdot \sum_{j=0}^{N_{nodes}(k+1)-1} w_{i,j}^{k,k+1}\, \delta_j^{k+1}.$$

[4] Die Zeit die zur sequentiellen Berechnung der Aktivierungen des Netzwerkes benötigt wird, wächst linear zur Anzahl der Verbindungen, d.h. im allgemeinen quadratisch zur Anzahl der Neuronen.

[5] Die *sigmoide* Funktion ist hier $\sigma(x) = \frac{1}{1+e^{-x}}$.

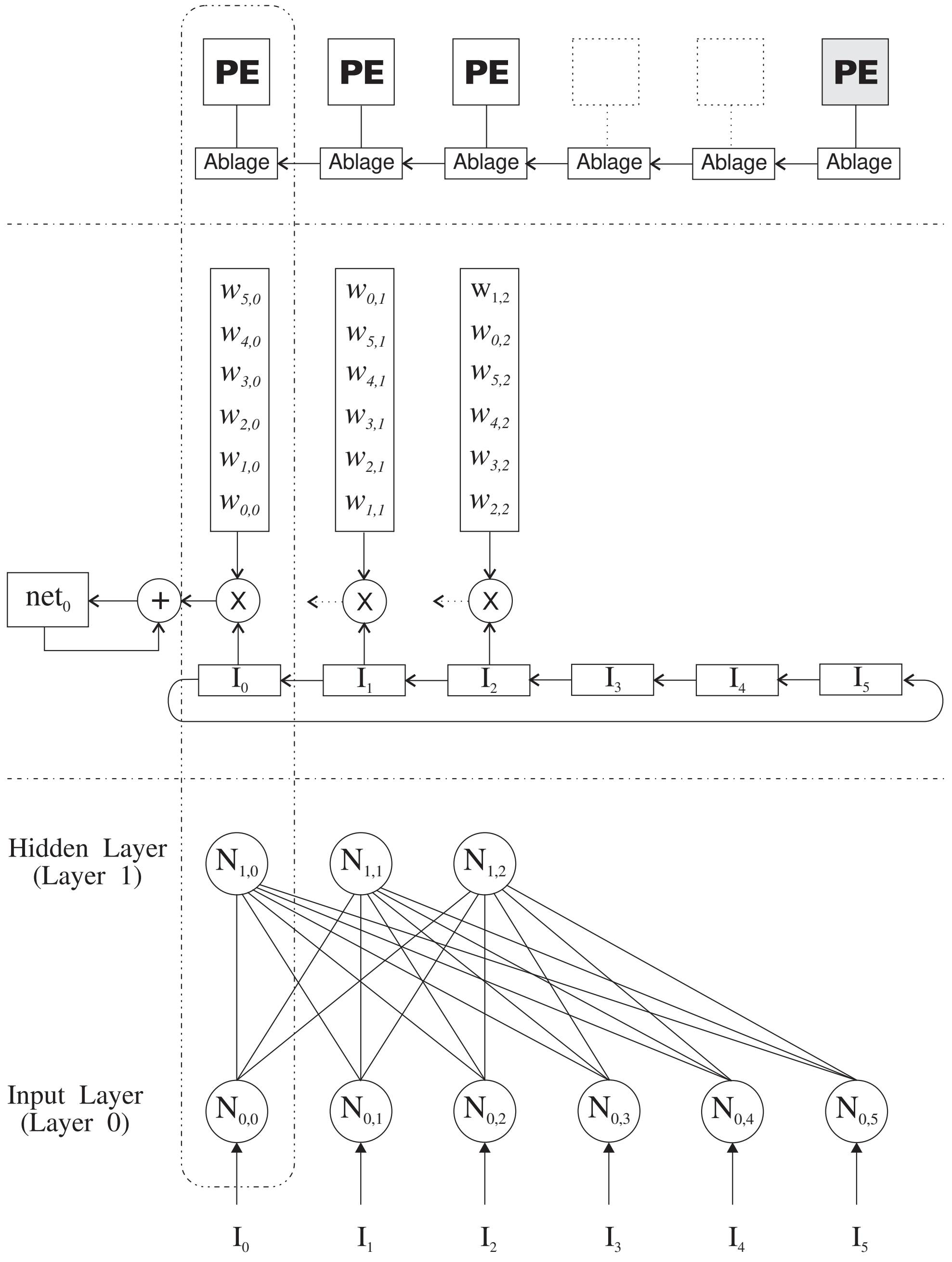

Abbildung 4.2: Die parallele Vorwärtspropagierung und Abbildung der Hidden-Units sowie deren Eingangsverbindungen beim *6-3-5* Netzwerk aus Abb. 1.1 auf die Prozessorelemente der MasPar. Jeder Prozessor berechnet die Netzeingabe der ihm zugeordneten Hidden-Unit i wie folgt: $net_i^1 = \sum_{j=0}^{N_{nodes}(0)-1} w_{j,i}^{0,1} I_j(n)$. Hierbei bezeichnet $N_{nodes}(0)$ die Anzahl der Neuronen im Layer 0, $w_{j,i}^{0,1}$ stellt das synaptische Gewicht der Verbindung zwischen dem Neuron j aus Layer 0 und dem Neuron i aus Layer 1 dar und I_j benennt das Element j des Eingabemusters n.

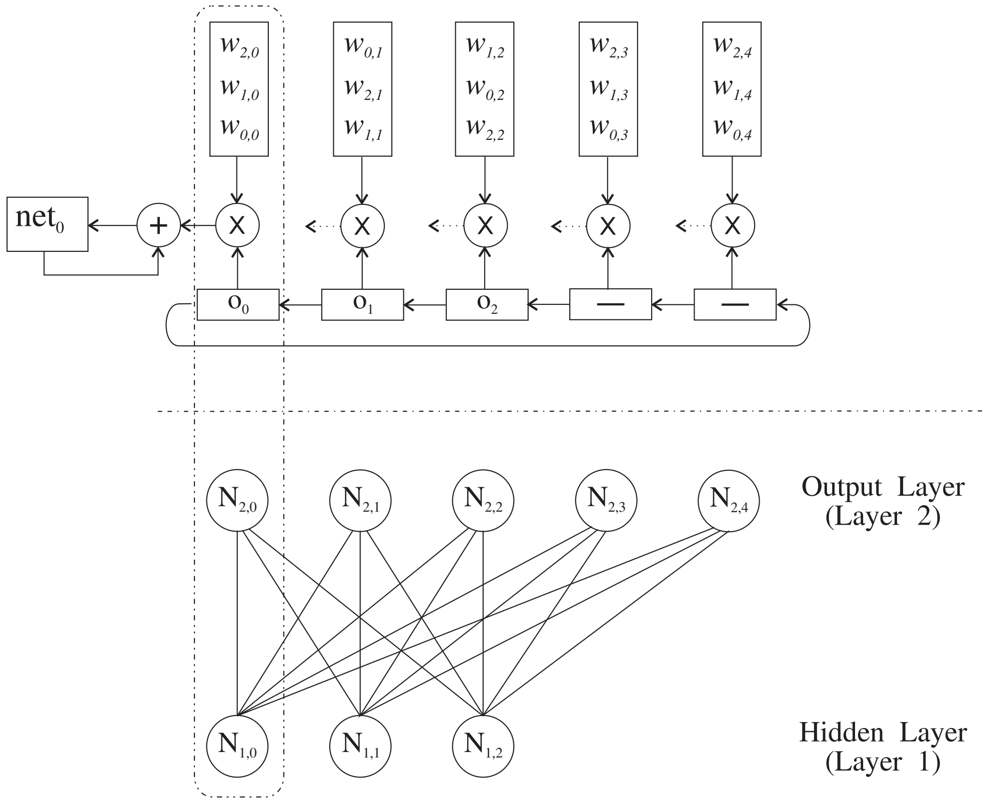

Abbildung 4.3: Die Parallele Arbeitsphase (forward propagation) und Abbildung der Output-Units sowie deren Eingangsverbindungen beim *6-3-5* Netzwerk aus Abb. 1.1 auf die Prozessorelemente der MasPar. Jeder Prozessor berechnet die Netzeingabe der ihm zugeordneten Output-Unit i wie folgt: $net_i^2 = \sum_{j=0}^{N_{nodes}(1)-1} w_{j,i}^{1,2}\, o_j^1$. Hierbei bezeichnet $N_{nodes}(1)$ die Anzahl der Neuronen im Layer 1, $w_{j,i}^{1,2}$ stellt das synaptische Gewicht der Verbindung zwischen dem Neuron j aus Layer 1 und dem Neuron i aus Layer 2 dar, und o_j^1 benennt den Ausgabewert des Neurons j aus Layer 1.

