

Selbstorganisierende Neuronale Netze auf Transputern

A. Ultsch, G. Guimaraes, D. Korus, H. Li
Universität Dortmund, FB Informatik, LS VI, *1)
Pf. 500 500, W-4600 Dortmund-50

Zusammenfassung

Künstliche Neuronale Netze (KNN), insbesondere Kohonen-Netze, sind zur Klassifizierung von Daten geeignet, da sie die topologischen Eigenschaften von Eingabedaten hoher Dimensionalität auf einen niedrigdimensionalen Raum (Ebene) abbilden können. Die Implementierung der Kohonen-Netze wird auf Transputern realisiert, da hier die inhärente Parallelität der Neuronen auf eine parallele Verarbeitung auf den Transputern fast vollständig übertragen werden kann. Demzufolge werden die Neuronen eines Gitters nach dem "interlaced"-Verfahren auf die Transputer verteilt. Dies führt zu einer geeigneten Auslastung der Transputer. Nach dem Anlernen werden die KNN mittels der U-Matrix-Methode visuell aufbereitet, um somit eine Klassifizierung vornehmen zu können.

Schlüsselworte: Neuronale Netze, Selbstorganisierende Merkmalskarten, Kohonen, Transputer, U-Matrix.

1. Einleitung

Künstliche Neuronale Netze (KNN) sind Rechnerarchitekturen, die sich an biologische Vorbilder, z.B. Neuronen im Gehirn, anlehnen. Eine interessante Eigenschaft natürlicher Neuronaler Netze ist die Bildung von rezeptiven Feldern in der Großhirnrinde, welche in ihrer Topologie die Struktur des Eingaberaumes (Sensorik) widerspiegeln. Dies läßt sich mit dem Modell von Kohonen [KOHONEN 89], auch Selbstorganisierende Merkmalskarte genannt, modellieren. Insbesondere wird die Eigenschaft dieser Merkmalskarten genutzt, einen Eingaberaum mit beliebig großer Dimensionalität auf einen niedrigdimensionalen Raum abzubilden. Mit einer von uns entwickelten Methode, der sog. U-Matrix-Methode lassen sich dabei Klassen in den Daten erkennen. Somit wird das Kohonen Modell zu einem neuronalen Klassifikator (siehe Kapitel 6.).

*1) ab 1.1.1993: Phillips Universität Marburg, FB Mathematik, FG Informatik; Hans-Meerwein-Straße, Lahnberge, W-3550 Marburg

2. Abbildung von Kohonen-Netzen auf Transputern

Neuronen sind hochgradig parallele Einheiten mit lokalen Informationen (im wesentlichen Synapsengewichte und Schwellenwert), einem hohen Vernetzungsgrad und i.a. einfachen Operationen (Skalarprodukt, Additionen, Multiplikationen). Transputer sind parallel verknüpfbare Prozessoren mit einer schnellen Floating Point Unit und lokalem Speicher. An Hardware für eine Verknüpfung bieten sie allerdings nur 4 Hardware-Kommunikationsverbindungen (Links). Eine vollständige Vernetzung läßt sich nur mittels virtueller Links realisieren. Eine 1-zu-1-Abbildung bietet sich aufgrund der hohen Leistungsfähigkeit und des Preises der Transputer nicht an, so daß bei der Abbildung eines Neuronalen Netzes auf ein Transputer-Netz einige Aspekte zu berücksichtigen sind.

