Diplomarbeit Fachhochschule Wedel
angefertigt am DESY/Hamburg

Schnelle Entscheidungen mit neuronalen Netzen

Hans-Thomas Duhme
betreut durch :
Dr. Straumann, Universität Zürich
Dipl. Ing. Poock-Haffmans, Fachhochschule Wedel

22. Januar 1993

Abstakt:

In dieser Arbeit werden die Möglichkeiten zum Einsatz eines neuronalen Netzes als Triggerkomponente des L2 Triggers des H1 Experimentes am HERA Elementarteilchenbeschleunigers (DESY/Hamburg) untersucht. Es wird ein Netz mit etwa 4000 Neuronen entwickelt. Da ein solches Netz nicht mehr mit den Standardansätzen konstruiert und trainiert werden kann, wird bei der Konstruktion ein neuer Weg beschritten. Dadurch, daß sich dieses Netz nicht mehr an das biologische Vorbild anlehnt, ist es möglich, die Neuronen so zu verändern, daß sie optimal durch digitale Bausteine realisiert werden können. Das Netz weist eine hybride Struktur auf und die Neuronen haben orthogonale Gewichtsvektoren. Die Entscheidung des Netzes wird aus der Abwägung von drei Netzen gebildet, die mit den Daten, den dazu inversen Daten und unsicheren Daten trainiert wurden. Durch die Entwicklung eines neuartigen Lernalgorithmus (Simultated Annealing) wird versucht, das Problem des Backpropagation Algorithmus mit lokalen Minima zu vermeiden, daß bei solchen grossen Netzen auftritt. Es werden Möglichkeiten aufgezeigt, das Netz zusätzlich zur Datenanlyse zu verwenden. Durch die geordnete Struktur ist es möglich, die Funktionsweise des Netzes direkt nachzuvollziehen. Ein solches Netz wird mit realen Daten simuliert und die Ergebnisse mit anderen Ansätzen verglichen. Ein Vorschlag für die Konstruktion eines solchen Netzes schließt sich an.

Inhaltsverzeichnis

Hiermit versichere ich, daß ich diese Arbeit selbständig verfaßt und keine als die angegebenen Quellen und Hilfsmittel benutzt habe.

Ich danke allen, die mich bei dieser Arbeit unterstützt haben.

1 Vorwort

Die Idee zu dieser Diplomarbeit entstand während meines Industriepraktikums am DESY in Hamburg bei der Arbeitsgruppe, die sich mit dem Trigger im H1 Experiment beschäftigt. Bei dem H1 Experiment werden Elektronen und Protonen aufeinandergeschossen. Diese Teilchen werden über mehrere Beschleuniger in den Doppelring Hera eingespeist, wo sie dann gegenläufig kreisen. An zwei Stellen kreuzen sich die Teilchenbahnen. An diesen Stelle stehen die Experimente Zeus und H1. Ein Experiment besteht im wesentlichen aus einer großen Anzahl von Detektoren, die um einen solchen Kollisionspunkt angeordnet sind. Diese Detektoren bestehen aus Driftkammern, um die Teilchenbahnen zu verfolgen, Kalorimeter um Energien zu messen sowie einige Spezialdetektoren.
In dem Heraspeicherring kreisen 220 Teilchenpakete mit annähernd Lichtgeschwindigkeit. Bei einem Umfang von etwa 6.5 km treten alle 96 ns Kollisionen auf. Pro Kollision liefert ein Experiment etwa 2 MByte Daten. Die Wahrscheinlichkeit für physikalische Ereignisse ist aber sehr gering. Ihre Rate beträgt etwa 3 Hz. Da es nicht möglich ist, alle Daten abzuspeichern, ist ein schneller Mechanismus zum Aussortieren der uninteressanten Ereignisse erforderlich. Dies wird bei H1 durch einen 4-stufigen Trigger geleistet. Jede Stufe reduziert die Datenrate um etwa Faktor 10-20. Die Triggerstufen werden L1 bis L4 genannt. Der erste Trigger übernimmt eine Art Plausibilitätsprüfung. Er stellt fest, ob überhaupt etwas passiert ist. Eine positive L2 Entscheidung startet den Datentransfer aller Detektordaten von der Detektorelektronik zur L4 Prozessorfarm. Dieser Datentransfare kann von einer L3 Entscheidung unterbrochen werden. Der L4 Trigger rekonstruiert aus den Daten die Flugbahnen und rechnet daraus auf den Ursprung des Ereignis zurück, wobei der L4 vollständig asynchron arbeitet. Dem L3 Trigger stehen ebenso wie L1 und L2 nur ein Teil der Daten zur Verfügung, der L4 Trigger rekonstruiert aus allen Daten des Detektores das Ereignis.
Das neuronale Netz soll als Bestandteil des L2 Triggers eingesetzt werden. Die technischen Randbedingungen sind in etwa: Aus 14 Kanälen mit je 8 Bit Auflösung muß in 1-2 μs eine Entscheidung getroffen werden. Da die Kriterien für diese Entscheidungen noch nicht feststehen und sich erst im Laufe der Experimente herausstellen, soll der Trigger möglichst universell und flexibel sein. Ein lernfähiges Netz, das nur mit entsprechenden Daten trainiert werden muß, scheint ein guter Ansatz zu sein. Abgesehen davon ist für eine neuronale Struktur die Rechenzeit nicht problematisch, da diese Struktur grundsätzlich parallel arbeitet. Die Verarbeitungszeit entspricht der Durchlaufzeit durch alle Schichten und ist unabhängig von der Anzahl der Neuronen in den Schichten. Da durch den

L1 Trigger alle trivialen Fälle aussortiert worden sind, muß diese Triggerstufe schon auf kombinierte Bedingungen selektieren. Dies sind Kombinationen aus Energiebetrachtungen und Flugbahnen. Ein besonderes Problem besteht auch darin, daß die Ereignisse, auf die der Trigger reagieren soll, nicht bekannt sind. Der Trigger muß also gute, schlechte und unbekannte Ereignisse erkennen. Zu dem neuronalen Ansatz gibt es mehrere alternative Verfahren, die parallel bei DESY entwickelt und eingesetzt werden. Hierin besteht die Chance, den neuronalen Ansatz mit klassischen Ansätzen direkt zu vergleichen.
Im Rahmen dieser Diplomarbeit soll kein kompletter L2 Trigger konstruiert werden. Es sollen vielmehr anhand eines konkreten Problems die Einsatzmöglichkeiten eines neuronalen Netzes beurteilt werden. Wie eine Abschätzung zeigt, ist für die hier gestellte Aufgabe ein Netz mit etwa 500-2000 Neuronen erforderlich. Die klassischen Verfahren zur Implementierung und Trainierung von neuronalen Netzen sind typischer Weise für kleinere Neuronenzahlen ausgelegt. Es soll im Rahmen dieser Diplomarbeit untersucht werden, wie die Ansätze für größere Neuronenzahlen modifiziert werden müssen und wie ein solches Netz konstruiert werden kann. Da die Daten der Experimente zur Verfügung stehen, können die dargelegten Überlegungen direkt durch entsprechende Simulationen überprüft und bewertet werden.

2 Einleitung

Würden die in den Experimenten anfallende Datenraten von 2 MByte/100 ns allein durch einen einzelnen Rechner bearbeitet werden, so müßte dieser eine Rechenleistung von 20 Mega-MIPS ($2 * 10^{13}$ Instruktionen pro Sekunde) besitzen, um die anfallenden Daten zu bearbeiten. Eine solche Rechenleistung ist zur Zeit nicht einmal ansatzweise zu realisieren. Durch das hier verwendete Pipelinekonzept und die Auswahl von nur einigen relevanten Daten des Detektors kann das Problem auf der L2-Ebene entsprechend reduziert werden. Das Hauptproblem bei einem klassischen Rechner ist nicht etwa, daß dieser nicht über genügend Schaltelemente verfügt, um eine solche Rechenleistung zu erreichen, sondern die Ursache liegt in der Struktur der Schaltelemente in einem solchen Rechner. Hierzu ein Beispiel: Die in dieser Arbeit verwendeten Simulation wurden auf einer HP-700 gerechnet. Dies ist eine RISK- Maschine mit 140 MByte Arbeitsspeicher und etwa 100 Mhz Taktfrequenz. Von diesen 140 MByte Arbeitsspeicher können aber nur etwa 3 Speicherzellen gleichzeitig bearbeitet werden (3, da dieser Rechner über drei getrennte Speicher für Daten und Programm verfügt). Die übrigen 139.99 MByte Speicher liegen völlig ungenutzt im Rechner. Da der Zugriff auf derartige Speicher auch nicht beliebig schnell sein kann (etwa 60 ns bzw 110 ns Zykluszeit) ergibt sich hieraus die 'geringe' Rechenleistung. Würde es gelingen, jede Speicherzelle durch ein Rechenwerk zu ergänzen, so daß alle Speicherzellen gleichzeitig rechnen können, würde die Rechenleistung erheblich steigen. Bei einem solchen Ansatz ist natürlich die Vernetzung und Programmierung dieser Struktur das entscheidende (ungelöste?) Problem. Solche Rechner wäre der Struktur von biologischen Gehirnen ähnlich. Bei einer biologischen Struktur ist stark vereinfacht ausgedrückt eine rechnende Speicherzelle durch eine Nervenzelle realisiert. Die klassischen Ansätze für entsprechende Rechnerstrukturen kommen deshalb auch nicht aus der Informatik oder Mathematik, sondern wurden in der Biologie entwickelt. So gehen die ersten Ansätze für einen derartigen Computer schon auf McCulloch und Pitts im Jahr 1943 zurück. Sie postulierten als erste, daß eine Nervenzelle einem logischen Schaltelement vergleichbar sei. Darauf aufbauend wurde im Jahre 1949 von dem Psychologen Hebb ein Lernalgorithmus für Nervenzellen vorgeschlagen. Bei diesem Algorithmus lernen die Zellen in Abhängigkeit von ihrer Aktivität lokal im Netz ohne den Gesamtzusammenhang des Netzes zu kennen. Dieses lokale Lernen (Selbstorganisation) ist für die Erklärung des Lernverhaltens von biologischen Netzen zwingend erforderlich, da dort keine übergeordnete Strukturen gefunden werden konnten. Dieser Lernalgorithmus wurde dann erstmalig 1958 von Rosenblatt für das von ihm vorgeschlagene Perzepton-Modell verwendet. In

diesem Zusammenhang kann auch gezeigt werden, daß dieser Ansatz für ein selbständiges Organisieren eines solchen Netzes tatsächlich funktioniert. Solche Ansätze wurden im Laufe der Zeit weiterentwickelt. Durch das Aufkommen von leistungsfähigen Computern konnten dann auch immer größere Strukturen simuliert bzw. aufgebaut werden. Heute sind im wesentlichen drei Modelle von neuronalen Netzen verbreitet. Dies ist einmal das assoziative Netz nach Hopfield (1982). Dieses ist ein rückgekoppltes boolsches Netz, welches aus unvollständigen oder gestörten Daten auf die gelernten schließen kann. Der Ansatz von Kohonens selbstorganisierenden Karten (1982) beschreibt ein rückgekoppltes analoges Netz. Dieses klassifiziert Daten in verschiedene Klassen. Besonders interessant an diesen Netz ist, daß die Klassifizierung in verschiedene Gruppen nicht zwingend vorgegeben sein muß. Das Netz kann die Daten selbstständig nach Merkmalen sortieren und diese zu entsprechenden Klassen zusammenfassen. Ein Durchbruch in der Neuroinformatik gelang mit der Einführung des Backpropagation Algorithmus (z.B. nach Rumelhart 1986). Dieses Modell beschreibt das Lernverhalten für ein mehrschichtiges nicht rückgekoppltes Netz. Eine solche Netzstruktur ist für eine technische Realisierung gut geeignet. Den meisten heutzutage eingesetzten neuronalen Netzen liegt dieses Modell zugrunde.[1]

Allen hier verwendeten Modellen liegt eine vereinfachte Form des biologischen Neurons zugrunde. Die Arbeitsweise eines biologischen Neurons wird durch die Vorstellung angenähert, daß die Signale über sogenannte Dendriten an den Zellkörper geführt werden, dort verrechnet und dann über das Axon weiter verbunden werden. Der Ausgang eines solchen Neurons ist also die gewichtete Summe der Eingangsgrößen, die durch eine Schwellwertfunktion auf den Ausgang abgebildet werden. Den oben vorgestellten Netzmodellen liegt dieses Neuronenmodell zugrunde. Sie unterscheiden sich durch die Art der Verschaltung und der Art der Schwellwertfunktion. Für eine Realisierung solcher Netze mit digitaler Hardware ist dieses Neuronenmodell aber sehr ungeeignet, da das Multiplizieren, Summieren und eine kontinuierliche Schwellwertfunktion nur durch aufwendige Rechenwerke realisierbar sind. Netze, die ein derartiges Neuronenmodell verwenden, sind somit zwangsläufig nur klein oder langsam. Außerdem erweist sich der Backpropagation Algorithmus für große und redundante Datensätze als problematisch (lokale Minima). Hieraus ergeben sich die Fragestellungen der Diplomarbeit:

[1] weitere Erläuterungen zu den Modellen sind nachzulesen in /Neuronale Netze / Helge Ritter, Thomas Martinetz, Klaus Schulten / Addison-Wesley Verlag / 1990

- Können Neuronen anders als durch digitale Rechenwerke konstruiert werden ?
- Kann das Neuronenmodell so modifiziert werden, daß es einer Realisierung mit digitalen Bauelementen angepaßt ist ?
- Kann der Backpropagation Algorithmus durch einen anderen Algorithmus ersetzt werden, der systematischer vorgeht ?

3 Die Funktionsweise eines neuronalen Netzes

Neuronale Netze lassen sich grob in zwei Klassen einteilen. Man unterscheidet Netze mit Rückkopplung und Netze, die in hierarchischer Struktur die Daten von einer übergeordneten Schicht in die nächst tiefere übergeben.

3.1 Rückgekopplte Netze

Zu der Klasse der rückgekopplten Netze zählen die schon erwähnten Netze von Hopfield und Kohonen. Das Netz von Hopfield besteht im wesentlichen aus einer Schicht, die auf sich selbst zurückgeführt wird. Das Netz wird nun so trainiert, daß die zu erkennenden Muster sogenannte Eigenwerte des nicht zurückgeführten Netzes sind. Werden bei einem solchen Wert die Ausgänge wieder auf die Eingänge zurückgeführt, so ist das Netz stabil, da Ein- und Ausgang die gleichen Werte haben. Wird nun aber eines den trainierten Daten ähnliches Muster auf die Eingänge gelegt, so schwingt sich das Netz auf den dem Eingangsmuster am ähnlichsten Eigenwert ein. Die abweichenden Komponenten in dem Eingangsmuster werden durch die Rückführung so lange korrigiert, bis sich ein stabiler Wert ergibt. Ein solches Netz kann mit einer Potentialfläche verglichen werden, in der die trainierten Daten Senken sind. Das Eingabemuster entspricht dann anschaulich einer Kugel, die in dieses Potentialgebirge gesetzt wird. Die Kugel rollt dann in die am nächsten befindliche Talsenke. Diese Talsenke ist dann das trainierte Muster, welches dem Eingabemuster am ähnlichsten ist.
Die selbstorganisierenden Karten nach Kohonen basieren auf dem Prinzip der lateralen Hemmung. Die Neuronen sind in einer zweidimensionalen Ebene angeordnet. Ein Neuron wird durch die Aktivierung von in der Nähe befindlichen Neuronen in seiner Aktivität gefördert, durch weiter entfernte Neuronen immer mehr gehemmt. Jedes Neuron ist mit den Eingabedaten verbunden. Beim Lernen werden die Gewichte der Neuronen schrittweise so verändert, daß der Bereich, in dem die Neuronen aktiviert werden, einer Klasse entsprechen. Durch die Hemmung der Neuronen untereinander bleiben nur die Neuronen mit der größten Aktivität übrig, alle weniger optimal reagierenden Bereiche von Neuronen werden in dieser Art Konkurrenzkampf unterliegen. Bei einer Klassifizierung bleiben bei einem solchen Netz nur einige abgegrenzte Bereiche übrig.

3.2 Nicht rückgekopplte Netze

Die Funktionsweise von nicht rückgekopplten Netzen kann geometrisch interpretiert werden. Dieser Ansatz eines neuronalen Netzes geht davon aus, daß ein Netz aus mehreren hintereinandergestaffelten Schichten aufgebaut ist. Eine Schicht besteht aus mehreren Neuronen, die in derselben Schicht nicht miteinander verbunden sind:

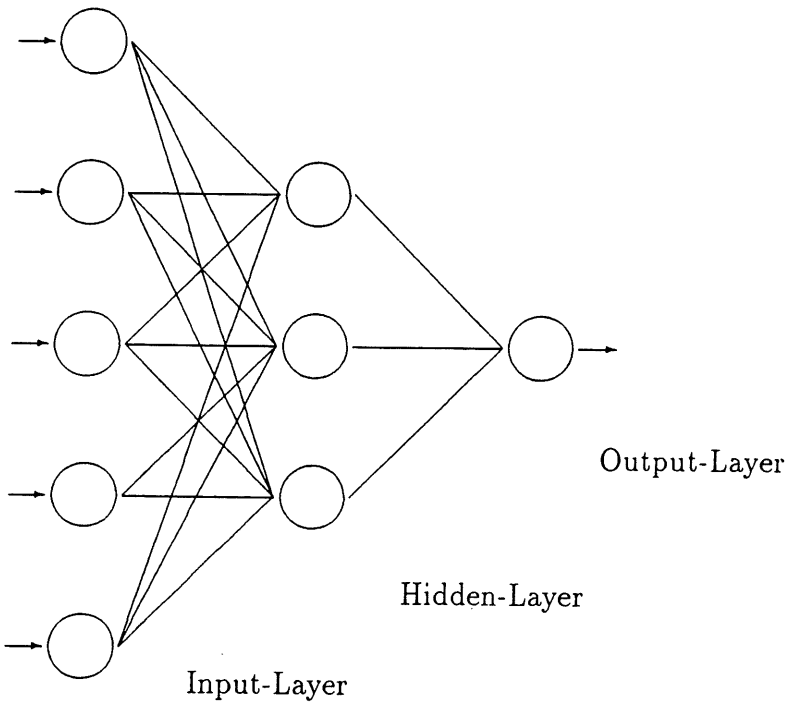


Abb. 1: feed forward Netz

Rechnerisch bildet ein Neuron das Skalarprodukt der Eingangsgrößen mit dem Gewichtsvektor des Neurons. Anschließend wird eine Schwellwertfunktion auf das Ergebnis angewandt.

$$Y = \theta\left(\sum_{i=0}^{n}(x_i * g_i) - s\right)$$

Y: Ausgangswert des Neurones.
g_i: Gewichtsvektor eines Neurons.
s: Schwellwert.
x_i: Eingangswert eines Neurones.
θ: Schwellwertfunktion.

Wird die Schwellwertfunktion zu einem Vergleich vereinfacht, so stellt dieser Ausdruck geometrisch betrachtet eine Art Aufteilung des Raumes in zwei Hälften dar. Durch das Skalarprodukt wird die Projektion des Eingangsvektors auf den Gewichtsvektor gebildet. Durch die Schwellwertfunktion wird dann praktisch senkrecht zum Gewichtsvektor der Raum aufgeschnitten (Cuts).

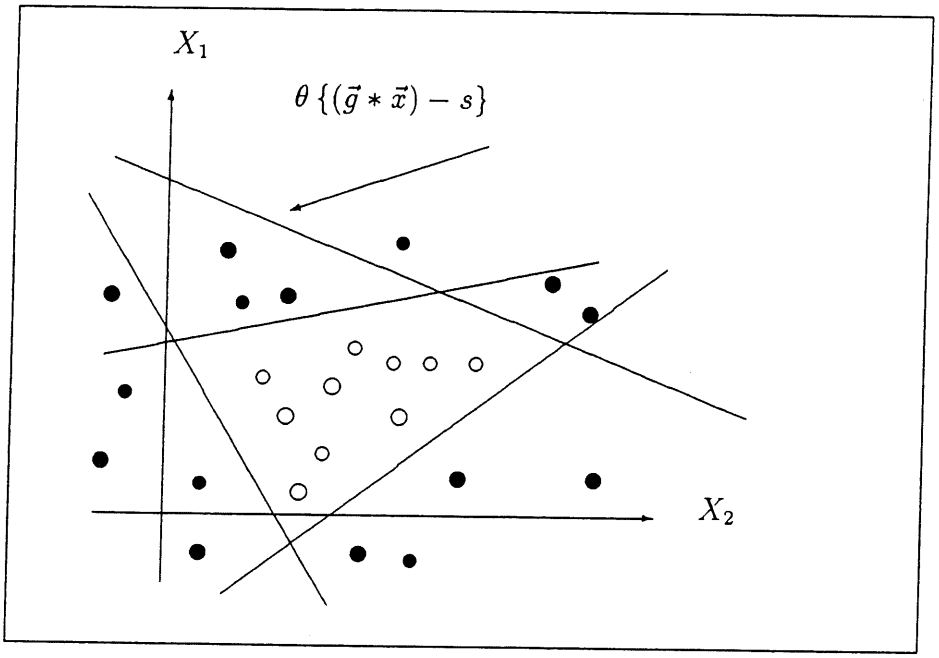


Abb. 2: Die Meßwerte in der X_1X_2- Ebene werden durch Geraden so umschlossen, daß alle Werte einer Eigenschaft in einem Polygon liegen

Durch mehrere Neuronen kann dann also ein Raumkörper umschrieben werden. Wird nun statt des Vergleiches eine kontinuierliche Sigmafunktion gewählt, so verschmelzen diese Cuts an den Stoßstellen. Die dabei entstehende Kurve kann auch als ein Polynom betrachtet werden. Die genaue Art des Polynoms hängt von der gewählten Sigmafunktion ab. So können z.B. die Werte W_p eines statistischen Neurons durch die Wahrscheinlichkeit von Pulsen abgebildet werden, deren Pulsbreite konstant ist. Die Gewichtsbildung geschieht durch Weglassen oder Hinzufügen von Pulsen, analog zu dem Multiplizieren des Eingangswertes mit einem Faktor bei einem kontinuierlichen Netz.[2] Die Summation und Schwellwertbildung geschieht durch einfache Veroderung der Pulse. Für die Ausgangswahrscheinlichkeit eines Neurons mit zwei Eingangsgrößen $P1 \cup P2$ ergibt sich :

$$P_{gesamt} = P1 \cup P2 = W_{P1} + W_{P2} - W_{P1} * W_{P2}$$

Für kleine Eingangsgrößen entspricht die Ausgangswahrscheinlichkeit der Summe der Eingangswahrscheinlichkeiten. Werden die Eingangswerte größer, so entsteht so etwas wie

[2]vgl. / Continuous Input RAM-Based Stochastic Neural Model / Denise Gorse, John G. Taylor / Neural Networks, Vol4 / pp657-665 / 1991

eine Sättigung durch den Produktterm. Steigt die Ausgangswahrscheinlichkeit bei kleinen Größen linear an, so nähert sie sich bei größeren Werten asymptotisch der 100% Grenze. Eine solche Sättigungsfunktion ist als Sigmafunktion geeignet. Wird $W_{P1} = -W_{P2}$ in die Gleichung eingesetzt, so wird der Eingangswert W_{P2} mit einer Konstante multipliziert. Werden die Eingangswahrscheinlichkeiten vor einem solchen Neuron klein skaliert, so ist es ein Addierer. Ein solcher Ansatz ist gut geeignet, das Verhalten eines Netzes auf ein Polynom oder bei einem rückgekopplten Netz auf eine Differentialgleichung zurückzuführen. Der Zustandsraum (d.h. der Raum aller Meßwerte x)ist beliebig dimensional. Das Netz lernt, indem bekannte Punkte im Zustandsraum durch Cuts umschlossen werden. Wenn nun die zu erkennenden Punkte im Zustandsraum einen geschlossenen Raumkörper bilden, dann liegen auch die noch unbekannten Ereignisse, die den gelernten ähnlich sind, in diesem durch die Cuts beschriebenen Raumkörper. Die Anzahl der Neuronen ist ein Maß für die Qualität dieser Approximation des Raumkörpers durch die Cuts. Ein kugelförmiger Körper wird durch Flächen umschrieben. Mehrere getrennte oder kompliziert unterteilte Raumkörper werden aus solchen kugelartigen Körpern zusammengesetzt. Dies Zusammensetzen der Raumkörper kann erst dann erfolgen, wenn diese einzelnt erkannt wurden. Deswegen müssen hier der ersten Schicht weitere folgen. Die Anzahl der Schichten ist also ein Maß für die Anzahl der nicht miteinander verbundenen oder konvex geformten Raumkörper.

Je nach Ansatz werden an die Neuronen unterschiedliche Anforderungen gestellt. Bei rückgekopplten Strukturen ist das genaue Rechenverhalten der Neuronen und das genaue Verhalten der Schwellwertfunktion nicht kritisch, da durch die Rückkopplung so etwas wie eine Gegenkopplung auftritt. Durch die Gegenkopplung werden Unlinearitäten oder Drifterscheinungen kompensiert. Für solche Netze ist es offensichtlich nicht erforderlich, die Neuronen durch exakt rechnende digitale Rechenwerke nachzubilden. Solche Netze lassen sich durch die schon erwähnten statistischen Neuronen oder durch analoge Schaltungen aufbauen. Ein analog aufgebautes Neuron könnte aus einem Widerstandsknoten bestehen. Die Skalarproduktbildung geschieht mit festen Koeffizienten. Diese Koeffizienten könnten durch die Widerstandswerte dargestellt werden. Ein Problem bei rückgekopplten Strukturen ist es, daß das Verhalten des Netzes nur schwer vorhersagbar ist und durch einen Einschwingzustand gekennzeichnet ist. Die Dauer dieses Einschwingens ist nicht vorhersagbar. Außerdem ist es nicht möglich, nach einem erfolgten Trainieren eines solchen Netzes zu erklären, warum es so reagiert, wie es reagiert. Gerade das Verhalten eines Netzes nach dem Ansatz von Kohonen ist nicht eindeutig zu erklären. So ist es

nicht möglich, zu entscheiden, welche der Eingabedaten für die Klassifizierung entscheidend sind. Es können also Fehler beim Trainieren entstehen, wenn mit nicht optimal ausgesuchten Daten trainiert wird. So kann ein unwesentlicher Unterschied in den Datenstrukturen zu einer ungewollt anderen Klassifizierung führen. Solche Netze sind deshalb nicht besonders geeignet, wenn es darum geht, Daten kontrolliert zu klassifizieren.
Die nicht rückgekopplten Strukturen sind in ihrem Verhalten besser zu verstehen. Durch die Beschreibung durch geometrische Cuts können die Datenbereiche für eine Klassifizierung anschaulich dargestellt werden. Im Prinzip können rückgekopplte Netze in nicht rückgekopplte Strukturen überführt werden, indem beliebig viele Netzschichten hintereinander geschaltet werden. Rückgeführte Netze sind also gewissermaßen ein Spezialfall der feed forward Netze. Der Aufwand für solche feed forward Strukturen ist aber größer und das Ergebnis hängt kritisch von dem Rechenverhalten der einzelnen Neuronen ab. Deswegen sollten solche Strukturen durch digitale Rechenwerke mit genügend großer Auflösung konstruiert werden. Trotz des größeren Aufwandes scheint mir nur eine solche forward Struktur geeignet zu sein, das beschriebene Triggerproblem zu lösen.

4 Möglichkeiten und Grenzen assoziativer Speicher

Die Funktion eines Netzes läßt sich als eine Art assoziativer Speicher verstehen. Bei einem solchen Speicher werden nicht wie bei einem herkömmlichen Speicher die Daten über eine Adresse abgerufen. Ein assoziativer Speicher sucht zu einem Eingangswert die Daten aus seinem 'Gedächtnis', die dem Eingangswert am ähnlichsten sind. Somit ist diese Art von Netz gut in der Lage, unbekannte Ereignisse auf bekannte zurückzuführen (vgl. Abb.3).Dieses Netz ist aber nicht in der Lage, Berechnungen, wie etwa aufwendige Transformationen, durchzuführen (es sein denn, sie würden Punkt für Punkt trainiert). Somit sollte im allgemeinen vor einem solchen Netz eine Vorverarbeitung der Daten erfolgen.

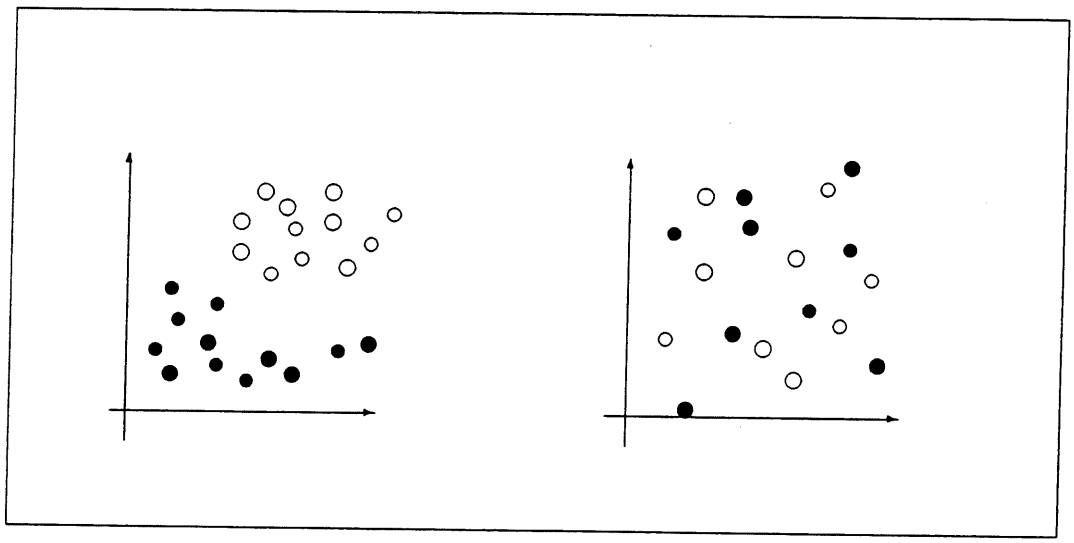


Abb. 3: Gut und schlecht geeignete Datenkonstellationen für einen assoziativen Speicher

Wichtig für die Beschreibung eines assoziativen Speichers ist der Begriff der Ähnlichkeit. Die Ähnlichkeit zwischen zwei Werten kann anschaulich als Abstand dieser Werte im Zustandsraum verstanden werden. Der einfachste Ansatz für einen assoziativen Speicher ist ein einfaches Suchverfahren. Als Metrik im Zustandsraum wird z.B. die kartesische gewählt ($d = \sqrt{x_1^2 + x_2^2 + \dots}$). Wird nun ein noch unbekannter Wert untersucht, so wird der Abstand zu jedem Punkt in diesem Raum berechnet. Aus diesen Abständen läßt sich, ähnlich wie bei einer Anziehungskraft, eine Bewegung für diesen Punkt errechnen. Der Punkt wird nun ein Stück in die Richtung gesetzt, bei der die Summe aller

Abstände ($\sum d$) kleiner ist. Das Verfahren wird nun solange wiederholt, bis die Summe der Abstände ein (lokales) Minimum erreicht hat. Die Punkte in der Umgebung sind nun dem neuen Punkt am Ähnlichsten. Der Wert dieser Punkte wird dann für den gesuchten Punkt eingesetzt. Interessant ist, daß auf diese Weise durch einige feste Punkte und eine Metrik ein geometrischer Raum konstruiert werden kann. Versucht man z.B., Punkte mit einem Abstand, die aus einem 3D-Raum entnommen wurden, so mit Hilfe einer Metrik in einen 2D- Raum zu sortieren, ist das Problem bei einigen Werten noch lösbar, fügt man aber immer mehr Werte in diesen Raum, so passen diese dann an mehrere Stellen. Es ergeben sich mehrere Minima. Es ist eigentlich auch klar, daß dieses Sortierproblem nur im 3D-Raum oder größeren Dimensionen gelöst werden kann.
Mit diesen Beispiel sollte gezeigt werden, daß ein geometrischer Raum allein durch einige feste Punkte und eine Metrik konstruiert werden kann. Ein assoziativer Speicher und somit auch ein neuronales Netz tun genau dieses.

5 Ein hybrides Netz mit orthogonalen Neuronen

Die Idee für diesen speziellen Netzansatz liegt eigentlich auf der Hand. Um den Rechenaufwand in den Neuronen zu minimieren, werden nur orthogonale Gewichtsvektoren zugelassen (vgl. *Abb.*4). Dadurch entfällt das Skalarprodukt, das Neuron vereinfacht sich zu einer Reihe von Vergleichern. Da in dem Gewichtsvektor aufgrund der Originalität nur eine Komponente von Null verschieden ist, wird beim Skalarprodukt auch nur die entsprechende Komponente des Eingangsvektors multipliziert und somit ist das Ergebnis das Produkt dieser beiden Komponenten. Der Betrag des Gewichtsvektors kann zu 1 normiert werden, indem die nachgeschaltete Schwellwertfunktion (Vergleich) entsprechend skaliert wird. Dadurch, daß die Gewichte der Neuronen nicht unabhängig voneinander geändert werden können, gibt es auch weniger Freiheitsgrade beim Trainieren des Netzes. Es kann vermutet werden, daß sich dadurch das Problem der lokalen Minima lösen läßt. Außerdem läßt sich diese Struktur einfach in Hardware aufbauen.
Die weiteren Schichten des Netzes werden nicht mit denselben Neuronen aufgebaut. Da die Neuronen der ersten Schicht aufgrund der vereinfachten Schwellwertfunktion nur boolsche Werte liefern, ist es sinnvoll, hier in den weiteren Schichten auch boolsche Neuronen einzusetzt. Boolsche Neuronen sind Neuronen, deren Eingänge nur digitale Informationen verarbeiten, also logische $UND/ODER$-Funktionen und deren Kombinationen. Eine derartige hybride Struktur ist zudem auch dem Datenfluß durch das Netz angepaßt. Die Eingangsdaten mit typischerweise je 8 Bit Auflösung werden durch das Netz auf 3 Bit reduziert. Deswegen muß irgendwo im Netz der Übergang aus der kontinuierlichen Welt in die boolsche geschehen. Bei den üblichen Ansätzen geschieht dieser erst am Ausgang des Netzes durch eine Schwellwertbildung. Das gesamte Netz rechnet dann mit kontinuierlichen Werten. Werden diese Schwellwerte aber schon in einer mittleren Schicht eingebaut, können in den ersten Schichten die Vorteile von kontinuierlichen Netzen genutzt werden, die anderen Schichten nutzen dann die Vorteile der boolschen Netze aus. Außerdem wird durch weniger kontinuierlichen Schichten die Realisierung vereinfacht, da sich boolsche Netze einfacher mit digitalen Schaltungen aufbauen lassen. Ein solcher Netzansatz entfernt sich sehr stark von dem biologischen Vorbild und orientiert sich an den Vorgaben der digitalen Welt.