Die Summation der $w_{i,j}^{k,k+1}\,\delta_j^{k+1}$ erfolgt wie in der Arbeitsphase durch Rotation der Elemente in den „Ablagen“ der Prozessoren. Im Gegensatz zur Arbeitsphase werden jedoch nicht die Eingabewerte rotiert, sondern die bei der Berechnung entstehenden Partialsummen (siehe Abbildung 4.4). Nach $\max(N_{nodes}(k), N_{nodes}(k+1))$-Durchläufen sind alle Partialsummen berechnet. Nach der Berechnung der Fehlerwerte δ_i^k kann die Fehlerpropagierung für die nächst-tieferliegende Schicht erfolgen. Es ist zu beachten, daß die Berechnung der Fehlerwerte durch Summation nur für die Neuronen aus den inneren Schichten des Netzwerkes erforderlich ist. Die Abbildung 4.4 zeigt die Fehlerpropagierung am Beispiel des *6-3-5* Netzwerkes.

4.1.3 Die Modifikation der Verbindungsgewichte

Nachdem die Fehlerwerte berechnet sind, können die Verbindungsgewichte und Schwellenwerte für alle Schichten $k = 1, \ldots, (N_{layers} - 1)$ justiert werden. Die Änderung des synaptischen Gewichts $\Delta w_{j,i}^{k-1,k}$ der Verbindung des Neurons j aus der Schicht $(k-1)$ mit dem Neuron i der Schicht k berechnet sich wie folgt:

$$\Delta w_{j,i}^{k-1,k} = \eta \cdot \delta_i^k \cdot o_j^{k-1}.$$

Die Änderung $\Delta\theta_i^k$ des Schwellenwertes θ_i^k des Neurons i ist:

$$\Delta\theta_i^k = \eta \cdot \delta_i^k.$$

η ist dabei der Lernfaktor und o_j^{k-1} ist der Ausgangswert des Neurons j aus der tieferliegenden Schicht des Netzwerkes. Ist $k = 1$, so gilt $o_j^0 = I_j(n)$, d.h. o_j^0 ist das Element j des n-ten Eingabemusters.

Die Berechnung der Gewichtsänderungen $\Delta w_{j,i}^{k-1,k}$ geschieht wie in der Arbeitsphase durch die Rotation der Ausgangswerte o_j^{k-1} in den „Ablagen“ der Prozessorelemente. Die Abbildungen 4.5 und 4.6 zeigen den Vorgang für die verdeckte Schicht und die Ausgabeschicht des Beispielnetzwerkes.

4.2 Parallele Berechnung verschiedener Trainingssätze

Normalerweise benötigt die Simulation eines Netzwerkes weniger Prozessoren als die MP-1 zur Verfügung stellt. Um möglichst viele Prozessoren auszulasten, werden alle Phasen des Backpropagation-Verfahrens, d.h. die Vorwärts- und die Rückwärtspropagierung, die Berechnung der Gewichtsänderungen sowie die Modifikation der Verbindungsgewichte, auf m Kopien des Netzwerkes für m verschiedene Trainingssätze parallel durchgeführt.

In den Netzwerkkopien werden die Verbindungsgewichte nicht nach jeder Lernphase geändert, sondern erst nach der Abarbeitung des gesamten Trainingssatzes. Dazu werden in den Netzwerkkopien die jeweiligen Gewichts- und Schwellenwertänderungen für die verschiedenen Trainingssätze aufsummiert. Nachdem die Berechnung der Trainingssätze abgeschlossen ist, erfolgt die Modifikation der Gewichte und Schwellenwerte durch die Addition der Änderungen Δw bzw. $\Delta\theta$ auf alle Netzwerkkopien. Dies entspricht der Änderung der Verbindungsgewichte nach jeweils m Trainingsmustern in einem einzigen Netzwerk.