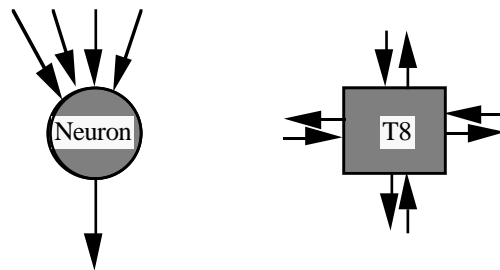


Abb. 1 Neuron versus Transputer

Beim Modell von Kohonen muß man zwischen dem biologischen Modell, bei dem sämtliche Neuronen miteinander verbunden sind und sich gegenseitig mit

Für jeden Lernschritt erledige:

1. Broadcasting des Lernvektors (Feuern des input-Neurons)
2. Berechnen des Abstands der Gewichte und des Lernvektors und Bestimmen des globalen Minimums
3. Broadcasting der Position des Best Matches
4. Adaption der Gewichte in der Nachbarschaftsumgebung

Abb. 2 Algorithmus (vereinfachte Darstellung) von Kohonen

tels lokaler Inhibition beeinflussen, und dem Computer-Modell (s. Abb. 3 u. Algorithmus in Abb 2.) unterscheiden [KOHONEN 89].

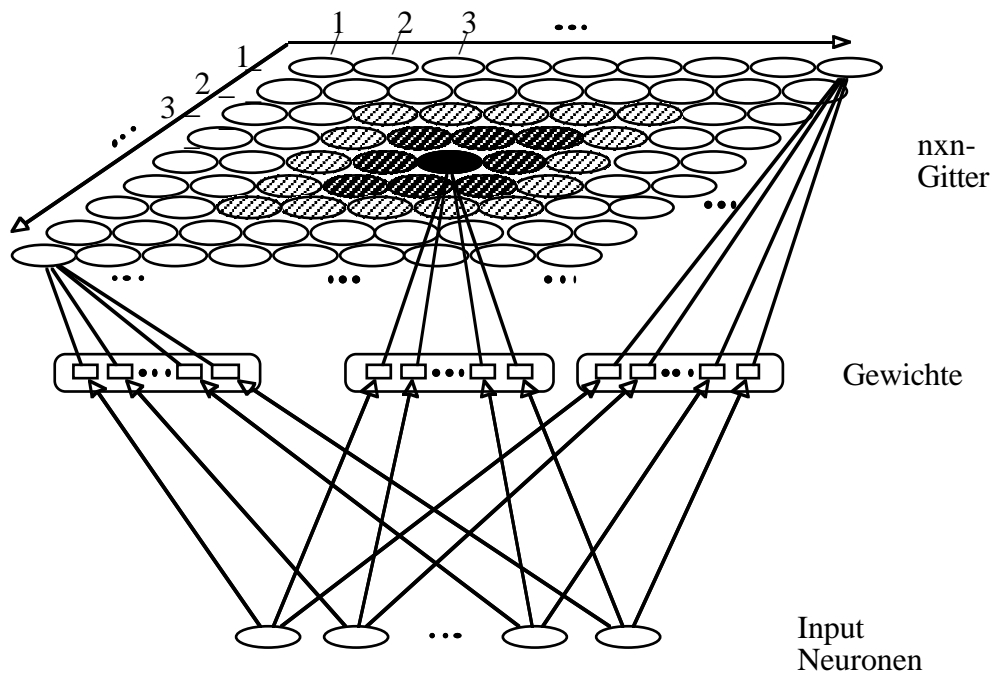


Abb. 3 Kohonen Netz

Die Lernvektoren und das Feuern der input-Neuronen lassen sich auf einem root-Transputer mit Anbindung zum Host - hier eine SUN-Sparcstation - implementieren. Die übrigen Transputer müssen mit dem root-Transputer so verknüpft sein, daß die Nachrichten (Broadcasting des Lernvektors und der Position des Best Matches) möglichst effizient an alle Neuronen verteilt werden. Untereinander kommunizieren die Neuronen um Gegensatz zum biologischen Modell nicht, mit Ausnahme zur Bestimmung des globalen Minimums, wozu z.B. eine Baum-Struktur genügt.

3. Implementierung von Kohonen-Netzen auf Transputern

Am einfachsten und bei uns z.Zt. realisiert ist die *Ringstruktur*. Das routing ist sehr einfach, da jeder Transputer genau einen Vorgänger und einen Nachfolger hat und die Kommunikation nur in eine Richtung erfolgt. Allerdings wächst der Kommunikationsaufwand linear mit der Anzahl der Transputer.