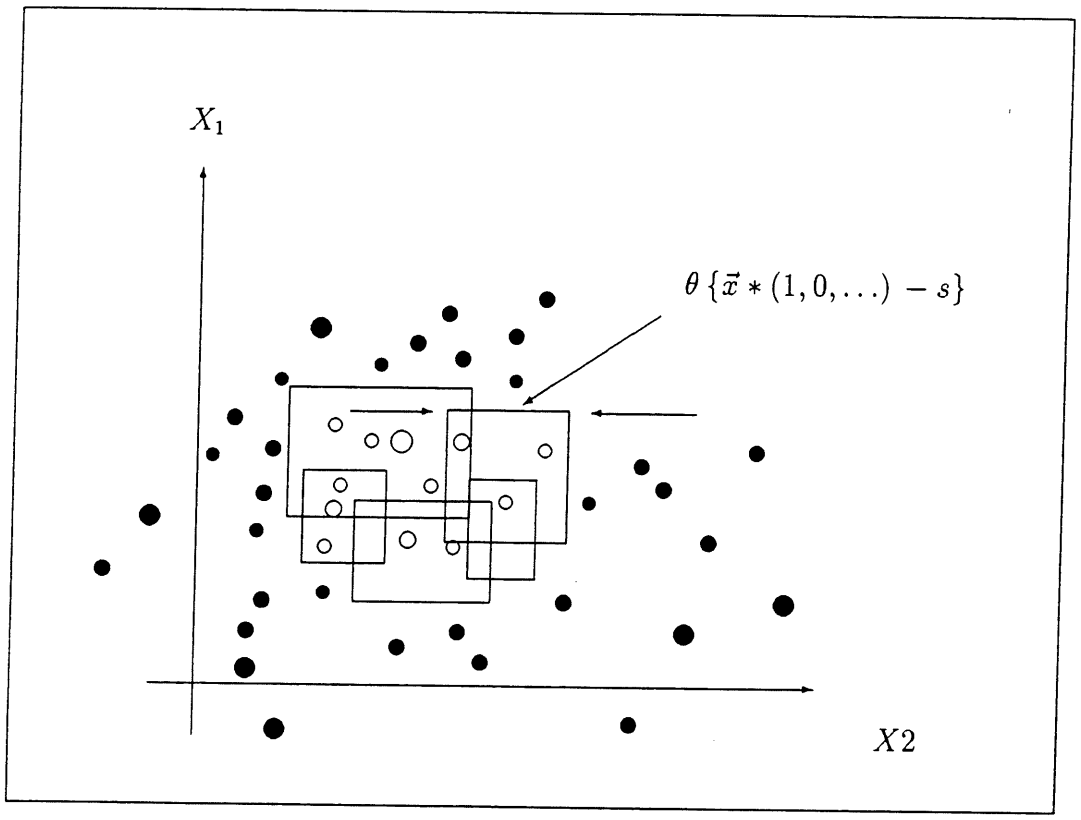


Abb. 4: Statt durch ein Polygon werden jetzt die indexs Meßwerte durch orthogonale Cuts umschrieben. Diese Cuts sind hier schon zu Raumwürfeln zusammengefaßt, wie dies von den später erklärten Lernalgorithmus vorgenommen wird. Da die Cuts orthogonal zu einem Koordinatensystem sind, vereinfacht sich die Darstellung dieser Cuts in diesem Koordinatensystem zu Vektoren mit nur einer von Null verschiedenen Komponente.

Prinzipiell ist die Zuordnung der Meßwerte in den Randbereichen nicht eindeutig. Durch die Verwendung von entsprechend geformten Cut werden lediglich Hypothesen über den Verlauf der Meßwerte in den Randbereichen gemacht. Dieses Problem wird dadurch berücksichtigt, daß das Netz einmal mit dem aktiven und dem dazu inversen Muster trainiert wird. Dadurch kann die Randzone erkannt werden, da hier inverses und aktives Netz nicht unterschiedlich reagieren.

Da es durchaus möglich ist, daß Daten widersprüchlich sind, d.h. zu denselben Eingangsdaten in den Trainingsdaten unterschiedliche Klassifizierungen vorliegen, ist es sinnvoll, diese Werte gesondert zu betrachten. Hierzu werden diese Werte durch einen Sortierprozeß erkannt und aus dem Trainingsdatensatz entfernt. Ein drittes Netz wird dann

gesondert auf diese Daten trainiert.

5.1 Der Lernalgorithmus

Der Lernalgorithmus für ein solches Netz ist an das Vorbild eines Kristalles angelehnt, der in übersättigter Lösung bei fallender Temperatur auskristallisiert (annealiert). Zuerst werden die Parameter der Neuronen in der Art zusammengefaßt, daß ein Neuron nicht nur einen Cut in den Raum legt, sondern daß ein sogenanntes Doppelneuron in jeder Dimension eine Ober- und Untergrenze mit seinen Eingangswerten vergleicht. Ein Doppelneuron beschreibt somit einen Würfelförmigen Bereich im n dimensionalem Raum. Liegen alle Komponenten des Eingangsvektors in diesen Grenzen, so schaltet das Neuron. Mit der Größe des Neurons kann somit auch sinnvoll der Rauminhalt dieses Würfels bezeichnet werden. Das Lernprogramm versucht zuerst, Neuronen mit einem großen Rauminhalt in den Raumkörper zu legen. Ist ein solches Neuron gefunden, versucht das Programm, durch immer kleinere Variation der Parameter des Neurons dieses in den Raumkörper optimal 'einzuparken'. Werden keine Neuronen mehr gefunden, wird die Temperatur gesenkt, d.h. es wird versucht, kleinere Neuronen zu finden, die noch in den Raumkörper passen. Sind alle Werte durch das Netz beschrieben, oder ist die Obergrenze der Neuronenzahl erreicht, so endet das Programm. Es wird bei diesem Algorithmus davon ausgegangen, daß durch ein langsames Absenken der Temperatur eine regelmäßige und somit systematische Struktur, wie bei einem Kristall, entsteht.

Der Kristallisationsalgorithmus unterscheidel sich grundlegend von Backpropagation Algorithmus. Bei dem Backpropagation Algorithmus wird von einem fertigen Netz ausgegangen, das dann schrittweise optimiert wird. Hieraus ergibt sich das Problem der Startwerte, mit denen bei der Optimierung begonnen wird. Bei sehr hochdimensionalen Problemen ist es sehr unwahrscheinlich, durch rein zufällige Wahl ein günstiges Startnetz zu finden. Durch diese unsystematische Komponente in diesem Algorithmus wird das Auftreten von lokalen Minima begünstigt. Die Tatsache, daß mit dem gesamten Netz die Optimierung begonnen wird, führt dazu, daß der Backpropagation Algorithmus nicht mit den wesentlichen Daten anfängt, das Netz zu trainieren und es ergeben sich Probleme beim Trainieren mit überbestimmten Daten. Diese beiden Problembereiche werden bei dem Kristallisationsalgorithmus vermieden.

Ein Netz wird erst beim Trainieren aufgebaut. Das Programm beginnt mit einem einzelnen Neuron und versucht mit diesem möglichst viele Daten zu klassifizieren. Ist dies geschehen, wird versucht, die noch nicht klassifizierten Daten mit einem zusätzlichen Neu-

ron zu klassifizieren (usw.).

In der Simulation hat sich auch gezeigt, daß offensichtlich die Anzahl der Neuronen, die bei einem bestimmten Eingangswert schalten, eine Größe ist, die weiter ausgewertet werden kann (vgl *Abb.*5, *Abb.*6). So schalten in der Mitte eines Raumkörpers sehr viele Neuronen, während am Rand nur wenige aktiv sind. Bei den mit dem inversen Muster trainierten Neuronen ist die Anzahl der aktiven Neuronen offenbar ein Maß für die Anzahl der Freiheitsgrade, die fehlerhaft sind. In *Abb.*6 in deutlich zu erkennen, daß die Anzahl der Neuronen an den Ecken ansteigt, da hier sowohl die x und y Komponente nicht übereinstimmt. An den Stellen, an denen nur eine Komponente fehlerhaft ist, ist deutlich ein zurückgehen der Neuronenzahl zu erkennen.

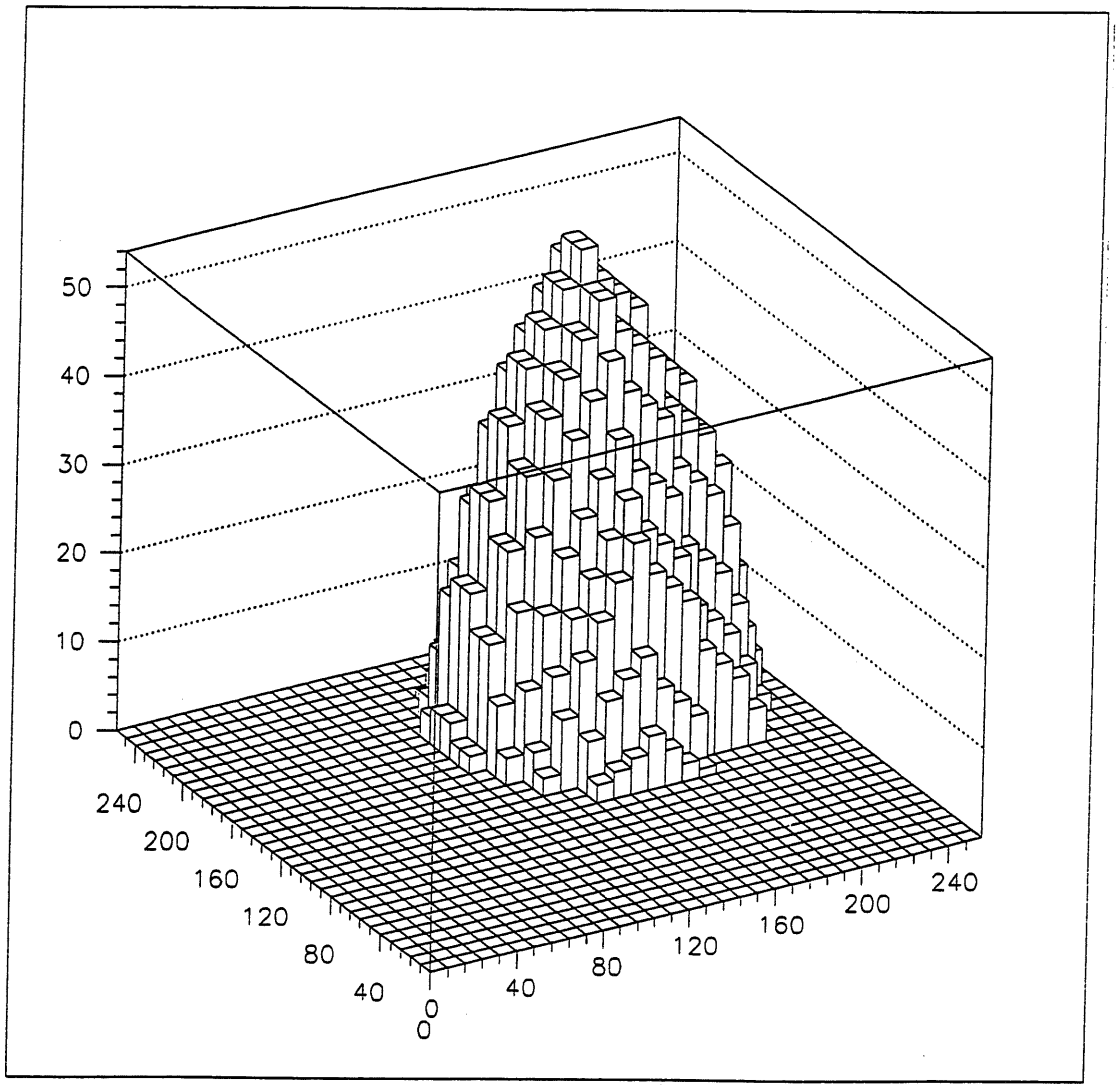


Abb. 5: Häufigkeit der Neuronen, die in einem Raumbereich schalten, wenn das Netz auf einen zweidimensionalen Kreis trainiert wurde

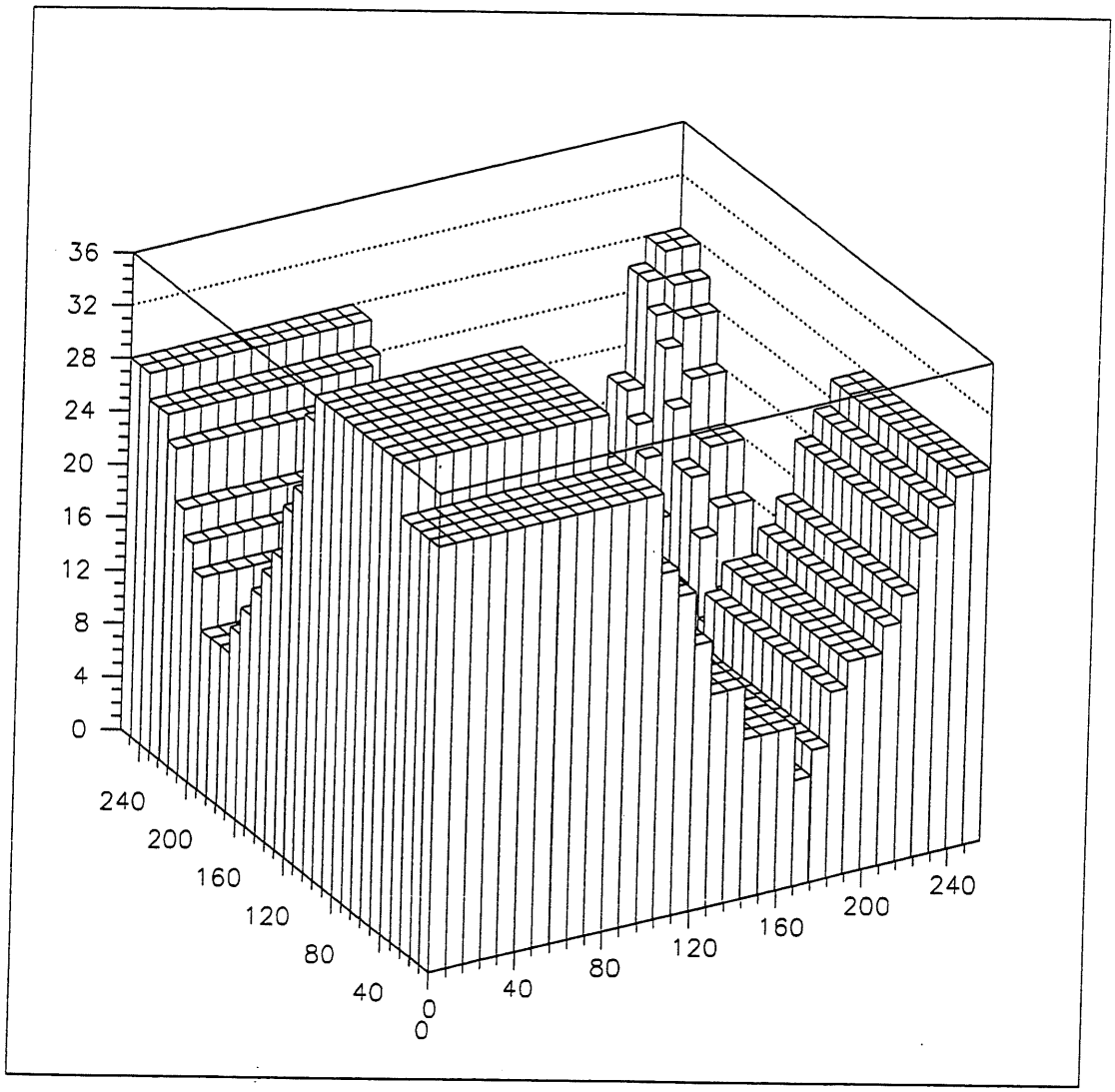


Abb. 6: Häufigkeit der Neuronen, die in einem Raumbereich schalten, wenn das Netz auf das inverse Muster eines zweidimensionalen Kreises trainiert wurde

5.2 Das Konvergenzverhalten eines Netzes

Mit der Konvergenz des Netzes ist hier das Verhalten des Fehlers E gemeint, den das Netz in Abhängigkeit zu der Anzahl der gelernten Werte n erzeugt. E ist die Anzahl der falsch erkannten Daten bezogen auf eine feste Anzahl von Refenenzdaten. In den *Abb.*7, *Abb.*8 und *Abb.*10 wurden der Fehler E als Anzahl der falsch erkannten Daten bezogen auf 10000 Referenzdaten angezeigt. Ein Fehler von $E = 200$ bedeutet somit, daß von 10000 Daten 200 falsch erkannt wurden. Der Fehler E ist somit eine abvlolute Zahl. Der Fehler E ist somit immer auf eine Anzahl von Referenzdaten bezogen. Mit der Fehler E soll im folgenden immer der Fehler bezogen auf 10000 Referenzdaten gemeint sein, da es so zu handliche Werten kommt.

Ein Netz, das digitalisierte Werte verwendet, konvergiert immer. Das liegt daran, daß

durch die Digitalisierung nur eine endliche Anzahl von Werten dargestellt werden kann. Entspricht die Anzahl der Neuronen der Anzahl der unterscheidbaren Eingangswerte, so ist immer eine Entscheidung möglich und der Fehler somit gleich Null. Die Werte werden in einem solchen Fall nicht mehr assoziiert, sondern sind abgespeichert. Es ist also nicht sinnvoll, die mathematische Definition eines Grenzwertes auf den Fehler anzuwenden. Unter dem Fehlergrenzwert soll hier so etwas wie ein Sättigungswert verstanden werden. Man ersetzt also aus praktischen Gründen $E = \lim_{n\to\infty} E(n)$ durch $\lim_{n\to N0}$. Wobei $N0$ so gewählt werden sollte, daß es die technisch maximale Größe darstellt. (Mit $E(n)$ soll der Fehler gemeint sein, der mit einem Netz erzieht wurde, daß mit n Werten trainiert wurde.) $N0$ wird also durch Rechenzeit und Speicherkapazität begrenzt. Außerdem muß $N0$ sehr viel kleiner als die Anzahl der unterscheidbaren Eingangswerte sein, da sonst das Netz seine assoziativen Eigenschaften verliert. Konvergiert der Fehler bis zu dieser Grenze nicht, so wird daraus auf die Konvergenz bei einer idealen Betrachtung geschlossen (mit $n \to \infty$).

Konvergiert ein Lernalgorithmus auf einer Datenstruktur, so sinkt der Fehler bei zunehmender Zahl der Trainigswerte bis zu einem Punkt, wo er dann quasi konstant ist. An diesem Punkt ist das Netz mit Werten gesättigt. Jeder zusätzlich trainierte Wert wird entweder durch das Netz richtig erkannt (vgl. *Abb.*8). Die Anzahl der Neuronen verhält sich anders. Sie steigt bei einem konvergierenden Algorithmus zuerst langsam an, da jetzt noch mit wenigen Neuronen viele Werte erfaßt werden können. Ist dieser Sättigungspunkt erreicht, verhält sie sich linear zur Anzahl der Trainingswerte. Die Anzahl der Neuronen steigt dann an, ohne die Qualität des Netzes zu verbessern (vgl. *Abb.*7). Somit ist die Errechnung dieses Sättigungspunktes eine wichtige Größe, um ein Netz zu minimieren. Läßt sich der Fehler hingegen nicht durch die Anzahl der Trainigswerte beeinflussen, kann das daran liegt, daß die Datenstruktur nicht geeignet, durch ein solches Netz beschrieben zu werden.

Die Existenz eines solchen Sättigungspunktes deutet auf zwei überlagerte Effekte hin. Es gibt zwei Klassen von Meßwerten. Eine Klasse wird durch ein solches assoziatives System ideal beschrieben. Das bedeutet, daß eine kleine Anzahl dieser Werte ausreicht, um auf die restlichen zu schließen. Die zweite Klasse von Meßwerten ist ungeordnet und kann nicht zu räumlichen Bereichen mit quadratischer Grundstruktur zusammengefaßt werden. Das Trennen der geordneten von den nicht geordneten Daten kann über eine Statistik der einzelnen Neuronen erfolgen. Dabei wird festgestellt, ob sich der Fehler des Netzes ändert, wenn ein Neuron aus diesem Netz entfernt wird. Ist dies nur unwesentlich der Fall, kann

dieses Neuron entfernt werden, da es offensichtlich nicht zum assoziativen Verhalten des Netzes beiträgt. Anhand von solchen Messungen kann die Güte des Netzes abgeschätzt werden. Ein wichtiger Parameter ist die Anzahl der erkannten Werte pro Neuron.
Ein genaueres Betrachten des Fehlerverhaltens läßt erkennen, daß der Fehler nicht stetig mit der Anzahl der Trainingswerte fällt, sondern zwischen verschiedenen Kurven springt. Wahrscheinlich ist die Anordnung der Neuronen zu den Trainingswerten bei nicht ausreichender Anzahl der Meßwerte nicht eindeutig. Der Netzfehler springt somit zwischen verschiedenen Netzkonfigurationen. Diese Tatsache hat dazu geführt, daß im Lernalgorithmus bewußt Zufallsmomente eingebaut wurden. Dadurch soll verhindert werden, daß sich durch regelmäßige Anwendung der Optimierungsvorschriften das Netz einseitig auf eine Netzkonstellation festlegt. Durch wiederholtes Trainieren mit einer anderen Zufallsfolge kann somit versucht werden, alle möglichen Netzzustände zu berechnen.
Soll ein Netz endgültig trainiert werden, muß durch Variation der Reihenfolge der Optimierungsschritte beim Lernvorgang geprüft werden, ob mehrere Zustände existieren. Die Existenz solcher 'metastabilen' Netzkonfigurationen läßt sich gut an der Anzahl der Neuronen erkennen. So kann in einem solchen Fall durch Variationen erreicht werden, daß die Anzahl der in der Trainingsphase benötigten Neuronen schwankt, obwohl alle Netze bei den Trainingswerten denselben Fehler erzeugen.

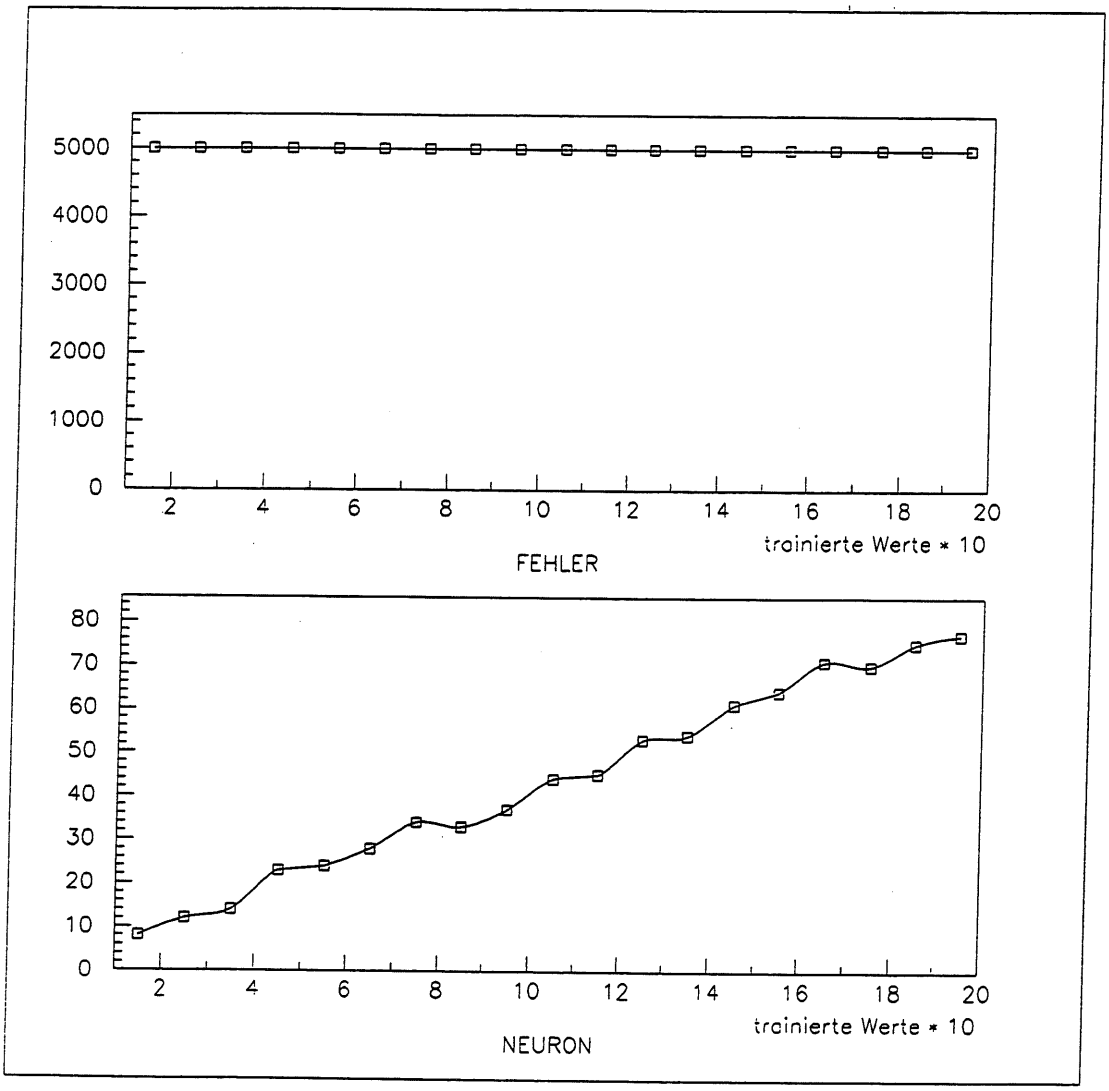


Abb. 7: Netz mit Zufallszahlen trainiert. Der Fehler des Netzes (oben) bei der Vorhersage der Zahlen bleibt unverändert, unabhänig davon wieviele Werte trainiert wurden. Die Zahl der Neuronen (unten) steigt jedoch fast kontinuierlich an.

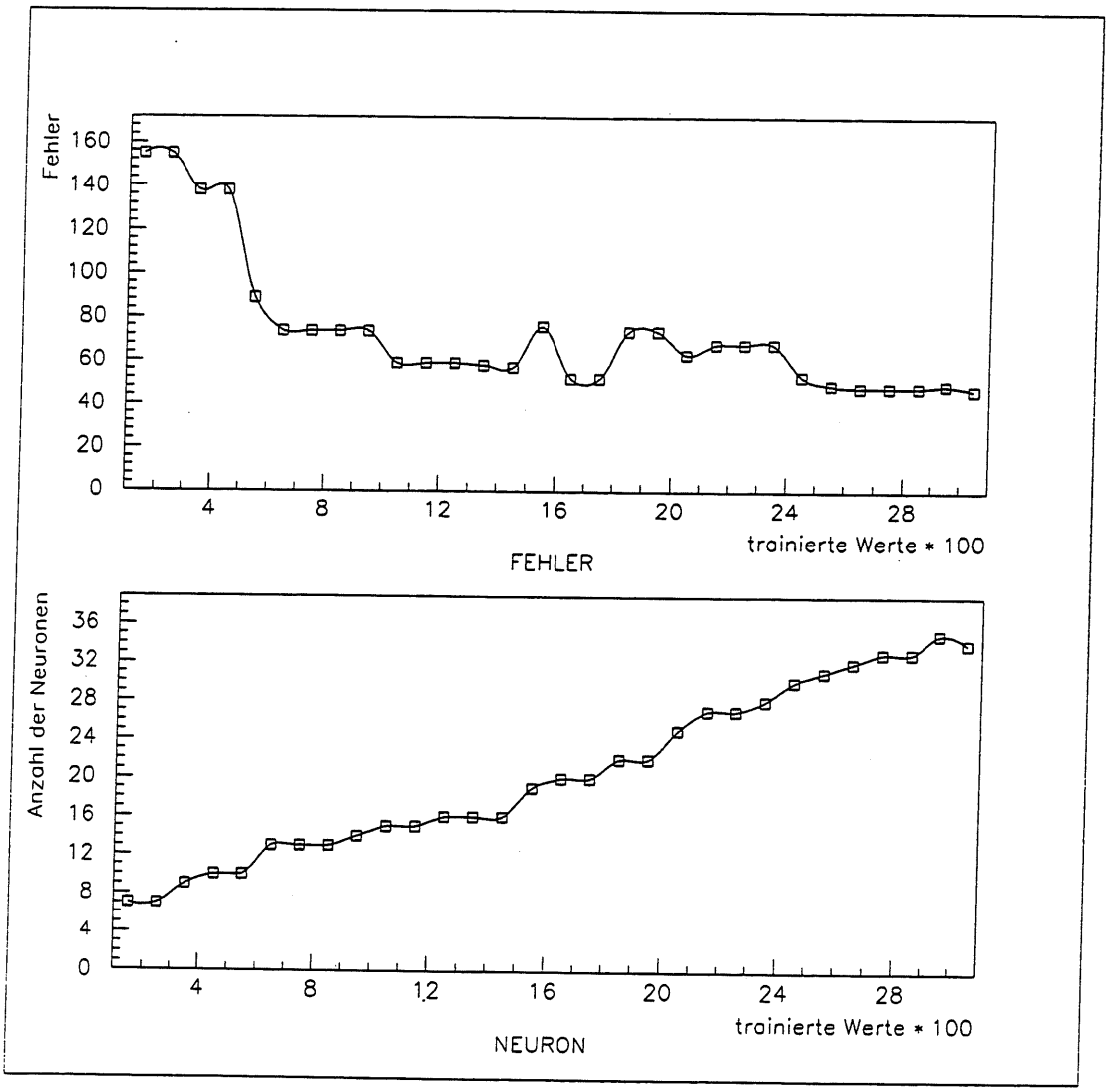


Abb. 8: Netz mit guten Daten trainiert. Der Fehler des Netzes (oben) sinkt bei zunehmender Anzahl von trainierten Werten bis zu einem Punkt, an dem er dann konstant bleibt. An diesem Punkt ist offensichtlich das Netz mit ausreichend vielen Daten trainiert. Aus der Grafik kann so etwas wie der Quotient $M = \frac{Wert}{Neuron}$ *abgelesen werden. Diese Größe beschreibt soetwas wie die Ordnung der Daten und ist äquivalent zu der in folgenden beschriebenen fraktalen Dimension. (Kap. 6.1)*

6 Das neuronale Netz zur Datenanalyse

Offensichtlich kann ein solches Netz nicht nur zur Klassifizierung der Daten eingesetzt werden, sondern aus dem Trainingsverlauf und einer Auswertung der Gewichtsvektoren kann auf die Struktur der Daten geschlossen werden. Die Konvergenzgeschwindigkeit des Netzes ist ein Maß für die Ordnung der Daten. Durch die einfache Struktur der Neuronen sind die Gewichtsvektoren eines solchen Netzes direkt als Karten zu lesen. So gibt jedes Neuron eine Bereichsunter und -obergrenze an, in der es aktiv wird.

6.1 Die fraktale Dimension eines neuronalen Netzes

Die Anzahl der Neuronen ist ein Maß für die 'Rauhigkeit' des Raumkörpers. So könnte ein einfacher glatter Würfel nur durch wenige Neuronen beschrieben werden, ein Schwamm mit seiner fast beliebig großen Oberfläche würde sehr viele Neuronen benötigen. Die Anzahl der Neuronen ist ein Maß für die fraktale Dimension des Raumkörpers. Die ersten Simulationen haben dies auch gezeigt. Die Anzahl der benötigten Neuronen hängt nicht von der Dimension des Eingangsvektors ab, sondern von der Struktur und Anzahl der trainierten Werte. Es hat sich sogar gezeigt, je mehr Komponenten der Eingangsvektor hat, desto weniger Neuronen reichen aus, um die gleiche Entscheidungsqualität zu erreichen. Dies Ergebnis war nicht zu erwarten, da auch vermutet werden konnte, daß die Neuronenzahl quadratisch oder exponentiell mit der Dimension der Eingangsgrößen steigt. So konnte mit Testdaten eine Entscheidung im 18- dimensionalen Raum mit einem Fehler von unter 1% mit nur etwa 80 Doppelneuronen getroffen werden. Für einen vollständigen Kreis zum Vergleich werden im nur zweidimensionalen Raum bei der gleichen Auflösung schon 40 Neuronen benötigt.

Ist ein neuronales Netz trainiert worden, so kann daraus die fraktale Dimension des Zustandsraumes bestimmt werden. Das Verfahren der orthogonalen Neuronen, bei dem ein Körper im Zustandsraum durch immer kleinere Würfel vermessen wird, ähnelt stark dem Verfahren, mit dem in der fraktalen Geometrie die Dimension bestimmt wird. So wird bei der fraktalen Dimensionsvermessung ein geometrischer Körper durch einen anderen vermessen. Eine mehrparametrige Größe wird durch eine einparametrige beschrieben. So wird eine Kugel mit dem Radius R z.B. mit einer Anzahl von Würfeln der Kantenlänge r aufgefüllt. Der Zusammenhang zwischen der Kantenlänge des Würfels und der Anzahl der Würfel, die in die Kugel passen, ist die dritte Potenz ($V = \frac{4}{3}\pi r^3$). Die Potenz ist die fraktale und hier natürlich auch geometrische Dimension der Kugel. Ein anderes Beispiel

ist die Vermessung einer Küstenlinie mit einem immer kürzeren Lineal. Bei der Küstenlinie von England hängt die Lineallänge und der Anzahl der Lineale etwa mit der Potenz 2,7 zusammen. Die fraktale Dimension von England ist somit 2,7.[3]

Offensichtlich ist die Anzahl der orthogonalen Neuronen in Abhängigkeit von der Neuronengröße vergleichbar mit der fraktalen Dimension des Körpers im Zustandsraum. Die Dimension ist nicht konstant, sondern ändert sich mit der Größe der Neuronen; so wird sie am Anfang ansteigen, bis sie dann aufgrund von Meßungenauigkeiten wieder abfällt, da die Werte aufgrund der Digitalisierung und der natürlichen Meßfehler nicht beliebig unterteilt werden können. So ist die Statistik der Neuronengröße über die Neuronenanzahl auch ein Maß für die Qualität der Meßwerte.

6.2 Auswahlverfahren der Triggerkomponenten

Mit Hilfe eines Netzes kann festgestellt werden, welche Datenkombination aus den etwa 50 Eingangskanälen (s. Anhang) eine hohe Aussagekraft haben und welche eher nur den Aufwand des Triggers vergrößern, ohne die Entscheidungsfähigkeit zu verbessern. Der zu konstruierende Trigger soll nicht alle Daten sehen, sondern mit etwa 10-18 Kanälen auskommen. Dies ist eine Vorgabe zur Konstruktion, die davon ausgeht, daß mit einer solchen Anzahl von Eingangskanälen eine ausreichend gute Entscheidung getroffen werden kann. Die Auswahl der Kanäle hängt maßgeblich von der Art des Ereignisses ab, auf die der Trigger reagieren soll. So kann für verschiedene Experimente durchaus eine andere Auswahl aus den 50 Kanälen sinnvoll sein. Um den Aufwand eines Triggers zu minimieren, sollte die Anzahl der Eingangskanäle möglichst klein sein.
Natürlich können auch nach einer Vorauswahl nicht alle Kombinationen von Eingangskanälen ausprobiert werden. Bei 50 Kanälen wären das Fakultät 50. Es wurden zuerst die Kanäle einzeln, dann alle zweier und dann alle dreier Kombinationen berechnet. Ein Auswahl kann nun nach verschiedenen Strategien erfolgen. Einmal können nun aus den so gebildeten einer, zweier und dreier Kombinationen die Besten herausgesucht werden und anschließend untereinander wieder mit den übrigen kombiniert werden. Oder es wird ein Netz mit allen Komponenten mit Ausnahme von ein bis drei Komponenten trainiert. Die erste Strategie findet die aussagekräftigen Werte, während die zweite die redundanten herausfiltert. Ein neuronales Netz kann also nicht nur als assoziativer Speicher eingesetzt

[3]vgl. / Die Entdeckung des Chaos / John Briggs, F. David Peat / Hansa Verlag / 1990

werden, durch das Trainieren mit Daten kann der Informationsgehalt der Daten und ihrer Komponenten bestimmt werden. Die Daten eines Ereignisses liegen als Vektor mit 50 Komponenten vor. Die tatsächliche Dimension der Daten ist aber nicht so hoch. Wie im Abschnitt über die fraktale Dimension schon beschrieben wurde, kann eine Dimensionsbestimmung der Daten durch ein solches orthogonales Netz erfolgen.