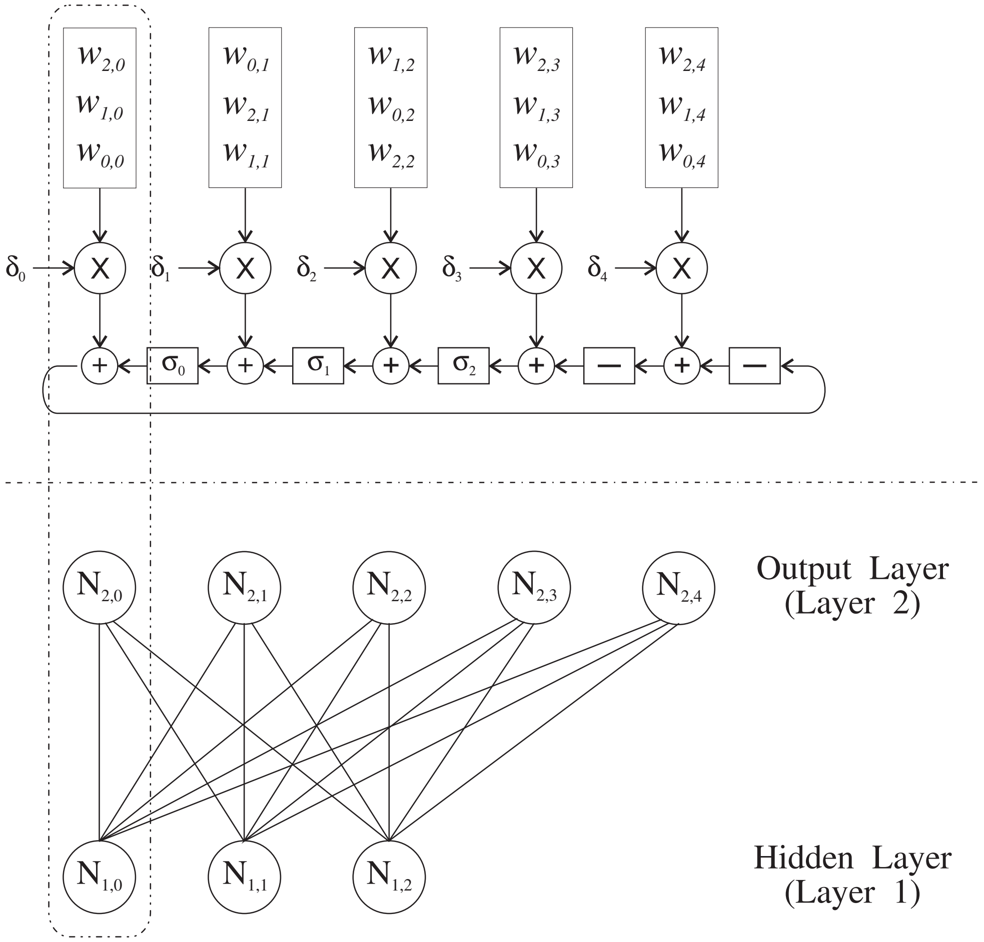

Abbildung 4.4: Die Parallele Berechnung der Fehlerwerte der inneren Neuronen und Abbildung der Ausgabeneuronen sowie deren Eingangsverbindungen beim *6-3-5* Netzwerk aus Abb. 1.1 auf die Prozessorelemente der MasPar. Jeder Prozessor berechnet den Fehlerwert einer Hidden-Unit i durch die gewichtete Summation des Fehlers der mit dieser Unit verbundenen Output-Units j wie folgt: $\delta_i^1 = o_i^1(1 - o_i^1)\sum_{j=0}^{N_{nodes}(2)-1} w_{i,j}^{1,2}\,\delta_j^2$. Hierbei bezeichnet $N_{nodes}(2)$ die Anzahl der Ausgabeneuronen, $w_{i,j}^{1,2}$ stellt das synaptische Gewicht der Verbindung zwischen dem Neuron i aus Layer 1 und dem Neuron j aus Layer 2, der Ausgabesschicht, dar. Den Fehler der Ausgabeneuronen j bezeichnet δ_j^2 und o_i^1 benennt den Ausgabewert der Unit i aus Layer 1.

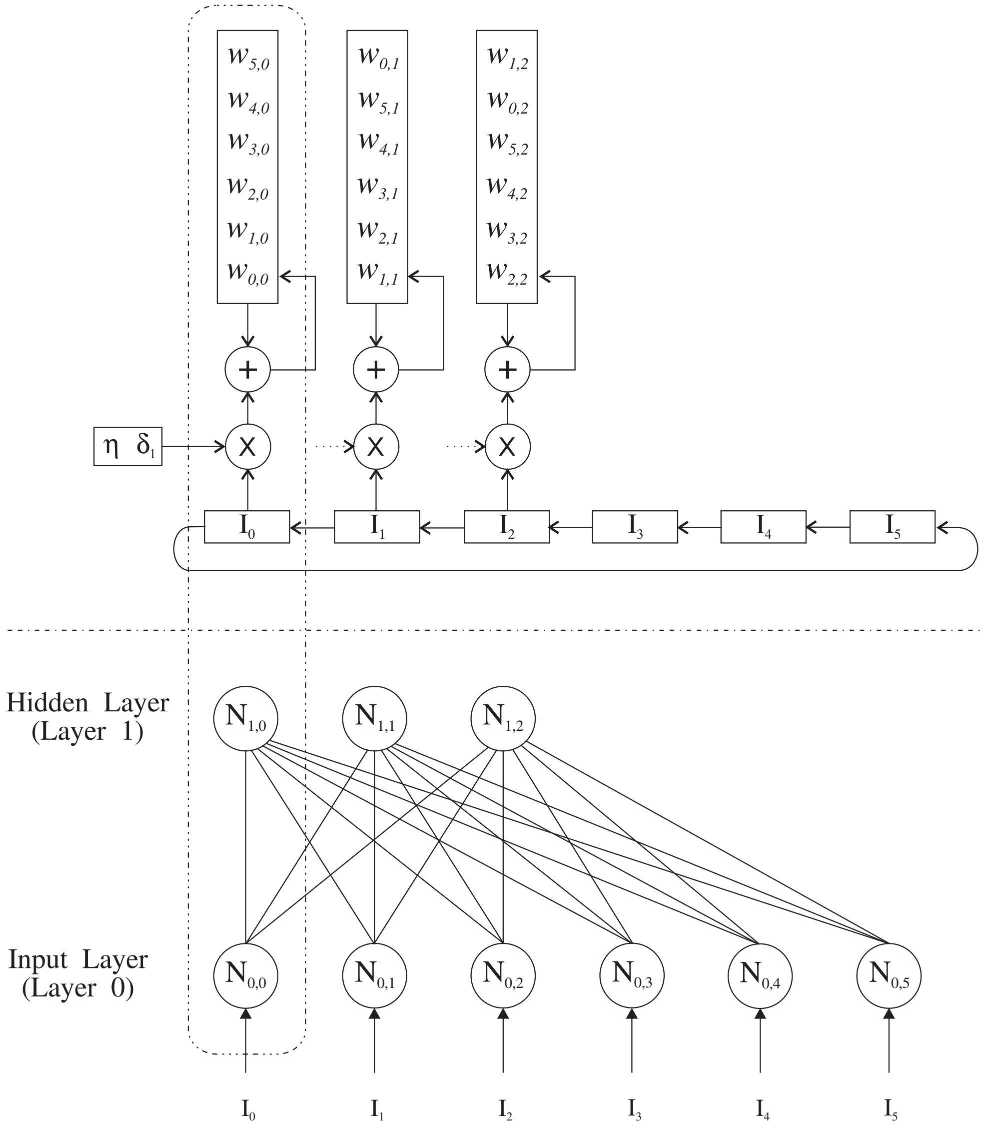

Abbildung 4.5: Die Berechnung der Gewichtsänderungen $\Delta w_{j,i}^{0,1} = \eta \cdot \delta_i^1 \cdot I_j$ der synaptischen Gewichte $w_{j,i}^{0,1}$ der Verbindungen der Eingabeneuronen j aus Schicht 0 mit den Neuronen i aus Schicht 1 des *6-3-5* Netzwerkes aus Abb. 1.1.

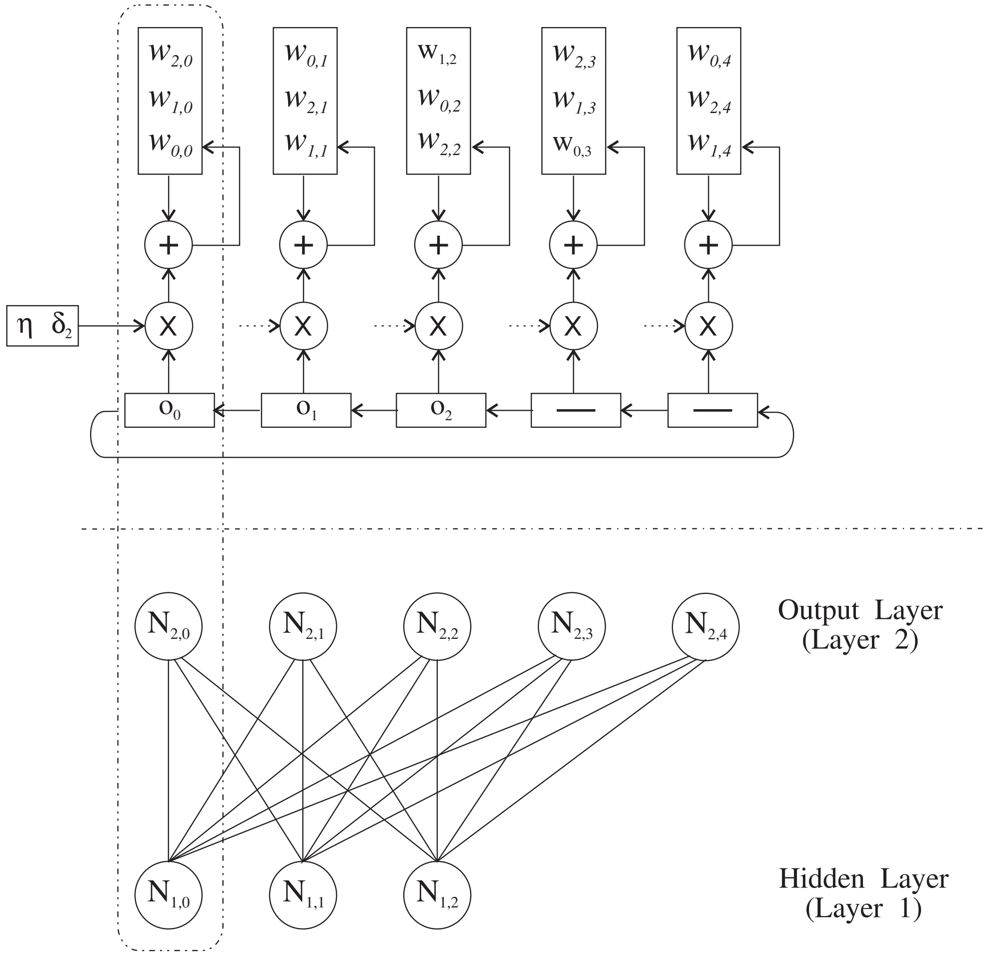

Abbildung 4.6: Die Berechnung der Gewichtsänderungen $\Delta w_{j,i}^{1,2} = \eta \cdot \delta_i^2 \cdot o_j^1$ der synaptischen Gewichte $w_{j,i}^{1,2}$ der Verbindungen der Neuronen j aus Schicht 1 mit den Ausgabeneuronen i aus Schicht 2 des *6-3-5* Netzwerkes.