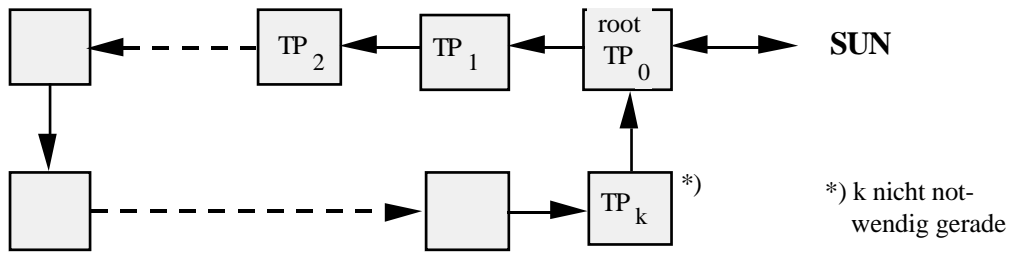


Abb. 4 Ringtopologie

Alternativ hierzu ließe sich eine *Baumstruktur* implementieren, wobei das routing etwas kompliziert ist: Jeder Transputer hat genau einen Vorgänger und 0 bis 3 Nachfolger und die Kommunikation erfolgt in beide Richtungen. Der Kommunikationsaufwand wächst im Verhältnis $O(\log_3 k)$. (s.a. [ELLINGER 92])

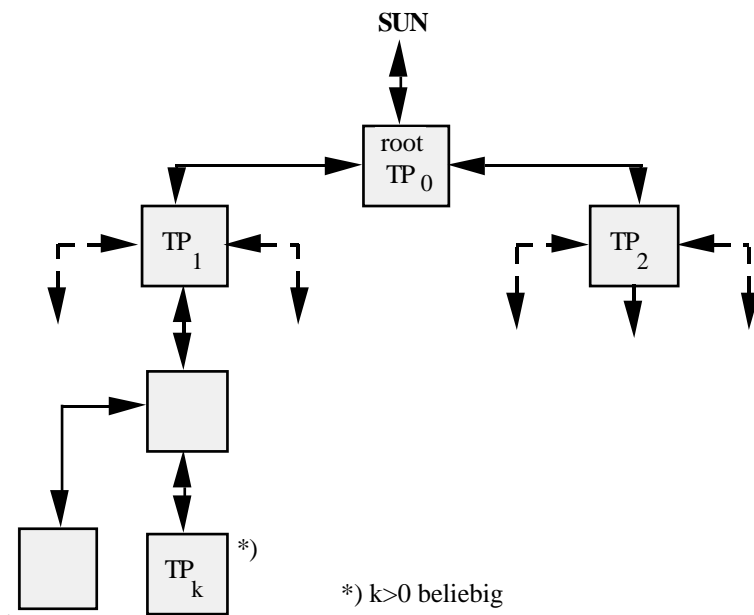


Abb. 5 Baumtopologie

Die Kommunikation auf dem Transputer-Netz ist mit dem Message-Passing-Konzept und dem varianten Kanalkonzept von Occam 2 realisiert. Da die Transputer in unserer Implementation zu einem Ring verknüpft sind, bietet sich eine Token-Ring-Strategie zum Propagieren von Nachrichten an.

Auf jedem Work-Transputer laufen zwei parallele Prozesse: der *Kommunikationsprozeß* (hohe Priorität), welcher das routing des aktuellen Transputers kon-

trolliert und Signale und Daten empfängt und weiterleitet, sowie der *Work-Prozeß* (niedrige Priorität), der vom Kommunikationsprozeß seine Handlungsanweisungen und evtl. Daten bekommt oder Daten an diesen zurückgibt, ansonsten aber die Prozeduren der Abstandsbestimmung und der Adaption der Gewichte enthält [SIEMON/ ULTSCH 90].