6.3 Die assoziative Dimension

Durch das hier beschriebene Verfahren wird versucht, aus den einzelnen Komponenten des Eingangsvektors eine minimale orthogonale Basis zu finden. Ein solches Problem kann schlecht durch ein übliches Determinatenverfahren gelöst werden, da die Dimension des Datenraumes nicht in allen Stellen gleich ist. So haben im Datenraum die meisten Stellen, an denen ähnliche Daten liegen, eine sehr niedrige fraktale Dimension. An den Stellen, an denen sich die Daten stark ändern, steigt die fraktale Dimension der Daten an. Es erscheint sinnvoll, für einen solchen Dimensionsbegriff den Begriff der assoziativen Dimension einzuführen, um Verwechselungen mit den Fraktalen zu vermeiden. Diese assoziative Dimension soll mit M bezeichnet werden und wird durch das im Abschnitt 'Konvergenz des Netzes' beschriebene Verfahren gemessen. Durch das Verfahren wird so etwas wie die globale Dimension bestimmt. Offensichtlich gibt es eine lokale Dimension m, die an den Randbereichen der Daten stark ansteigt. Dieser Wert gibt an, wieviel Neuronen pro Daten erforderlich sind. Dieser Wert ist analog zu der Anzahl der orthogonalen Vektoren, die zur Beschreibung eines Punktes in einem geometrischen Raum benötigt werden. Die Anzahl dieser Vektoren in einem geometrischen Raum ist die Dimension des Raumes. Aus diesem Grund erscheint es sinnvoll, diesen Wert M ebenfalls als Dimension zu bezeichnen.

6.4 Modellbildung durch ein neuronales Netz

Ein Netz wird mit bekannten Werten trainiert. Anhand dieser Werte kommt das Netz dann zu Hypothesen über noch unbekannte Werte. Die Art dieser Hypothesen ist durch die Struktur des Netzes vorgegeben. So ist bei einem feed forward Netz das Ähnlichkeitskriterium für diese Hypothesen entscheidend. Bei den selbstorganisierenden Karten nach Kohonen bilden sich diese Hypothesen selbständig durch den Optimierungsvorgang. Je nach Art des Optimierens können entsprechende Vorgaben gemacht werden.
Mathematisch gesehen bildet ein Netz den Eingangsraum in den Zustandsraum der Neuronen ab. Durch die Optimierung und den Vergleich der Ergebnisse des Netzes mit den

erwarteten Werten wird versucht, daß das Netz das gleiche Verhalten wie die wirkliche Welt zeigt. Ist ein solches Netz dann erfolgreich trainiert worden, so kann das Netz als ein vereinfachtes (bzw. auf das Wesentliche reduzierte) Modell der Wirklichkeit verstanden werden. Offensichtlich gelingt durch die Transformation in den Zustandsraum der Neuronen so etwas wie eine Abbildung der realen Welt in den Zustandsraum des Netzes. Um ein Netz zur Datenanalyse zu benutzen, kann alternativ zu der Analyse der realen Daten die Struktur des Netzes untersucht werden. Wie schon im Abschnitt über die assoziativen Speicher angedeutet, ist ein solches Netz in der Lage, nur aus Meßwerten einen dreidimensionalen Raum zu entwickeln.

Andererseits kann diese Modellbildung durch ein solches Netz dazu eingesetzt werden, um den Rechenaufwand für entsprechende Probleme zu minimieren. So ist es denkbar, daß der Strahlverfolgungsalgorithmus, wie er zum Beispiel für die Berechnung von natürlichen Bildern verwendet wird, durch ein neuronales Netz verbessert werden kann. Hierzu könnte dann Netze mit den optischen Eigenschaften der geometrischen Grundelemente trainiert. Höhere geometrische Ordnungen würden dann aus solchen Netzen zusammengesetzt (superpositioniert). Durch das Trainieren der Netze könnten von den geometrischen Objekten Modelle erzeugt werden und die weiteren Berechnungen wird dann mit diesen Modellen durchgeführt werden. Die Rechengenauigkeit dieser Modelle ist dann natürlich wesentlich kleiner als die der exakten Lösung. Für die Berechnung von solchen Bildern ist es aber eigentlich nicht notwendig, mit einer Auflösung von z.B. 80 Bit Floatingpoint zu rechnen. Um die Position eines Schattens oder die Reflexion einer Lichtquelle zu beschreiben, ist eigentlich nur eine Auflösung in der Größenordnung der Bildschirmdarstellung erforderlich. So konstruiert ein Maler ja seine Bilder auch nicht mit einem Taschenrechner, sondern er hat gelernt, die Schatten und Reflexionen an die richtigen Stellen zu setzen.

Wird bei einem solchen Algorithmus nicht mit der exakten Lösung gerechnet, sondern werden dazu neuronale Netze als Modelle der Objekte verwendet, so wird der Rechenaufwand erheblich reduziert, ohne daß sich der optische Gesamteindruck wesentlich verändert. Werden diese Netze dann durch entsprechende Hardware (z.B. mit orthogonalen Neuronen) realisiert, ist eine um Größenordnungen kleinere Verarbeitungszeit zu erwarten.

7 Der Lernprogramms

Das Lernprogramm weicht von den klassischen Verfahren ab. Bei den klassischen Verfahren wird eine fertige Netzstruktur mit einer festen Anzahl von Neuronen an den Anfang gestellt. Die Verknüpfungsgewichte sind zufällig oder durch sinnvolle Abschätzung gewählt. Der Fehler eines solchen Netzes wird durch einen Satz von bekannten Daten bestimmt, der durch das Netz berechnet wird. Die Antwort des Netzes wird mit der gewünschten Antwort verglichen. Der Gesamtfehler über alle Daten wird anschließend gewichtet aufsummiert. Durch Variation der einzelnen Gewichte kann der Fehlergradient bestimmt werden. Durch entsprechende Optimierungsverfahren werden die Gewichte dann solange verändert, bis der resultierende Gesamtfehler ein Minimum erreicht hat.
Der hier verwendete Algorithmus geht anders vor. Aufgrund der Orthogonalisierung der Neuronen können die Gewichte der Neuronen unabhängig voneinander verändert werden. Der Gesamtfehler kann aus den Fehlern der einzelnen Neuronen superpositioniert werden. Diese beiden Eigenschaften macht sich der Algorithmus zunutze. Ausgegangen wird von einem Neuron, daß so gewählt wird, daß es die Daten optimal beschreibt. Ist noch ein Restfehler vorhanden, so wird ein nächstes Neuron erschaffen, das dann wiederum solange variiert wird, bis der verbleibende Fehler minimiert ist. Das Verfahren wird solange wiederholt, bis kein Fehler verbleibt. Das Programm gliedert sich in folgende Schritte:

1. Starttemperatur setzen = $t1$

2. Kann ein neues Neuron der Breite $t1^n$
   gefunden werden?
   Wenn ja , dann 5.

3. $t1$ erniedrigen und bei 2. weitermachen, bis $t1 = 0$.
   Dann 4.

4. Überflüssige Neuronen aussortieren und Programm beenden

5. Temperatur $t2$ setzen

6. Wählt eine Komponente des Neurones aus ;
   variieren dieser Komponente mit $t2$;
   bewerte die Variation.

7. Wenn die Variation das Netz verbessert hat,
   übernehme die Variation

8. Wenn keine Variation mit $t2$ zum Erfolg führt,
   verkleinere $t2$ und gehe zu 6.

9. Suche nächstes Neuron, gehe also zu 2.

Beschreibung zu Punkt 2: Ein Neuron kann schnell gefunden werden, indem ein Wert aus der zu lernenden Tabelle zur Berechnung benutzt wird. Hierzu werden dann die Komponenten des zu trainierenden Wertes in die Komponenten des zu Neurones eingesetzt. Als Schaltbreiter dieses Doppelneurons wird dann die Temperatur $t1$ als Schaltbreite eingesetzt. Da mit einer sehr großen Temperatur begonnen wird, werden zuerst die Neuronen mit der größten Schaltbreite gefunden. Anschaulich betrachtet sind sie die Punkte, die zu den Rändern des zu beschreibenden Raumkörpers die größten Abstände haben, also in der Mitte des Raumkörpers liegen. Diese Punkte dienen quasi als Kristallisationskeim.
Beschreibung zu Punkt 6: Das in Punkt 2 gefundene Neuron wird jetzt 'eingeparkt'. Hierzu werden die Parameter systematisch in immer kleineren Schritten variiert. Hierzu können verschiedene Variationsregeln aufgestellt werden. Die Auswahl der Variationsregeln kann nach unterschiedlichen Verfahren erfolgen. Bei dem Programm hat sich experimentell folgende Strategie als erfolgreich erwiesen: Es gibt drei Variationsregeln. Die beiden Seiten getrennt zu verschieben und beide Seiten mit der halben Weite gleichzeitig

nach außen zu schieben. Die drei Variationen werden ausgeführt und die am besten bewertete wird dann übernommen. Die Variationsregeln können über jede Komponente des Neurons ausgeführt werden. Es hat sich herausgestellt, daß es nicht sinnvoll ist, die Variation der Komponenten in fester Reihenfolge nacheinander durchzutesten. Die Auswahl erfolgt mit einem Zufallsgenerator. Die Neuronen diffundieren bei diesem Schritt also in ihre beste Position, d.h. zu dem kleinsten Fehler hin.

Bei der Bewertung eines Neurons werden zuerst die Übereinstimmungen mit den zu lernenden Werten gezählt. Falsches Verhalten wird so schlecht bewertet, daß die Übernahme einer solchen Variation ausgeschlossen werden kann. Ein zweiter Faktor ist die Fläche des Neurons. Eine Ausdehnung des Neurons wird positiv gezählt, um damit eine Überlappung der Neuronenbereiche zu erreichen. Das Flächenkriterium wird aber nur bis zu einer Minimalbreite angewendet, da sonst bei kleinen Meßwertdichten Neuronen zu weit in verbotene Bereiche wachsen können, bis sie auf einen Wert stoßen (vgl. *Abb.*9).

Der hier beschriebene Algorithmus zeichnet sich durch eine hohe Rechengeschwindigkeit aus. Diese ist auch wichtig, da große Netze und viele Trainingswerte verarbeitet werden sollen. Weiterhin sind die Ergebnisse des Lernvorganges direkt zu interpretieren, da die Liste der Neuronen als eine Art Karte gelesen werden kann in der die signifikanten Bereiche aufgelistet sind. Um diese Art der Interpretation zu begünstigen, werden die Neuronen durch ihre Mitte und der entsprechenden Ausbreitung von dort aus abgespeichert.

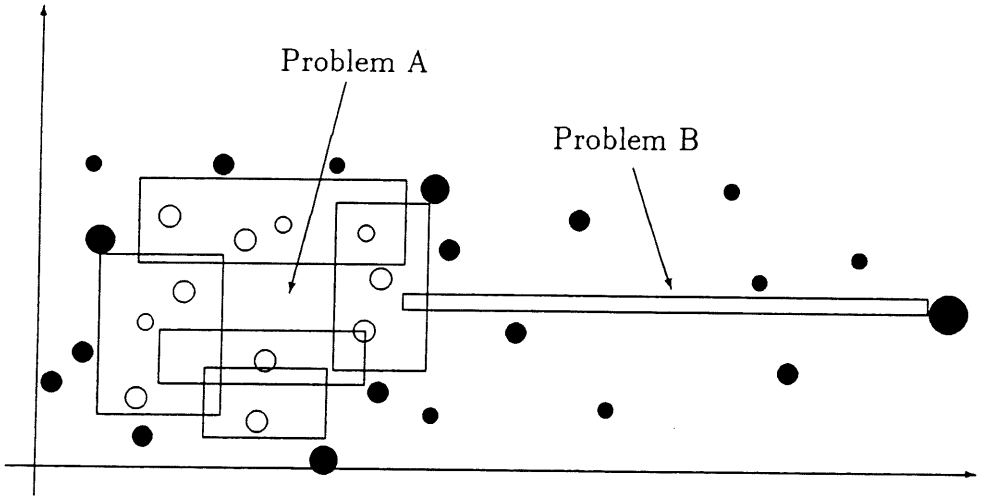


Abb. 9: Problem A soll durch das Flächenbewertungskriterium umgangen werden. Dadurch, daß die Flächen maximal expandiert werden, kann Problem B entstehen. Durch eine minimale Seitenbreite, ab der das Flächenbewertungskriterium nicht mehr angewand wird, soll dies Problem umgangen werden.

Das in der Diplomarbeit erstellte Simulationsprogramm ist in der Lage, ein Netz mit orthogonalen Neuronen zu trainieren. Die Anzahl der Dimensionen, Neuronen und Trainingswerte ist beliebig und nur durch maschinelle Größen begrenzt. Zusätzlich wurden einige Hilfsprogramme zur Datenkonvertierung und zur Filterung entwickelt. Ein Programm vergleicht ein errechnetes Netz mit Referenzdaten und errechnet den Fehler. Die Programme wurden klein gehalten und werden durch eine UNIX-Shell verbunden. Der Datenaustausch geschieht über das Filesystem und Pipes, so daß die entsprechenden Zwischenschritte im Texteditor verfolgt werden können. Die Programme sind eine Art 'neuronaler Baukasten', die flexibel durch eine entsprechende Shell gesteuert werden.

Im wesentlichen wurde folgende Teilprogramme erstellt:

- **TRANS-IBM** : Dieses Programm überträgt Daten aus dem zentralen Rechner auf die hier verwendete HP- Workstation über Ethernet. Dieses Programm ist erforderlich, um Zugang zu den Daten der Experimente zu bekommen.
- **DOUBLE** : Dieses Programm sortiert die Daten vor. Doppelte und widersprüchliche Daten können erkannt und entsprechend behandelt werden.
- **OPTI** : Durch dieses Programm wird der Wertebereich der Daten so skaliert, daß

der gesamte Zahlenbereich benutzt wird.

- **CUTF** : Dies ist ein universeller Filter, um nach gewüschten Komponenten oder Eigenschaften zu suchen oder diese zu verändern.
- **SUFFEL** : Durch dieses Programm werden die Daten in einem File gemischt, um eine Abhängigkeit der Daten von deren Reihenfolge beim Trainieren des Netzes auszuschließen.
- **NTOOL** : Dieses Programm trainiert ein orthogonales Netz mit Daten und erzeugt eine universelle Liste mit den Gewichtsvektoren.
- **BACKP** : Dieses Programm erzeugt eine Liste von Gewichtsvektoren mit Hilfe des Backpropagation Algorithmus. [4]
- **WORKNET** : Dieses Programm benutzt die Liste der Gewichtsvektoren und errechnet aus einem Datensatz die entsprechende Antwort eines Netzes.
- **L1-TRIGG** : Dieses Programm simuliert den L1 Trigger. Dieser Trigger ist dem Netz vorgeschaltet und übernimmt eine Vorentscheidung über die Daten.

Das Format der Gewichtsvektorenliste ist so gestaltet, daß sowohl das orthogonale wie auch das Backpropagation Netz alternativ verwendet werden können. Die Programme Ntool und L1-Trigg sind in Ausschnitten im Anhang ausgedruckt.

[4] vgl : / Ein Backpropagation Netz unter Turbo Pascal / Walter Kirchner / c't 11-90

8 Vergleich des orthogonalen Netzes mit einem Backpropagation Netz

Zum Vergleich wurde hier ein 3-Schichten-Netz verwendet. Es gliedert sich in eine Eingangsschicht, einer verdeckten Schicht und einer Ausgangsschicht. Verwendet wurde ein solches Modell, da das orthogonale Netz einen Spezialfall dieser Architektur darstellt. Bei dem orthogonalen Netz wurden die Schwellwertfunktionen durch diskrete Vergleicher ersetzt. Bei dem 3-Schichten-Netz wurde in allen drei Schichten eine Rechenauflösung von 80 Bit-Floatigpoint verwendet, während das andere Netz mit der technisch realisierbaren Auflösung von 9 Bit in der 1. Schicht und jeweils 1 Bit in der 2. Schicht simuliert wurde. Der Lernvorgang beider Netze ist von Ansatz her völlig verschieden. Der Backpropagation Algorithmus ist eine Optimierungsvorschrift für ein fertiges Netz:
Dazu wird ein Eingangswert auf das Netz gegeben und der Ausgangswert errechnet (Feedforward Schritt). In einem zweiten Schritt wird die Abweichung zu dem gewünschten Wert bestimmt. Bei dem Feedforward Schritt wurden alle Gewichte markiert, die das Ergebnis vergrößert oder verschlechtert haben. Soll der Ausgangswert vergrößert werden, so werden nun alle hemmenden Gewichte verkleinert und alle positiv wirkenden Gewichte vergrößert (Feedbackward Schritt). Das Verfahren für die Gewichtsveränderung ist an das newtonsche Nullstellenverfahren angelehnt. Die Änderungsrate für eine Iteration ist variabel und ändert sich im Laufe des Trainierens. So wird mit einer großen Änderungsrate begonnen, bis der Fehler ein lokales Minimum erreicht hat, dann wird sie solange verkleinert, bis das Netz ausreichend trainiert ist. Da bei diesem Algorithmus alle Gewichte unabhängig voneinander verändert werden können, ist zu erwarten, daß ein solches Optimierungsverfahren anfällig für lokale Minima ist, da die Anzahl der Freiheitsgrade sehr groß ist. Das Verhalten des Netzes hängt stark von den Anfangswerten der Gewichte ab. Um dieses Problem einzugrenzen, wurde für diesen Vergleich jeweils fünf mal ein Netz mit zufällig gewählten Gewichten verwendet. Aus diesen Netzen wurde das am besten trainierte ausgewählt.
Untersucht wurde dann das assoziative Verhalten der Netze. Dazu wurden die beiden Netze mit bekannten Werten trainiert. Aufgetragen wurde dann der Fehler der Netze auf 10000 unbekannten Daten. Die Anzahl der Trainigswerte wurde schrittweise erhöht. Aus *Abb.*10 kann der Fehler des Netzes auf unbekannte Daten und die Assoziativfähikeit abgelesen werden.

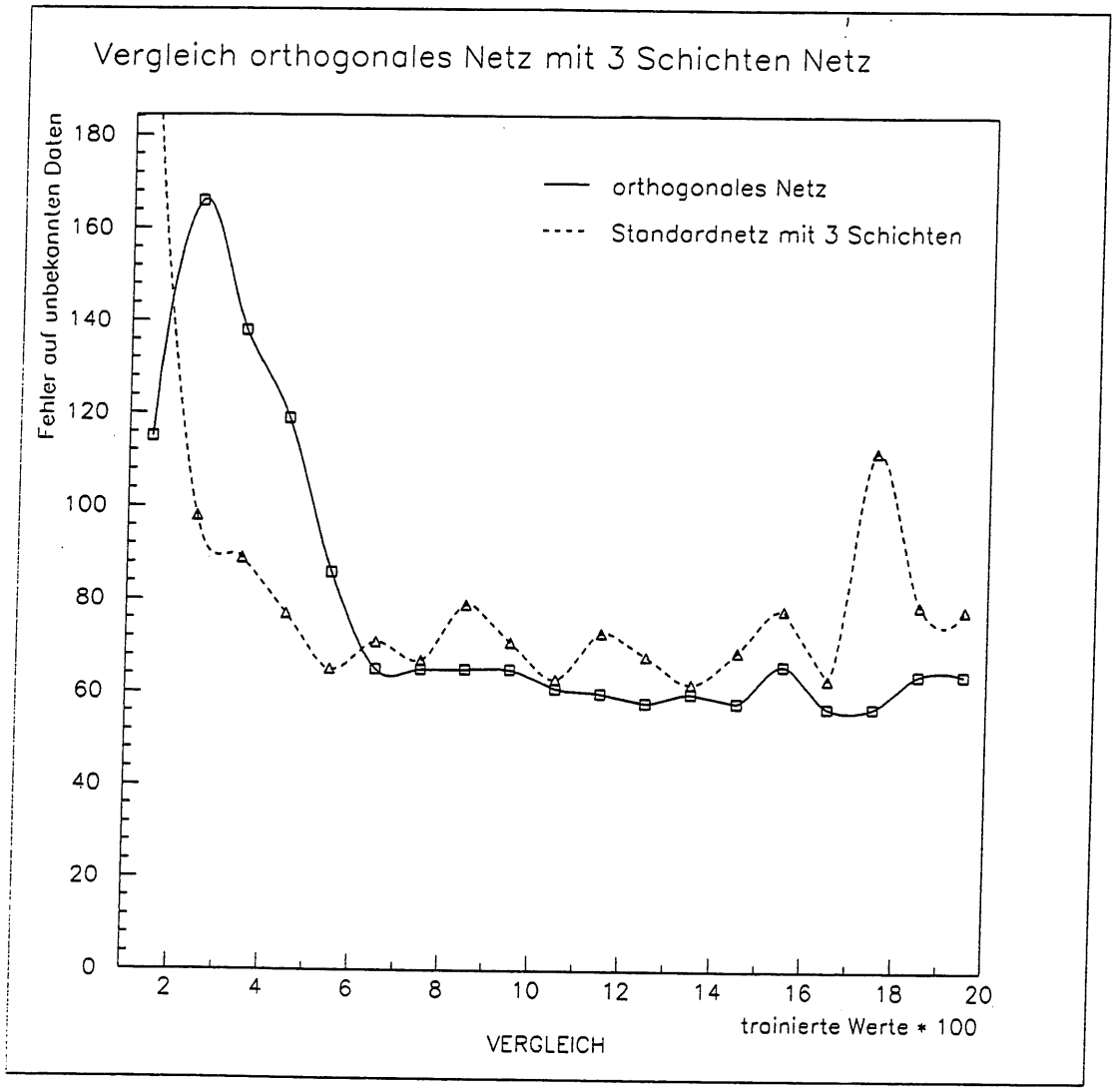


Abb. 10: Vergleich des orthogonalen Netzes mit einem vergleichbaren Backpropagation Netz

Um die Netze vergleichen zu können, wurde in einem ersten Schritt ein orthogonales Netz mit der maximalen Anzahl der Trainingswerte berechnet. Da sich bei diesem Algorithmus die Anzahl der Neuronen erst beim Trainingsvorgang ergibt, bei dem Standardverfahren aber mit einer festen Neuronenzahl begonnen werden muß, konnte so die maximale Neuronenzahl bestimmt werden. Dies waren in diesem Beispiel 20 Doppelneuronen. (Verwendet wurden in diesem Beispiel eine Auswahl von 6 Kanälen aus den Triggerdaten, die durch das weiter oben beschriebene Optimierungsverfahren bestimmt wurden.) Um diese Neuronen auf das andere Netz abzubilden, müssen 40 Neuronen in der verdeckten Schicht verwendet werden. Mit diesem Wert wurden die Simulationen für das Netz durchgeführt. Bei dem Vergleich zeigt sich folgendes:

Im Bereich bis etwa 300 Trainingswerten ist das Verhalten der Netze eher zufällig. Die Anzahl der Daten reicht nicht aus, um auf unbekannte Daten zu schließen. Im Bereich bis 600 Trainingswerten ist das Standardnetz besser in der Lage zu assoziieren. Es reichen weniger Trainingswerte aus, um die unbekannten Werte zu berechnen. Diese Fähigkeit wird vom orthogonalen Netz erst ab etwa 600 Trainingswerten erreicht. Offensichtlich zeigen beide Netze ein Sättigungsverhalten mit Daten. Ab diesem Sättigungspunkt verbessert sich die Qualität der Netzes nicht mehr. Beide Netzstrukturen erscheinen gleichwertig. Bei dem Standardnetz ist deutlich das Auftreten von lokalen Minima zu erkennen, da der Fehler stark schwankt und nur an einigen Stellen den Fehler der orthogonalen Netze erreicht. Das Problem mit den lokalen Minima tritt bei dem Kristallistionsalgorithmus des orthogonalen Netzes nicht auf, da hier keine Anfangswerte benötigt werden. Ein anderer Effekt deutet sich am Ende der Kurve an. Durch eine zu große Anzahl von Trainingswerten verschlechtert sich das Verhalten des Standardnetzes wieder. Offensichtlich hat der Backpropagationsalgorithmus Schwierigkeiten mit redundanten Datensätzen.
Da bei dem hier verwendeten Triggerproblem die Daten in fast beliebiger Anzahl zur Verfügung stehen, ist das spätere Eintreten des Sättigungspunktes kein wirklicher Nachteil. Beachtet man dagegen das Wegfallen der Multiplikationen durch die Orthogonalisierung, den schnelleren Lernalgorithmus und die geringere Anfälligkeit für lokale Minima, ist der Ansatz mit den orthogonalen Neuronen erstaunlich gut, obwohl er eine starke Vereinfachung ist.

9 Genetische Lernalgorithmen

Dieser Algorithmus soll als Erweiterung des Kristallisationsalgorithmus eingesetzt werden. Bei einem solchen Algorithmus werden Lösungen in einer Art evolutionären Prozeß gezüchtet. Dazu werden die besten Lösungen kombiniert, in der Hoffnung, daß sich aus der Kombination noch bessere Lösungen ergeben. Ein gutes Beispiel für einen solchen Algorithmus stellen simulierte Einzeller in einer Ursuppe dar. In dieser Ursuppe sind die Einzeller Umwelteinflüssen ausgesetzt, auf die sie reagieren sollen. Einzeller, die Umwelteinflüsse besser als andere vorhersagen können, überleben, die anderen werden aussterben. Das Programm eines solchen Einzellers wird als Chromosom bezeichnet. Die Chromosomen können durch Mutationen verändert werden. Dazu wird ein zufälliges Stück des Chromosom verändert. Ebenso können die Chromosomen durch Kombinationen mit anderen verändert werden. Dazu werden Stücke aus den Chromosomen zweier Einzeller ausgetauscht (Crossing-over). Es zeigt sich, daß auf diese Weise Lösungen für komplizierte Probleme gezüchtet werden können.[5] Das hier untersuchte Triggerproblem kann auf ein solches Ursuppenproblem zurückgeführt werden. Der verwendete Trigger muß aus den Eingangsdaten, die den Umwelteinflüssen entsprechen, die Lösung vorhersagen. Offensichtlich erfüllt die Codierung der Neuronen in einem Netz mit orthogonalen Neuronen die Bedingungen für einen genetischen Algorithmus. Jedes Neuron ist für einen abgegrenzten Bereich von Meßwerten zuständig. Entfernt oder verändert man eines, so ändert sich nur das Ergebnis, das durch die Veränderung dieses Neurones betroffen ist, die anderen Neuronen werden dadurch nicht beeinflußt. Netze können einfach durch Crossing-overs kombiniert werden, indem die Neuronen der Netze ausgetauscht werden. Ein solcher Algorithmus scheint eine vielversprechende Erweiterung des Kristallisationsalgorithmus zu sein.

9.1 Genetische Codierung

Das Programm in den Chromosomen muß bestimmte Eigenschaften aufweisen. Ein normales Computerprogramm würde nie durch einen genetischen Algorithmus optimiert werden können, da eine Änderung des Codes fast immer zu einem unbrauchbaren Programm führt. Ein Algorithmus muß also so codiert werden, daß er nicht kritisch auf solche Änderungen reagiert. Bei dem oberen Beispiel ist in den Chromosomen eine Übergangstafel verborgen. Ändert sich eine Stelle in der Tafel, so ändert sich das Verhalten des Einzellers

[5] vgl. / Genetische Algorithmen / A. K. Dewdney / Computer Kurzweil 1 / Spektrum der Wissenschaft: Verständliche Forschung / 1988

nur auf bestimmte Ereignisse, der Rest der Tafel funktioniert aber weiter. Das entscheidene Problem bei Genetischen Algorithmen ist es, eine solche geeignete Darstellung des Problems zu finden. Daraus ergibt sich die Forderung, daß ein derartiges Programm in unabhängige Stücke zerlegt werden kann, die dann verändert werden können, ohne das Gesamtprogramm zu stark zu beeinflussen. Die Stücke des Chromosoms sollen als Gene bezeichnet werden. In einem Gen sind alle Regeln für eine bestimmte Eigenschaft zusammengefaßt. Die Gene sind in einem Chromosom hintereinander in einer Kette aufgeführt und bilden so das Chromosom. Die Position der Gene auf dem Chromosom kann sich ändern, so daß ähnliche Gene zusammen wandern können. Durch eine Crossing-over Operation wird ein ganzes Stück mit mehreren Genen getauscht.

9.2 Auffinden von abstrakten Regeln

Ist eine solche Codierung gefunden, hofft man, daß durch Optimierung der einzelnen Teile das Gesamtprogramm verbessert wird. Es wird erhofft, daß es zu evolutionären Sprüngen kommt. Dies bedeutet, daß das Ergebnis nicht nur die Summe der einzelnen Optimierungen ist, sondern sich durch ständiges Optimieren sprungartig ein Programm mit völlig neuen Eigenschaften ergibt. Dazu kommt es wahrscheinlich, weil durch die Optimierung der einzelnen Gene die Gene nicht mehr nur einfache Übergangstafeln darstellen, sondern abstrakte Funktionen übernehmen. Die abstrakte Codierung ist eine optimale Lösung für ein Gen. Ist dieses Gen für die Temperaturerwartung zuständig, so werden sich in diesem Gen durch Crossing-over Optimierungen im Laufe der Zeit alle guten Informationen zur Temperaturvorhersage ansammeln, die irgendwann einmal in der Population aufgetaucht sind. Durch eine geeignete Codierung und mit Hilfe von Mutationen werden diese Bedingungen dann optimiert. Eine solche Optimierung geschieht nicht durch Rechenregeln, sondern durch einfaches Ausprobieren und Testen. Offensichtlich ist es wichtig, daß eine Eigenschaft nicht durch ein einziges Gen bestimmt wird, sondern daß mehrere Gene darum konkurrieren, eine Aktion zu bewirken. Zuerst wird jedes Gen für eine gewisse Eigenschaft eine Aktion bewirken. Eine abstrakte Beschreibung ergibt sich dadurch, daß sich Gene für allgemeine Eigenschaften herausbilden und spezielle Gene Ausnahmefälle regeln.[6] Ein solcher Algorithmus hat es dann geschafft, eine Reihe von Eingangswerten auf ein Grundprinzip zurückzuführen und Ausnahmefälle von diesen Prinzip zu erkennen. Ist eine solche Abstraktion gelungen, kommt es zu den gewünschten Evolutionssprung.

[6] vgl / Genetische Algorithmen / John H. Holland / Spektrum der Wissenschaft 9-92

9.3 Parameter für genetische Algorithmen

Für genetische Algorithmen sind mehrere Parameter entscheidend. Dies sind einmal die Mutationsrate und die Art der Mutation. Ebenso ist die Art des Crossing-overs entscheidend und nach welchen Kriterien die Gene auf dem Chromosom zusammenwandern und die Bewertung der konkurrierenden Regel in einem Gen. Klassische Parameter für evolutionäre Prozesse sind die Populationsgröße und der Evolutionsdruck. Durch diese Parameter wird die Variationsbreite der Gene in einer Population geregelt. Eine Veränderung der Gene, die zu einer abstrakteren Beschreibung führen, wird meistens zuerst zu einer Verschlechterung der Gesamteigenschaften führen, da hier noch genügende Ausnahmeregeln fehlen. In großen Populationen und einem kleinen Selektionsdruck werden solche Einzeller nicht sofort aussterben, sondern können sich weiterentwickeln. Ein wichtiger weiterer Einfluß ist das Auftreten von isolierten Populationen, die sich unabhängig von der Gesamtpopulation entwickeln können und später mit der Gesamtpopulation wieder rekombiniert werden.

9.4 Die Rechenzeit des Kristallisationsalgorithmus

Die Rechenzeit bei dem Kristallisationsalgorithmus läßt sich abschätzen durch das Produkt der Anzahl der Trainingswerte $N_{Messwerte}$ und der Anzahl der Neuronen N_{Neuron}. Dies ergibt sich aus der Überlegung, daß bei dem Algorithmus für die Berechnung jedes Neurons alle Meßwerte in einer Schleife nacheinander verglichen werden. Dies ist natürlich eine Vereinfachung, da die genaue Rechenzeit von der Struktur der Daten abhängt.
Da die Anzahl der Neuronen aber mit der Anzahl der Meßwerte über eine Konstante M zusammenhängt (s. assoziative Dimension), ergibt sich die Rechenzeit für ein Netz in Abhängigkeit von der Anzahl der Meßwerte als :

$$Z \approx N_{Msswerte} * N_{Neuron} = N_{Messwerte} * \frac{N_{Messwerte}}{M} = a * N^2_{Messwerte}$$

Vermutlich ist die Rechenzeit für sehr große Datenmengen bei einem genetischen Algorithmus kleiner, da hier zwar mehrere Netze gleichzeitig trainiert werden, aber aufgrund der Selektion nur die Gene variiert werden, die eine Verbesserung bewirken. Optimale Gene werden sich in einer Population durchsetzen, daß durch eine Crossing-over Operation nur die noch nicht stabilen Gene neu optimiert werden. Durch eine genügend große

Population kommt es zu einer Art kollektiven Lernens, da in dem Genpool einer Population die Erfahrung der vorherigen Entwicklungen steckt. Gene, die in der Vergangenheit schon als schlecht erkannt wurden, sind aussortiert worden und können jetzt nicht erneut auftreten und den Entwicklungsprozeß hemmen. In der Gesamtheit der Gene steckt also die Information über den Entwicklungsprozeß der Population.
Es ist denkbar, daß bei einem solchen Algorithmus der Rechenaufwand so optimiert werden kann, daß die Rechenzeit linear mit der Meßwertanzahl steigt, da die Gene unabhängig voneinander lernen.

9.5 Strategie der isolierten Populationen

Netze können zuerst mit unterschiedlichen Klassen trainiert werden. Das heißt, es werden Netze in verschiedene Ursuppen gesetzt, wobei jede Ursuppe eine andere Klasse von Ereignissen darstellt. Haben sich so optimale Netze für jede Ereignisklasse herausgebildet, werden alle Netze in eine Ursuppe gesetzt, in der alle Ereignisklassen vorkommen. Durch Crossing-over und Mutation werden sich Netze entwickeln, die auf die gesamten Ereignisklassen optimal reagieren. Es ist zu erwarten, daß ein solcher Algorithmus schneller arbeitet, als wenn ein Netz von Anfang an mit allen zur Verfügung stehenden Daten optimal trainiert wird.

9.6 Züchten von Lösungen

Bei dieser Strategie wird von einer Population ausgegangen, die auf eine Ereignisklasse trainiert wurde. Schrittweise werden nun die Anforderungen an die Population erweitert oder verändert. Dadurch ändert sich der Selektionsdruck auf die Population. Es wird nun davon ausgegangen, daß sich durch Mutation der Gene oder dem Wiederauftreten von rudimentären Genen die Population an die veränderten Umweltbedingungen anpaßt. Rudimentäre Gene sind Gene, die bislang zwar vorhanden waren, aber keine Aktionen bewirkt haben.

9.7 Erweiterung des hybriden Netzansatzes

Bei dem hier verwendeten Kristallisationsalgorithmus kann es nicht zum Auftreten von abstrakten Klassifizierungsregeln kommen. Durch diesen Algorithmus werden möglichst viele Werte durch ein Neuron beschrieben. Diese Zuordnung ist bei geringen Meßwertdich-

ten nicht eindeutig. So kann es durch diesen Algorithmus zwar zu Hypothesen über die nicht ausreichend definierten Bereiche kommen. Diese Hypothesen könnten dann durch einen genetisch-evolutionären Prozeß optimiert werden. Eine abstrakte Regel unterscheidet sich von solchen Hypothesen dadurch, daß sie von einer größeren Ordnung ausgeht, die aber nicht durch reines Wiederholen von quasi abgespeicherten Reaktionen oder entsprechend nach Ähnlichkeitskriterien assoziierten Reaktionen gebildet werden kann. Das liegt daran, daß der hier verwendete Algorithmus alle Werte, die ihm zur Verfügung stehen, in das Netz trainiert. Der Algorithmus wird erst dann beendet, wenn alle Referenzwerte durch das Netz erkannt wurden. Somit würde eine einzige Ausnahme von einer abstrakten Regel verhindern, daß diese erkannt wird.