Kapitel 5

Aspekte der Implementierung

5.1 Abbildung des Netzwerkes auf die Prozessoren der MasPar

Der parallele Backpropagation-Algorithmus mußte aufgrund einer, durch die Hardware bedingten, Fehlfunktion im globalen Router der MasPar, die während der Implementierungsphase dieser Arbeit nicht behoben werden konnte, modifiziert werden. Die Änderung des Backpropagation-Verfahrens beeinflußt vor allem die Anordnung der einzelnen Netzwerke im Array der Prozessorelemente. Die Netzwerke werden in der derzeitigen Implementierung des Backpropagation Verfahrens immer orthogonal zueinander angeordnet. Dadurch sinkt in ungünstigen Fällen die Auslastung des Prozessorarrays ganz erheblich [1]. Dieser Nachteil wird jedoch durch die schnellere *Xnet*-Kommunikation zwischen benachbarten Prozessoren, die durch die orthogonale Anordnung der Prozessorelemente die *Router*-Kommunikation ersetzen kann, teilweise wieder ausgeglichen. Bei einem zweidimensionalen $N_{PExprocs} \times N_{PEyprocs}$ Prozessorarray ist die Anzahl $N_{networks}$ der Netzwerke, die parallel berechnet werden können, wie folgt definiert:

$$N_{networks} = \begin{cases} int\left(\frac{N_{PExprocs}}{N_{procs}}\right) \cdot N_{PEyprocs} & \text{falls } N_{procs} <= N_{PEyprocs}, \\ int\left(\frac{N_{PEyprocs}}{int\left(\frac{N_{PExprocs}}{N_{procs}-1}+1\right)}\right) & \text{sonst.} \end{cases}$$

Die beiden folgenden Abbildungen veranschaulichen die Darstellung der Netzwerke im Prozessorarray der MasPar sowie die Belegung der Prozessorelemente mit den einzelnen Netzwerkkopien.

[1] Im ungünstigsten Fall beträgt die Auslastung der Prozessoren: $\frac{1}{2} + \frac{2}{N_{PExprocs}}$, hierbei bezeichnet $N_{PExprocs}$ die Anzahl der Prozessoren in X-Richtung, d.h. die „Breite“ des Prozessorarrays. Bei einer MasPar MP-1216 mit 128^2 Prozessoren ist die Auslastung im schlechtesten Fall circa 51.6 %.

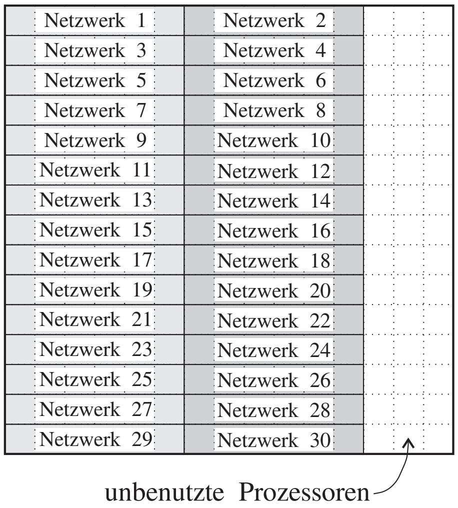

Abbildung 5.1: Ist die Anzahl der zur Darstellung eines Netzwerkes verwendeten Prozessorelemente kleiner oder gleich der Breite des Prozessorarrays, d.h. $N_{procs} \leq N_{PExprocs}$, so werden die einzelnen Netzwerke linksbündig und ohne Umbruch im Prozessorarray angeordnet.

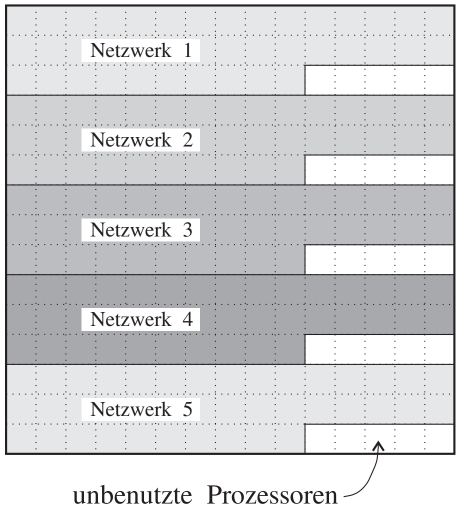

Abbildung 5.2: Ist die Anzahl der zur Darstellung eines Netzwerkes verwendeten Prozessorelemente größer als die Breite des Prozessorarrays ($N_{procs} > N_{PExprocs}$), werden die Netzwerke linksbündig im Prozessorarray angeordnet.

5.2 Die Schnittstellen des Simulatorkerns

Beim Entwurf des parallelen Netzwerksimulators wurde der Forderung nach Erweiterbarkeit in Bezug auf andere Netzwerktopologien besonders Rechnung getragen. Ein weiteres wichtiges Kriterium bei der Entwicklung des Simulators war die Kompatibilität der Anwenderschnittstellen des parallelen Simulatorkerns mit denen des sequentiellen Simulatorkerns. Die Schnittstellen des parallelen Simulatorkerns sind daher abwärtskompatibel mit den Schnittstellen des Simulatorkerns der Version 2.0. Um die zukünftige Erweiterung des parallelen bzw. sequentiellen Simulatorkerns zu erleichtern, wurde die funktionale Anwenderschnittstelle vollständig reimplementiert und eine weitere Schnittstelle zwischen der Anwenderschnittstelle und den Funktionen des Simulatorkerns eingefügt. Über diese interne Schnittstelle wird zum einen die Kommunikation zwischen dem sequentiellen und dem parallelen Teil des Simulators abgewickelt und zum anderen setzt die Anwenderschnittstelle auf dieser internen Schnittstelle auf. In Abbildung 5.3 sind die Schnittstellen des Simulatorkerns dargestellt.