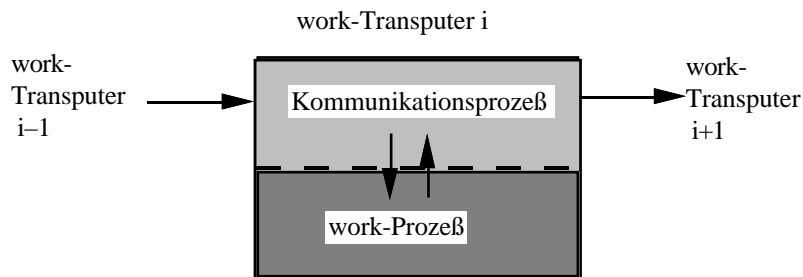


Abb. 6 Kommunikation

Eine typische Eigenschaft des Kohonen-Netzes ist es, nur in der Nachbarschaft des Best Matches zu lernen. Anfangs umfaßt diese Nachbarschaft zwar möglicherweise das ganze Netz, ist aber im weiteren Verlauf auf die nähere Umgebung des Best Matches beschränkt.

Würde man nun bei einem $n \times n$ -Neuronengitter und k Transputern, $n \geq k$, i.a. $n > k$, OE $n = c \cdot k$, $c \in \{2, 3, 4, \dots\}$ die ersten $c = n/k$ Zeilen auf den ersten Transputer, die nächsten c auf den zweiten Transputer u.s.w. verteilen, so würde bei einem Best Match in Gitterzeile j und einer fortgeschrittenen Nachbarschaftsweite $r \leq c$ nur der Transputer $(j \text{ div } c) + 1$ mit insgesamt $(2 \cdot r + 1)^2$ Updates beschäftigt sein, während sich die anderen im Wartezustand befinden. Eine bessere Auslastung der Transputer während der Adaption-Phase eines Lernschrittes erhält man, wenn man die Neuronen im *interlaced*-Verfahren auf die Transputer verteilt (s. Abb.7). Hierbei sind während der Adaption-Phase $2 \cdot r + 1$ Transputer, OE $2 \cdot r + 1 \leq k$, mit je $2 \cdot r + 1$ Updates beschäftigt. Eine feinere Verteilung der Neuronen, in dem man z.B. das erste Neuron auf den ersten Transputer abbildet u.s.f. bringt nichts, da hier $(c \in \{2, 3, 4, \dots\})$ z.B. nur Zeilen und Spalten vertauscht werden, dafür aber die Identifizierung der Neuronen ungleich mühsamer wird. Dies wurde auch schon bei einer ersten Implementierung berücksichtigt [SIEMON/ ULTSCH 90].

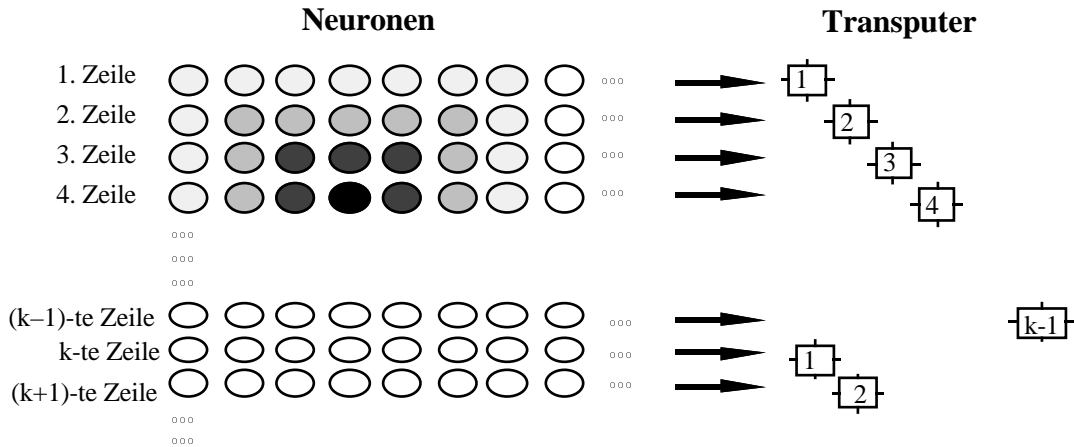


Abb. 7 Verteilung der Neuronen

4. Beobachten des Lernens bei Kohonen Netzen

Um Informationen über den Zustand des Neuronalen Netzes während der Lernphase zu bekommen, lassen wir das Netz nach jeder Lernepoche die Lage der Best Matches dieser Epoche und die Summe der Fehler ausgeben. Diese Werte werden grafisch angezeigt. Dabei gilt:

$$\text{Fehlersumme} = \sum_{i=1}^{\#\text{Lernvek}} \left| \text{Lernvek}_{\sigma(i)} - \text{BestMatch}(\text{Lernvek}_{\sigma(i)}) \right|_2$$

Auf dem Root-Transputer läuft dazu parallel ein Prozeß, der die Daten via stdout-Kanal an die SUN-Workstation weiterleitet. wo der aufrufende Prozeß - die X-Window-Oberfläche - einen Child-Prozeß startet, so daß die Daten in zwei X-Window-Fenstern kontinuierlich angezeigt werden. Wir beabsichtigen die Analyse des Neuronalen Netzes in der Lernphase soweit auszuweiten, daß wir die zugehörige U-Matrix während des Lernens beobachten können. Ebenso soll das lernen zu einem beliebigen Zeitpunkt unterbrochen werden können, daß das Netz in einem aussagekräftigen Zustand befindetet. Hierbei ist allerdings der Flaschenhals, d.h. die begrenzte Datenübertragungsrate, vom root-Transputer über den SBus-Adapter zur SUN-Grafikstation das größte (Hardware-) Problem.

5. Nachteile und mögliche Alternativen

Für den oben beschriebenen Kohonen-Algorithmus wäre optimal, daß man via Broadcasting vom root-Prozeß sämtliche Neuronen mit einem Mal benachrichti-

gen bzw. mit Daten versorgen kann. Dies ist bei Transputern aufgrund der Hardware-Architektur jedoch nicht möglich.

Eine Alternative hierzu wäre eine Architektur auf Basis von i860-Boards an, die unter verschiedenen Kommunikationsarchitekturen auch einen Broadcast-Mechanismus realisiert. In diesem Konzept können die Speicherbereiche der verschiedenen Boards in einem RAM-Fenster auf dem Host-Rechner eingeblendet werden (s. a. [KLEIN 92]). Das Parallelisierungskonzept beruht im Gegensatz zum eleganten Message-Passing auf dem LINDA-Konzept, das von D. Gelernter und N. Carriero [GELERNTER87] auf der Yale-Universität entwickelt wurde. Es arbeitet auf einem gemeinsamen "Datenpool", der aus beliebigen Tupeln besteht, und verwendet dort die Befehle `IN()` – Ablegen eines Tupels im Tupelraum –, `READ()` – Lesen eines Tupels des Tupelraums – und `OUT()` – Entfernen eines Tupels aus dem Tupelraum.

Da die Hardware jedoch schneller, billiger und massiv paralleler wird, spielen obige Nachteile vermutlich in Zukunft keine so große Rolle mehr. Eher fehlt es an verfügbaren Programmiersprachen zur Implementation Neuronaler Netze auf massiv parallelen Systemen. Wichtig wären dabei Sprachkonstrukte die weg von der Adreß-Programmierung eines C gehen und dynamischere Möglichkeiten als Occam 2 bieten. Letzteres bedeutet vor allem dynamische Zeiger, Listen und Prozeduren, rekursive und lokale Prozeduren, Records, Datenkapselung, aber auch dynamische Prozesse und vor allem dynamische Prozeßallokation. Das alles dazu noch mit einer sauberen, seiteneffektfreien Typbindung. Von der Sprache Ada z.B. ließe sich da sicherlich einiges lernen, objekt-orientierte Ansätze müssen zusätzlich übernommen werden.