Die Struktur des hier verwendeten Netzes, bei dem die Entscheidung aus der Abwägung von drei Netzen gebildet wird, ist aber durchaus in der Lage, solche abstrakten Regeln zu implementieren. Erweitert man das Netz um eine zusätzliche Schicht, welche mit Ausnahmen trainiert werden kann, so lassen sich abstrakte Regeln mit Ausnahmen abbilden. Um ein solches Netz zu trainieren, scheiden die klassischen Algorithmen aus, da sie im allgemeinen auf einem mehr oder weniger systematischen Ausprobieren basieren. Da das Auftreten von abstrakten Regeln aber gerade durch das Auftreten eines Evolutionssprunges gekennzeichnet ist, werden Strategien, die lediglich auf der schrittweisen Modifikation eines Netzes nach einem Gradientenverfahren basieren, nur schwer zu Lösungen führen. Ein brauchbarer Ansatz für ein solches Netz kann auf einem genetischen Algorithmus basieren, bei dem nicht ein einziges Netz trainiert wird, sondern sich die Information in dem Genpool einer Population befindet. Ein solcher Algorithmus könnte folgendermaßen:
Zuerst wird eine Population von Netzen mit Hilfe des Kristallistionsalgorithmus trainiert. Hierbei werden die Netzschichten für die Ausnahmeregeln nicht trainiert, sie bleiben leer. In einem zweiten Schritt werden die Netze dann um eine Netzschicht für Ausnahmeregeln erweitert. Durch Mutation oder dem Austauschen von Neuronen aus den anderen Netzschichten wird diese Schicht dann mit Werten gefüllt. Durch Selektionsmechanismen werden dann schlechte Veränderungen aussortiert. Durch die Crossing-over Veränderungen, daß heißt das Austauschen von Neuronen oder Teilen von diesen aus anderen Schichten, werden die neuen Schichten nicht wahllos trainiert. Dadurch wird das Lernen beschleunigt oder gar erst ermöglicht, da hier die Information über Klassifizierungen verwendet werden, die die Population schon früher einmal erworben hat.

Das boolsche Netz, das sich an diese Schichten anschließt, erfüllt die Bedingungen für einen genetischen Algorithmus nicht, da hier kleine Modifikationen gleich das ganze Er-

gebnis global verändern. Das Problem wird dadurch umgangen, indem das boolsche Netz vereinfacht wird. Jeder Eingang kann wahlweise invertiert werden. Dadurch lassen sich entsprechende Wenn-Dann-Beziehungen aufbauen; z.B. wenn $NETZ_A$ und $NETZ_B$ und $NETZ_C$ nicht, dann $Aktion_1$. Die Gesamtaktion ist dann die Veroderung der Einzelaktionen. Eine beliebige Verknüpfung von Und-Oder-Funktionen läßt sich immer in so einer reinen Und-Oder-Form darstellen. Die Anzahl der möglichen $UND - ODER$ Formen ist endlich. Da diese Zahl schnell sehr groß wird, beschränkt man sich auf eine Veroderung einer maximalen Anzahl von UND-Formen. Bei einer Veroderung von zwei Und-Formen und vier Eingangsschichten sind dies $16 * 15 = 240$. Um eine genetische Variation der Schicht zu vermeiden, werden alle Kombinationen dieser boolschen Schicht gleichzeitig betrachtet und die beste ausgewählt. Auf diese Weise kann der evolutionäre Prozeß umgangen werden. Ein Evolutionssprung ist dann dadurch gekennzeichnet, daß sich die Konfiguration des boolschen Netzes ändert.

Bei den hier untersuchten Triggerdaten hat sich nicht die Notwendigkeit herausgestellt, den Kristallisationsalgorithmus zu erweitern. Die verwendeten Daten konnten auch ohne einen solchen Algorithmus immer in zufriedenstellender Form in das Netz trainiert werden. Durch den Verzicht auf abstrakte Regeln wird sich die Anzahl der Neuronen erhöhen. Bei dem hier verwendeten Netztyp ist die Anzahl der Neuronen aber nicht problematisch. Wie sich im nächsten Kapitel der Arbeit zeigen wird, kann ohne Schwierigkeiten ein Netz mit 4096 Neuronen konstruiert werden. So ist es möglich, mit einem einfachen und leicht nachvollziehbaren Lernalgorithmus zu arbeiten. Die dadurch entstehenden Einschränkungen können durch eine größere Anzahl an Neuronen kompensiert werden. Die schlechteren assoziativen Eigenschaften können durch eine größere Anzahl an Trainingswerten ausgeglichen werden. Bei dem hier verwendeten Trigger können solche Referenzwerte in beliebiger Anzahl erzeugt werden. Durch das Überlagern von inversem und aktivem Netz können nicht ausreichend trainierte Bereiche erkannt werden. Dies ist in diesem Fall sicher besser, als diese Bereiche durch hypothetische Annahmen zu beschreiben, da in solchen Grenzbereichen gerade die interessanten Ereignisse vermutet werden. Da diese Bereiche ja nicht in den Referenzwerten aufgetreten sind, stellen sie somit unerwartete Ereignisse dar.

Die abstrakten Regeln sind für die Datenanalyse interessant, da dadurch Schlußfolgerungen auf die Bedeutung der Triggerkanäle gezogen werden können. Eine solche Interpretation wird nach physikalischen Gesichtspunkten erfolgen. Das hier verwendete Netz soll in erster Linie auch nicht zur Datenanalyse eingesetzt werden, sondern ein technisches Werkzeug zum schnellen Aussortieren von unbrauchbaren Daten sein. Somit möchte ich einem

einfachen Algorithmus den Vorzug geben. Hat sich ein solcher Algorithmus bewährt, kann dieser dementsprechend erweitert werden.
Genetische Algorithmen sind zu Analyse von noch komplexeren Aufgaben geeignet. So werden sie z.B. erfolgreich zur Optimierung von Turbinen eingesetzt, wo das Ergebnis auf der Abwägung von etwa 50 Forderungen und auf der Modifikation von mindestens 100 Variablen basiert.[7]

10 Simulation mit L2 Daten

Da das Readout der Detektoren auf L2 Ebene nur etwa 2 μs dauern darf, ist die Auswahl der Daten für den L2 Trigger beschränkt. Auf dieser Ebene sind die Flugbahnen der Teilchen noch nicht rekonstruiert worden. Um die Flugbahnen der Teilchen auswerten zu können, wird versucht, nur aus den inneren und äußeren Driftkammeren auf den Kollisionspunkt zu schließen. Liegt dieser in dem erwarteten Bereich, ist dies eine wichtige Information für den Trigger. Dazu wird für jeden Punkt auf dieser Z-Achse eine mögliche Flugbahn errechnet, die zu den gemessenen Driftkammeren paßt. So entsteht entlang der Achse ein Histogramm. In diesem Histogramm werden dann Maxima erkannt und gewichtet. Aus dieser Information kann dann auf den Kollisionspunkt geschlossen werden. Diese Informationen stehen dem L2 Trigger zur Verfügung. Weitere wichtige Daten sind die Energieverteilung in den Kalorimetern. Da hier natürlich nicht alle Kalorimeter benutzt werden, wird die Energiesumme in den verschiedenen Richtungen gebildet. Die Anzahl der gefundenen Muonen, das Auftreten von Teilchen, die in den Detektor geflogen sind (TOF = Time Of Flight Messungen) und die Anzahl der positiv und negativ gekrümmten Bahnen liegt auf L2 Level vor. Diese Informationen werden zu dem sogenannten L2 Cocktail zusammengefaßt. Dieser umfaßt etwa 50 Komponenten (s. Anhang).

10.1 Auswahl der optimalen Komponenten

Offensichtlich sind nicht alle Komponenten gleich wichtig oder redundant. Im vorherigen Kapitel wurde bereits ein Verfahren zur Auswahl der besten Komponentenauswahl mit Hilfe des Netzes vorgeschlagen. Bei der hier vorliegenden Datenmenge hat sich dieses Ver-

[7] vg / Genetische Algorithmen / John H. Holland / Spektrum der Wissenschaft 9-92

fahren aber als zu langsam herausgestellt. Um die Komponenten durch ein systematisches Verfahren zu verbessern, wurden die Kanäle jetzt nicht mehr in ein Netz trainiert, sondern es wurde die Aussagekraft der Kanäle direkt untersucht. Dazu wurde gezählt, wieviel Ereignisse unterschieden werden können, wenn nur ein Kanal aus den 50 zur Verfügung steht. Wurde der Kanal mit der größten Signifikanz gefunden, so wurde dieses Verfahren dann mit allen Zweierkombinationen mit diesem Kanal wiederholt. Das Verfahren wurde so lange angewandt, bis eine Auswahl von Kanälen gefunden wurde, mit der alle Referenzdaten eindeutig unterschieden werden können. Das bedeutet, daß jetzt eindeutig durch die Eingangsgrößen definiert ist, um welche Ereignisklasse es sich handelt. Je nach Ereignisklasse waren 4 bis 9 Kanäle ausreichend um diese eindeutig zu erkennen. Als Referenz hierzu wurde eine Auswahl von Kanälen untersucht, die nach physikalischen Überlegungen gewonnen wurde. Vor dieser Analyse wurden aus jeder Ereignisklasse 500 Werte entfernt und getrennt gespeichert. Mit Hilfe dieser Daten wurde dann der Fehler und somit die Qualität des Netzes bestimmt. Diese Werte dürfen natürlich nicht zum Auswahlverfahren der Kanäle und zum Trainieren des Netzes herangezogen werden. Für die einzelnen Ereignisklassen ergab sich folgened Auswahl der optimalen Komponenten :

- CCBAR : *Veto_Tof, E_BACK, Etot, E_IF, Peak_pos, ZVTX_qua, DC_trks*
- CCNORAD : *Etmiss, Etot, ZVTX_sum, E_FORW*
- JET-JET : *Muon_TOT, Eele_tag, N_bigray, DC_loneg, E_BARR, E_FORW, E_IF, Etot, FW_rays*
- MTB1 : *EmaxBemc, E_CB, Etot, Veto_Tof, E_PLUG, E_BEMC, Posi_Tof, Peak_pos*

10.2 Der L1 Trigger

Vor dem L2 Trigger ist der L1 Trigger geschaltet. Dieser Trigger überwacht gewisse Schwellwerte und Randbedingungen. Der L1 Trigger kann mit verschiedenen Schwellwerten geladen werden. Je höher diese Schwellwerte gesetzt werden, desto weniger Ereignisse werden durch diesen Trigger durchgelassen. Man spricht davon, daß mit Hilfe dieser Schwellwerte die Frequenz der Signale hinter diesem Trigger abgestimmt wird. Werden die Schwellwerte niedrig angesetzt, werden praktisch alle Physikereignisse durch den Trigger gelassen, der Anteil der Backgroundereignisse ist dann aber sehr hoch und die Daten würden die nachfolgenden Triggerstufen überfordern. Werden die Schwellwerte sehr hoch

angesetzt, so werden auch sehr viele interessante Ereignisse aussortiert.

Da die Daten lokal in den Detektoren gehalten werden, werden solange keine neuen Meßwerte aufgenommen, bis der L4 Trigger alle Daten aus den Detektoren ausgelesen hat oder das Ereignis verworfen wurde. Deshalb ist es nicht sinnvoll, den L1 Trigger mit einer hohen Frequenz abzustimmen, da jetzt nur sehr kurze Zeit gemessen wird und der Trigger die meiste Zeit die Daten auswertet. Offensichtlich gibt es ein optimales Verhältnis zwischen der Eingangsfrequenz der Signale in den Trigger und seiner Entscheidungszeit. Da alle Eingangswerte des L1 Triggers auch dem L2 Trigger zur Verfügung stehen, ist dieser eigentlich überflüssig, der L1 Trigger ist aber in der Lage, seine Entscheidung in Echtzeit (96 ns) zu treffen, der L2 Trigger bräuchte 20 μs. Die Datenrate hinter den L1 Trigger bei der Standardabstimmung entspricht 886 Hz. Das Verhältnis zwischen Physikereignissen und Hintergrundereignissen ist jetzt etwa 1:200.
Aus den 10 Millionen möglichen Kollisionen pro Sekunde im Detektor bleiben nach den 4 Triggerstufen etwa 3 übrig. Aus diesem Grund ist es durchaus sinnvoll, den Detektor für die Zeit des Triggerns blind zu schalten, da es aufgrund dieses ungleichen Verhältnisses nicht wahrscheinlich ist, daß in der Blindzeit ein zweites interessantes Ereignis auftritt. Das optimale Abstimmen der Triggerstufen hängt aber von sehr vielen Größen ab, die zur Zeit noch nicht bekannt sind. Die wichtigste ist die Luminisität, d.h. der Wirkungsgrad der Kollisionen.

	Tuned to L1 rate			
Event Class	10.7KHz	4.4KHz	886Hz	94Hz
BEAMGAS2	93%	88%	50%	24%
BEAMWAL2	99%	97%	74%	40%
JETJET	94%	92%	86%	73%
MTB1	100%	100%	98%	89%
CCNORAD	99%	97%	89%	79%
CCBAR	71%	63%	50%	30%

Tabelle 1: Durchlässigkeit des L1 Triggers bei verschiedenem Abgleich. Die ersten beiden Ereignisse sind unerwünscht. Auf die letzten vier Ereignisse soll getriggert werden. Diese treten mit etwa 3 Hz auf. Die Tabelle wurde mit dem Programm L1-Trigg (s. Anhang) berechnet.

Bei dem hier simulierte L1 Trigger werden auf die Energieverteilung in den verschieden Raumrichtungen Schwellwerte angewendet. Diese Schwellwerte werden durch das Auftreten von signifikanten Teilchen oder erkannten Spuren modifiziert (s. Anhang).
Für den L1 Trigger muß also eine minimale Menge an Energie in dem Detektor deponiert worden sein. Dadurch wird ein großer Teil der Ereignisse aussortiert, in denen sich die Teilchen in den Paketen nicht getroffen haben. Durch die zusätzliche Verwendung von Spurdaten werden z.B. kosmische Muonen aussortiert. Die gewünschten Ereignisse sollen durch den Trigger möglichst nicht beeinträchtigt werden. In der Tabelle 1 sind 7 Ereignisklassen aufgeführt, die nach dem L1 Trigger im wesentlichen übrig bleiben. Die gewünschten Ereignisse ($CCBAR, JETJET, MTB1, CCNORD$) treten mit einer Rate von etwa 3 Hz auf. Die Störereignisse machen den übrigen Anteil der Frequenz aus. Das Signal-Stör-Verhältnis beträgt vor dem L1 Trigger somit $3 : 10^6$, nach dem L1 Trigger bei einem Abgleich von z.B. 886 Hz dann $2,4 : 884$. Die Aufgabe des L2 Triggers ist es somit, die Störereignisse noch weiter zu reduzieren, ohne die gewünschten Ereignisse noch weiter zu dämpfen.

10.3 Trainieren des L2 Triggers

Um den L2 Trigger zu trainieren, können verschiedene Daten verwendet werden. Zum einen wird der Trigger mit einem systematisch gebildeten Subset der Daten trainiert, zum anderen mit einer nach physikalischen Überlegungen gebildeten Auswahl. Da es bei den hier verwendeten Daten vier Klassen von guten Daten und zwei Klassen von schlechten Daten gibt, kann das Netz einerseits auf das gewünschten Ereignis trainiert werden, die anderen Ereignisklassen bleiben dann unberücksichtigt. Andererseits können die anderen Ereignisklassen als unerwünscht definiert werden. Untersucht wurde jetzt ein Trigger, der auf das $CCBAR$-Ereignis trainiert wurde. Der L1 Trigger wurde auf 4,4 KHz abgeglichen. Untersucht wurde einmal ein Netz, welches nur auf die Daten trainiert wurde. Dem gegenübergestellt wurde ein Netz, welches aus drei Netzen seine Antwort abwägt. Für ein solches Netz ist dann neben der Triggerrate die Zurückweisungsrate entscheidend. Die Fehlerrate dieses Netzes in den nachfolgenden Tabellen ist die Raten nach den dem die zurückgewiesen Ereignisse aussortiert wurden. Diese Zahl ist also mit der Fehlerrate des einfachen Ansatzes vergleichbar. Die Verbesserung dieser Raten nach dem Aussortieren ist ein Maß für die Güte der Aussortierung. Um die effektive Durchlässigkeit des Triggers

zu bestimmen, müssen Zurückweisungsrate und Fehler zusammengefaßt werden. Für die Berechnung der Frequenzen, mit denen die Ereignisse nach dem Trigger auftreten, wurde angenommen, daß die Ereignisse BEAMGAS und BEAMWAL mit je 2 KHz nach dem L1 Trigger auftreten, die übrigen Ereignisse mit je 100 Hz. Zur Kontrolle wurden auf den so gebildeten Trigger auch völlig zufällige Daten gegeben. Hierdurch soll kontrolliert werden, wie das Netz auf Werte reagiert, die nicht gelernt wurden.

Ereignis	Triggerrate	Hz	zurückgewiesen	korrigierte Triggerrate	Hz
BEAMGAS2	8.06%	161Hz	4.63%	4.03%	76Hz
BEAMWAL2	1.41%	28Hz	1.41%	0.20%	4Hz
MTB1	97.78%	98Hz	50.60%	97.78%	48Hz
CCBAR	91.94%	92Hz	28.43%	93.75%	68Hz
JETJET	91.73%	92Hz	30.65%	92.74%	64Hz
CCNORAD	91.13%	91Hz	29.23%	91.33%	63Hz
RANDOM	93.40%	—	95.00%	0.40%	—
Background : Event	470:92=5.0		255:68=3.7		

Tabelle 2: Netz mit CCBAR, $\overline{BEAMGAS}$, $\overline{BEAMWAL}$ trainiert. Die Triggerkomponenten wurden durch ein systematisches Verfahren ausgewählt. (Veto_Tof, E_Back, Etot, E_IF, Peak_Pos, ZVTX_qua, DC_trks)

Ereignis	Triggerrate	Hz	zurückgewiesen	korrigierte Triggerrate	Hz
BEAMGAS2	11.90%	238Hz	10.28%	1.81%	32Hz
BEAMWAL2	3.02%	60Hz	2.62%	0.40%	7Hz
MTB1	17.14%	17Hz	16.33%	17.54%	14Hz
CCBAR	76.41%	76Hz	52.22%	80.85%	38Hz
JETJET	42.74%	43Hz	35.28%	44.96%	33Hz
CCNORAD	81.85%	81Hz	15.52%	18.15%	15Hz
RANDOM	13.20%	—	15.20%	0.00%	—
Background : Event	439:76=5.7		101:38=2.6		

Tabelle 3: Netz mit CCBAR, $\overline{BEAMGAS}$, $\overline{BEAMWAL}$, $\overline{MTB1}$, $\overline{JETJET}$, $\overline{CCNORAD}$ trainiert. Die Triggerkomponenten wurden durch ein systematisches Verfahren ausgewählt. (Veto_Tof, E_Back, Etot, E_IF, Peak_Pos, ZVTX_qua, DC_trks)

Ereignis	Triggerrate	Hz	zurückgewiesen	korrigierte Triggerrate	Hz
BEAMGAS2	10.69%	214Hz	6.05%	3.83%	72Hz
BEAMWAL2	1.81%	36Hz	1.81%	0.20%	4Hz
MTB1	86.69%	87Hz	43.15%	90.73%	51Hz
CCBAR	90.32%	90Hz	26.61%	91.73%	67Hz
JETJET	94.54%	94Hz	29.64%	96.17%	68Hz
CCNORAD	83.47%	83Hz	29.84%	92.74%	65Hz
RANDOM	94.60%	—	99.60%	0.00%	—
Background : Event	514:90=5.7		260:67=3.8		

Tabelle 4: Netz mit CCBAR, $\overline{BEAMGAS}$, $\overline{BEAMWAL}$ trainiert. Die Triggerkomponenten wurden nach physikalischen Gesichtspunkten ausgewählt. (ZVTX_max, ZVTX_sum, ZVTXsig2, Ncoinc_i ,N_bigray, FW_rays, DC_trks, Eele_tag, Nbpc1hit, EtotBemc, EcluBemc, Veto_Tof, Posi_Tof, Et, Etweight, E_IF, Elec_Lar, Muon_TOT)

Ereignis	Triggerrate	Hz	zurückgewiesen	korrigierte Triggerrate	Hz
BEAMGAS2	9.27%	185Hz	5.44%	3.02%	57Hz
BEAMWAL2	2.62%	52Hz	2.42%	0.20%	3Hz
MTB1	20.77%	21Hz	20.77%	21.98%	18Hz
CCBAR	75.20%	75Hz	49.81%	80.85%	41Hz
JETJET	51.61%	52Hz	43.35%	53.43%	30Hz
CCNORAD	13.10%	13Hz	11.90%	13.91%	12Hz
RANDOM	85.80%	—	90.80%	0.00%	—
Background : Event	323:75=4.3		120:41=2.9		

Tabelle 5: Netz mit CCBAR, $\overline{BEAMGAS}$, $\overline{BEAMWAL}$, $\overline{MTB1}$, $\overline{JETJET}$, $\overline{CCNORAD}$ trainiert. Die Triggerkomponenten wurden nach physikalischen Gesichtspunkten ausgewählt. (ZVTX_max, ZVTX_sum, ZVTXsig2, Ncoinc_i ,N_bigray, FW_rays, DC_trks, Eele_tag, Nbpc1hit, EtotBemc, EcluBemc, Veto_Tof, Posi_Tof, Et, Etweight, E_IF, Elec_Lar, Muon_TOT)

10.4 Interpretation der Ergebnisse

Unabhängig von dem gewählten Verfahren kann festgestellt werden, daß der hier verwendete Trigger funktioniert. Von einer Datenrate von 4,4 KHz am Eingang des Triggers bleiben nach diesem je nach verwendetem Verfahren eine Rate von 600-160 Hz übrig. Das entspricht einer Reduktionsrate von 7-26. Das Signal-Stör-Verhältnis verbessert sich von 43:1 auf 5,7-2,6 : 1. Die geforderten Randbedingungen an einen L2 Trigger werden also durch ein solches neuronales Netz erfüllt. Durch die Korrektur des Netzes mit Hilfe eines inversen Netzes kann das Signal-Stör-Verhältnis verbessert werden. Ein Netz ohne Korrektur ist nicht in der Lage, unbekannte Werte zu erkennen. So triggert ein solches Netz auf zufällige Werte fast immer. Durch die Korrektur mit einem Inversen und einem Fuzzy Netz werden nicht trainierte Werte zuverlässig erkannt und aussortiert. Durch die Korrektur werden nicht alle Eigenschaften des Netzes verbessert. So sinkt die Durchlässigkeit für gute Daten stark. Bei den hier verwendeten Daten werden durch ein korrigiertes Netz etwa nur noch die Hälfte der gewünschten Daten durchgelassen.
Durch ein systematisches Auswahlverfahren können die Eingangskanäle optimiert werden. Offensichtlich wird durch eine optimierte Auswahl der Eingangskanäle ein ähnliches Ergebnis erreicht, wie durch eine Auswahl nach physikalischen Gesichtspunkten. Durch eine Optimierung konnte in diesem Beispiel die Anzahl der Eingangskanäle halbiert werden. Dadurch halbiert sich auch die Größe des Netzes.

	Correctly classified events		
	$beam-gas$	$beam-wall$	$c\bar{c}$
Linear Discriminant	90.1%	85.2%	61.8%
Analog Net	89.3%	94.5%	77.8%
Orthogonal Net	94.4%	99.4%	68.3%

Tabelle 6: Vergleich eines orthogonalen Netzes mit einem Standardverfahren und einem kontinuierlichen neuronalen Netz. Die Zahlen sind unabhänig von dieser Arbeit gewonnen worden. [8]

Der Vergleich zeigt, daß die Vereinfachungen, die bei dem orthogonalen Netz gemacht wurden, die Funktion nicht einschränken. Die Datensätze für die Simulation waren nicht identisch, aber vergleichbar. Offensichtlich kann durch die Verwendung der großen Anzahl

[8] entnommen aus / Proposal for a Second Level Neural Network Trigger / J. Fent, A. Gruber, C. Kiesling, H. Oberlack, P. Ribarics / DESY 17. März 92

der Neuronen bei dem orthogonalen Netz und der Optimierung der Eingangsgrößen die Fähigkeit verbessert werden, Hintergrundereignisse zu erkennen.

11 Das H1 Triggerkonzeptes

Bei dem H1 Experiment werden Elektronen und Protonen aufeinandergeschossen. Diese Teilchen werden über mehrere Beschleuniger in den Doppelring Hera eingespeist, wo sie dann gegenläufig kreisen. An zwei Stellen kreuzen sich die Teilchenbahnen. An diesen Stellen stehen die Experimente Zeus und H1. Ein Experiment besteht im wesentlichen aus einer großen Anzahl von Detektoren, die um einen solchen Kollisionspunkt angeordnet sind. Um diesen Kollisionspunkt ist ein Zylinder aus Driftkammern angeordnet. Eine Driftkammer ist im Prinzip ein Draht, der in einem elektrischen Feld von einem speziellen Gas umströmt wird. Fliegt ein Teilchen durch dieses Gas, so werden dort die Atome ionisiert, und driften im elektrischen Feld zu dem Draht, an dem dann ein Strom abgenommen werden kann. Die meisten Driftkammern sind längs der Strahlrichtung angeordnet. Durch eine solche Kammer können dann die X, Y- Koordinaten der Teilchenbahn bestimmt werden. Die Z- Koordinate, daß heißt die Stelle entlang eines solchen Drahtes, kann durch mehrere Verfahren bestimmt werden. Zu einen sind in regelmäßigem Abstand Kammern in X, Y- Richtung gespannt. Ebenso sind die Kammern leicht gegeneinander versetzt, so daß eine stereoskopische Auswertung erfolgen kann (beim Zeus-Detektor). Ein anderes Verfahren mißt den Strom an beiden Enden der Kammer (beim H1-Detektor). Die genaue Position, an der ein Teilchen durch diese Kammer geflogen ist, kann zum einen durch die Laufzeitunterschiede dieser Signale an beiden Enden bestimmt werden, ein anderes Verfahren nutzt den elektrischen Widerstand des gespannten Drahtes aus. So ist das Verhältnis der Ströme dann ein Maß für die Position des Teilchendurchganges. Dieser Zylinder ist an beiden Seiten noch einmal durch solche Driftkammern abgeschlossen. Diese ganzen Driftkammern sind von einem Paket von Kalorimetern umschlossen. Ein solches Kalorimeter soll alle Teilchen absorbieren und diese abbremsen, um so die kinetische Energie zu messen. Diese Kalorimeter ist mit flüssigem Argon (H1-Experiment) gefüllt und in zwei Stufen hintereinander aufgebaut. Aufgrund dieser Konstruktion werden im vorderen Teil der Kalorimeter alle Elektronen abgebremst, Teilchen, die im hinteren Kalorimeter noch nachgewiesen werden können, sind dann schwerere Teilchen. Die ganze Apparatur ist von einer supraleitenden Spule umschlossen. Durch diese wird ein Magnet-

feld in dieser Apparatur erzeugt. Die Flugbahnen geladener Teilchen werden durch das Magnetfeld zu Kreisbahnen geformt. Aus dem Radius in Zusammenhang mit den Kalorimeterdaten können so Masse und Ladung bestimmt werden. Die Krümmungsrichtung ist gleichbedeutend mit dem Ladungsvorzeichen. Um den magnetischen Fluß zu schließen, ist um die supraleitende Spule ein Eisenjoch konstruiert (vgl. *Abb.*11). Hierin sind die Muonenkammern untergebracht. Diese sind spezelle Driftkammern. Die Muonen durchdringen die innere Apparatur praktisch unbeschadet und werden dann in diesen Kammern nachgewiesen. Ebenso werden durch diese Kammern kosmische Muonen, die von außen in den Detektor fliegen, gemessen. Durch verschiedene Maßnahmen kann die Flugrichtung der Teilchen bestimmt werden. Solche Teilchen können einmal aus dem Kosmos in den Detektor fliegen oder durch Sekundärreaktionen im Strahlrohr oder Kollisionen im Strahlrohr mit Restgasatomen entstehen. Solche Ereignisse machen eine Messung natürlich unbrauchbar.

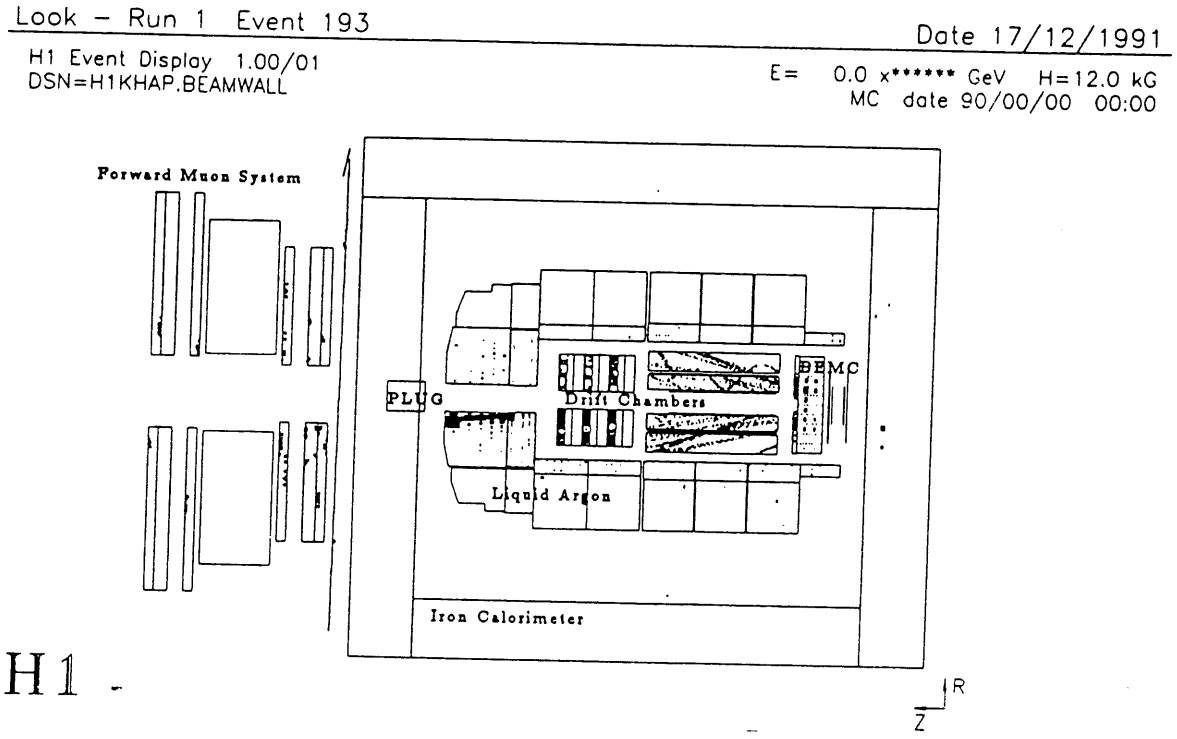


Abb. 11: Skizze der inneren Kammeren des H1 Detektors, wie es beim Event Display verwendet wird

Einige Spezialdetektoren außerhalb des eigentlichen Detektors messen Teilchen, die aus

der Bahn geworfen wurden. Dies geschieht, wenn die Elektronen oder Protonen abgebremst werden. Durch das Ansprechen dieser Detektoren können bestimmte Ereignisse erkannt werden. Werden alle Daten der Detektoren zusammengefaßt, ergibt sich eine Datenmenge von 2 MByte alle 96 ns. Da es unmöglich ist, solche Datenmengen direkt zu bearbeiten, werden diese Daten zuerst lokal gespeichert. Dazu verfügt jeder Detektor über einen Speicher, der in Form eines Schieberegisters als Pipeline ausgebildet ist. Die Pipeline ist 32 Stufen tief. Die erste Triggerstufe entscheidet innerhalb von 2 μs, ob die Daten übernommen werden sollen, oder nicht. Triggert diese Stufe, so werden alle Pipelines angehalten. Die Pipelines, die für den L2 Trigger interessant sind, werden nun ausgelesen. Dieses Auslesen dauert weitere 2 μs. Der L2 Trigger trifft nach 18 μs seine Entscheidung über weiteres Bearbeiten der Daten oder Verwerfen dieser Daten. Eine positive L2 Entscheidung startet den Datentransfer aller Detektordaten von der Detektorelektronik zur L4 Prozessorfarm. Während dieses Auslesens arbeitet der L3 Trigger und entscheidet während des Auslesens der Detektoren über ein vorzeitiges Abbrechen des Readouts. Sind alle Daten aus den Detektoren ausgelesen, wird im L4 Trigger aus den Daten das Ereignis rekonstruiert. Diese Rekonstruktion geschieht völlig asynchron zum Experiment (vgl. *Abb*12). Von den vorher 10 Millionen Ereignissen pro Sekunde bleiben nach der L4 Stufe etwa 1 bis 5 Ereignisse übrig.

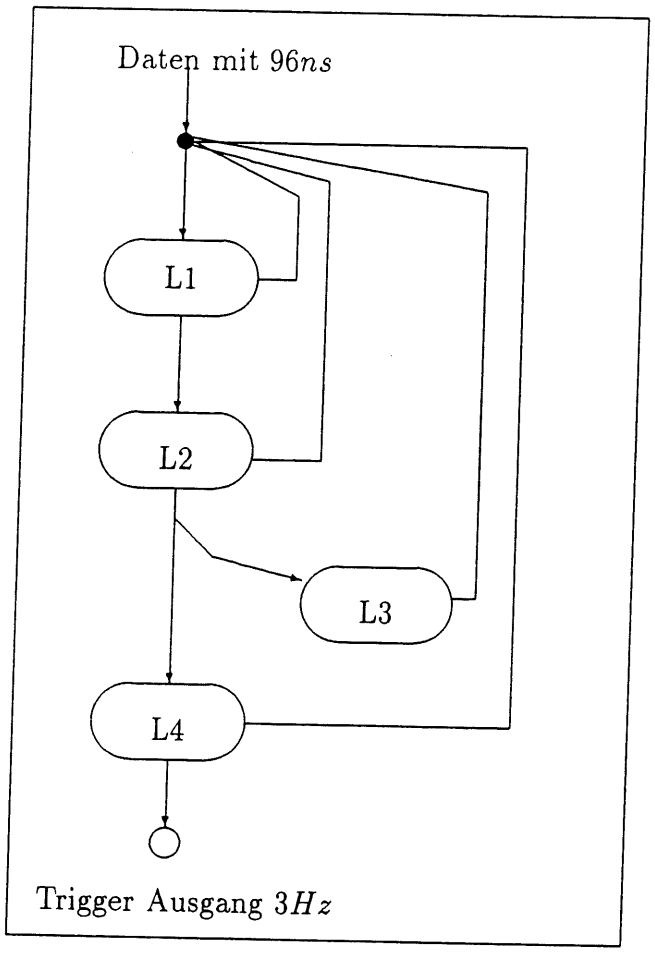


Abb. 12: Der L1 Trigger sieht die Daten in Echtzeit (96 ns). Nach etwa 2 μs entscheidet dieser über das Anhalten der Pipline. Der L2 Trigger liest dann ein Subset der Daten aus den Detektoren und trifft dann innerhalb von 18 μs eine genauere Entscheidung über eine Weiterverarbeitung der Daten oder ein Verwerfen der Messung. Eine positive L2 Entscheidung startet den Datentransfer aller Detektordaten von der Detektorelektronik zur L4 Prozessorfarm. Dieser Datentransfare kann von einer L3 Entscheidung unterbrochen werden.Der L4 Trigger liest alle Daten aus den Detektoren aus und rekonstruiert daraus vollständig asynchron das Ereignis.