5.2.1 Die erweiterte Anwenderschnittstelle des parallelen Simulatorkerns

Die Anwenderschnittstelle des parallelen Simulatorkerns ist im Vergleich zur Version 2.0 des sequentiellen Simulators um einige wenige Funktionen erweitert

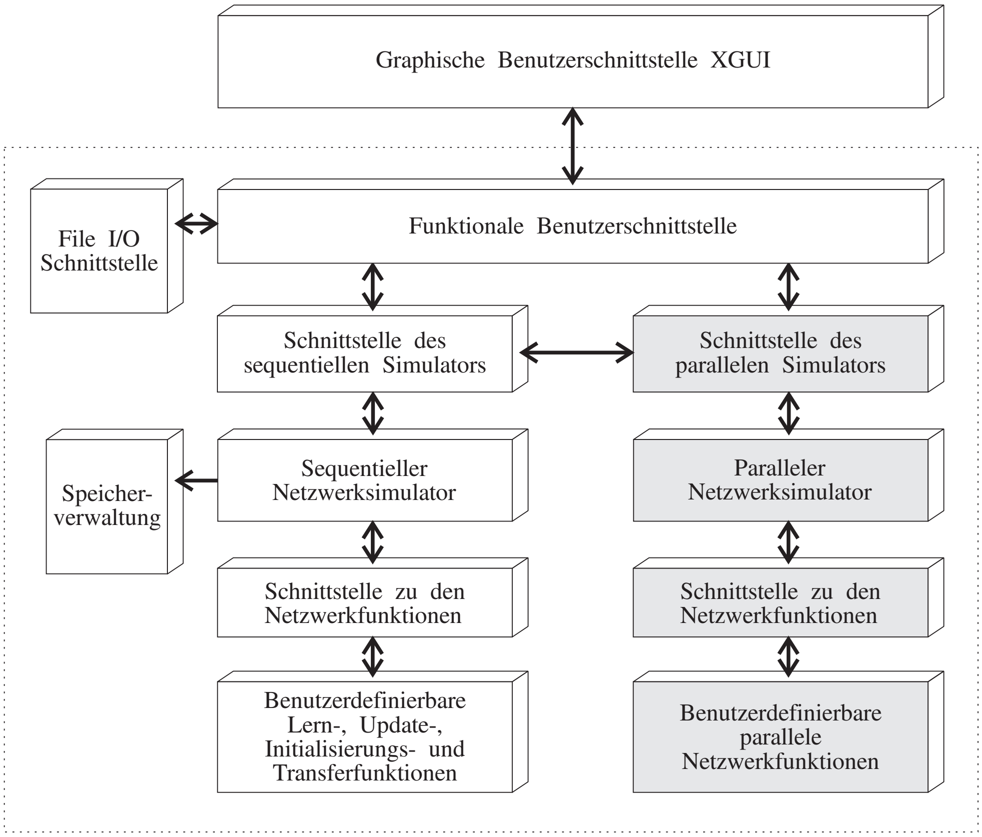

Abbildung 5.3: Die Schnittstellen des Netzwerksimulators.

worden. Die zusätzlichen Funktionen sind sowohl in dem parallelen als auch in dem sequentiellen Modus des Netzwerksimulators verfügbar. Die wichtigste dieser Funktionen dient dazu, zwischen der sequentiellen und der parallelen Betriebsart des Simulators hin- und herzuschalten. Die übrigen Erweiterungen sind Funktionen zur Erzeugung und Verwaltung der Ein- und Ausgabemuster des Netzwerkes. Im folgenden Abschnitt werden die neu hinzugekommenen Schnittstellenfunktionen beschrieben.

`krui_err  krui_setSpecialNetworkType( int  net_type )`
schaltet den Simulatorkern in eine spezielle, durch `net_type` bestimmte, Betriebsart. Zur Zeit gibt es zwei Betriebsarten:

- `NET_TYPE_GENERAL`
  ist die normale, sequentielle Betriebsart des Netzwerksimulators. In dieser Betriebsart ist die Simulation von Netzwerken mit beliebiger Topologie möglich.

- `NET_TYPE_MASPAR_FF1`
  ist die parallele Betriebsart des Netzwerksimulators und ermöglicht die Simulation von schichtweise vernetzten Multilayer Perceptons auf der MasPar MP-1.

Beim Umschalten in den parallelen Modus des Simulators werden die Transferfunktionen der Neuronen sowie die Topologie des Netzwerkes überprüft. Falls der parallele Simulatorkern die Topologie des Netzwerks nicht abbilden kann, oder falls die Transferfunktion eines Neurons dem parallelen Kern unbekannt ist, gibt die Funktion einen Fehlercode zurück. Die Fehlercodes können mit der Funktion `krui_error( int error_code )` in einen Zeichenkette umgewandelt werden. Zukünftige Erweiterungen des Simulatorkerns können mit der Schnittstellenfunktion `krui_setSpecialNetworkType( ... )` recht gut in den vorhandenen Kern eingebunden werden. Mittels dieser Funktion könnte beispielsweise der parallele Simulator zur Simulation anderer Netzwerktopologien umgeschaltet werden.

`int  krui_getSpecialNetworkType( void )`
liefert die aktuelle Betriebsart des Netzwerksimulators. Es sind zwei Rückgabecodes möglich:

- `NET_TYPE_GENERAL` und

- `NET_TYPE_MASPAR_FF1`.

Die Rückgabecodes haben dieselbe Bedeutung wie die Codes zur Umschaltung der Betriebsarten des Simulators.

```
krui_err  krui_newVectorPatternPair( float  *input_patterns,
                                     int  no_of_input_units,
                                     float  *output_patterns,
                                     int  no_of_output_units )
```
erzeugt ein neues Trainingsmuster durch das Kopieren der gegegebenen Arrays in die internen Musterarrays des Simulators. Befindet sich bereits ein Netzwerk im Speicher, so muß die Anzahl der Ein- und Ausgabeneuronen des Netzwerks mit der Größe des Ein- und Ausgabemusters übereinstimmen.

```
krui_err  krui_getPatternArrays( float  * *input_patterns,
                                 int  *no_of_input_patterns,
                                 float  * *output_patterns,
                                 int  *no_of_output_patterns )
```
ermittelt die Startadressen und die Größe der internen Musterarrays. Mit Hilfe dieser Funktion können die durch den Simulator verwalteten Musterarrays vom Anwendungsprogramm direkt modifiziert werden. Vor der Modifikation der Trainingsmuster durch das Anwendungsprogramm sollten die Adressen der Musterarrays durch den Aufruf von `krui_getPatternArrays(...)` jedesmal neu bestimmt werden, da sich die Startadressen nach der Anforderung bzw. Freigabe von Trainingsmustern ändern können.

```
krui_err  krui_allocatePatterns( int  no_of_input_patterns,
                                 int  no_of_output_patterns,
                                 int  no_of_pattern_pairs )
```
fordert vom Simulatorkern Speicher für neue Trainingsmuster an. Die Muster können nach der Ermittlung ihrer Adressen mit `krui_getPatternArrays(...)` direkt modifiziert werden.

```
krui_err  krui_setNoOfPatterns( int  no_of_pattern_pairs )
```
übermittelt dem Simulator die Anzahl der insgesamt verfügbaren Trainingsmuster. Mit dieser Funktion können nach der Anforderung zusätzlicher Muster mit `krui_allocatePatterns(...)` die angeforderten Muster den Netzwerkfunktionen zugänglich gemacht werden.
Die meisten Schnittstellenfunktionen liefern einen Rückgabewert vom Typ `krui_err` zurück. Dieser Rückgabewert ist auch in der globalen Variable `KernelErrorCode` verfügbar. Falls der Rückgabecode ungleich `KRERR_NO_ERROR` ist, trat während der Ausführung der Funktion ein Fehler auf. Der Fehlercode kann mittels der Funktion `krui_error( int error_code )` in eine Zeichenkette umgewandelt werden. Im Anhang A sind die Fehlermeldungen des Simulatorkerns aufgelistet.