6. Neuronaler Klassifikator

Die in den vorherigen Kapiteln beschriebene Realisierung der Kohonen Netze auf den Transputern sind nur ein Teil eines bereits implementierten Gesamtsystems, den Neuronalen Klassifikator. Hier sind (siehe Abb.8) alle Schritte die zur Klassifizierung der Rohdaten enthalten. Bevor die Daten im Neuronalen Netz angelern werden können, ist zunächst eine Vorverarbeitung der Daten notwendig. Hier werden Ausreißer durch statistische Methoden und korrelierte Komponenten durch Scatter-Plots erkannt und eliminiert. Weiterhin werden alle Komponenten auf einen Wertebereich normiert, da sonst das Neuronale Netz eine unterschiedliche Gewichtung der Komponenten vornehmen würde. Anschließend werden die Daten angelern, wobei die Beobachtung dieses

Vorgangs möglich ist und über die Oberfläche eingestellt werden kann. Mit Hilfe der U-Matrix-Methode [ULTSCH/ SIEMON 90] werden die vom Neuronalen Netz erkannten Mauern zwischen den Klassen zur Visualisierung aufbereitet. Nun kann eine Klassifizierung der Daten sowohl automatisch als auch per Hand am Bildschirm erfolgen, indem die Klassengrenzen festgelegt werden.

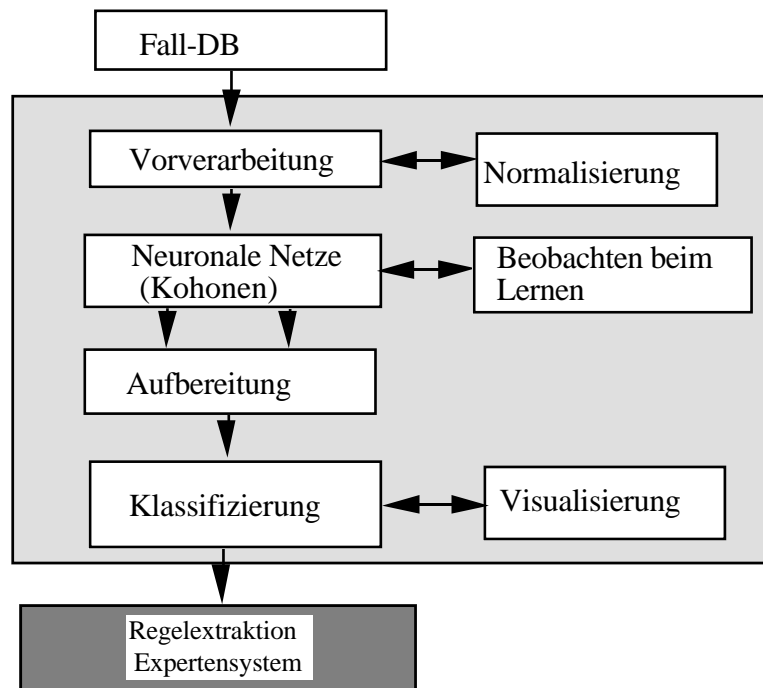


Abb. 8 Neuronaler Klassifikator

Um das gewonnene Wissen auf subsymbolischer Ebene dem Experten in Form von Regeln zu erklären, werden die durch den Neuronalen Klassifikator erkannten Klassen und die Daten benötigt. Ein von uns speziell entwickeltes Regelextraktionsverfahren sig* generiert hieraus die Regeln für das Expertensystem [ULLTSCH 91].

7. Benutzeroberfläche

Die Integration der einzelnen Module in ein Gesamtsystem unter einer einheitlichen Oberfläche ermöglicht nicht nur eine menügesteuerte Führung des Benutzers, sondern verringert durch Parametereinstellungen über die Oberfläche

erheblich die Fehleranfälligkeit des Systems. Weiterhin wird dadurch erst die von uns gewünschte Protokollierung der Arbeitsschritte ermöglicht. Die Protokollierung der bereits durchgeführten Bearbeitungsschritte ist notwendig, da es sich bei unserem Verfahren um eine explorative Datenanalyse handelt. Wir gehen folgendermaßen vor: Nach Abarbeitung jedes einzelnen Moduls werden die Parameter in einer Protokolldatei abgespeichert. Der Benutzer kann nun die Sitzung an einem beliebigen Punkt abbrechen, und diese an einer vorherigen Stelle weiterführen bzw. ganz von vorne anfangen. Die Parameter werden dann einfach bis zu dem neuen Ansatzpunkt in die neuen Protokolldatei kopiert.