Das neuronale Netz soll als Bestandteil des L2 Triggers eingesetzt werden. Die technischen Rahmenbedingungen sind in etwa : Aus 14 ausgesuchten Kanälen mit je 8 Bit Auflösung muß in 18 μs eine Entscheidung getroffen werden. Da die Kriterien für diese Entscheidungen noch nicht feststehen und sich erst im Laufe der Experimente herausstellen, soll der Trigger möglichst universell und flexibel sein.

Da durch den L1 Trigger alle trivialen Fälle aussortiert worden sind, muß diese Trig-

gerstufe schon auf kombinierten Bedingungen selektieren. Dies sind Kombinationen aus Energiebetrachtungen und Flugbahnen.

Die Aufgabe des L2 Triggers kann als ein Filter verstanden werden. Die Hauptaufgabe ist es, die Datenrate zu reduzieren. Bei dieser Reduktion sollen alle unerwünschten Ereignisse möglichst gut aussortiert werden und die gewollten möglichst alle durch diese Triggerstufe fallen. Dadurch ergeben sich mehrere Parameter, nach denen ein solcher Trigger beurteilt werden kann. Dies sind einmal die Prozentanteile, mit denen die verschiedenen Ereignisklassen den Trigger durchdringen und die Frequenz der Ereignisse nach dem Trigger, daß heißt die nach der Häufigkeit der Ereignisklassen gewichteten Prozentzahlen.

12 Die Hardwarefunktionen des Triggers

Die Hardware gliedert sich in mehrere Baugruppen:

- Interface zu den Triggerdaten
- Interface zu einem Rechner, um das Netz zu programmieren
- Das Netz
- Ausgabeinterface

12.1 Das Interface zum Experiment

Die Triggerdaten werden in einer Time and Space Map angeliefert. Dies ist ein Bus, in dem die Daten sowohl räumlich parallel als auch zeitlich seriell anliegen. Dies sind 7 Kanäle mit je 8 Bit Auflösung und einen Steuerkanal, über den 2 μs lang Daten im 100 ns Raster geschickt werden. Über diesen Bus werden zuerst die wichtigsten Daten geschickt, später immer unwichtigere. Dadurch soll es möglich sein, die ersten Daten schon während der Übertragung zu verarbeiten.
Anschaulich betrachtet ist dieses Datenformat eine Tabelle mit $7 * 20$ Fächern (vgl. *Abb.*13). Die Zuordnung der Meßwerte zu diesen Fächern ist programmierbar und wird sich im Laufe der Experimente ändern. Somit sollte diese Zuordnung zu den Fächern in einem solchen Trigger einfach zu ändern sein.

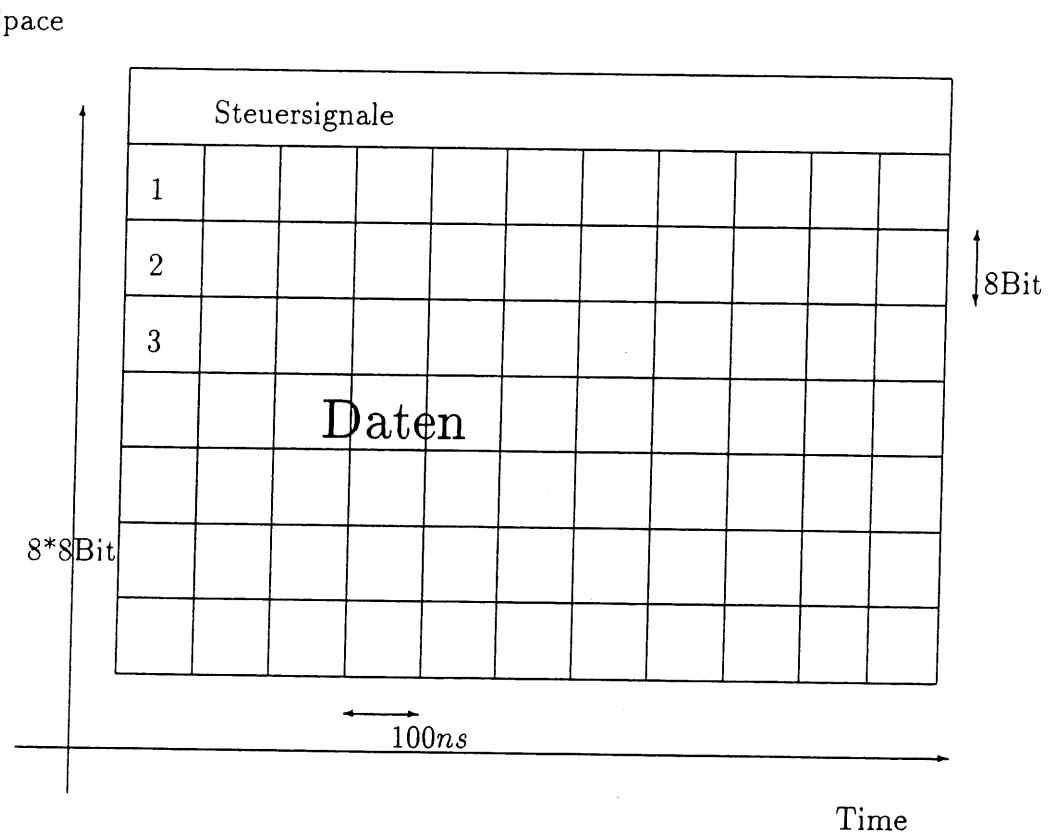


Abb. 13 : Zeitliche und räumliche Verteilung der Triggerwerte in der Time and Space Map

Das Experiment liefert Daten in 100 ns Abständen. Diese Daten werden zuerst im L1 Trigger bearbeitet. Dieser Trigger ist aus verschiedenen Gründen nicht in der Lage, nach 100 ns eine Entscheidung zu treffen (Unterschiedliche Laufzeiten in den Kabeln, einige Detektoren liefern erst nach längerer Zeit Daten). Deshalb werden alle Daten des Experimentes in eine 16stufige Pipe (ein großes Schieberegister) geschoben. Der L1 Trigger hat eine Durchlaufzeit von etwa 800 ns. Wird ein Ereignis erkannt (Signal L1 Keep), so wird diese Pipe angehalten. Die entsprechenden Daten werden dann aus dieser Pipe ausgelesen und in die Time and Space Map eingetragen. Diese Daten werden anschließend in der nächsten Triggerstufe L2 verarbeitet. Der neuronale Trigger soll in dieser L2 Stufe eingesetzt werden. Der L2 Trigger hat eine Zeit von etwa 18 μs, um diese Daten weiter zu untersuchen. Tritt in dieser Zeit ein neues Ereignis im L1 Trigger auf, wird dieses neue Ereignis ignoriert. Der L2 Trigger entscheidet nun, ob die Daten verworfen werden sollen, dann arbeitet der L1 Trigger weiter, oder ob die Daten an den L3 Trigger weiter gegeben werden sollen. Für den L2 Trigger stehen die ersten 20 Fächer in der Time and Space Map zur Verfügung. An diese Fächer schließen sich dann die Daten für den L3 Trigger an.

Elektrisch werden die Datenbits auf 100Ω twisted Lines abgebildet. Diese Daten müssen

also auf dem Trigger zuerst durch entsprechende Receiver mit Trennverstärkern aufbereitet werden. Dann muß die Time and Space Map entsprechend der gewünschten Fächer aufgespalten werden. Die entsprechenden Daten sollen auf dem Trigger in Registern gespeichert werden. Ein Register muß also einmal mit dem richtigen Kanal an den Bus verbunden werden und im richtigen Moment die Daten übernehmen. Das Takten der Register kann einfach über Zählen der entsprechenden Takte des Busses erfolgen. Die Zuordnung zu den Kanälen ist aufwendiger. Es gibt 7 Kanäle, die auf etwa 18 Register verteilt werden sollen. Somit muß jedes der 18 Register auf jeden der 7 Kanäle geschaltet werden können. Bei einer Lösung mit Multiplexer sind das 126 Standard TTL Bausteine. Alternativ wäre eine feste Verdrahtung in einer aufsteckbaren Platine denkbar, in der dann auch die zeitliche Komponente codiert sein könnte. Diese Variante reduziert den Aufwand zwar erheblich, schränkt aber die Flexibilität stark ein. Gerade diese Flexibilität ist ein Vorteil des neuronalen Netzes, da es diesem völlig egal ist, mit welchen Daten es trainiert wird. Ein Umändern der Daten bei laufendem Experiment ist sehr wichtig und sollte ohne Eingriff in die Hardware möglich sei.

Sinnvoll erscheint es, eine eigene Huckepackplatine mit diesen Multiplexer, Receivern, Zähler und Registern zu entwickeln, die dann auf einen Trigger aufgesteckt wird. Eine solche Platine könnte dann auch von anderen Triggern genutzt werden. Es würde bei einer solchen Entwicklung also ein Standardmodul entstehen. Zur Zeit ist ein solches Modul noch nicht entwickelt worden.

Da dieses Netz von einem Rechner geladen werden muß, sollte der Trigger eine Schnittstelle zu einem Rechner besitzen. Das Konzept für eine Rechnerkopplung ist folgendes: Die gesamte Triggerelektronik befindet sich in zwei Containern direkt neben dem Detektor. Die Steuerung des Triggers erfolgt aus einem Kontrollraum in größerer Entfernung. Die Elektronik ist möglichst dicht bei dem Detektor, um die Länge der Signalleitungen zu minimieren. Der Kontrollstand befindet sich außerhalb, da die Experimentenhalle bei eingeschaltetem Experiment nicht betreten werden kann.

Der Trigger wird mit einem VME-Bus-Interface ausgerüstet. So können mehrere Triggerplatinen auf einen Bus gesteckt werden. Die Kommunikation mit dem Kontrollstand geschieht dann über eine VME-Bridge. Dies ist eine Einsteckkarte, die sich auf den VME-Bus als Prozessor verhält. Am anderen Ende im Kontrollstand kann dann ein entsprechendes Programm laufen. Auf die einzelnen Register im VME-Bus kann zugegriffen und diese entsprechend ausgelesen und gesetzt werden. Reicht die Rechenleistung über diese Bridge nicht aus, so kann ein zusätzlicher Prozessor auf den VME-Bus gesteckt wer-

den, der dann entspreched aus dem Kontrollstand programmiert wird. Über die Bridge kann dann jederzeit die Funktion überwacht werden und im laufenden Betrieb debugged werden. Für ein solches Verfahren sind entsprechende Module und Softwarepakete vorhanden.

12.2 Das Netz

Das Netz soll in einer hybriden Struktur aufgebaut werden. Es soll aus zwei Schichten bestehen. Die erste Schicht verarbeitet die Daten in quasi analoger Auflösung (8 Bit), die zweite Schicht ist ein boolsches Netz (1 Bit Auflösung). Die erste Schicht besteht im Prinzip aus Komparatoren. Diese Komparatoren können gut durch Speicherbausteine realisiert werden. Benutzt man z.B. einen Standard 128 KBit SRAM, so können durch Verwendung von 8 Adreßleitungen als Eingang 8 Komparatoren gleichzeitig und durch Umschalten der übrigen 9 Adressen 512 Komparatoren nacheinander abgebildet werden. Da für die Entscheidung des Netzes etwa 18 μs zur Verfügung stehen, ist eine Zugriffszeit von 35 ns für den Speicher erforderlich. Geht man von einer Zugriffszeit von 70 ns aus, können durch einen einzelnen Speicher 2048 Vergleicher realisiert werden. Geht man bei der Realisierung von 18 Eingangskanälen aus, so wird die erste Schicht durch 18 Speicherbausteine realisiert. Würde man diese Funktion auf einem normalen Netz ohne orthogonale Neuronen abbilden, so würde das einer Neuronenzahl von 2048 Neuronen(!) entsprechen. Ein orthogonales Neuron wird durch folgende Funktion beschrieben :

$$((X_1 > U_1) \wedge (X_1 < O_1)) \vee ((X_2 > U_2) \wedge (X_2 < O_2)) \vee ((X_3 > U_3) \wedge (X_3 < O_3)) \ldots$$

Wobei X_n die Eingangsgrößen, U_n die Untergrenzen und O_n die Obergrenzen für die Komparatoren darstellen.
Die Speicherbausteine liefern die Vergleicher, deren Ergebnis dann noch entsprechend durch eine Und-Funktion verknüpft werden muß. Nach dieser Und-Verknüpfung liegen dann bei Verwendung von einer Speicherbank 8 Ausgänge vor. Für die zweite Schicht bietet sich ein Xilinx Baustein an. Dieses ist ein programmierbares Gatearray mit etwa 900 Zellen. Jede Zelle besteht aus zwei Registern, 5 Eingängen und 2 Ausgängen. Eingänge, Register und Ausgänge können fast beliebig über boolsche Funktionen in einer Zelle verknüpft werden. Die Zellen werden untereinander durch ein Geflecht von Leitungen verdrahtet. Die Programmierung dieses Xilinxs geschieht durch Landen eines im Xilinxs befindlichen RAM und kann ständig geändert werden. Somit können sowohl die Gewichte

der Neuronen, als auch die Verknüpfungen in der zweiten Schicht völlig frei programmiert werden. Da auch die Eingangskomponenten frei aus dem Eingangsstrom ausgewählt werden können, handelt es sich um einen völlig universell einsetzbaren neuronalen Prozessor. Der Ausgang des Netzes soll drei Entscheidungen liefern:

- Trigger erkannt ($Trigger$)
- Entscheidung ist unsicher, da in diesem Bereich widersprüchliche Daten trainiert wurden ($Fuzzy$)
- Daten sind unbekannt, da zu diesen Daten keine Werte trainiert wurden ($Undef$)

Ein Netz wird auf positive Ereignisse (P) trainiert, ein anderes auf inverse Ereignisse (I) und ein drittes auf widersprüchliche (F). Die Ausgänge werden wie folgt gebildet :

- $Trigger = P \wedge \bar{I} \wedge \bar{F}$
- $Fuzzy = F$
- $Undef = P \wedge I \wedge \bar{F} \vee \bar{P} \wedge \bar{I} \wedge \bar{F}$

13 Konstruktionsvorschlag eines neuronalen L2 Triggers

Die Erstellung eines neuronalen Netzes teilt sich in zwei getrennte Schritte. Zuerst muß ein Lernalgorithmus entwickelt werden. Mit diesem werden dann auf einem Rechner die Gewichte für ein Netz berechnet. Die Lernphase eines Netzes ist von der späteren Einsatzphase getrennt. Eine Technik, die das Netz während des Einsatzes ständig selbständig anpaßt und verbessert, ist hier nicht implementiert worden. Das Netz wird also mit fertig berechneten Daten geladen und lernt dann nicht mehr weiter. Die Lernphase ist von der Einsatzphase getrennt. Eine Verbesserung der Parameter des Netzes ist nur durch Analyse des Netzes im Einsatz und entsprechendes Nachtrainieren mit dort erkannten Fehlern möglich. Bis jetzt wurde hauptsächlich das Lernverfahren diskutiert. Jetzt soll der Entwurf einer solchen Netzstruktur erfolgen, die den Anforderungen im L2 Trigger genügt.

Bei der Konstruktion wurde Wert darauf gelegt, daß die Schaltung einfach, verständlich und testbar ist. Deshalb wurde auf die Verwendung von programmierbaren Bausteinen (EPLDs, Gals,...) verzichtet, da diese hier nicht unbedingt erforderlich sind. Dadurch ist die Schaltung übersichtlicher, die Funktion kann in jedem Punkt der Schaltung bei Bedarf einfach gemessen werden. Sollte sich im Laufe des Einsatzes ein Fehler in der Schaltung zeigen, so kann diese auch mit einfachen Digitaltechnikkenntnissen erkannt und mit Hilfe von Drahtbrücken (etc.) vor Ort behoben werden. Alle Bausteine werden nicht am Rande der Spezifikation betrieben, um so die Schaltung zuverlässig zu machen. Solche Entwicklungsrichtlinien wurden deshalb gewählt, da es sich hier um einem Experimentalaufbau handelt. Durch diese Schaltung soll gezeigt werden, daß es gelungen ist, durch die Modifiktion des neuronalen Standardansatzes mit Hilfe von orthogonalen Neuronen ein solch großes Netz zu realisieren.
Wenn ein solcher Trigger einmal eingesetzt wird, werden sich mehrere Arbeitsgruppen mit der Hardware befassen und diese gegebenenfalls an ihre Bedürfnisse anpassen. Deshalb ist es nicht sinnvoll, eine solche Schaltung hinsichtlich des Aufwandes oder der Leistung zu optimieren. Die hier vorgeschlagene Schaltung ist in erster Linie als ein Vorschlag für eine Realisierung gedacht, um dem Aufwand für einen solchen Trigger besser beuteilen zu können.

13.1 Die Time and Space Map

Die Elektronik teilt sich in zwei Baugruppen. Einmal ist dies eine Karte, die Triggerdaten aus dem Time and Space Bus herausnimmt und in paralleler Form (16 * 7 Bit) zur Verfügung stellt. Diese Karte ist so programmierbar, daß jedes Fach der Map auf jeden Ausgang gelegt werden kann. Diese Karte ist im wesentlichen eine Matrix aus Registern. Der Eingangsvektor ist die Time and Space Mape (TSMap), die gewünschten Kanäle bilden den Ausgangsvektor. Die Matrixelemente sind die Register. Da die Daten sowohl durch ihre Position auf den Bus, als auch durch einen Zeitpunkt während der Übertragung lokalisiert sind, wird jedem Register der entsprechende Zeitpunkt zugeordnet, an dem es die Daten aus dem Bus übernehmen soll. Pro Zeile darf nur ein Register aktiv sein, da der Augangswert nicht eine Überlagerung aus mehreren Eingangswerten sein soll. Realisiert wurde diese Struktur für 7 Eingangskanäle, über die nacheinander etwa 160 Werte angeliefert werden und 16 Ausgangskanäle. Jeder Ausgangskanal ist mit den 7 Eingangskanälen über ein Tristateregister verbunden. Jeder Ausgangskanal ist durch eine 8 Bit Zahl codiert. Die unteren 3 Bit wählen eines der 7 Register aus, das auf die Ausgangskanäle geschaltet werden soll. Die oberen 5 Bit geben den Zeitpunkt an, an dem die Register einen Wert aus den Eingangskanälen übernehmen sollen. Dazu werden die Eingangstakte gezählt und durch einen Komparator auf Übereinstimmung mit dem gewünschten Zeitpunkt geprüft. Somit stehen dem Netz 16 Eingangskanäle zur Verfügung.

13.2 Das Netz

Die zweite Karte ist das eigentliche Netz. Die erste Schicht des Netzes besteht aus 16 Vergleichern, die je eine Ober- und eine Untergrenze (ein Fenster) detektieren. Diese Vergleicher sind durch Speicherbausteine realisiert. Verwendet wird hier ein Standard 128 KBit statischer Speicher mit 8 Daten- und 15 Adreßleitungen. Die unteren 8 Adressen werden als Dateneingänge verwendet. Über die oberen 9 Adressen werden die Fenster ausgewählt. Der Speicher wird so programmiert, daß auf der so gebildeten Adresse der gewünschte Wert steht. Im einem solchen Speicher können somit $8*512$ Fenstervergleiche durchgeführt werden. Ein solcher Fenstervergleich entspricht dem schon vorher beschriebenen Doppelneuron. In dem Netz sind 16 Speicher parallel geschaltet. So wird in einem 16-dimensionalen Raum jeweils ein rechteckiger Bereich erkannt. Die Datenausgänge der Speicher sind über Und-Gatter verbunden. Die so gebildeten Werte werden an die zweite

Schicht weitergegeben.[9] Hier liegen dann jeweils 8 Bit vor, wobei jedes Bit angibt, ob die Eingangswerte in einem Bereich liegen. Es sind 8 Kanäle, da die Speicher 8 Ausgänge haben und somit 8 Vergleiche gleichzeitig durchführen können. Durch Weiterschalten der anderen 9 Adressen des Speichers werden andere Vergleiche durchgeführt. Das Weiterschalten und das Zusammensortieren der Ergebnisse übernimmt auch die zweite Schicht. Eine Besonderheit dieses Netzes ist es, daß die Entscheidung aus der Abwägung von drei verschiedenen Netzen gebildet wird. Ein Netz wurde auf positive Daten trainiert, ein anderes auf die inversen Daten und ein letztes auf widersprüchliche Daten. Diese drei Netze wurden nicht einzelnd aufgebaut, sondern werden nacheinander durch die gleiche Hardware bearbeitet. Dazu wird zusätzlich zu den Speichern der ersten Schicht ein weiterer Speicher benutzt, der die Entscheidung der ersten Schicht mit einem Attribut versieht. Diese Attribute geben an, zu welchem Netztyp die in der ersten Schicht gebildete Entscheidung gehört. Die zweite Schicht wertet dann die Daten entsprechend aus. Durch diesen Attributspeicher können dann auch mehrere Netztypen implementiert werden, so daß später einmal auch die schon erwähnten Ausnahmeregeln dargestellt werden können. Die zweite Schicht des Netzes wird durch einen Xilinxbaustein realisiert. Dies ist ein programmierbares Gatearray mit etwa 900 Gattern und Registern (vgl *Abb.*14). Die Programmierung geschieht über den VME Bus. Die Programmierung ist nicht fest in dem Baustein eingebrannt, sondern wird in diesem Baustein in einem RAM geladen. Dadurch kann auch diese zweite Schicht völlig frei programmiert werden.

[9]Die Schicht mit den UND-Gattern kann auch als zweite Schicht bezeichnet werden, die nachfolgende Schicht dann als dritte Schicht. Durch eine solche Benennung wird die Ähnlichkeit zu einem dreischichtigen Backpropagation Netz deutlicher, soll hier aber nicht verwendet werden.

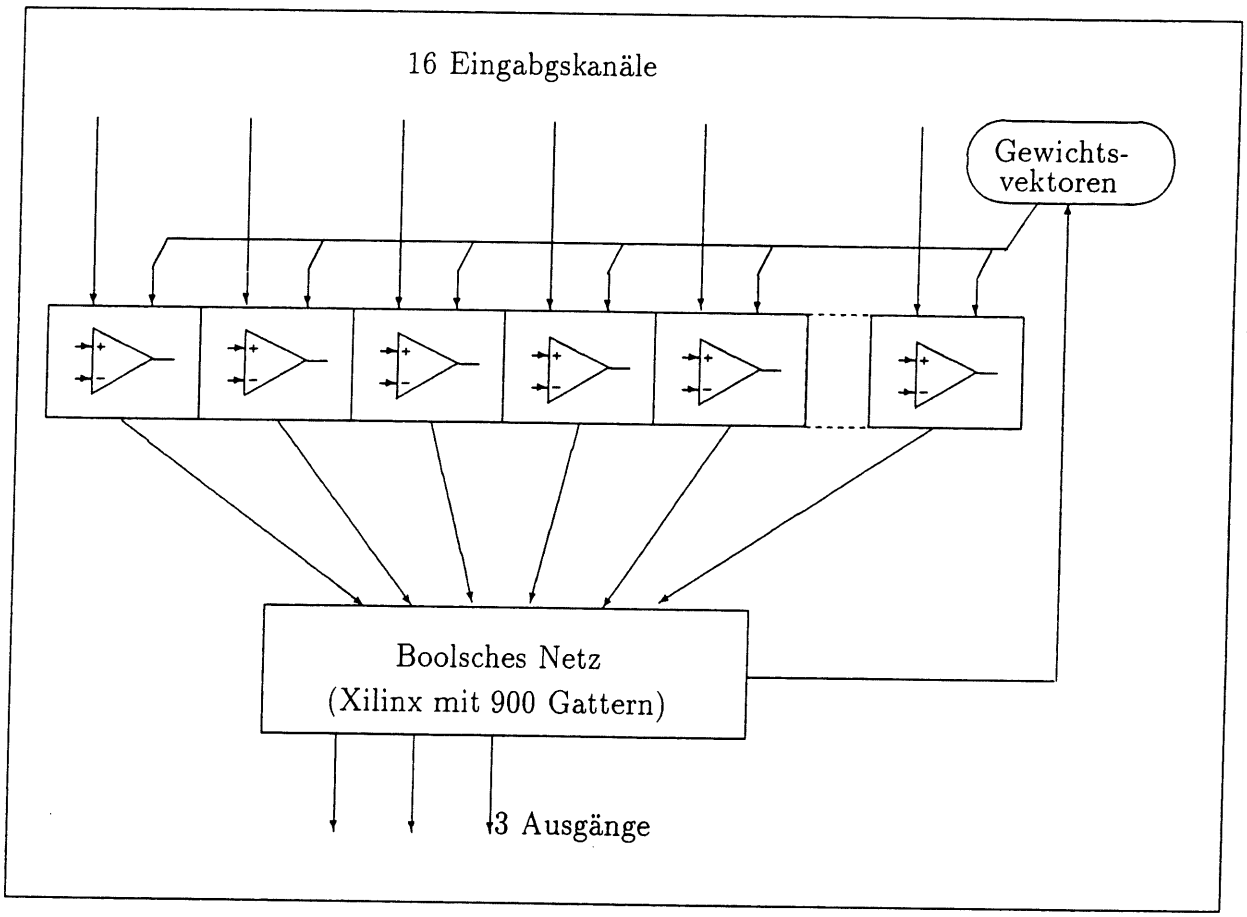


Abb. 14: Grundstruktur des Netzes

Um das Netz zu testen, um gezielt Kanäle auszuschalten oder mit festen Werten zu versehen, können alle 16 Eingangskanäle alternativ zu den Daten mit festen Werten beschaltet werden. Der Ausgang des Netzes kann von VME Bus ausgelesen werden.
Programmiert und initialisiert wird das Netz über den VME Bus. Dazu ist die Netzkarte mit einem VME Businterface ausgerüstet. Mechanisch soll die Netzkarte die Basiskarte sein. Alle Stecker befinden sich auf diese Karte. Die Karte soll ohne Huckepackkarte mit Interface zur TSMap testbar sein. Die Signale der TSMap werden von der unteren Karte auf die obere Karte geführt, die gewandelten Signale werden dann wieder zurückgeführt. Dieses wird über fünf 64-polige Stecker geleistet.
Auf der oberen Karte sollen sich alle Treiber zur Map und die dazugehörigen Abschlußwiderstände befinden. So können die Treiber leicht getauscht werden und die Abschlußwiderstände je nach Bedarf gesteckt werden.

13.3 Schaltungsbeschreibung

Die Time and Space Map Blatt 1: Eingangsinterface

Die Signale der TSMap werden auf zwei 64-poligen Verbindern angelegt. Diese teilen sich in 7 Kanäle mit je 8 Bit und einem Steuerkanal auf. Alle Signale sind als Differentialsignale ausgelegt. Die Signale werden durch einen Receiverbaustein auf TTL-Pegel konvertiert. Hierfür wurde der MC3450 verwendet. Dieser wird schon sei langer Zeit am DESY für solche Aufgaben eingesetzt und hat sich als robust herausgestellt. Außerdem zeichnet er sich durch eine kleine Durchlaufzeit aus. In einem 16-pol. Gehäuse sind 4 dieser Receiver untergebracht. Die Versorgungsspannung von +/- 5 Volt wird vom VME Bus genommen, wobei die -5V aus den -12V des VME Bus heruntergeregelt werden.
Die Leitungen können wahlweise mit einen 100Ω Abschlußwiderstand versehen werden. Dies ist bekanntlich dann erforderlich, wenn die Receiver am Ende der Leitung liegen. Hören die Receiver nur an einer vorhandenen abgeschlossenen Strecke mit, so dürfen diese nicht bestückt werden. Zusätzlich wird über zwei hochohmige Widerstände ein Strom auf jeden Receiver eingeprägt. Das hat zwei Gründe. Da die Empfängerbausteine über einen Trennverstärker verfügen, ist der Eingang galvanisch von der restlichen Elektronik getrennt. Dies ist erforderlich, da sich sonst Masseschleifen und somit Brummen oder andere Störeffekte ergeben würden. Da die einzelnen Geräte weit voneinander entfernt stehen können, haben nicht alle Geräte dasselbe elektrische Potential. Diese Potentialdifferenz würde ohne elektrische Trennung über den Empfänger fließen. Durch die hochohmige Kopplung der Potentiale kann dieser Ausgleichsstrom nicht abfließen, es wird aber verhindert, daß sich durch elektrische Aufladungen große Spannungen auf den Leitungen aufbauen. Dadurch, daß ein Strom eingeprägt wird, haben alle Leitungen einen definierten Pegel, wenn die Leitungen nicht angeschlossen sind, oder ein Kabelbruch vorliegt. Es hat sich gezeigt, daß die Receiver bei offenen Leitungen sonst zum Schwingen neigen. An die Receiver schließt sich ein Register an, welches die Signale mit dem ebenfalls übertragenen Takt wieder synchronisiert. Alle Flanken der Datensignale sind dann wieder gleich, auch wenn sie auf dem Bus durch unterschiedliche elektrische Eigenschaften der Leitungen auseinander gedriftet sind. Die Daten sind durch das Ablatchen um einem Takt nach hinten verschoben worden. Neben den 7 Datenkanälen liefert diese Schaltung noch den Takt, mit dem die Daten seriell über dem Bus geschoben werden und das Startsignal, das den Beginn der Übertragung anzeigt.

Blatt 2 :

Auf diesem Blatt befinden sich drei 64-polige Stecker, mit denen die 16 Ausgangskanäle auf die Netzkarte übertragen werden. Außerdem wird hierüber das Board mit Strom versorgt. Über einen kleinen Adreß- und Datenbus werden die Register zur Auswahl der Kanäle programmiert. Das Taktsignal und das Startsignal der Übertragung werden geliefert. Über zwei Demultiplexer wird aus den Adressen und dem Datenstrobe die Strobeimpulse zum Übernehmen der Daten in die entsprechenden Register gebildet. Ein Zähler zählt nach entsprechendem Startsignal die Taktimpulse. Dieser Zähler ist so konstruiert, daß er nicht überläuft, wenn er seinen Maximalwert erreicht hat, sondern daß er anhält. Dies ist wichtig, da über diesen Bus nicht nur die Daten für den L2 Trigger übertragen werden, sondern anschließend können noch zusätzlich Daten übertragen werden. Da das Zählerergebnis später mit Komparatoren verglichen werden soll, muß hier ein Synchronzähler verwendet werden. Dieser wird hier dadurch konstruiert, indem die Zählerbits des Asynchronzählers 74HC393 durch ein Register mit dem Takt synchronisiert werden. Dadurch wird zusätzlich erreicht, daß die Verzögerung durch das Ablatchen der Daten wieder kompensiert wird, da der Referenzzähler nun auch verzögert zählt. Das Latch ist so geschaltet, daß die Daten auf der negativen Flanke des Taktes übernommen werden. Da sich die Datenbits auf der positiven Flanke des Taktes ändern, kann somit aus dem Zähler ein Übernahmepuls gebildet werden, dessen Flanken immer auf stabilen Daten liegen. Ein Problem stellt die Ausgangsleistung des Datentreibers und des Zählerregisters dar, da diese 16 Eingänge treiben müssen. Bei dem hier verwendeten HCMos Bausteinen ist dies noch im Bereich der Spezifikation, sollte es aber zu Problemen kommen, können diese durch ACMos Bausteine ausgetauscht werden

Blatt 3/4/5/6:

Auf diesem Blatt werden die 7 Eingangskanäle auf die 16 Ausgangskanäle verteilt. Jeder Ausgangskanal ist über jeweils 7 Register mit den 7 Eingangskanälen verbunden. Die Auswahl eines Faches aus der TSMap geschieht über ein 8 Bit Register. Dieses wird über einen kleinen Adreß- und Datenbus von der unteren Karte programmiert. Die unteren 3 Bit selektieren über einen Demultiplexer eines dieser 7 Register. Die oberen 5 Bits werden mit dem Taktzähler verglichen und ergeben im richtigen Augenblick einen Übernahmeimpuls für die Register. Dieser Vergleich geschieht über einen Komparator 74HC688. Da es zu undefinierbaren Zuständen kommen kann, wenn die Eingänge geändert werden,

wird dieser Komparator zusätzlich synchron zum Takt enabeld. Dies geschieht bei einem LOW auf der Taktleitung. Die Daten ändern sich bei ansteigender Flanke des Taktes. Aufgrund der Durchlaufzeiten durch die Datenlatches (Blatt1) und den Komparator[10], ist der Komparator ausgeschaltet, wenn die Daten geändert werden. Die Blätter 4/5/6 unterscheiden sich nur in der Benennung der Ausgangskanäle und sind nicht extra ausgedruckt.

Das Netz Blatt 1:

Diese Schaltung ist das VME-Bus Interface. Das Interface ist einfach ausgelegt, da sich die Kommunikation mit dem Rechner nur auf die Initialisierung des Netzes und entsprechende Kontrollfunktionen beschränkt. Das Interface unterstützt den 16 Bit Datenbus mit 24 Adreßbits. Auf die spezielle Shortadressierung, bei der nur 16 Adressen benutzt werden, wurde verzichtet. Das Interface unterstützt den Blocktransfer Modus. Bei diesem Modus werden Daten unter Umgehung der CPU direkt vom Speicher in das Interface geschoben. Ein solcher Buszyklus wird durch eine entsprechende Kombination der Adreßmodifer angezeigt. Ein solcher Zugriff beginnt wie ein normaler Zugriff durch Anlegen der Adressen und Setzen von Adreßstrobe und den Datenstrobes. Beschrieben wird jetzt ein Blocktransfer aus dem Speicher in das Interface. Nach der Selektierung werden die Daten aus dem Speicher gelesen und angelegt. Abweichend von einem normalen Zugriff wird dieser Zyklus nicht nach der Übergabe des ersten Datums beendet. Nach der Übergabe des ersten Wortes werden die Datenstrobes ungültig, der Adreßstrobe bleibt aber aktiv, so daß die Karte weiter selektiert bleibt. Zu diesen Zeitpunkt haben sich die Adressen geändert. Das Interface muß also die Adressen für die Dauer eines Zugriffes speichern. Das nächste Wort wird übertragen, indem die Daten angelegt werden und die Datenstrobes wieder aktiv werden. Das Interface muß seine Adresse jetzt selbst weiterzählen. Dies wird durch einen Zähler geleistet. Der Zyklus wird beendet, wenn der Adreßstrobe wieder ungültig wird. Auf diese Weise können durch einen einzigen Zugriff der CPU sowohl der Speicher ausgelesen und gleichzeitig die Daten in das Interface transferiert werden. Die Daten werden also nicht wie sonst üblich erst von einer CPU oder DMA Unit gelesen und in einem zweiten Zugriff an ihr Ziel gespeichert. Auf diese Weise kann die große Speicherbank (2 MByte) des Netzes schnell geladen werden. Außerdem wird durch das automatische

[10]Das Enable an diesem Komparator hat eine definiert kürzere Durchlaufzeit als der Vergleich, so daß es auch bei gleichzeitigem Ändern von Daten und disabeln des Komparators nicht zu einem Spike kommen kann.