5.3 Restriktionen der Netzwerktopologie

Effiziente Algorithmen zur massiv parallelen Simulation von neuronalen Netzwerken sind im allgemeinen nur zur Simulation bestimmter Netzwerktopologien brauchbar. Die im Rahmen dieser Arbeit entwickelten parallelen Algorithmen sind aus diesem Grund ausschließlich zur Simulation von feedforward-Netzwerken geeignet. Die Topologie des Netzwerkes unterliegt jedoch weiteren

Beschränkungen, die zum einen durch die Hardware des verwendeten Rechnersystems und zum anderen durch die Implementierung der Algorithmen bedingt werden.
Die Topologie der Netzwerke unterliegt in der derzeitigen Implementierung des parallelen Simulators den folgenden Einschränkungen:

- Es ist ausschließlich die Simulation von vorwärtsgekoppelten (feedforward) Netzwerken möglich.
- Es sind nur Verbindungen von Neuronen in benachbarten Schichten möglich, Verbindungen über mehrere Schichten hinweg sind unzulässig. Die Erweiterung des Simulators zur Simulation von beliebig vernetzten feedforward-Netzwerken ist mit geringem Aufwand möglich und wurde bereits bei der Definition der Datentypen berücksichtigt.
- Das Netzwerk darf aus maximal acht Schichten bestehen (inclusive der Eingabe- und der Ausgabeschicht).
- Die Simulation von teilweise vernetzten Netzwerken ist möglich, jedoch ist der Zeitverbrauch zur Berechnung eines teilweise vernetzten Netzwerkes genau gleich wie die Zeit, die zur Berechnung eines vollvernetzten Netzwerkes benötigt wird, d.h. die Effizienz bei der Simulation des Netzwerkes sinkt.
- Die Anzahl der Verbindungen zwischen den Neuronen in einer „Spalte“ des Netzwerkes ist auf etwa 2000 Verbindungen beschränkt. Diese Einschränkung ist durch den maximal 16384 Byte großen lokalen Speicher der Prozessorelemente bedingt. Die Beschränkung der Verbindungsanzahl bedeutet für vollvernetzte, dreischichtige Netzwerke eine maximale Größe von etwa 660 Neuronen pro Schicht, d.h. die maximale Anzahl der Verbindungen in einem solchen Netzwerk liegt etwa bei 870.000 Verbindungen.

Im nächsten Abschnitt werden Lösungsvorschläge zur Aufhebung dieser Beschränkungen diskutiert.

5.4 Erweiterungen des parallelen Simulators

Die Erweiterung des parallelen Simulators könnte auf zwei Arten erfolgen. Die erste Möglichkeit zur Erweiterung des Simulators ist die Beseitigung der obengenannten Beschränkungen der Topologie der Netzwerke. Es ist möglich den Simulator um spezielle Algorithmen für teilweise vernetzte Netzwerke zu erweitern, sodaß sich die Effizienz bei der Berechnung des Netzwerkes nicht reduziert. Der Unterschied zwischen den Algorithmen für teilweise vernetzte Netzwerke und den in Kapitel 4 besprochenen Verfahren für vollvernetzte Netzwerke liegt in der Art und Weise, wie die Inhalte in den „Ablagen“ der Prozessorelemente rotiert werden. Bei vollvernetzten Netzwerken werden die Werte in den „Ablagen“ der Prozessorelemente immer nach links rotiert, während bei teilweise vernetzten Netzwerken die Inhalte in den „Ablagen“ abwechselnd je einmal nach links und nach rechts rotiert werden können. Die Anzahl der Berechnungsschritte verkürzt sich dadurch.

Auch die Begrenzung der maximalen Anzahl der Verbindungen des Netzwerkes kann durch die Speicherung der Verbindungsgewichte in mehreren Prozessorelementen umgangen werden. Jedoch erfordert die Erweiterung der besprochenen Algorithmen durch Verfahren, die die Parallelität der Verbindungen ausnutzen, einen nicht unerheblichen Implementierungsaufwand.
Die zweite Möglichkeit zur Erweiterung des Simulators ist der Ausbau der parallelen Netzwerkfunktionen zur Simulation anderer Netzwerktopologien, z.B. zur Simulation der selbstorganisierenden Karten von Kohonen.

5.5 Beschränkungen der MasPar MP-1

Die Entwicklung leistungsfähiger, paralleler Algorithmen wird sehr durch die Hardwarestruktur des parallelen Rechnersystems beeinflusst. Im folgenden werden die Eigenschaften der MasPar MP-1 genannt, die beim Entwurf paralleler Algorithmen maßgebend sind.

Die wichtigste Vorgabe bei der Entwicklung paralleler Algorithmen ist die Kommunikationsstruktur des Rechnersystems. Die MasPar MP-1 verfügt über ein 2-dimensionales Kommunikationsnetz und einen Router zur Kommunikation beliebiger Prozessoren. Für die Simulation von neuronalen Netzwerken wäre ein höherdimensionales (mindestens 3-dimensionales) Kommunikationsnetzwerk der Prozessoren wesentlich besser geeignet. Zwar ist die Emulation beliebiger Kommunikationsstrukturen durch Software möglich, jedoch geht dieses Vorgehen sehr stark zu Lasten der Ausführungsgeschwindigkeit. Die „1-dimensionale" Darstellung der feedforward-Netzwerke durch die Prozessorelemente der MasPar ist daher eine Konsequenz aus der 2-dimensionalen Kommunikationsstruktur. Eine weitere Beschränkung ergibt sich aus der Größe des lokalen Speichers der Prozessorelemente, da die maximale Größe des lokalen Speichers mit 16K Byte RAM für die Simulation von neuronalen Netzwerken recht knapp bemessen ist.

Eine weit gravierendere Einschränkung ist die langsame Geschwindigkeit des Datentransfers zwischen der MasPar und dem Subsystem. Diese wirkt sich maßgeblich auf die Verarbeitungsgeschwindigkeit der Maschine, in Bezug auf die Simulation von neuronalen Netzwerken, aus. Für den Datenaustausch zwischen der MasPar und dem Subsystem gibt es eine ganze Reihe von Funktionen. Die schnellste Methode des Datenaustausches ist der sogenannte *Direkt Memory Access (DMA)*. Bei dieser Art von Datentransfer greift eine spezielle Hardware auf den Arbeitsspeicher des Subsystems bzw. den Speicher der Prozessorelemente der MasPar zu und überträgt, ohne Zutun der Prozessoren der Rechnersysteme, selbstständig die Daten zwischen den beiden Systemen. Die hiermit erzielbaren Transferraten sind relativ hoch und belaufen sich auf einige Megabyte in der Sekunde. Leider ist die Latenzzeit vom Aufruf der DMA-Funktion bis zum Beginn der Datenübertragung sehr lange und beträgt einige Millisekunden [2]. Da die Trainingsmuster des Netzwerkes aus Platzgründen weder in der

[2] Die Zeit, die für die Initiierung der Datenübertragung benötigt wird, ist größeren Schwan-

Array Control Unit (ACU) noch in den Prozessorelementen gespeichert werden können, müssen die Muster vom Subsystem in die Prozessorelemente transferiert werden. Für den Transfer der Ein- und Ausgabemuster werden in der derzeitigen Implementierung zwei Funktionsaufrufe der DMA-Transferfunktion pro Netzwerkkopie benötigt. Zum Beispiel nimmt die Übertragung für ein einziges Trainingsmuster pro Netzwerkkopie bei 128 Netzwerkkopien etwa eine Sekunde Zeit in Anspruch. Die Übertragungszeit für die Muster steht daher in keinem Verhältnis zu dem Zeitbedarf, der für die Berechnung der Netzwerke benötigt wird [3]. Glücklicherweise kann diese Übertragungszeit jedoch noch deutlich reduziert werden, indem man im Subsystem die Trainingsmuster derart aufbereitet, daß die Ein- bzw. Ausgabemuster der Netzwerkkopien mit insgesamt nur zwei Aufrufen der Transferfunktion übertragen werden. Dazu ist es notwendig die Aufbereitung der Muster im Subsystem parallel zu der Berechnung der Netzwerke auf der MasPar vorzunehmen. Für dieses Verfahren lagen zum Zeitpunkt der Abgabe dieser Arbeit noch keine Leistungsmessungen vor, da an der Implementierung noch gearbeitet wurde.