Die Parameter für die Neuronale Netze können auf zwei Ebenen eingestellt werden: Laien-Ebene und Experten-Ebene. Der Kohonen-Algorithmus wird dann von der Oberfläche aus auf den Transputern als Child-Prozeß geladen, indem der iserver des inmos-Toolset gestartet wird.

8. Schluß

Neuronale Netze eignen sich aufgrund ihrer hochgradigen Parallelität zur Implementierung auf Transputern. Die Transputertarchitektur und -topologie sowie die hier zur Verfügung stehenden Programmiersprachen schränken jedoch die Struktur der Algorithmen für eine effiziente Implementierung ein. Im einzelnen zu berücksichtigen sind: die Topologie der Transputer und deren Kommunikation untereinander, sowie die Verteilung der Neuronen auf den Transputern. Um eine gute Auslastung der Transputer zu erreichen, werden die Neuronen nach dem "interlaced"-Verfahren auf die Transputer verteilt. Statt einem Broadcasting nach dem Kohonen-Modell werden die Lernvektoren und die Positionen der Bestmatches durch das Netz (Ring- oder Baumstruktur) propagiert.

Die Neuronale Netze sind hier als Teil eines Gesamtsystems zu sehen, dem Neuronale Klassifikator. Um das Verhalten der Neuronale Netze während des Anlernens des Netzes zu beobachten, kann die Fehlerkurve auf dem Host-Rechner ausgegeben werden. Die Animation der U-Matrix in der Lernphase ist wegen des Flaschenhalses bei der Kommunikation zwischen Transputern und Host-Rechnern bisher noch nicht möglich.

Literatur

- [ELLINGER 92] Ellinger, Diplomarbeit Fachbereich Elektrotechnik, Dortmund 1992.
- [GELERNTER 87] Gelernter, D. "Das Programmieren modernster Computer" Spektrum der Wissenschaft, 12/1987, S. 74–82.
- [GOORHUIS et.al. 90] Goorhuis, H., Gürman, N., Montigel, M., Thalmann, L.: Neuronale Netze und Regelbasierte Systeme: Ein hybrider Ansatz. Gelbe Berichte des Dept. Informatik der ETH Zürich, Nr. 131. Zürich, 1990.
- [KLEIN 92] Klein, R.D. "Der Einsatz von T22x Transputern als Kommunikationsprozessor in auf i860 Prozessoren basierenden MIMD-Supercomputer" Abstraktband TAT'92, Aachen 1992, S.81.
- [KOHONEN 89] Kohonen, T. "Self-Organization and Associative Memory" Springer 1989³.
- [PALM/RÜCKERT/ULTSCH 91] Palm, G., Rückert, U., Ultsch, A.: Wissensverarbeitung in neuronaler Architektur. in: Proceedings des 4 Int. GI-Kongreß Wissensbasierter Systeme, München, 1991.
- [SIEMON/ULTSCH 90] Siemon, H.P. & Ultsch, A. "Kohonen Networks on Transputers: Implementation and Animation" INNC Paris 1990, S. 643–646.
- [ULTSCH/ SIEMON 90] Ultsch, A. & Siemon, H:P. "Kohonen's Self Organizing Feature Maps for Exploratory Data Analysis" INNC Paris 1990, S.305-308.
- [ULTSCH 91] Ultsch, A.: Konnektionistische Modelle und ihre Integration mit wissensbasierten Systemen, Internal Report Nr. 396, University of Dortmund, Germany, Februar 1991.