Hochzählen der Adresse vermieden, daß das Interface einen großen Adreßraum im VME Bus belegt, da die Daten nacheinander auf die gleiche Adresse geschrieben werden. Da sich ein normaler Zugriff auf diese Adresse und ein Blocktransfer mit 1 Wort Länge nur durch die Adreßmodifier unterscheiden, ist es auch möglich, diesen Mode zu nutzen, wenn die CPU keinen Blocktransfer unterstützt. Dazu müssen dann die Daten nur nacheinander auf die gleiche Adresse geschrieben werden. Um dies zu ermöglichen, werden auf dem Interface die Adreßmodifier für normalen und Blocktransfer nicht unterschieden. Zu Testzwecken kann das automatische Hochzählen der Adresse abgeschaltet werden und die Adresse über einen Port gesetzt werden. Dadurch wird es möglich, gezielt Bytes zu setzen. Ebenso ist es zu Testzwecken möglich, den gesamten Speicher wieder zurückzulesen.
Die Adreßselektierung geschieht durch zwei 8 Bit Komparatoren und dem Auscodieren der Adreßsmodifier. Die unteren 8 Bit der Adresse selektieren die Register auf der Karte. Wegen des Blocktransfers werden diese in einem Register für die Dauer eines Zugriffes gehalten. Um die Laufzeit durch die Komparatoren zu kompensieren, wird das Adreßstrobe durch einen RC-Paß verzögert. Eine Schottky Diode sorgt dafür, daß diese Verzögerung nur beim Aktivieren des Adreßstrobe wirkt. Beim Deaktivieren entlädt sich der Kondensator über diese Diode dann schnell. Neben der Verzögerung des Adreßstrobes wirkt dieser Paß als Integrator, der eventuelle Spikes auf dem Adreßstrobe ausintegriert. Solche Spikes hätten gerade beim Blocktransfermode Folgen, da sich der Adreßzähler verzählen würde. Durch das nachgeschaltete Register wird ein erkannter Zugriff für die Dauer des Adreßstrobes gespeichert. Dies ist erforderlich, da sich die Adressen und somit die Selektion beim Blocktransfer auch bei aktivem Adreßstrobe ändern. Das Adreßstrobe ist also abweichend von bekannten 68000-Timing, von dem der VME Bus abgeleitet ist, ein flankenaktives Signal. Mit fallender Flanke vom Adreßstrobe beginnt der Zugriff, das Ungültigwerden von AS beendet den Zugriff. Genau dieses wird durch das schon erwähnte Register geleistet. Die drei nachgeschalteten Gatter sind ein spezielles RS-Flipflop. Dieses Flipflop wird durch gültige Adreß- und Datenstrobes gesetzt. Gelöscht wird es durch Ungültigwerden des Datenstrobes unabhängig vom Adreßstrobe. Dieses ist deshalb erforderlich, da es beim VME Bus möglich sein muß, das Adreßstrobe und somit Adressen ändern zu können, während ein Zugriff auf eine Karte noch nicht abgeschlossen ist. Dadurch können auf den Bus schon die nächsten Adressen angelegt werden und die Einschwingzeiten der Adressen auf dem Bus verkürzt werden. Das Interface muß also in einem solchen Fall die Adresse speichern. Das nachgeschaltete Schieberegister steuert die zeitliche Abfolge eines Zugriffes. Der Datentreiber wurde schon vor dem Schieberegister aktiviert. Da-

nach werden die Adreßdecodierer auf dem Interface aktiviert. Sind diese stabil, wird ein Schreibpuls erzeugt. Dieser Puls wird für den Schreibzugriff auf die Speicherbausteine benötigt, da diese erst dann beschrieben werden sollen, wenn alle Signale stabil sind. Mit diesem Impuls wird das *Write* für die Speicher erzeugt. Dieses *Write* ist dann von stabilen Signalen umschlossen. Das Interface erzeugt drei Arten von Selektsignalen: für Schreibzugriffe, für Lesezugriffe und welche für Schreib-/Lese- Zugriffe. Des weiteren sind 8 Schaltbits und 8 Bit zum Auslesen auf diesem Interface. Über diese Bits können entsprechende Funktionen des Netzes geschaltet werden oder der Status ausgelesen werden. Das Setzen der Bits geschieht über einen 74HC259. Dies ist ein Decoderlatch mit Rücksetzeingang. Durch diesen Rücksetzeingang werden nach einem Systemreset alle Register auf 0 eingestellt. Durch den Decoder wird jedes Bit auf eine eigene Adresse abgebildet. Somit ist es möglich, die Bits unabhängig voneinander zu ändern. Zwei LEDs zeigen die Aktivität des Interfaces an. Eine LED pulst bei jedem Zugriff auf das Board, die andere ist programmierbar, nach einem Reset aber definiert eingeschaltet. Über diese LED kann dann angezeigt werden, ob das Board erfolgreich initialisiert wurde.

Blatt 2:

Auf diesem Blatt sind die Steckverbinder abgebildet. P1 ist Verbinder zum VME Bus, J6 ist ein Blindstecker, der auf den zweiten VME Busstecker gesteckt werden soll. Über diesen können dann beliebige Leitungen verlegt werden. Die Stecker J7 und J8 befinden sich an der Vorderseite der Platine und werden mit den Signalen des Triggersignalbusses verbunden (TSMap). Diese Stecker sind nur aus mechanischen Gründen auf der Platine. Die Signale werden unverändert auf die Stecker J4 und J5 gegeben, wo sie dann von der vorher beschriebenen Karte verarbeitet werden. Diese Karte erzeugt 16 Datenkanäle, die dann über die Stecker J1/J2/J3 wieder auf die Platine gelangen. Über J2 werden noch zusätzlich Steuersignale übertragen.

Blatt 3:

Auf diesem Blatt ist die zweite Schicht des Netzes dargestellt. Im wesentlichen besteht diese aus einem Xilinx. Dieser Xilinx ist mit dem VME-Datenbus und einigen Steuerleitungen verbunden, um programmiert zu werden. Nach einem Reset und einem entsprechenden Pegel auf der $*Programm$ Leitung erwartet der Baustein Daten auf den

Datenleitungen. Durch einen Puls auf der $*SEL - Xilinx$ Leitung werden die Daten eingelesen. Der Datentranfer wird über den $BUSY$ Ausgang des Xilinxs gesteuert. So werden nacheinander etwa 45 KByte in den Baustein geladen. Der Xilinx verfügt über einen internen Taktgenerator, der die Steuerung des Tansfares übernimmt.
Über die Signalleitungen CO(0:8) werden die Speicher der ersten Schicht adressiert. Diese Signale werden zur Programmierung der Speicher entweder durch ein Register oder durch einen Zähler gebildet. Wird das Netz nicht programmiert, werden diese Adressen vom Xilinx gebildet, um so in der ersten Netzschicht die Gewichtsvektoren auszuwählen. Zusätzlich befindet sich der Attributspeicher auf diesem Blatt. Durch diesen werden die adressierten Gewichtsvektoren einem bestimmten Netztyp zugeordnet. Der Xilinx kann dann die Ergebnisse der ersten Schicht entsprechend bewerten. Der programmierbare Zähler zählt die Takte bei der Übertragung der Triggerdaten. Sind alle Daten übertragen, kann das Netz gestartet werden. Dieser Zähler kann nicht fest auf einen Wert eingestellt werden, da es durchaus sein kann, daß die Daten für den Trigger in den ersten Fächern der TSMap eingetragen sind. Würde die gesamte Map abgewartet werden, bis das Netz gestartet wird, würde Zeit verschwendet. Dies ist deshalb wichtig, da sich die Anzahl der Neuronen des Netzes dadurch vergrößern, je mehr Zeit das Netz bekommt, da so mehr Gewichtsvektoren in der ersten Schicht mit den Eingangswerten verglichen werden können. Der Xilinx wird über fünf Modeleitungen gesteuert. Dadurch soll es möglich sein, die Funktion des Bausteines zu verändern, ohne die Programmierung zu ändern. Diese Modeleitungen müssen dann im Xilinxsprogramm abgefragt werden. Über fünf Ausgänge werden die Ergebnisse und der Status des Netzes ausgegeben. Das Auslesen der Daten geschieht über ein Latch, das mit aktivem Datenstrobe die Daten festhält. Dies ist erforderlich, da sich diese Daten asynchron zum VME-Bus ändern. Durch das Latch ist gewährleistet, daß sich die Daten während des Auslesens auf dem Bus nicht mehr ändern können. Dieses könnte sonst zu unübersehbaren Fehlern im VME-Bus-Timing führen. Der Xilinx wird mit einem externen Takt versorgt, da die Funktionen im Xilinx durch eine Statemaschine gebildet werden.

Blatt-4/5 :

Diese ist die erste Schicht des Netzes. Sie besteht im wesentlichen aus Speicherbausteinen. Die unteren 8 Adressen werden durch die Eingangskanäle gebildet. Durch Umschalten der übrigen Adressen können so die Eingangskanäle mit Daten aus dem Speicher verglichen werden. Die so gebildeten Vergleicher werden durch Und-Gatter verknüpft und an die

zweite Schicht des Netzes weitergegeben. Zur Programmierung dieser Speicher können die unteren 8 Adressen der Speicher durch Register gesetzt werden. Die Datenleitungen sind dann mit einem Treiber mit den Datenleitungen des VME-Busses verbunden. Durch die Register kann dann eine Eingabe des Netzes simuliert werden. Dazu werden diese dann anstelle des Dateneinganges geschaltet. So kann dann auch ohne Daten der Trigger getestet werden. Zusätzlich ist der gesamte Speicher rücklesbar, um diesen Testen zu können. Blatt 5 unterscheidet sich von Blatt 4 nur durch die Benennung der Ein- und Ausgangskanäle und ist deshalb nicht extra ausgedruckt.

13.4 Der Xilinx als Parallelprozessor

Die Funktion des Xilinxs kann nicht nur, wie bei Gatearrays üblich, als Gatterschaltung oder als boolsche Funktion beschrieben werden, sondern auch als ein Programm. Aufgrund der parallelen Struktur dieses Bausteines und der frei verdrahtbaren Verbindungen können so parallele Prozesse realisiert werden. Bei der Erstellung eines solchen Programmes wird aus dem Algorithmus eine Statemaschine abgeleitet. Eine solche Statemaschine kann dann durch die Xilinxsoftware übersetzt werden. Für die Erstellung der Statemaschine ist ein Compiler denkbar. Ein solcher ist aber nicht erhältlich, so daß diese Arbeit von Hand erledigt werden muß. Aufgrund der besonderen Struktur des Xilinxs ist es möglich, mehrere unabhängige Statemaschinen auf einem Chip zu realisieren.

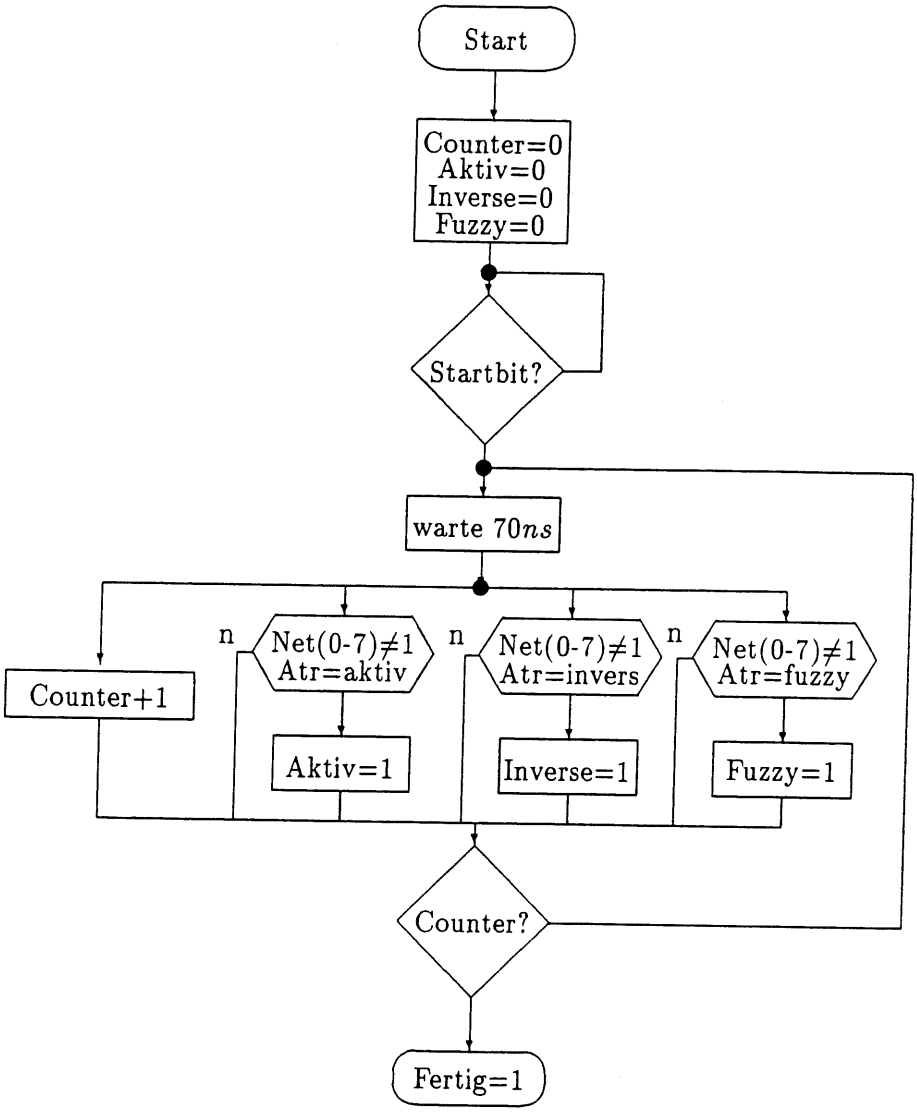


Abb. 15: Einfacher Algorithmus zur Realisierung des boolschen Netzes. Es wird festgestellt, ob mindestens ein Neuron einer gewissen Klasse aktiv ist.

Der hier dargestellte Algorithmus kann in 5 Tasks zerlegt werden. Jede Task wird durch eine Statemaschine realisiert. Eine Task realisiert den Adreßcounter, drei weitere Tasks überwachen je die Aktiven , Inversen und die Fuzzy Neuronen. Die letzte Task wird gestartet, wenn das Netz seine Arbeit beendet hat und trifft aus den drei Netzen eine Entscheidung. Die Kommunikation der Statemaschinen erfolgt über gemeinsame Register (interne Pins).

13.5 Auflistung einer Xilinxstatemaschine

```
Output Pin : co(0..8),trigger-bit,undef-bit,fuzzy-bit, trigger-ready;
Input Pin : clock,startbit,atraktiv,atrundef,atrfuzzy, net(0:7);
Intern Pin : undef,aktiv,fuzzy,reset;


STATE-MASCHINE akitv-net (state1,clock)
CASE state1 IS
WHEN start
   aktiv = 0
   IF (not startbit) THEN state1 = start
   ELSE state1 = wait
   END IF;
WHEN wait
   aktiv = aktiv
   state2 = test ;
WHEN test
   IF (reset) THEN state1 = start
   ELSEIF (net(0:7) = 0,0,0,0,0,0,0,0) THEN state1 = wait
   ELSEIF (atraktiv)
    aktiv = 1
    state1 = wait
   ENDIF;
WHEN ready
   IF (startbit * /reset) THEN state1 = ready
   ELSE state1 = start
   ENDIF;
END STATE MACHINE aktiv-net.


STATE-MACHINE invers-net (state2,clock)
   wie STATE-MACHINE 1, aber für inverses Netz


STATE-MACHINE fuzzy-net (state3,clock)
   wie STATE-MACHINE 1, aber für fuzzy Netz


STATE-MASCHINE counter (state4,clock)
CASE state4 IS
WHEN start
   co(0..8) = 0,0,0,0,0,0,0,0
   IF (not startbit) THEN state4 = start
   ELSE state1 = waitforcount
```

```
    END IF;
  WHEN waitforcount
    co(0..8) = co(0..8)
    IF (state1 = wait) THEN state4 = count /* state1 aus statemachine 1 */
    ELSEIF (reset) THEN state4 = start
    ELSE state4 = waitforcount;
  WHEN count
    co(0) = /co(0)
    co(1) = /co(0)*/co(1)+co(0)*co(1)
    co(2) = /co(2)*/co(1)+/co(2)*co(1)*/co(0)+co(2)*co(1)*co(0)
    co(3) = ( es folgen die Terme für einen synchronen 9bit Zähler)
    IF (co(8) = 1 state4 = ready
    ELSE state4 = waitforcount;
  WHEN ready
    IF (startbit * /reset) THEN state4 = ready
    ELSE state4 = start;
END STATE MACHINE counter.


STATE-MASCHINE ergebnis (state5,clock)
CASE state5 IS
WHEN start
   trigger-bit = 0
   undef-bit = 0
   fuzzy-bit = 0
   trigger-ready = 0
   IF (state4 = ready) THEN state5 = setoutputs
   ELSE state5 = start
   END;
WHEN setoutputs
   trigger-bit = aktiv * /inverse * /fuzzy
   fuzzy-bit = fuzzy
   undef-bit = aktiv * invers * /fuzzy + /aktiv * /invers * /fuzzy
   trigger-ready = 0
   state5 = ready
WHEN ready
   trigger-ready = 1
   trigger-bit = trigger-bit
   fuzzy-bit = fuzzy-bit
   undef-bit = undef-bit
   IF (startbit * /reset) THEN state5 = ready
   ELSE start5 = start
END STATE MACHINE ergebnis.
```

14 Neuronale Netze ohne orthogonale Neuronen

Zur Zeit sind mehrere verschiedene Ansätze von neuronalen Strukturen gebräuchlich. Diese unterscheiden sich sowohl durch die Implementation als auch im theoretischen Ansatz :

- Rückgekopplte Strukturen (feed backward Net)
- Mehrschichtige Strukturen ohne Rückkopplung und klare Trennung der Schichten (feed forward Net)
- Ungeordnete Strukturen (chaotic Net)

Rückgekopplte Strukturen zeichnen sich dadurch aus, daß das Ergebnis einer Berechnung durch eine Gleichgewichtslage bestimmt wird. Durch diese Gleichgewichtslage sind solche Strukturen sehr unempfindlich gegenüber Störungen in den Eingangsdaten. Das genaue Verhalten eines Neurones ist nicht entscheidend für den Ausgangswert, sondern das Ergebnis wird in einer Art Konkurenzprozeß der Lösungen gebildet. Eine gebräuchliche Anwendung für solche Netze sind sensorische Karten. Bei einer solchen Karte sind die Neuronen in einer zweidimensionalen Ebene angeordnet. Jedes Neuron ist mit dem Eingangsvektor und allen übrigen Neuronen verbunden. Neuronen in der räumlichen Nähe eines Neurons begünstigen dieses, weiter entfernte hemmen das Neuron. Dadurch bleibt nach einer gewissen Zeit nach der Aktivierung nur ein Bereich mit der stärksten Aktivität übrig, andere Bereiche werden durch die Hemmung unterdrückt. In einem solchen Netz können nun gewisse Bereiche in der XY-Ebene mit gewünschten Klassifikationen gleichgesetzt werden. Der Lernalgorithmus für ein solches Netz beginnt mit zufällig gewählten Gewichten. Beim Trainieren eines solchen Netzes werden bekannte Werte auf das Netz gegeben. Die durch das Netz daraus errechneten Werte werden mit den gewünschten verglichen. Die Gewichte werden dann in Richtung zum gewünschten Ergebnis hin vergrößert und in Richtung weg von dem gewünschten Ort verkleinert.
Solche Netze sind sehr interessant, da sie sich am stärksten von allen Netzansätzen an den Neurocomputer Gehirn anlehnen. Das Gehirn ist im wesentlichen eine zweidimensionale Struktur. Die Nervenzellen befinden sich an der Oberfläche des Gehirns, im Inneren befinden sich die Verbindungen der Nervenzellen untereinander. Solche sensorischen Karten konnten in Gehirnen von Fledermäusen zur Ultraschallortung nachgewiesen werden. [11]

[11] vgl. / Kohonens Modell am Beispiel des auditiven Kortex der Fledermaus / Neuronale Netze / Helge Ritter, Thomas Martinetz, Klaus Schulten / Addison-Wesley Verlag / 1990

Ein Problem solcher Netze ist, daß der Aufwand der Verbindungen zu stark mit der Anzahl der Neuronen ansteigt, da aufgrund der gegenseitigen Hemmung alle Neuronen miteinander verbunden sind. Um dieses Problem zu umgehen, kann statt der gegenseitigen Hemmung eine globale Hemmung eingeführt werden. Hat ein Neuron ein bestimmtes Aktivitätspotential überschritten, so werden global alle anderen Neuronen immer mehr gehemmt. Durch eine solche Struktur kann der Aufwand der Hemmung auf eine einzige Verbindung reduziert werden, die mit allen Neuronen verbunden ist [12]. Ein besonderer Aspekt solcher Strukturen ist, daß diese offensichtlich gestartet werden und nach einiger Zeit nur wenige Neuronen übrig bleiben, die alle anderen hemmen. Ist ein solcher Punkt erreicht, muß die globale Hemmung gelöscht werden, damit das Netz erneute Berechnungen durchführen kann. Ein solches Netz reagiert also nicht kontinuierlich auf die Eingangswerte, sondern wird gestartet und kommt dann zu einem Ende. Ein solches Netz muß also schwingen, wenn damit kontinuierliche Ergebnisse erzielt werden sollen. Verblüffenderweise wird ein solches Schwingen bei biologischen Netzen beobachtet. So werden im menschlichen Gehirn Frequenzen im Bereich von 10 Hz bis 100 Hz gemessen. Diese Frequenzen ändern sich mit der Aktivität einzelner Gehirnteile lokal. Dieses Frequenzraster ist dem Reaktionszeitraum des Gehirnes ähnlich. Es ist zu vermuten, daß ein biologisches Gehirn in Teilen ähnlich wie ein solcher Algorithmus arbeitet.

Zur Implementierung eines solchen Netzes bieten sich statistische Neuronen an. Bei solchen Neuronen werden die kontinuierlichen Eingangswerte durch Modulation von digitalen Signalen abgebildet. Durch Mischen dieser Signale mit anderen können Neuronen nachgebildet werden.[13] Für solche Neuronen ist eine rückgekopplte Struktur erforderlich, da hier das Ergebnis nicht kritisch von einem einzelnen Neuron abhängt. Dadurch, daß die analogen Werte auf digitale abgebildet werden, kann ein solches Netz gut durch digitale Hardware aufgebaut werden. Die biologische Datenübertragung auf Nervenfasern benutzt genau einen solchen Code. Ein Aktivierungswert wird durch die Häufigkeit von Pulsen auf einer Nervenfaser übermittelt. Eine Nervenfaser kann aufgrund ihres Aufbaus nur Pulse gleicher Intensität und Dauer übertragen, durch die Häufigkeit der Pulse wird die Information übertragen.

Bei einem feed-forward Netz findet keine Rückkopplung der Neuronen statt. Das Signal läuft von der Eingangsschicht in definierter Zeit bis zur Ausgangsschicht. Ein derartiges

[12] vgl. / Complex Dynamics in Winner- Take-All Neural Nets With Slow Inhibition / Bard Ermentrout / Neuronal Networks / Vol5 / pp415-431 / 1992

[13] vgl. / Continuous Input RAM-Based Stochastic Neural Model / Denise Gorse, John G. Taylor / Neural Networks / Vol4 / pp657-665 / 1991

Netz ist klar in Schichten unterteilt. Die Kommunikation erfolgt ausschließlich von einer übergeordneten Schicht zu der nächsten Schicht. Solche Netze zeichnen sich durch kurze Durchlaufzeiten aus. Das Verhalten dieser Netze ist gut vorhersagbar, zu einem unkontrollierten Schwingen kann es nicht kommen. Durch geeignetes Pipelining der Schichten kann die Verarbeitungszeit für ein solches Netz auf die Durchlaufzeit einer Schicht optimiert werden. Für nicht rückgekopplte Netze ist die Rechengenauigkeit und Reproduzierbarkeit der einzelnen Neuronen entscheidend. Diese Netze werden häufig zur Klassifizierung verwendet. Als Lernverfahren für ein solches Netz wird standardmäßig der Backpropagation Algorithmus verwendet. Bei diesem Algorithmus werden die Gewichte der Neuronen schrittweise durch ein Optimierungsverfahren berechnet. Dieser Algorithmus konvergiert schnell, hängt aber kritisch von den Anfangswerten ab. Prinzipiell ist es möglich, ein rückgekoppltes Netz auf ein feed-forward Netz mit unendlichen Schichten abzubilden.
Ein anderer Lernalgorithmus versucht nicht, wie bei dem Backpropagation Algorithmus die Meßwerte durch Polynome zu umschreiben, sondern benutzt Gaußfunktionen. Diese Gaußfunktionen werden so gebildet, daß sie möglichst viele Werte approximieren. Der Algorithmus beschreibt zuerst jeden trainierten Wert durch eine solche Funktion und faßt in einem zweiten Schritt dann möglichst viele solcher Funktionen zusammen (Cluster Bildung).[14] Das Zusammenfassen von Funktionen zu Clustern scheint mir in hochdimensionalen Räumen problematisch, da hier ein Cluster als zusammenhängender Raumkörper nicht die Bedeutung wie etwa in zwei Dimensionen hat. Aufgrund der großen Anzahl der Dimensionen ist die Trainingswertdichte in vielen Bereichen sehr gering. So kann ein Würfel mit der Kantenlänge von 10 Einheiten in zwei Dimensionen vollständig durch 100 Werte beschrieben werden. In 12 Dimensionen sind das aber schon 10^{12} Werte. Dieses Beispiel zeigt, daß die Meßwertdichte in 12 Dimensionen bei einer begrenzten Anzahl von Trainingswerten sehr gering sein muß. Deshalb kann der Versuch, Raumbereiche zu Clustern zusammenzufassen, daran scheitern, daß es in einer Dimension immer eine Verbindung zwischen diesen Raumkörpern geben kann, wenn diese dicht beieinander liegen, oder zwei zusammenhängende Bereiche nicht zusammengefaßt werden, wenn sie sich in einer unwichtigen Dimension zu sehr unterscheiden.
Der hier entwickelte Kristallisationsalgorithmus berücksichtigt die Aspekte besser. Zum einem benutzt dieser Algorithmus als Basis keine Gaußfunktionen, sondern Sigmafunktionen, um den Rechenaufwand bei der Implementierung zu verkleinern. Im Unterschied zum Clusteralgorithmus wird hier mit einem Meßwert begonnen, der als Kristallisati-

[14] vgl./ On the Training of Radial Basis Funktion Classifiers / M. T. Musavi, W. Ahmed, K. H. Chan, K. B. Faris, D. M. Hummels / Neuronal Networks / Vol 4 / pp595-603 / 1992

onskeim dient. Ausgehend von diesem Meßwert und einiger anderen Randbedingungen wird nun versucht, ein orthogonales Neuron (d.h. eine mehrdimensionale Sigmafunktion) um diesem Wert zu legen, so daß möglichst optimal alle anderen Werte erkannt werden. Auf diese Weise wird das Problem der Clusterbildung in höheren Dimensionen umgangen. Entscheidend für die Berechnung der Neuronen ist nur die Verbesserung des Netzes durch die Variation der Neuronen. Eine Abhängigkeit von Anfangswerten, wie bei dem Backpropagation-Algorithmus, wird durch diesen Algorithmus vermieden. Der Rechenaufwand und somit die Rechenzeit ist sehr viel kleiner, als bei dem Clusteralgorithmus, da hier nicht mit Gaußfunktionen gerechnet werden muß und die aufwendige Nachbarschaftssuche in hohen Dimensionen entfällt auch. Die Ergebnisse des Algorithmus können direkt für ein Netz verwendet werden und müssen nicht wie bei dem Clusteralgorithmus noch entsprechen auf ein Netz transformiert werden.

Boolsche Netze werden häufig in ungeordneter Form verwendet. Ein solches Netz besteht dann aus einer Vernetzung von *UND–*, *ODER–*, *EOR–* Gattern und Invertern. Es ist nachweisbar, daß bei einer bestimmten Konzentration jedes Gattertyps ein derartiges Netz stabile Zustände aufweist, die in Abhängigkeit der Eingangswerte zu brauchbaren Klassifizierungen führen. [15] Solche Netze sind interessant, da die Neuronen hier nicht mehr an ein biologisches Vorbild angelehnt sind, sondern direkt auf digitalen Gattern beruhen. Durch spezielle Lernalgorithmen können diese Netze effektiv trainiert werden. Es ist zu vermuten, daß diese Netze eine optimale Implementierung von neuronalen Strukturen auf einer digitalen Hardware ermöglichen, da hier direkt die in digitalen Schaltungen zur Verfügung stehenden Elemente benutzt werden und nicht andere Strukturen simuliert werden müssen. Solche oder ähnliche Netze werden als assoziative Speicher eingesetzt. Das bekannteste Modell ist der Ansatz von Hopfield [16]
Bei diesem wird ein boolsches Netz so verwendet, daß die Ausgänge des Netzes wieder auf die Eingänge wirken und somit das Eingabemuster korrigiert. Die Klassifizierung einer Eingabe ist dann durch die Eigenwerte eines solchen Netzes gekennzeichnet $x = f(x)$. Da hier Probleme auftreten können, wenn sich eine solche Funktion an ungewollten Stellen (lokale Minima) 'verklemmt', werden solche Netze häufig mit Rauschgeneratoren erweitert, die dafür sorgen, daß die Lösung den Eigenwert bestimmt, der am wahrscheinlichsten

[15] vgl. / Dynamic behaviour of Boolean networks / D.Martland / Neural Computing Architectures pp217-235 / North Oxford Academic / 1989

[16] vgl. / Das Hopfield-Modell / Helge Ritter, Thomas Martinetz, Klaus Schulten / Neuronale Netze / Addison-Wesley Verlag / 1990

zu dem Eingabemuster paßt. Durch das Rauschen wird erreicht, daß dieser Eigenwerte dann am häufigsten auftritt und bei einem langsamen Absenken des Rauschenpegels einfriert.

15 Alternative Implementierung neuronaler Netze

Bei neuronalen Netzen ohne orthogonale Neuronen ist ein Hauptproblem die Bildung des Skalarproduktes in jedem Neuron. Dieses Produkt ist die Summe der Produkte der Eingangswerte mit den Gewichtswerten der Neuronen. Bei einem Netz mit 3 Schichten, 16 Eingangswerten, 50 Neuronen in der inneren Schicht und einem Ausgang sind dies 950 Multiplikationen und 950 Additionen, die berechnet werden müssen. Die Rechenauflösung der Operation hängt von der verwendeten Architektur ab. Eine kleinere Auflösung kann durch mehr Neuronen in der innere Schicht kompensiert werden. Wird ein rückgekoppltes Netz verwendet, ist nur eine relative Genauigkeit der Rechenoperationen erforderlich.
Verwendet man für das oben angeführte Problem einen Signalprozessor (z.B. DSP 56000), so ist dieser in der Lage, ein solches Skalarprodukt optimal zu berechnen, da Signalprozessoren auf die Berechnung von digitalen Filtern hin optimiert wurden und somit Skalarprodukte effektiv berechnen können. Der DSP56000 ist in der Lage, in 100 ns eine Addition und Multiplikation auszuführen. Demnach würde dieser für die oben beschriebene Aufgabe 95 μs benötigen. Der Aufwand für die Schwellwertbildung und die Kommunikation kann mit 20% abgeschätzt werden. Somit ergibt sich eine Rechenzeit von 114 μs. Da das Problem in etwa 18 μs berechnet werden soll, müßten 7 solcher Prozessoren verwendet werden.
Die Ähnlichkeit dieses Skalarproduktes zu den Operationen in einem digitalen Transversalfilter kann dazu benutzt werden, solche Filterbausteine als Neuronen einzusetzen. So ist z.B. ein Tranversalfilter erhältlich (Inmos IMS 110), der innerhalb von 400 ns 32 solcher Additionen und Multiplikationen mit 16 Bit Auflösung ausführt. Wird die Rechenauflösung bei diesem Baustein auf 12 Bit begrenzt, verdoppelt sich die Rechengeschwindigkeit. Leider werden die Daten und Gewichtsvektoren seriell in den Baustein geschoben. Dieser Baustein kann nur zwei getrennte Koeffizientensätze verwalten, das Laden des Bausteines mit neuen Koeffizienten dauert etwa 1 μs. Durch solche Bausteine könnte ein rein digitales Netz aufgebaut werden, das innerhalb von etwa 400 ns + 1 μs Ladezeit eine Entscheidung trifft. Für ein solches Netz müßten dann aber sehr viele dieser Bausteine (Stückpreis $300) verwendet werden.
Für digitale Bildverarbeitung und Radarauswertung sind sogenannte Convoluter oder

Correlater erhältlich. Diese Bausteine bilden ebenfalls das Skalarprodukt mit Eingangswerten und festen Koeffizienten. Da diese Bausteine dafür ausgelegt sind, zweidimensionale Daten zu bearbeiten, können hier im Gegensatz zu den Transversalfiltern mehrere Koeffizientensätze verwaltet werden. Ein geeigneter Baustein für eine solche Anwendung ist der IPRMM/Oxford Computer INC.[17] Dieser Baustein in der Lage, das Problem etwa 4mal so schnell wie ein Signalprozessor zu bearbeiten. Somit würden zwei dieser Bausteine ausreichen, um das Netz aufzubauen. Die Schwellwertbildung müßte dann extern durch Speichertabellen realisiert werden. Ein solcher Baustein ist erhältlich und kostet etwa $800.

Die Skalarproduktbildung in einem neuronalen Netz stellt nicht die gleichen Qualitätsanforderungen an die Linearität, wie z.B. ein Tranversalfilter, da diesem eine nichtlineare Schwellwertfunktion folgt. Bei einem neuronalen Netz ist es wichtig die Daten schnell in den Chip zu bekommen. Würden an einen Chip die oben angeführten 16 Eingänge mit 8 Bit Auflösung geführt werden, müßte dieser schon 128 Pins als Eingang verwenden.

Ein andere Möglichkeit, eine neuronalen Chip zu konstruieren, ist es diesen analog aufzubauen. Die Skalarproduktbildung mit festen Koeffizienten könnte dann durch einfache Festwiderstände erfolgen, in denen dann Ströme addiert werden. Die Anzahl der Eingangspins ist dann die Anzahl der Eingänge, da die Daten als Ströme analog an den Chip geführt werden. Ein solcher Chip wird von der Firma INTEL/80170NX angeboten.[18] Dieser Chip ist komplett analog aufgebaut. Die Gewichtsvektoren werden in Epromzellen analog gespeichert. Durch entsprechende Referenzzellen wird versucht, entsprechende Temperatur- und Altersdrifterscheinungen zu kompensieren. Der hier beschriebene Chip besteht aus zwei hintereinandergeschalteten Schichten mit je 64 Neuronen. Die Schichten können parallel geschaltet werden, oder die zweite Schicht kann auf sich selbst zurückkoppln. Der Chip verfügt über 128 analoge Eingangsleitungen und 10 analoge Ausgänge und kann begrenzt kaskadiert werden. Die Auflösung der Rechenoperation wird mit äquivalenten 8 Bit angegeben. Dieser Chip ist erhältlich und kostet etwa $1000. Das Trainieren des Chips erfolgt zuerst komplett als Simulation auf einem Rechner. Die so berechneten Gewichte werden dann in den Chip gebrannt. Da aufgrund des analogen Aufbaues des Chips dieser nicht exakt simuliert werden kann, erfolgt nach diesen Schritt eine systematische Kontrolle der Eigenschaften und die Gewichte werden entsprechend modifiziert und

[17] Datasheet for Intelligent Pattern Recognition Memory Module (IPRMM) / Oxford Computer inc / August 1990

[18] Datasheet The 80170NX Electrically Trainable Analog Neural Network (ETANN) Chip / Intel Corporation / 1991

erneut in den Chip gebrannt. Besonders unangenehm ist dabei, daß das Löschen durch UV-Licht erfolgen muß und die Lebensdauer dadurch begrenzt ist. Durch das wiederholte Beschreiben und Löschen verändern sich auch die elektrischen Eigenschaften der Epromzellen, so daß die korrigierten Gewichte nicht auf den durch das Löschen veränderten Chip optimiert sind. Außerdem muß der Chip zum Löschen aus der Fassung genommen werden, da sich das Quarzfenster auf der Unterseite befindet. Dieser Chip ist andererseits sehr leistungsfähig und die direkte Verarbeitung von analogen Daten macht gerade bei Verarbeitung von Meßwerten eine Digitalisierung überflüssig. Somit könnten durch einen einzigen Chip die Anforderungen des L2 Triggers erreicht werden. Die Laufzeit durch den Chip beträgt 2 μS pro Schicht und entspricht der Entscheidungszeit für den L1 Trigger. Somit könnte durch 20 solcher Chips, die zeitlich um je 96 ns versetzt die L1 Daten sehen, ein neuronaler Trigger gebaut werden, der die Eigenschaften des L2 Triggers besitzt, aber schon auf L1 Ebene seine Entscheidung trifft. Diese Geschwindigkeit wird deshalb erreicht, da diese Chips im Gegensatz zu den vorigen Ansätzen wirklich parallel arbeiten.