5.6 Leistungsmessungen

In den folgenden Tabellen wird die Performance verschiedener leistungsfähiger Netzwerksimulatoren mit der Verarbeitungsgeschwindigkeit des hier beschriebenen Simulators für neuronale Netzwerke auf der MasPar MP-1216A verglichen. Die Werte für die Connections bzw. Connection Updates pro Sekunde sind einmal für das NETtalk Netzwerk aus [SR86] mit 203 Eingangsneuronen, 60 Neuronen in der verdeckten Schicht und 26 Neuronen in der Ausgabeschicht, angegeben. Die übrigen Leistungsangaben beziehen sich auf die maximale Leistung der Rechnersysteme. Die Maximalleistung der MasPar MP-1216A wurde mit einem vollvernetzten 128-128-128 Netzwerk ermittelt. Die Angaben der Gewichtsänderungen pro Sekunde (CUPS) beruhen auf dem Lernverfahren nach [RHW86].

Die Leistungsmessungen auf der MasPar MP-1216A beinhalten aufgrund der in Kapitel 5.5 erläuterten Einschränkungen nicht die Transferzeit der Trainingsmuster.

kungen unterworfen. Die Angabe bezieht sich auf die mittlere Latenzzeit.

[3] Die Forwärtspropagierung und die Berechnung der Fehlerwerte eines 128-128-128 Netzwerks benötigt etwa $6 \cdot 10^{-3}$ Sekunden.

Rechnertyp	**Max. Leistung**
VAX 8600	2.0
Convex C-1	3.6
SAIC SIGMA-1	5-8
Butterfly (64)	8
TRW MK V (16)	10
ANZA plus	10
Cray X-MP	50
MasPar MP-1216A	176
CM-2 [a]	180

[a]Implementierung nach [Zha90]

Tabelle 5.1: Die maximale Leistung verschiedener Rechnersysteme in Millionen Verbindungen pro Sekunde (Connections Per Second, CPS).

Rechnertyp	**NETtalk**	**Max. Leistung**
ANZA plus	-	1.5
NeuroTurbo	-	2
Delta	-	2.7
NeuMan	-	8.5
CM-1	2.8	13
WARP	17	17
CM-2 [a]	-	40
MasPar MP-1216A	42	68
Sandy/6 [b]	118	141
SX-2	72	180
Sandy/8 [b]	118	567

[a]Implementierung nach [Zha90]
[b]Spezielle Hardware zur Simulation von neuronalen Netzwerken

Tabelle 5.2: Die Leistung verschiedener Rechnersysteme in Millionen Gewichtsänderungen pro Sekunde (Connection Updates Per Second, CUPS).

Anhang A

Die Fehlermeldungen des Simulatorkerns

Ein Großteil der Schnittstellenfunktionen des Simulatorkerns übergibt als Funktionswert einen Fehlercode. Es sind folgende Fehlermeldungen möglich:

```
KRERR_INSUFFICIENT_MEM:  Insufficient memory
KRERR_UNIT_NO:           Invalid unit number
KRERR_OUTFUNC:           Invalid unit output function
KRERR_ACTFUNC:           Invalid unit activation function
KRERR_SITEFUNC:          Invalid site function
KRERR_CREATE_SITE:       Creation of sites isn't permitted
  because unit has direct input links
KRERR_ALREADY_CONNECTED: Creation of a link isn't permitted
  because there exists already a link between these units
KRERR_CRITICAL_MALLOC:   Memory allocation failed during
  critical operation. Have lost some pointers, but
  consistency of the network is guaranteed
KRERR_FTYPE_NAME:        Ftype name isn't definite
KRERR_FTYPE_ENTRY:       Current Ftype entry isn't defined
KRERR_COPYMODE:          Invalid copy mode
KRERR_NO_SITES:          Current unit doesn't have sites
KRERR_FROZEN:            Can't update unit because
  unit is frozen
KRERR_REDEF_SITE_NAME:   Redefinition of site name
  isn't permitted (site name already exists)
KRERR_UNDEF_SITE_NAME:   Site name isn't defined
KRERR_NOT_3D:            This is not a 3D-Kernel
KRERR_DUPLICATED_SITE:   This unit has already a site
  with this name
KRERR_INUSE_SITE:        Can't delete site table entry
  because site is in use
```

```
KRERR_FTYPE_SITE:        Current Ftype site isn't defined
KRERR_FTYPE_SYMBOL:      Ftype symbol isn't defined
KRERR_IO:                I/O error:
KRERR_SAVE_LINE_LEN:     Creation of output file failed
  (line length limit exceeded)
KRERR_FEW_LAYERS:        The network has not enough layers:
KRERR_NO_UNITS:          No Units defined
KRERR_EOF:               Unexpected EOF
KRERR_LINE_LENGTH:       Line length exceeded
KRERR_FILE_FORMAT:       Incompatible file format
KRERR_FILE_OPEN:         Can't open file
KRERR_FILE_SYNTAX:       Syntax error at line
KRERR_MALLOC1:           Memory allocation error #01
KRERR_TTYPE:             Invalid topologic type
KRERR_SYMBOL:            Symbol pattern invalid
  (must match [A-Za-z]^[|, ]*)
KRERR_NO_SUCH_SITE:      Current unit doesn't have a
  site with this name
KRERR_NO_HIDDEN_UNITS:   No hidden units defined
KRERR_CYCLES:            Network contains cycle(s):
KRERR_DEAD_UNITS:        Network contains dead unit(s):
KRERR_INPUT_PATTERNS:    Pattern file contains not the
  same no. of input units as the network
KRERR_OUTPUT_PATTERNS:   Pattern file contains not the
  same no. of output units as the network
KRERR_CHANGED_I_UNITS:   No. of input units have changed
KRERR_CHANGED_O_UNITS:   No. of output units have changed
KRERR_NO_INPUT_UNITS:    No input units defined
KRERR_NO_OUTPUT_UNITS:   No output units defined
KRERR_NO_PATTERNS:       No patterns defined
KRERR_INCORE_PATTERNS:   In-Core patterns are incompatible
  with the current network (remove In-Core patterns before
  loading a new network)
KRERR_PATTERN_NO:        Invalid pattern number
KRERR_LEARNING_FUNC:     Invalid learning function
KRERR_PARAMETERS:        Invalid parameters
KRERR_UPDATE_FUNC:       Invalid update function
KRERR_INIT_FUNC:         Invalid initialisation function
KRERR_DERIV_FUNC:        Derivation function of the
  activation function doesn't exist
KRERR_I_UNITS_CONNECT:   Input unit(s) with input
  connection(s) to other units:
KRERR_O_UNITS_CONNECT:   Output unit(s) with output
  connection(s) to other units:
KRERR_TOPOMODE:          Invalid topological sorting mode
```

```
KRERR_LEARNING_SITES:    Learning function doesn't
  support sites
KRERR_SITES_NO_SUPPORT:  Sites are not supported
KRERR_NO_MASPAR_KERNEL:  This isn't a MasPar Kernel
KRERR_NOT_NEIGHBOUR_LAYER:  Connection(s) between unit(s) in
  non-neighbour layers are not supported:
KRERR_MUCH_LAYERS:       The network has too much layers:
KRERR_NOT_FULLY_CONNECTED:   The network layers aren't
  fully connected
KRERR_MODE_FF1_INVALID_OP:   This operation isn't allowed in the
  parallel kernel mode
KRERR_NET_TRANSFORM:  Change of network type isn't
  possible in parallel kernel mode
KRERR_NO_CURRENT_LINK:       No current link defined
KRERR_NO_CURRENT_UNIT:       No current unit defined
KRERR_UNIT_NO_INPUTS:        Current unit doesn't have
  any inputs
KRERR_TOPO_DEFINITION:       Invalid parameter in topologic
  definition section
KRERR_BAD_CONNECTION:        Creation of link between these
  units isn't permitted
KRERR_MASPAR_NO_RESPOND:     MasPar don't respond
KRERR_NOT_IMPEMENTED_YET:    This function isn't implemented yet
KRERR_NOT_PARALLEL_MODE:     Kernel isn't in parallel mode
KRERR_MASPAR_OUT_OF_PE_MEM:  MasPar ran out of memory
KRERR_MASPAR_COMMUNICATE:    MasPar communication error
KRERR_MASPAR_OUT_OF_PE:      MasPar ran out of processors
KRERR_MISSING_DEFAULT_FUNC:  Missing default function
  (check function table)
KRERR_MASPAR_MULTI_OUT_FUNC: MasPar kernel doesn't support
  multiple unit output functions
KRERR_MASPAR_MULTI_ACT_FUNC: MasPar kernel doesn't support
  multiple unit activation functions
```