Eine besonders effektive Implementation eines neuronalen Netzes ist mit Hilfe von optischen Komponenten möglich. Hierzu werden die Gewichtsvektoren in einem Hologramm gespeichert.[19] Bei dem Aufbau wird ein Laserstrahl aufgeweitet und durch eine LCD-Matrix geleitet. Durch das LCD werden die Eingangswerte entsprechend der Lichtdurchlässigkeit der Matrix codiert. Der so veränderte Laserstrahl wird durch ein Hologramm geführt. Durch die dort auftretenden Beugungen und Interferenzen entsteht nach dem Hologramm ein entsprechend verändertes Bild der Eingangsdaten. Durch Beugung kann jeder Punkt der Eingangsmatrix an jede Stelle der Ausgangsmatrix abgebildet werden. Durch Interferenz können diese Punkte entsprechend der gewünschten Gewichte aufaddiert oder ausgelöscht werden. Ein geeigneter CCD Sensor kann die Daten dann wieder aufnehmen. Durch spezielle Kombinationen aus CCD Sensoren und LCD Matrizen können Elemente konstruiert werden, die Licht mit nichtlinearer Kennlinie durchlassen. Solche Elemente könnten direkt aus Halbleitertransistoren und LCD Elementen auf einen Glasträger aufgebaut sein und die Schwellwertfunktion der Neuronen nachbilden. Durch erneute Einspiegelung des Ausgangsstrahles in den Eingangsstrahl können dann rückgekopplte optische Netze aufgebaut werden. Die Herstellung von synthetischen Hologrammen mit Dia- Belichtern ist möglich. Diese werden zur Herstellung von naturgetreuen Bildern verwendet. Solche Belichter sind mit einer Auflösung bis etwa 2048

[19]/ Optical implementations / J. E. Midwinter and D. R. Selviah / Neural Computing Architectures pp268-278 / North Oxford Academic / 1989

dots/inch im Schwarzweißbereich erhältlich. Diese Auflösung reicht für Hologramme im langwelligen Infrarotbereich aus. Solche Netze sind hinsichtlich der Neuronenzahl und der Verarbeitungsgeschwindigkeit den herkömmlichen Netzen um Größenordnungen überlegen. Verzichtet man auf rückgekopplte Strukturen, so ist ein solches Netz durchaus mit verfügbaren Mitteln aufbaubar. Ein solcher holographischer Trigger könnte mit einer Eingangsmatrix von $100 * 100$ Punkten arbeiten und somit alle Daten des Experimentes gleichzeitig verarbeiten. Die Rechengeschwindigkeit würde nur durch die Geschwindigkeit der Eingabe-LCD-Matrik und des CCD-Sensors begrenzt.

A Anhang :

A.1 Das Programm L1 Trigger

```
/* +++++ C Programm zur Simulation des L1 Triggers +++++ */
 int L1_trigg (et,etmiss,eif,efor,eback,ebarr,ebemc,elect,ebemcc,tof,dc)
    float et,etmiss,eif,efor,eback,ebarr,ebemc,elect,ebemcc,tof,dc;
{ /* 886 Hz  Tuning rate */
     #define ET1      12.0
     #define ET2      20.0
     #define ET3      35.0
     #define ETMISS1   5.0
     #define ETMISS2  12.0
     #define ETMISS3  25.0
     #define EFORW1    5.0
     #define EIF1      5.0
     #define EBARR1    5.0
     #define EBARR2    8.0
     #define EBARR3   12.0
     #define EBEMC1    5.0
     #define EBACK3   25.0

     if (  et > ET3 ) return (1);
     if ( (et > ET2 ) && (tof == 0) ) return (1);
     if ( (et > ET1 ) && (tof == 0) && (dc > 0) ) return (1);

     if (  etmiss > ETMISS3 ) return (1);
     if ( (etmiss > ETMISS2 ) && (tof == 0) ) return (1);
     if ( (etmiss > ETMISS2 ) && (dc > 0) ) return (1);
     if ( (etmiss > ETMISS1 ) && (tof == 0) && (dc > 0) ) return (1);

     if ( (eif < EIF1) && ( eback > EBACK3) && (tof == 0) ) return (1);

     if ( (efor < EFORW1) && ( eback > EBACK3) && (tof == 0) ) return (1);

     if (  ebarr > EBARR3 ) return (1);
     if ( (ebarr > EBARR2 ) && (tof == 0) ) return (1);
     if ( (ebarr > EBARR1 ) && (tof == 0) && (dc > 0) ) return (1);

     if ( (ebemc > EBEMC1 ) && (tof == 0) && (dc > 0) ) return (1);

     if ( (elect > 0) && (tof == 0) ) return (1);

     if (ebemcc > 10.0 ) return (1);

     return (0);
}
```

A.2 Abgleichparameter für den L1 Trigger [20]

	L1 rates(Hz)			
Quantity/threshold	94Hz	886Hz	4355Hz	10717Hz
$E_{T\ and\ TOF\ and\ DC}$	13	12	5	3
$E_{T\ and\ TOF}$	28	20	6	4
E_{T}	44	35	10	4
$E_{Tmiss\ and\ TOF\ and\ DC}$	13	5	3	2
$E_{Tmiss\ and\ TOF}$	28	12	5	3
E_{Tmiss}	44	25	8	5
E_{forw}	5	5	2	2
E_{IF}	5	5	2	2
$E_{barrel\ and\ TOF\ and\ DC}$	10	5	1	1
$E_{barrel\ and\ TOF}$	16	8	3	2
E_{barrel}	24	12	5	3
$E_{BEMC\ and\ DC\ and\ TOF}$	20	5	1	1
E_{back}	25	25	8	6

[20] entnommen aus / Proposal for a Second Level Neural Network Trigger / J. Fent, A. Gruber, C. Kiesling, H. Oberlack, P. Ribarics / DESY 17. März 92

A.3 Das Programm NTOOL

```
/*  ----  C Programm zum Trainieren eines orthogonalen Netzes -----  */
#include "global.inc"                        /* Variablen und Datentypen */
#include "neurolib.inc"                            /* Standardprozeduren */
/*  ------------------------------------------------------------------ */
/*                                                                     */
/*    #define      space         512                                  */
/*    #define      str_temp_net  (space/2)                            */
/*    #define      str_temp_net2 (space/2)                            */
/*                                                                     */
/*    #define      slow_down    -1          \fuer Neuronensuche       */
/*    #define      slow_down2   /2 -1       \zum Einparken der Neur.  */
/*                                                                     */
/*    typedef       \Grundelement in der Welt des neuronalen Netzes   */
/*      struct                                                         */
/*      { int xy[max_dimension+1];       \Koordinate in Raum           */
/*        int wert;                      \Wert auf der Koordinate      */
/*      }                                                              */
/*    point_of_world ;                                                 */
/*                                                                     */
/*    typedef    \Datentype Neuron                                     */
/*      struct         \Angabe des Schaltbereiches durch Mittelpunkt   */
/*      { int  center[max_dimension+1];              \und Ausdehnung    */
/*        int  width [max_dimension+1];                                */
/*        int  aktiv;      \Zustand des Neurons aktiv-passiv im Netz   */
/*        int  temp;       \aktuelle Temperatur im Abkuelungsprozess   */
/*        int  nr_in_net;      \eindeutige Nummer zur Unterscheidung   */
/*        int  type;            \aktiv,invers,fuzzy                    */
/*       }                                                             */
/*    my_new_neuron;                                                   */
/*                                                                     */
/*    typedef             \Trainingswert werden als dynamische Liste   */
/*      struct             \verwaltet                                  */
/*      { int number;                                                  */
/*        point_of_world point;                                        */
/*        int *next;                                                   */
/*      }                                                              */
/*    sample;                                                          */
/*                                                                     */
/*    typedef      \Das Netz ist ein Array aus Neuronen                */
/*      my_new_neuron                                                  */
/*    netz [nr_neuronen];                                              */
/*  ------------------------------------------------------------------ */

/* ------------------------------------------------------------------ */
/* Schnittstelle zum Unixbetriebssystem.                              */
```

```
/* Durch Angabe entsprechender Optionen kann die Funktion des        */
/* des Lernalgorithmus modifiziert werden                            */
/*  ---------------------------------------------------------------- */

main (argc,argv)
 int argc;                    /* Zeiger auf Parameter des Programmaufrufes */
 char *argv[];

    {  char          cmd[100];
       int           x,i;
       my_new_neuron neuro;
       char filename [maxstring];        /* Name der Daten zum trainieren   */
       char list     [maxstring];        /* Filename der Trainingsliste     */
       char out      [maxstring];        /* Filename der Liste mit Neuronen */

       int aktiv =1;                                  /* aktives Netz erzeugen */
       int invers = 0;                               /* inverses Netz erzeugen */
       int loadnet = 0;                 /* vorhandene Neuronenliste einlesen */
       int fuzzy_n = 0;                                  /* fuzzy net erzeugen */
       int optim  = 1;                   /* redundante Neuronen aussortieren */

                     /* Filename und Optionen aus Programmaufruf einlesen */
       if (argc == 1 )
            {printf(" one parameter is needed ! \n");exit (-1);};
       if (argc > 1)  strcpy (filename,*++argv);
            { strcpy (list,filename); strcat(list,".list");
                 strcpy (out,filename); strcat(out,".net"); }
       for ( i = 1 ; i < argc ; i++)
           { strcpy (cmd,*++argv);
             if (strcmp ("-a",cmd) == 0)  { aktiv = 0;};
             if (strcmp ("+a",cmd) == 0)  { aktiv = 1;};
             if (strcmp ("-f",cmd) == 0)  { fuzzy_n = 0;};
             if (strcmp ("+f",cmd) == 0)  { fuzzy_n = 1;};
             if (strcmp ("-i",cmd) == 0)  { invers = 0;};
             if (strcmp ("+i",cmd) == 0)  { invers = 1;};
             if (strcmp ("-s",cmd) == 0)  { show = 0; };
             if (strcmp ("+s",cmd) == 0)  { show = 1; };
             if (strcmp ("-s1",cmd) == 0) { show1 = 0; };
             if (strcmp ("+s1",cmd) == 0) { show1 = 1; };
             if (strcmp ("-o",cmd) == 0)  { optim = 0; };
             if (strcmp ("+o",cmd) == 0)  { optim = 1; };
             if (strcmp ("-n",cmd) == 0)  { loadnet = 0; };
             if (strcmp ("+n",cmd) == 0)  { loadnet = 1; };
                                          /* Zufallsgenerator modifizieren */
             if (strcmp ("+r",cmd) == 0)  { i++; if ( i > argc) break;
                                                strcpy (cmd,*++argv);
                                                srand(atoi(cmd) + 10); }
       }
```

```
        read_samplelist(list);              /* Liste mit Trainingswerten in
                                                  dynamische Liste einlesen */
        last_aktiv_neuron = 0;          /* vorhandene Neuronenliste einlesen */
        if ( loadnet)  { read_net(out); }

        stmain(aktiv,invers,fuzzy_n);       /* eigentliches Programm starten */

        write_netlist(out);                     /* Neuronenliste abspeichern */
        free (first_sample);                    /* und fertig */
        exit (0);
  } ;

/* ------------------------------------------------------------------ */
/* Hauptprogramm                                                      */
/* Netz nach entsprechenden Optionen trainieren und redundante        */
/* Neuronen aussortieren                                              */
/* ------------------------------------------------------------------ */
stmain (aktiv,invers,fuzzy_n)
  int aktiv;
  int invers;
  int fuzzy_n;
    {
     int i,j,k;
     my_new_neuron  dummy_neuron;
     int type_neuron;

     str_list();  /* dynamisch Liste mit Trainigswerten auf 1. Werte */
     if (aktiv)   train_net(aktiv_neuron);    /* entsprechene Netze  */
     if (invers)  train_net(invers_neuron);   /* trainieren          */
     if (fuzzy_n) train_net(fuzzy_neuron);

                          /*  sind  Neuronen doppelt, wenn ja, aussortieren */
     for (i = 0 ; i < last_aktiv_neuron -1; i++)
       { if (x_net[i].aktiv > 0 )
         for ( j = i+1 ; j < last_aktiv_neuron; j++)
           { if ( cmp_neuron(&x_net[i],&x_net[j]))
                    { x_net[j].aktiv = -1;}
           }
       }
              /* verbessert ein Neuron das Netz nicht, dann aussortieren */
      if (optim)
        { for ( i= 0 ; i < last_aktiv_neuron ; i ++)
           { for (x_net[i].aktiv)
              { x = error_net ( x_net[i].type );
                x_net[i].aktiv = -1;
                k = error_net ( x_net[i].type);
                if ( x < k ) x_net[i].aktiv = 1;
```

```
            }
          }
        }
    }

  /* ------------------------------------------------------------- */
  /* Trainieren eines Netzes mit angegebenem Type                  */
  /* Zuerst wird versucht, ein Neuron zu finden. Durch die Variable */
  /* tempf werden die Kriterien dafuer immer weiter gefasst. Danach */
  /* wird das gefundene Neuron optimiert.                          */
  /* Das Programm endet, wenn keine Neuronen mehr gefunden werden. */
  /* Die Neuronen werden in das Netz eingetragen. Die Variable     */
  /* last_aktiv_neuron zeigt auf das letzte Neuron der Liste.      */
  /* ------------------------------------------------------------- */
  train_net (type_neuron)
    int type_neuron;

    { int  i,corr,tempf;
      my_new_neuron  dummy_neuron;

      tempf = str_temp_net;          /* tempf wird als Temperatur verstanden */
      str_list();                  /* dynamische Liste der Traingswerte reset */
      for (i = last_aktiv_neuron; i < max_neuronen; i++)
        {  last_aktiv_neuron = i;
                          /* senkt solange tempf, bis Neuron gefunden wurde */
           do { corr = find_neuron(&x_ne t[i],type_neuron,tempf);
                if ( corr < 1 ) tempf = tempf slow_down;
              }
           while ( (corr < 1) && (tempf >= 0) );
           if ( corr < 1 ) break;            /* gefundenes Neuron optimieren */
           train_neuron (&x_net[i]);
           x_net[i].nr_in_net = i;
           /* nach Wunsch Kontrollmeldungen ausgeben */
           if (show1) print_neuron (&x_net[i]);
           if (show) printf (" %d----------------------------\n",i);
        }
    }

/* ------------------------------------------------------------- */
/* Uebergebenes Neuron optimieren                                */
/* Die Funktion correlation bewertet das Neuron. Durch die Funktion */
/* gradient_neuron wird der gradient dieser Correlation fuer jede */
/* Komponente berechnet. Wird das Neuron durch einen solchen     */
/* Gradientenabstieg verbessert, wird dieser uebernommen         */
/* ------------------------------------------------------------- */
int  train_neuron(neuron)
  my_new_neuron   *neuron;
```

```
    { int corr,corr_alt,j,i,moves;
      corr = correlation(neuron);
      for (j = 0 ; j < 10000 ; j++ )
        { corr_alt = corr;
          moves = 0;
          permutat_dim(-1); /* durch diese Funktion wird die Zahlenfolge */
            /* 1..dimension statistisch umsortiert. Durch argument (-1) */
            /* wird eine neue Folge erzeugt. Argument (i) uebergibt     */
            /* jeweils einen Eintrag */

          for( i=0;i<dimension;i++)   /* kann Neuron mit Gradient verbessert */
            { corr = gradient_neuron(                                /* werden */
              neuron,permutat_dim(i),neuron->temp,corr,&moves
                                    );
              if (corr_alt < corr) break;
             }
          if ( i == dimension + 1 )          /* wenn keine Verbesserung, dann
                                                globale Temperatur senken */
            { neuron->temp = neuron->temp slow_down2;
            }
            if (neuron->temp <1) break;
          }
      return (corr);
  }

/* ------------------------------------------------------------------ */
/* suche ein Neuron der Breite width vom Typ type                     */
/* Fuer diese Suche dienen die Trainigsdaten als Kristallisations-    */
/* keime. Das bedeutet, die Suche nach einem solchen Neuron beginnt   */
/* bei einem Wert aus der Liste der Trainingswerte.                   */
/* Hierdurch wird der Suchvorgang erheblich beschleunigt              */
/* ------------------------------------------------------------------ */
int find_neuron (n1,type,width)
  my_new_neuron  *n1;
  int width;
  int type;

    { int corr;
      int j,nnalt;
      my_new_neuron ntest;
      point_of_world p;

      corr = -1;
      new_neuron(&ntest,type);
      if (tst_list() ==0) str_list();          /* wenn letztes Element in */
           /* der Liste der Trainingswerte errechnet, dann wieder von vorne */
      switch (ntest.type)
        case aktiv_neuron :                     /* aktives Neuron suchen */
```

```
    { while( (tst_list() != 0) && ( corr < 1) )
      { read_list(&p);
        if (p.wert == fuzzy) p.wert = 1;
             /* Wert aus der Trainigsliste suchen, der noch nicht durch
                                                    das Netz erkannt wird */
                          /* Aus dem Traininigswert ein Neuron erzeugen */
        if ( (p.wert > 0) && (ask_net(&p,type) == 0)  )
        { for ( j = 0 ; j < dimension ; j ++)
          { ntest.center[j] = p.xy[j];
            ntest.width[j] = width;
            if(( p.xy[j]- width) < 0)
              ntest.width[j] = p.xy[j];
            if(( p.xy[j]+ width) > space )
              ntest.width[j] = space - p.xy[j] ;
          }
        ntest.aktiv = 1;
        ntest.nr_in_net = -1;                    /* noch nicht im Netz */
        corr =  correlation_2(&ntest);         /* berechne vereinfachte */
                               /* correlation fuer das gefundene Neuron */
        if (corr > 0) break;    /* corr >0 bedeutet dass dieses neuron  */
                                /* moeglich ist                         */
      }
    } /* while */
  } break; /* case */
  case invers_neuron :                              /* siehe aktiv_neuron */
    { while(tst_list() != 0)
      { read_list(&p);
        if (p.wert == fuzzy) p.wert = 0;
        if ( (p.wert == 0)   && (ask_net(&p,type) == 0)  )
          { for ( j = 0 ; j < dimension ; j ++)
            { ntest.center[j] = p.xy[j];
              ntest.width[j] = width;
              if(( p.xy[j]- width) < 0)
              ntest.width[j] = p.xy[j];
              if(( p.xy[j]+ width) > space )
              ntest.width[j] = space - p.xy[j] ;
            }
            ntest.aktiv = 1;
            ntest.nr_in_net = -1;                /* noch nicht im Netz */
            corr =  correlation_2(&ntest);
            if (corr > 0) break;
          }
        }   /* while */
    } break; /* case */
case fuzzy_neuron :                               /* siehe aktiv Neuron */
  { while(tst_list() != 0)
    { read_list(&p);
      if ( (p.wert == fuzzy) && (ask_net(&p,type) == 0) )
```

```
                { for ( j = 0 ; j < dimension ; j ++)
                  { ntest.center[j] = p.xy[j];
                    ntest.width[j] = width;
                    if(( p.xy[j]- width) < 0)
                      ntest.width[j] = p.xy[j];
                    if(( p.xy[j]+ width) > space )
                      ntest.width[j] = space - p.xy[j] ;
                  }
                  ntest.aktiv = 1;
                  ntest.nr_in_net = -1;                    /* noch nicht im Netz */
                  corr =  correlation_2(&ntest);
                  if (corr > 0) break;
                }
              }   /* while */
            } /* break; */
      } /* switch */

      if (corr > 0)                        /* gefundenes Neuron in Netz einfuegen */
        { nnalt = n1->nr_in_net;
          copy_neuron (&ntest ,n1);
          n1->nr_in_net = nnalt;
        }
      return (corr);
    }

/* ------------------------------------------------------------- */
/* Berechnet den Differentialquotienten des Neurons n1 und der   */
/* Komponente komp fuer beide Seiten des Neurons.                */
/* Die beiden Seiten des Neurons werden um temp1-2 veraendert    */
/* Diese Veraenderung wird bewertet und zurueckgegeben           */
/* Ist die Veraenderung illegal, so wird ein negativer Wert      */
/* zurueckgegeben, um sich so eine Fehlerbehandlung in der       */
/* Optimierungsstrategie zu ersparen                             */
/* ------------------------------------------------------------- */
int diff_neuron (n1,corr,komp,temp1,temp2)
  my_new_neuron  *n1;
  int komp,corr;
  int temp1,temp2;

  { int new_corr;
    int i,err;
    my_new_neuron n2;
    i = komp;
    err = 0;
    copy_neuron (n1,&n2);                  /* mit Copy des Neurons arbeiten */

        /* Neuron veraendern und pruefen, ob es noch im Wertebereich liegt */
    n2.center[komp] = n1->center[komp] + temp1;
```

```
    if ((n2.center[komp] < 0) ||(n2.center[komp] > space))
                              { return (corr-30000); }
    n2.width[komp] = n1->width[komp] + temp2;
    if ((n2.width[komp] < 0) ||(n2.width[komp] > max_width))
                              { return (corr-30000); };
    if ( ( (n2.center[komp] + n2.width[komp]) > space)
                  ||( (n2.center[komp] - n2.width[komp]) < 0 )  )
                              { return (corr-30000); }

     new_corr = correlation(&n2);    /* veraendertes Neuron bewerten */
     if (new_corr > corr)            /* wenn Verbesserung, dann neues Neuron */
       { n2.temp = n1->temp;         /* uebergeben */
         copy_neuron (&n2,n1);       /* sonst altes */
         corr = new_corr;
       };
     return (corr);
  }


/* ------------------------------------------------------------------ */
/* Das Neuron wird nach angegebener Strategie veraendert. Diese       */
/* Aenderungen werden bewertet und zurueckgegeben                     */
/* ------------------------------------------------------------------ */
int vari_neuron (neurox,komp,temp,corr,stratigy)
  my_new_neuron  *neurox;
  int temp;
  int komp;
  int stratigy;
  int corr;

  { switch ( stratigy )
    { case 0:                        /* Schaltbereich nach rechts verschieben */
        return(diff_neuron(neurox,corr,komp,temp/2,temp/2)); break;
      case 1:                         /* Schaltbereich nach links verschieben */
        return(diff_neuron(neurox,corr,komp,-1*(temp/2),temp/2)); break;
      case 2:              /* Schaltbereich bei gleichem Centrum vergroessern */
        return(diff_neuron(neurox,corr,komp,0,temp/2)); break;
    }
  }

/* ------------------------------------------------------------------ */
/* Diese Funktion erzeugt einen Vektor (moves), der in die Richtung   */
/* der groessten positiven Aenderung zeigt. moves ist also aehnlich   */
/* dem Gradientenvektor. Abgeleitet wird hier nach                    */
/*          d(Correlation)                                            */
/*          -------------   = moves                                   */
/*          d(Streategie)                                             */
/* ist moves = 0, so konnte keine positive Veraenderung gefunden      */
```

```
/* werden.                                                              */
/* -------------------------------------------------------------------- */
int gradient_neuron(neurox,komp,temp,corr,moves)
  my_new_neuron  *neurox;                          /* untersuchtes Neuron */
  int temp;                                      /* Groesse der Veraenderung */
  int komp;                                       /* untersuchte Eigenschaft */
  int *moves;                                  /* zurueckgegebene Optimierung */
  int corr;                 /* alte correletion, um Verbesserung zu erkennen */
  { my_new_neuron n[nr_of_strategy];
    int cor;
    int  i,max;

    max = -1;
    permutat_stra(0);          /* erzeugt zufaellige Folge von 1..strategie */
    for ( i = 0 ; i < nr_of_strategy ; i ++ )
      { copy_neuron(neurox,&n[i]);
        cor = vari_neuron( &n[i],komp,temp,corr,permutat_stra(i) );
        if (corr < cor) { max = i;corr = cor;break; };
      }        /* um Rechenzeit zu optimieren, werden nicht alle Strategien */
               /* untersucht, sondern bei der ersten Verbesserung wird      */
               /* Optimierung beendet. Das fuehrt schneller zum gleichen    */
               /* Ergebnis */                                                */
    if (max > -1)
      { copy_neuron( &n[max],neurox);corr = cor; (*moves)++;
      };
    return (corr);
}

/* -------------------------------------------------------------------- */
/* Bewertet Neuron mit Hilfe der Trainingsliste. Wird durch das Neuron  */
/* ein Wert aus der Liste falsch klassifiziert, so wird ein negativer   */
/* Wert zurueckgegeben. In eine positive Bewertung gehen die Anzahl der */
/* richtig klassifizierten Trainigswerte und Schaltgroesse des Neurones */
/* positiv ein. Die Schaltgroesse wird nur bis zu einer Untergranze     */
/* beruecksichtigt (s. Text )                                           */
/* -------------------------------------------------------------------- */
int   correlation (neuro)
  my_new_neuron *neuro;

  { int i,j,respon;
    int  counts,net_counts,in_net,flach;
    point_of_world        x;
    sample *merker;

    merker = xwert;
    counts = 0; in_net = 0 ; flach = 0;
    str_list();
    switch (neuro->type)    /* verschiedene Neuronentypen getrennt bewerten */
```

```
{ case aktiv_neuron :
  { while (tst_list() != 0 )
    { in_net = 0;
      read_list(&x);                    /* Wert aus dynamischer Liste holen */
      if (x.wert == fuzzy) { x.wert = 1; }
      respon = neuron(neuro,&x);
      if  (x.wert == 0)
        { if (respon == 1) counts =- 30000;
        }
      else if (respon == 1)              /* richtige Klassifizierung   */
        counts = counts + 1;
    } /* while */
    flach = 0;                          /* Flaechenbewertungskriterium */
    for(i=0; i<dimension;i++)
      { flach = flach + neuro->width[i];
        if ( neuro->width[i] <= 1)                    /* Untergrenze */
          { flach = 0; break;
          }
      };
   } break; /* case */
 case invers_neuron :
   { while (tst_list() != 0 )
     { in_net = 0;
       read_list(&x);                   /* Wert aus dynamischer Liste holen */
       if (x.wert == fuzzy) { x.wert = 0; }
       respon = neuron(neuro,&x);
       if  (x.wert == 1)
         { if (respon == 1) counts =- 30000;
         }
       else if (respon == 1)            /* richtige Klassifizierung   */
         { counts = counts + 1;
         }
     } /* while */
     flach = 0;                         /* Flaechenbewertungskriterium */
     for(i=0; i<dimension;i++)
       { flach = flach + neuro->width[i];
         if ( neuro->width[i] <= 1)                   /* Untergrenze */
           { flach = 0; break;
           }
       };
  } break; /* case */
case fuzzy_neuron :
  { while (tst_list() != 0 )
    { in_net = 0;
      read_list(&x);                    /* Wert aus dynamischer Liste holen */
      respon = neuron(neuro,&x);
      if  (x.wert != fuzzy)
        { if (respon == 1) counts =- 30000;
```

```
                }
              else if (respon == 1)          /* richtige Klassifizierung   */
                { counts = counts + 1; }
            } /* while */
            flach = 0;
            for(i=0; i<dimension;i++)       /* Flaechenbewertungskriterium */
              { flach = flach + neuro->width[i];
                if ( neuro->width[i] <= 1)                  /* Untergrenze */
                  { flach = 0; break;
                  }
              };
           } break; /* case */
      } /* swtich */
      xwert =  merker;
      if ( counts < 0 ) return counts;
      return (counts * space + flach);
    }

/* ------------------------------------------------------------------- */
/* Vereinfacht Funktion correlation. Bei dieser Funktion wird ein Neuron  */
/* nur auf Widerspruch zu den Trainingsdaten geprueft. Dieses             */
/* vereinfacht die Pruefung erheblich.                                    */
/* ------------------------------------------------------------------- */
int correlation_2 (neuro)
  my_new_neuron *neuro;

  { int   i,j,respon;
    int   counts;
    point_of_world   x;
    sample *merker;

    merker = xwert;
    counts = 1;
    str_list();
    switch (neuro->type)
      case  aktiv_neuron:
        { while (tst_list() != 0)
          { read_list(&x);              /* Wert aus dynamischer Liste holen */
            if (x.wert == fuzzy) { x.wert = 1; }
              respon = neuron(neuro,&x);
            if ((x.wert == 0) && (respon == 1)){counts =0 ;break;}
            else if (respon == 1)
              { for (j = 0; j < last_aktiv_neuron; j++)
                { if ( (neuron(&x_net[j],&x) == 1)
                         &&
                       (x_net[j].type == neuro->type) )
                  { counts = 0 ; break; }                   /* Widerspruch */
                }
```

```
                    if (counts == 0 ) break;
                  }
              }   /* while */
            } break; /* case */
          case  invers_neuron:
            { while (tst_list() != 0)
              { read_list(&x);                    /* Wert aus dynamischer Liste holen */
                if (x.wert == fuzzy) { x.wert = 0; }
                respon = neuron(neuro,&x);
                if ((x.wert == 1) && (respon == 1)) { counts =0 ; break;}
                else if (respon == 1)
                  { for (j = 0; j <= last_aktiv_neuron; j++)
                    { if ( (neuron(&x_net[j],&x) == 1)
                            &&
                           (x_net[j].type ==neuro->type) )
                      { counts = 0 ; break; }                    /* Widerspruch */
                    }
                      if (counts == 0 ) break;
                  }
              }   /* while */
            } break; /* case */
          case  fuzzy_neuron:
            { while (tst_list() != 0)
              { read_list(&x);
                respon = neuron(neuro,&x);
                if ((x.wert != fuzzy) && (respon == 1))
                  { counts =0 ; break;
                  }
                else if (respon == 1)
                  { for (j = 0; j <= last_aktiv_neuron; j++)
                    { if ((neuron(&x_net[j],&x) == 1)
                            &&
                           (x_net[j].type ==neuro->type))
                      { counts = 0; break;
                      }
                    }
                    if (counts == 0 ) break;
                  }
              }   /* while */
            } break; /* case */
          }  /* switch */
          xwert = merker;
          return (counts);
      }
/* ------------------------------------------------------------ */
```

A.4 Der L2 Cocktail

```
*********************************************
* Guide for using N-tuples for trigger studies *
*********************************************

      Author:     Stephan Egli       8.3.92
```

Standard N-tuple files (both for LOOK and PAW) have been created for various files containing background and physics events. The n-tuple has two parts: the first part contains trigger quantities and the second part quantities derived from the generator output (if available). Each N-tuple row is one event.

Nr	Name	Comment
1	'Event_No'	Useful for event display of selected events
2	'Run_No '	see above
3	'Weight '	use it to fill your histograms for weighted events
4	'ZVTX_max'	# of entries in peak bin of z-vertex histogram
5	'ZVTX_sum'	total # of entries in z-vertex histogram
6	'ZVTX_qua'	=1 if no entries outside window of 3 neighboured bins otherwise 0 (also 1 for empty histogram)
7	'ZVTXsig1'	"old" ZVTX significance: (MAX-BGR)/SQRT(MAX) where BGR=(SUM-MAX)/15
8	'ZVTXsig2'	"new" ZVTX significance SQRT(MAX-2.5)/(BGR+1./15.)
9	'Ncoinc_i'	total # of pad-coincidences in 16 phi sectors for inner MWPCs, only last 20 pads at -z side counted
10	'Ncoinc_o'	same for outer MWPCs, only -z side counted (pad 0..9)
11	'N_bigray'	# of backward going bigrays (ITHETA>=11)
12	'Peak_pos'	position of peak in ZVTX histogram (0..15, -1 if no entries)
13	'FW_rays '	forward ray multiplicity
14	'DC_trks '	# of tracks found by level 1 DC-rphi trigger
15	'DC_lopos'	# of positive charged tracks in low pt range (< ca. 800MeV)
16	'Eele_tag'	energy deposited in e-tagger
17	'Epho_tag'	energy deposited in photon tagger
18	'Veto_tag'	=1 if veto counter in front of photon tagger was on
19	'Nbpc1hit'	# of BPC planes with at least 1 hit
20	'Nbpc2hit'	# of BPC planes with at least 2 hits

```
                  For all BEMC quantities: triangular stacks ignored
 21  'EtotBemc'   total energy in BEMC (sum over stacks with E>800MeV)
 22  'EcluBemc'   total cluster energy , cluster initiator E>4GeV
 23  'NcluBemc'   # of clusters in BEMC, cluster initiator E>4 GeV
 24  'EmaxBemc'   energy in hottest BEMC stack

 25  'Veto_Tof'   =1 if coincidence between first and second TOF wall in
                  background timewindow, TOF planes are ored (no geometrical
                  correlation !)
 26  'Posi_Tof'   same for physics timewindow

 27  'Et      '   Total transverse energy derived from LAr BT 0..13
 28  'Etmiss  '   dito for missing transverse energy
 29  'Etweight'   theta weighted Et, weighting function see below
 30  'Etot    '   E_PLUG + E_IF + E_FB + E_CB + E_BEMC = total energy
 31  'E_CB    '   central barrel energy          (big tower 9..13)
 32  'E_FB    '   forward barrel energy          (big tower 6..8)
 33  'E_BEMC  '   BEMC energy                    (big tower 14..15)
 34  'E_IF    '   inner forward part energy      (big tower 0..5)
 35  'E_PLUG  '   plug energy                    (big tower 16)
 36  'E_FORW  '   E_PLUG + E_IF
 37  'E_BARR  '   E_FB + E_CB
 38  'E_BACK  '   E_FB + E_CB + E_BEMC
 39  'Elec_Lar'   =1 if at least one identified electron with medium thresh.
                  =2 if at least one identified electron with high   thresh.
 40  'Lar_t0  '   =1 if LAr t0 bit (no bigray validation) is set

 41  'Muon_BAR'   # of muon candidates (2 of 5 layers) in barrel region (c+f)
 42  'Muon_FWD'   # of muon candidates (3 of 5 layers) in forward endcap
 43  'Muon_BCK'   # of muon candidates (3 of 5 layers) in backward endcap
 44  'Muon_TOT'   total # of muon candidates
 45  'FWD_Muon'   not yet implemented
 46  'DC_hipos'   # of pos. charged tracks in high pt range (> ca.800MeV)
 47  'DC_hineg'   # of neg. charged tracks in high pt range (> ca.800MeV)
 48  'DC_loneg'   # of neg. charged tracks in low  pt range (< ca.800MeV)
 49  '        '
-----------------------------------------------------------------------
              generator quantities:

 50  'LOG_X   '   log10(XBjoerken)
 51  'LOG_Q2  '   log10(Q**2)
 52  'LOG_Y   '   log10(Y)
 53  'Pthat   '   transverse momentum in CM of hard collision (one parton)
 54  'LOG_Xres'   log10(momentum fraction of the parton on the lepton side
                   as used in the photon parton density function)
 55  'LOG_Xglu'   log10(momentum fraction of the parton on the proton side
                   as used in the proton parton density function)
```

```
 56  'PsRap_p1'   for photo production: pseudo rapidity of hard interaction
                                  parton being most in central region
                  for DIS: theta (rad) of electron
 57  'PsRap_p2'   for photo production: pseudo rapidity of hard interaction
                                  parton being less in central region
                  for DIS: theta (rad) of struck quark
 58  'Pt_part1'   for photo production: transv. momentum of hard interaction
                                  parton being most in central region
                  for DIS: total energy of electron
 59  'Pt_part2'   for photo production: transv. momentum of hard interaction
                                  parton being less in central region
                  for DIS: total energy of struck quark
 60  'Gen_Et  '   transverse momentum summed from charged and neutral
                  particles in range 5 deg < theta < 150 deg
 61  'Gen_Etwg'   dito but with theta weighting as described below
```

Weighting function for theta weighting used:

```
IF(THETA.LT.0.122173)THEN
  THWGHT=0.0
ELSEIF(THETA.LT.2.7)THEN
  THWGHT=1.0-EXP((0.122173-THETA)/0.279252)
ELSE
  THWGHT=3.7-THETA
ENDIF
```

Comments to simulation from 9.March 1992: (V01 files)

For all files the forward digi step was redone to take into account the list of dead FMWPC pads. Also all trigger modules were repeated with the most up-to-date version available at Feb. 92. At this stage noise was added for the LAr and BEMC triggers and a 2.5 sigma cut was applied. Note that the energy scales for the LAR trigger quantities are not yet determined correctly, therefore the threshold values applied e.g. for Et have no absolute meaning yet - the generated Et is usually roughly a factor 1.5 to 2 larger than what is derived from the trigger.

Available Source files:

```
H01EGL.TRIGGER     contains all jobs to produce Ntuples, steering of
                   trigger modules etc. There is also a LOOK-->PAW
                   conversion program in member #CONPAW
H01EGL.PHAN.CMZ    contains the routines to extract all the information
```

from the various trigger banks

Available N-tuple files:

For LOOK Ntuples replace "..." by "TRIGGER.LOOK" .
For PAW Ntuples replace "..." by "TRIGGER.PAW" .

N-tuple file	# of events	derived from file
Background files:		
H01EGL....BEAMGAS1.V01	1369	F11CAR.BEAMGAS.SPECIAL.CENT.TO
H01EGL... BEAMGAS2.V01	6216	H01EGL.BEAMGAS.H1SIM.S01TOS07.TOBIT
H01EGL....BEAMGAS3.V01	9872	F11CAR.BEAMGAS.TOTAL.CENT.TO.OUT01
H01EGL....BEAMWAL1.V01	17093	F11CAR.BEAMWAL.TOTAL.CENT.TO
H01EGL....BEAMWAL2.V01	4195	H01EGL.BEAMWALL.MODB.CENT.TO
Physics files:		
H01EGL....NC.MTB1.V01	3000	HERA02.LEPTO52.NC.MTB1.H1SIM001.DISK
H01EGL....CC.NORAD.V01	1000	HERA02.PYTHIA55.CC.DIS.NORAD.H1SIM001
H01EGL....JETJET.V01	2000	HERA02.PYTHIA56.JETJET.H1SIM01.DISK
H01EGL....CCBAR.V01	2000	HERA02.PYTHIA56.CCBAR.H1SIM001.DISK
H01EGL....JPSIDIFF.V01	7361	F11CAR.JPSIMU.SIM2060.D2101.NOC10K

B Begriffserklärung

Annealing: *engl. auskristallisieren. Beschreibt einen Vorgang, bei dem durch langsame Verkleinerung einer statistischen Variationsgröße (Temperatur) eine globale Optimierung durchgeführt wird. Der Vorgang lehnt sich an das Auskristallisieren einer übersättigten Lösung bei fallender Temperatur an.*

aktive Neuronen: *bezeichnet Neuronen, die auf Daten trainiert wurden, die zu einer positiven Klassifizierung gehören.*

assoziative Dimension: *bezeichnet die Anzahl der orthogonalen Neuronen, die zur minimalen Beschreibung einer Datenstruktur benötigt werden. Die orthogonalen Neuronen übernehmen die Funktion von Basisvektoren. Die assoziative Dimension ist ein Maß für den Grad der Ordnung einer Struktur.*

Backpropagation Algorithmus: *Standardverfahren zum Trainieren von Neuronalen Netzen nach Rumelhart 1986.*

Chromosomen: *Bestandteil bei der genetischen Codierung eines Algorithmus.*

Crossing over: *Vorgang bei dem Optimieren von genetischen Codes.*

DESY: *Deutsches Elektronen Synchrotron in Hamburg.*

Driftkammer: *Komponente zum nachweisen von Elementarteilchenspuren im Detektor.*

Feed forward Netz: *Neuronales Netz, in dem die Schichten nicht zurückgeführt werden.*

Fuzzy Neuron: *Neuronen, die auf Daten trainiert wurden, zu denen widersprüchliche Klassifizierungen vorliegen.*

Gen: *Bestandteil bei der genetischen Codierung eines Algorithmus.*

H1: *eines der beiden Experimente am HERA Speicherring.*

Hebb'sche Lernregel: *Von Hebb 1949 vorgeschlagener Mechanismus, bei dem Nervenzellen unabhängig von einander lernen.*

HERA: *Speicherring am DESY, an dem die Experimente H1 und Zeus durchgeführt werden.*

Inverses Neuron: *bezeichnet Neuronen, die auf Daten trainiert wurden, die zu einer negativen Klassifizierung gehören.*

Kalorimeter: *Detektor, um die kinetische Energie von Elementarteilchen zu messen.*

Kristallisationsalgorithmus: *s. Annealing*

L1-L4 Trigger: *Bezeichnet die vier Ebenen des vierstufigen Triggers.*

L2 Cocktail: *Auswahl aus den allen dem Trigger zur Verfügung stehenden Daten, die für die L2 Entscheidung ausreichen ist.*

orthogonale Neuronen: *Neuronen, deren Gewichtsvektoren orthogonal zu einender stehen.*

Perzepton Modell: *einfaches Modell eines Neuronalen Netzes.*

Sättigungspunkt: *Punkt, ab dem das trainieren mit zusätzlichen Daten ein Netz nicht mehr verbessert.*

Sigmafunktion: *Schwellwertfunktion.*

Temperatur: *Parameter beim Kristallisationsalgorithmus.*

Time and Space Map (TSMap): *Datenformat, um die Triggerdaten aus dem Detektor auszulesen.*

Xilink: *programmierbares Gatearray.*

Zeus: *eines der beiden Experimente am HERA Speicherring.*

Mathematische Abkürzungen

d: Abstand zwischen zwei Punkten bei vorhandener Metrik.

E: Fehler bei der Anwendung eines Netzes auf unbekannten Daten. Die Zahl gibt die Anzahl der falsch erkannten Werte bei eine festen Anzahl von Referenzwerten an. (meist 10000)

g_i: Gewichtsvektor eines Neurons.

M: Quotient Wert/Neuron.

s: Schwellwert.

X_i: Eingangswert eines Neurones.

Y: Ausgangswert des Neurones

θ: Schwellwertfunktion.

C Literaturverzeichnis

1. Neuronale Netze / Helge Ritter, Thomas Martinetz, Klaus Schulten / Addison-Wesley Verlag / 1990

2. Continuous Input RAM-Based Stochastic Neural Model / Denise Gorse, John G. Taylor / Neural Networks, Vol4 / pp657-665 / 1991

3. Die Entdeckung des Chaos / John Briggs, F. David Peat / Hansa Verlag / 1990

4. Ein Backpropagtion Netz unter Turbo Pascal / Walter Kirchner / c't 11-90 /

5. Genetische Algorithmen / A. K. Dewdney / Computer Kurzweil 1 / Spektrum der Wissenschaft: Verständliche Forschung / 1988

6. Genetische Algorithmen / John H. Holland / Spektrum der Wissenschaft 9-92

7. Proposal for a Second Level Neural Network Trigger / J. Fent, A. Gruber, C. Kiesling, H. Oberlack, P. Ribarics / DESY 17. März 92

8. Kohonens Modell am Beispiel des auditiven Kortex der Fledermaus / Neuronale Netze / Helge Ritter, Thomas Martinetz, Klaus Schulten / Addison-Wesley Verlag / 1990

9. Complex Dynamics in Winner-Take-All Neural Nets With Slow Inhibition / Bard Ermentrout / Neuronal Networks / Vol5 / pp415-431 / 1992

10. On the Training of Radial Basis Function Classifiers / M. T. Musavi, W. Ahmed, K. H. Chan, K. B. Faris, D. M. Hummels / Neuronal Networks / Vol 4 / pp595-603 / 1992

11. Dynamic behaviour of Boolean networks / D.Martland / Neural Computing Architectures pp217-235 / North Oxford Academic / 1989

12. Das Hopfield-Modell / Helge Ritter, Thomas Martinetz, Klaus Schulten / Neuronale Netze / Addison-Wesly Verlag / 1990

13. Optical implementations / J. E. Midwinter and D. R. Selviah / Neural Computing Architectures pp268-278 / North Oxford Academic / 1989

14. Datasheet for Intelligent Pattern Recognition Memory Module (IPRMM) / Oxford Computer inc / August 1990

15. Datasheet The 80170NX Electrically Trainable Analog Neural Network (ETANN) Chip / Intel Corporation / 1991

16. The Progammable Gate Array Data Book / XILINX / 1991

D Schaltzeichnungen

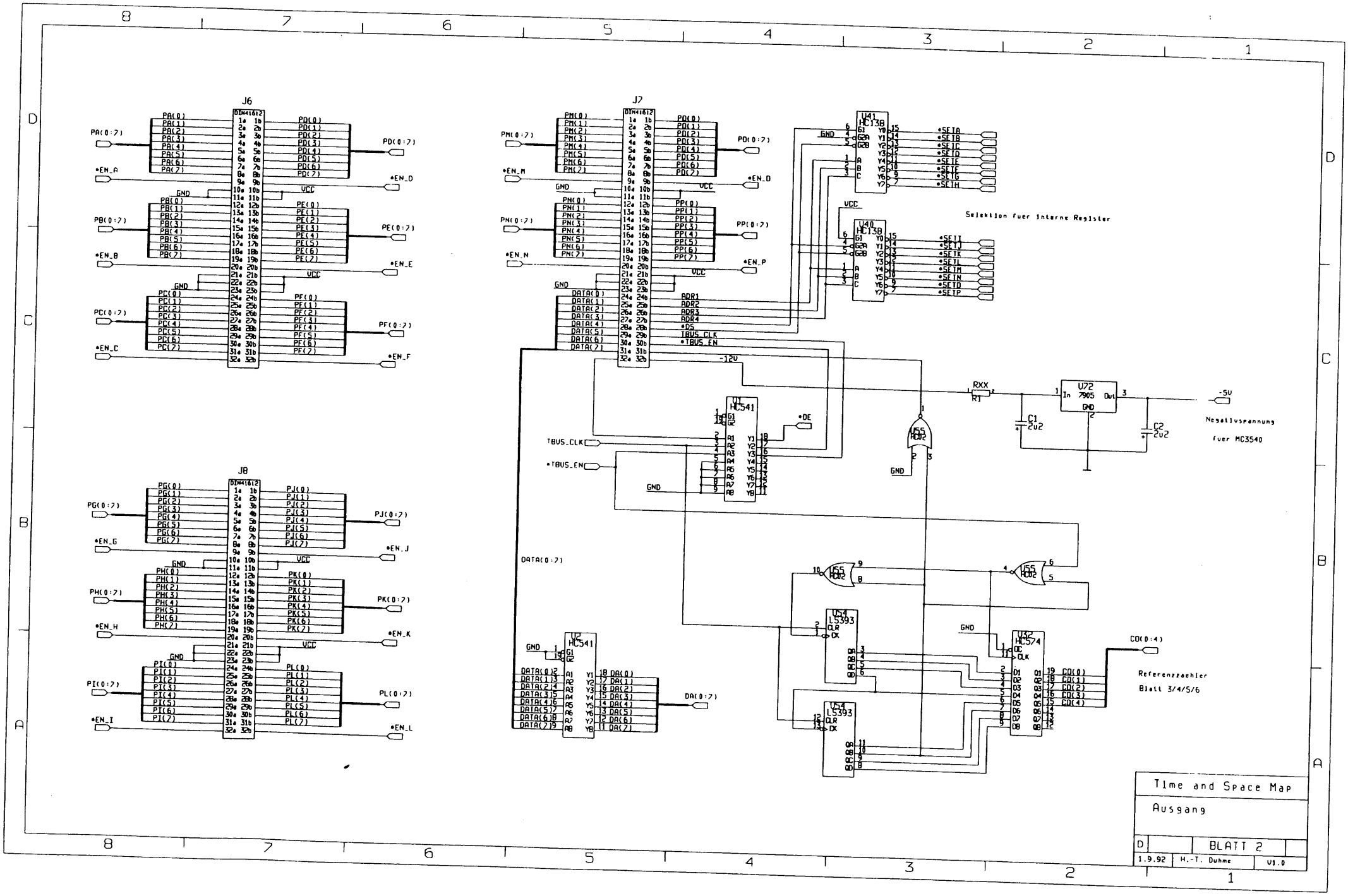
Schaltplan zur Time and Space Map

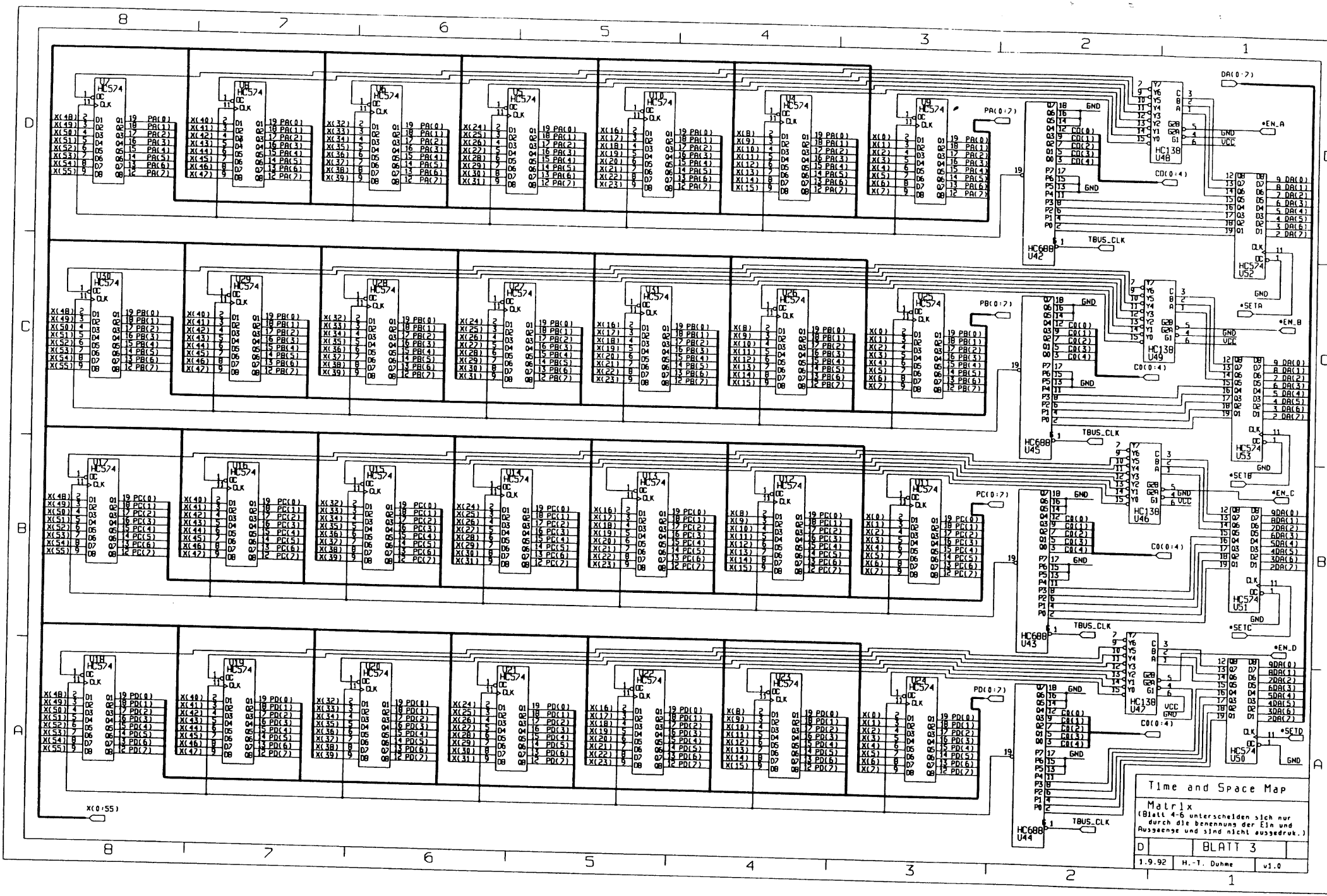
- Blatt 1, Eingangsinterface
- Blatt 2, Ausgänge
- Blatt 3 /4/5/6 , Matrix

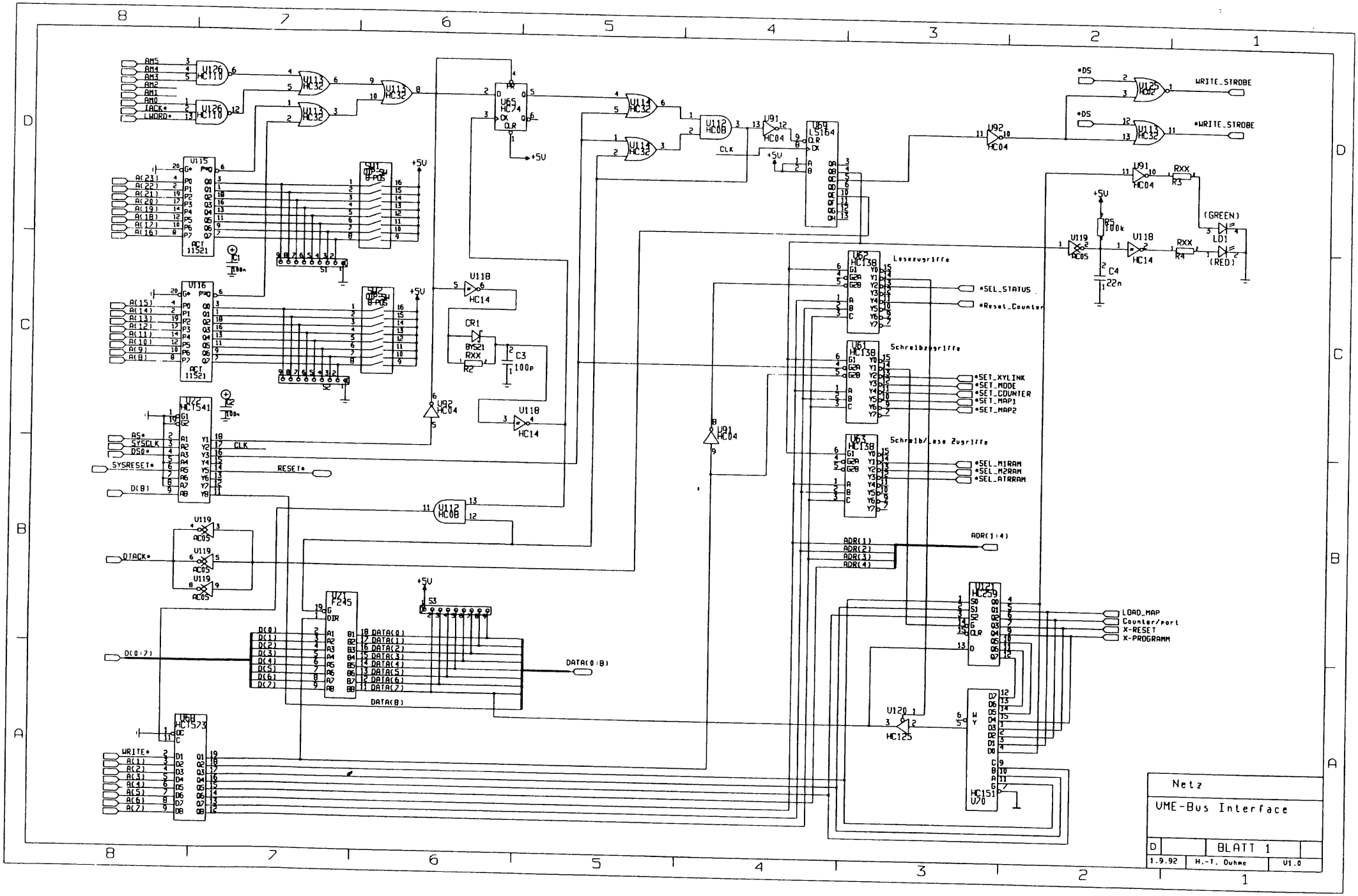
Schaltplan des Netzes

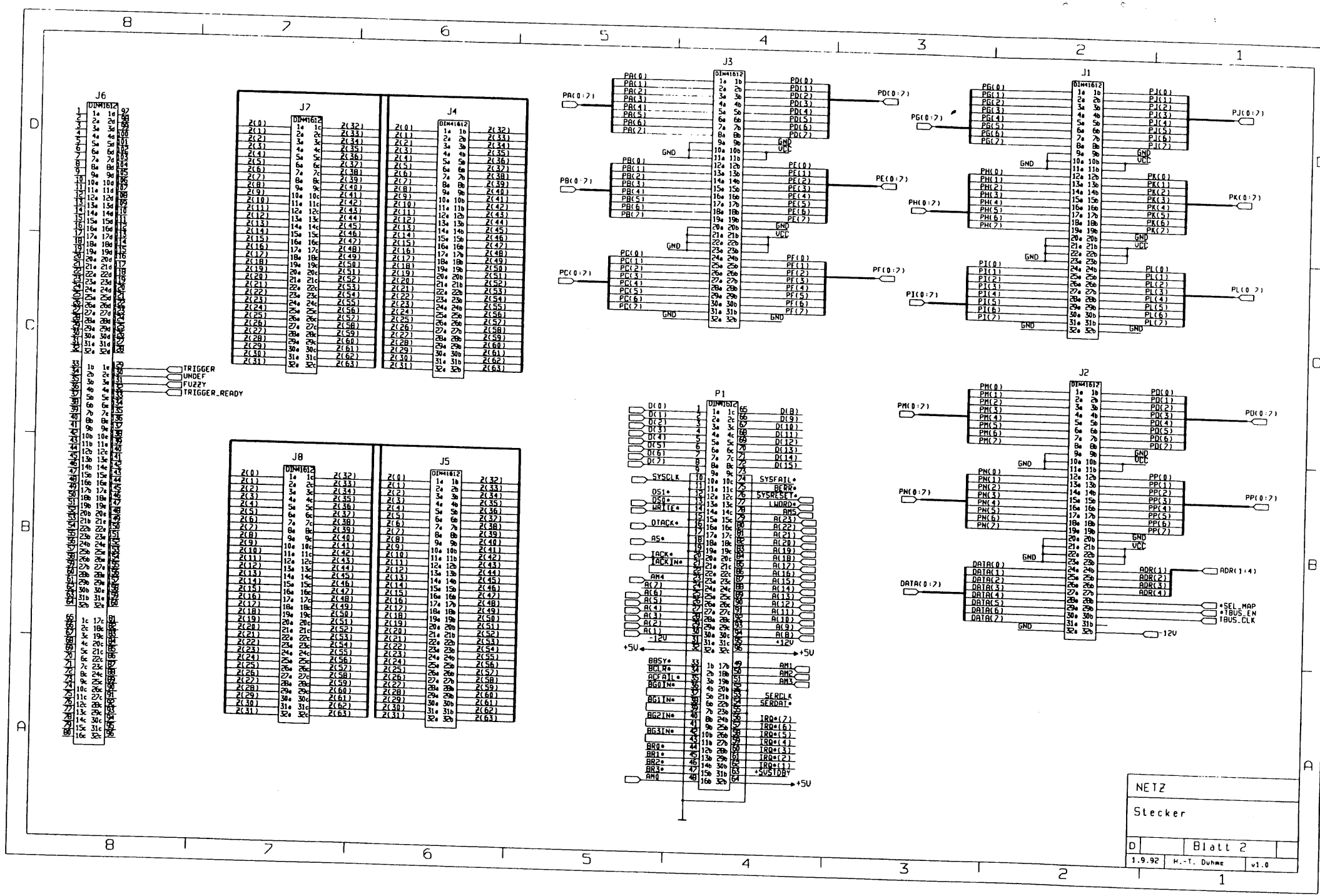
- Blatt 1, VME-Bus Interface
- Blatt 2, Stecker
- Blatt 3, Schicht 2
- Blatt 4 /5 Schicht 1

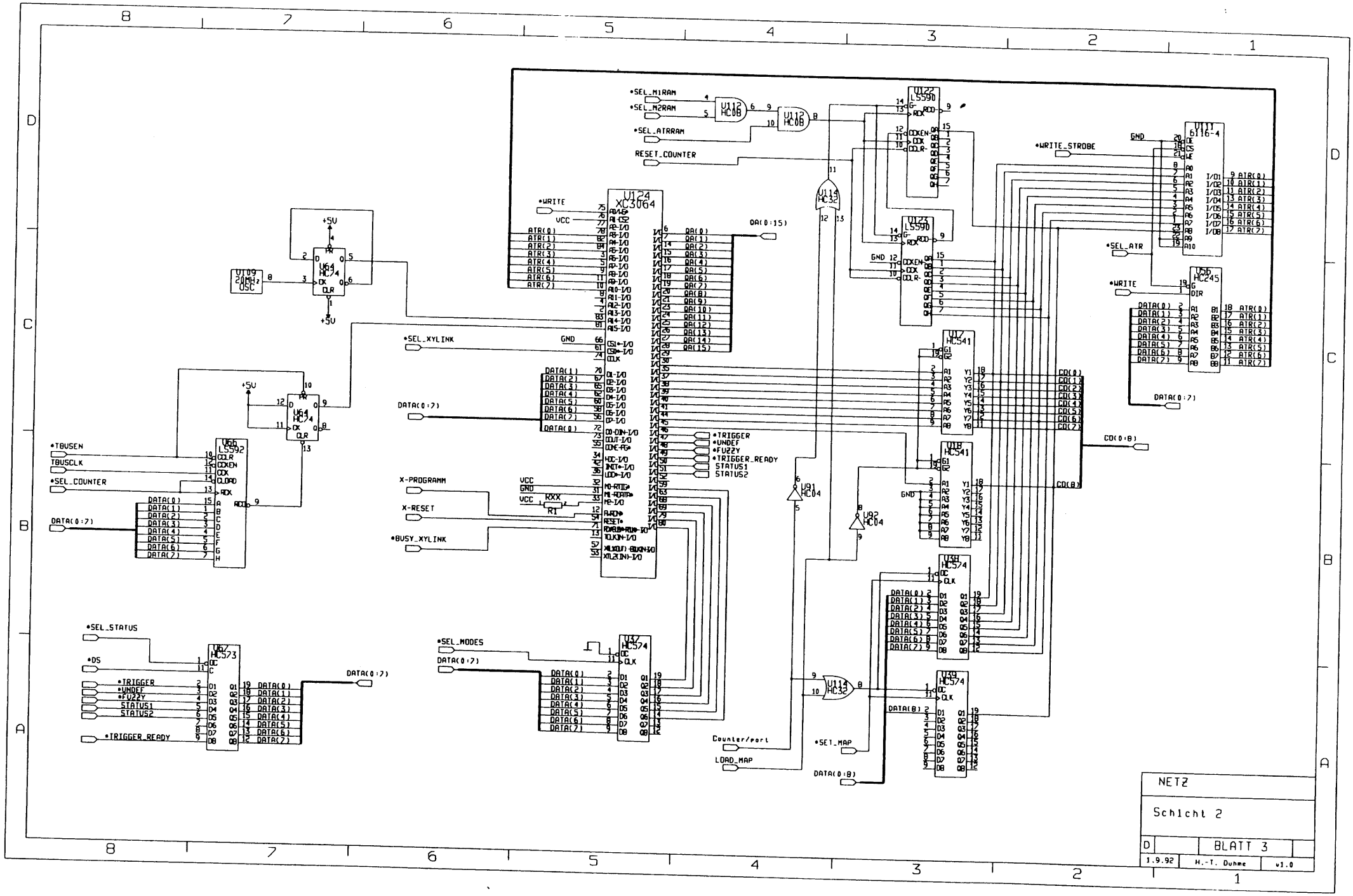

A(1:32)
J5
J4
X(0:55)
*TBUS_EN
TBUS.CLK
GND
VCC
Time and Space Map
Eingangsinterface
D
Y1 0000
BLATT 1
1.9.92
H.-T. Duhme FHIT
V1.0

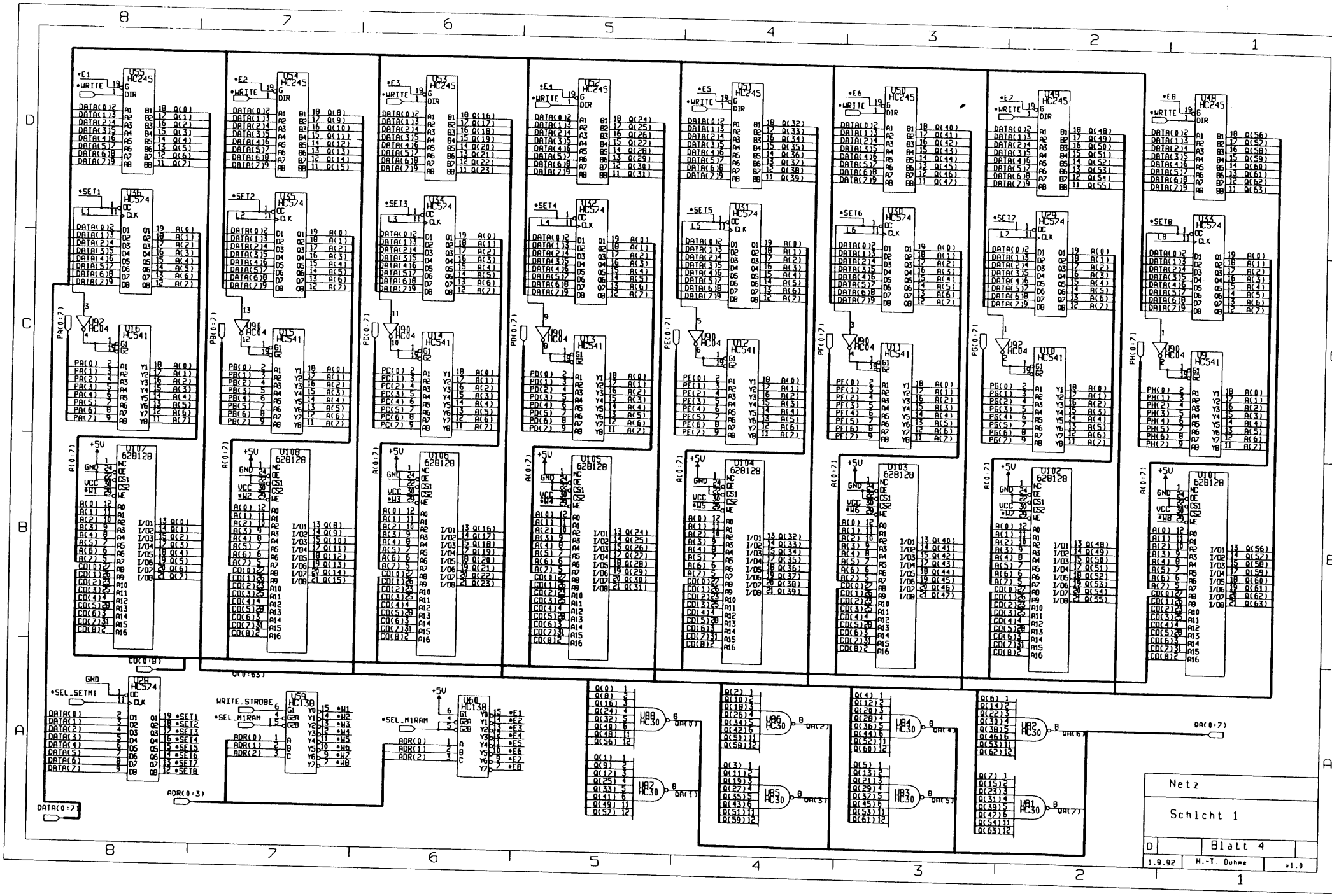

J6
J7
J8
PA(0:7)
PB(0:7)
PC(0:7)
PD(0:7)
PE(0:7)
PF(0:7)
*EN_A
*EN_B
*EN_C
*EN_D
*EN_E
*EN_F
PM(0:7)
PN(0:7)
PP(0:7)
*EN_M
*EN_N
*EN_P
DATA(0:7)
ADR1
ADR2
ADR3
ADR4
*DS
TBUS_CLK
*TBUS_EN
-12V
PG(0:7)
PH(0:7)
PI(0:7)
PJ(0:7)
PK(0:7)
PL(0:7)
*EN_G
*EN_H
*EN_I
*EN_J
*EN_K
*EN_L
GND
VCC
U41 HC138
U40 HC138
*SETA
*SETB
*SETC
*SETD
*SETE
*SETF
*SETG
*SETH
*SETI
*SETJ
*SETK
*SETL
*SETM
*SETN
*SETO
*SETP
Selektion fuer interne Register
U1 HC541
U2 HC541
*OE
U55 HC02
U54 LS393
U52 HC574
RXX
R1
U72 7905
C1 2u2
C2 2u2
-5V
Negativspannung
fuer MC3540
CO(0:4)
Referenzzaehler
Blatt 3/4/5/6
Time and Space Map
Ausgang
BLATT 2
1.9.92
H.-T. Duhme
V1.0


Time and Space Map
Matrix
(Blatt 4-6 unterscheiden sich nur durch die benennung der Ein und Ausgaenge und sind nicht ausgedruk.)
BLATT 3
1.9.92
H.-T. Duhme
v1.0
X(0:55)
DA(0:7)
PA(0:7)
PB(0:7)
PC(0:7)
PD(0:7)
CO(0:4)
TBUS_CLK
*EN_A
*EN_B
*EN_C
*EN_D
*SETA
*SETB
*SETC
*SETD
GND
VCC


Netz
VME-Bus Interface
D
BLATT 1
1.9.92
H.-T. Duhme
V1.0
WRITE_STROBE
*WRITE_STROBE
*SEL_STATUS
*Reset_Counter
*SET_XYLINK
*SET_MODE
*SET_COUNTER
*SET_MAP1
*SET_MAP2
*SEL_M1RAM
*SEL_M2RAM
*SEL_ATRRAM
ADR(1:4)
LOAD_MAP
Counter/Port
X-RESET
X-PROGRAMM
DATA(0:8)
RESET*
CLK
(GREEN)
(RED)
Lesezugriffe
Schreibzugriffe
Schreib/Lese Zugriffe


J6
J7
J4
J3
J1
J8
J5
P1
J2
TRIGGER
UNDEF
FUZZY
TRIGGER_READY
PA(0:7)
PB(0:7)
PC(0:7)
PD(0:7)
PE(0:7)
PF(0:7)
PG(0:7)
PH(0:7)
PI(0:7)
PJ(0:7)
PK(0:7)
PL(0:7)
PM(0:7)
PN(0:7)
PP(0:7)
DATA(0:7)
ADR(1:4)
*SEL_MAP
*TBUS_EN
TBUS_CLK
GND
VCC
+5V
-12V
NETZ
Stecker
D
Blatt 2
1.9.92
H.-T. Duhme
v1.0


NETZ
Schicht 2
BLATT 3
1.9.92
H.-T. Duhme
v1.0
U124 XC3064
U122 L5590
U123 L5590
U117 6116-4
U56 HC245
U7 HC541
U8 HC541
U55 L5592
U67 HC573
U57 HC574
U58 HC574
U59 HC574
U109 20MHz OSC
U64 HC74
U61 HC74
U112 HC08
U113 HC08
U114 HC32
U115 HC32
U91 HC04
U92 HC04
*SEL_M1RAM
*SEL_M2RAM
*SEL_ATRRAM
RESET_COUNTER
*WRITE_STROBE
*SEL_ATR
*WRITE
*SEL_XYLINK
DATA(0:7)
X-PROGRAMM
X-RESET
*BUSY_XYLINK
*TBUSEN
TBUSCLK
*SEL_COUNTER
*SEL_STATUS
*DS
*TRIGGER
*UNDEF
*FUZZY
STATUS1
STATUS2
*TRIGGER_READY
*SEL_MODES
Counter/port
LOAD_MAP
*SET_MAP
DATA(8)
CD(0:8)
ATR(0:7)
DA(0:15)


Netz
Schicht 1
Blatt 4
1.9.92
H.-T. Duhme
v1.0