Literaturverzeichnis

[Akl89] Selim G. Akl.
The Design and Analysis of Parallel Algorithms.
Prentice-Hall, 1989.

[Bay91] Harald Bayer.
Massiv parallele Simulation der selbstorganisierenden Karten von Kohonen auf einem SIMD-Rechner.
Master's thesis, Universität Stuttgart, Fakultät Informatik, 1991.

[Bla90] Tom Blank.
The MasPar MP-1 Architecture.
In *Proceedings of the IEEE Spring 1990, IEEE*, Feb 1990.

[CGC+90] G. Chinn, K. A. Grajski, C. Chen, C. Kuszmaul, and S. Tomboulian.
Systolic Array Implementations of Neural Nets on the MasPar MP-1 Massively Parallel Processor.
In *Proc. Intl. J. Conf. on Neural Networks, San Diego*, San Diego, CA., 1990. IEEE TAB Neural Network Commitee.

[FFA90] Naoyuki Fukuda, Yoshiji Fujimoto, and Toshio Akabane.
A Transputer Implementation of Toroidal Lattice Architecture for Parallel Neurocomputing.
In *Proc. Intl. J. Conf. on Neural Networks, Washington, D.C.*, San Diego, CA., 1990. IEEE TAB Neural Network Commitee.

[GR88] Alan Gibbson and Wojcieck Rytter.
Efficient Parallel Algorithms.
Cambridge University Press, 1988.

[Gra90] Kamil A. Grajski.
Neurocomputing using the MasPar MP-1.
Technical report, Ford Aerospace Advanced Development, San Jose, CA 95161, 1990.

[Hil85] W.D. Hillis.
The Connection Machine.
MIT Press, 1985.

[HS86a] W.D. Hillis and G.L. Steel.
Data parallel algorithms.
ACM, 29(12):1170–1183, 1986.

[HS86b] W.D. Hillis and G.L. Steel.
Massively Parallel Computers: The Connection Machine and NON-VON.
Science, 231(4741):975–978, 1986.

[Kor90] Thomas Korb.
Entwurf und Realisierung einer Beschreibungssprache für neuronale Netze.
Master's thesis, Universität Stuttgart, Fakultät Informatik, 1990.

[KT89] Hon Keung Kwan and Pang Chung Tsang.
Systolic Implementation of Multi-Layer Feed-Forward Neural Network with Backpropagation Learning Scheme.
In *Proc. Intl. J. Conf. on Neural Networks, Washington, D.C.*, San Diego, CA., 1989. IEEE TAB Neural Network Commitee.

[KYIA90] Hideki Kato, Hideki Yoshizawa, Hiroki Iciki, and Kazuo Asakawa.
A Parallel Neurocomputer Architecture towards Billion Connection Updates Per Second.
In *Proc. Intl. J. Conf. on Neural Networks, Washington, D.C.*, San Diego, CA., 1990. IEEE TAB Neural Network Commitee.

[Mac90] Niels Mache.
Entwurf und Realisierung eines effizienten Simulatorkerns für neuronale Netze.
Studienarbeit 895, Universität Stuttgart, Fakultät Informatik, 1990.

[Mas91a] MasPar Computer Corporation.
MasPar Commands Reference Manual, jul 1991.

[Mas91b] MasPar Computer Corporation.
MasPar MP1 Parallel Application Language, Reference Guide, jul 1991.

[Mas91c] MasPar Computer Corporation.
MasPar MP1 Parallel Application Language, User Guide, jul 1991.

[Mas91d] MasPar Computer Corporation.
MasPar Progamming Environment Reference Manual, jul 1991.

[Mas91e] MasPar Computer Corporation.
MasPar Progamming Environment Tutorial, jul 1991.

[Nic90] John R. Nickolls.
The Design of the MasPar MP-1: A Cost Effective Massively Parallel Computer.
In *Proceedings of the IEEE Spring 1990, IEEE*, Feb 1990.

[PPDG89] A. Petrowski, L. Personnaz, G. Dreyfus, and C. Girault.
Parallel Implementations of Neural Network Simulations, pages 205–213.
Elsevier Science Publishers B.V., North-Holland, 1989.

[Qui87] Michael J. Quinn.

Designing Efficient Algorithms for Parallel Computers.
McGraw-Hill, 1987.

[RCB88] C.R. Rosenberg, R. Charles, and G. Belloch.
Connectionist Models and their Implementations, chapter 5, An implementation of network learning on the Connection Machine.
NJ: Ablex Publishing, 1988.

[RHW86] D.E. Rumelhart, G.E. Hinton, and R.J. Williams.
Parallel Distributed Processing, volume 1, chapter 8, Learning Internal Representations by Error Propagation, page 318 ff.
MIT Press, 1986.

[RM86] D.E. Rumelhart and J.L. McClelland.
Parallel Distributed Processing, volume 1.
MIT Press, 1986.

[RMS90] H. Ritter, T. Martinetz, and K. Schulten.
Neuronale Netze.
Addision-Wesley, 1990.

[Sin90] Alexander Singer.
Implementations of Artificial Neural Networks on the Connection Machine.
Technical report, Thinking Machines Corporation, 245 First Street, Cambridge, MA 02142, 1990.

[Som90] Tilman Sommer.
Eine graphische Benutzeroberfläche unter X-Windows zur Manipulation und Visualisierung neuronaler Netze.
Master's thesis, Universität Stuttgart, Fakultät Informatik, 1990.

[SR86] T. J. Sejnowsky and C. R. Rosenberg.
NETtalk: A Parallel Network that Learns to Read Aloud.
Technical report, JHU/IEECS-86/01, The Hopkins University, 1986.

[Zha90] X. Zhang et al.
An efficient implementation of the back-propagation algorithm on the Connection Machine CM-2.
In *Advances in Neural Information Processing Systems*, volume 2, pages 801–809, 1990.

[ZKMS91] A. Zell, T. Korb, N. Mache, and T. Sommer.
SNNS, Stuttgarter Neuronale Netze Simulator, Benutzerhandbuch.
Technical Report 1/91, Universität Stuttgart, Fakultät Informatik, 1991.

[ZMSK91a] A. Zell, N. Mache, T. Sommer, and T. Korb.
Recent Developments of the SNNS Neural Network Simulator.
In *Proc. Applications of Neural Networks Conf., SPIE*, volume 1469, pages 708–719, 1991.

[ZMSK91b] A. Zell, N. Mache, T. Sommer, and T. Korb.
The SNNS Neural Network Simulator.
In *GWAI-91, 15. Fachtagung für künstliche Intelligenz, Informatik Fachberichte 285, Springer*, pages 254–263, 1991.