# NeNEB - Neuronales Netzwerk zur Echtzeit-Klassifizierung von Bildern

*L. Larsson, S. Krol, K. Lagemann*
Universität Hamburg, Fachbereich Informatik / Arbeitsbereich TECH
Vogt-Kölln-Straße 30, D-22527 Hamburg, Germany
e-mail: larsson@informatik.uni-hamburg.de

**Zusammenfassung**

*Die vorliegende Arbeit beschreibt die digitale Realisierung eines auf dem Backpropagation-Algorithmus basierenden neuronalen Netzwerks zur Klassifizierung von Bildern in Echtzeit. Dabei stand der Aspekt der Systemintegration, mit den sich daraus ergebenden Randbedingungen, im Vordergrund. Dieses sind insbesondere der sequentielle Charakter des Videosignals gebräuchlicher Videokameras sowie Eigenschaften handelsüblicher Speicher. Durch Verarbeitung der Videosignale an deren Quelle wird der Flaschenhals bezüglich Datenübertragung und Speicherung vermieden. Geringe Größe, die durch Integration auf einem Chip erreicht werden kann, ist für mobile Systeme von Bedeutung. Möglichen Abweichungen zwischen durch Simulation vorhergesagter und tatsächlich in einer Systemumgebung erreichbarer Verarbeitungsgeschwindigkeit wird durch einstellbare Auflösungsreduzierung Rechnung getragen. Elementare Operationen des neuronalen Netzwerks werden approximiert, um sie effizient in Hardware realisieren zu können.*

## 1 Motivation

Inspiriert wurde die vorliegende Arbeit durch *ALVINN*, vorgestellt in [1, 2] von D. Pomerleau, wo ein Backpropagation-Netz mit einer verdeckten Schicht verwendet wird, um ein Fahrzeug selbständig anhand von Bilddaten in Echtzeit einen Straßenverlauf folgen zu lassen. Die hier vorgestellte VLSI-Implementation eines neuronalen Netzwerks kann neben der Klassifizierung von Bildern andere Applikationen mit abdecken, wenn diese mit den implementierten Netzwerkparametern kompatibel sind.

*Published by GMD, in Proc. 8. E.I.S. - Workshop, Entwurf Integrierter Schaltungen, Hamburg, Germany, 8-9 April 1997, in L. Peters, K. Lagemann (Hrsg.), Tagungsband, S. 57-66, GMD 1997.*

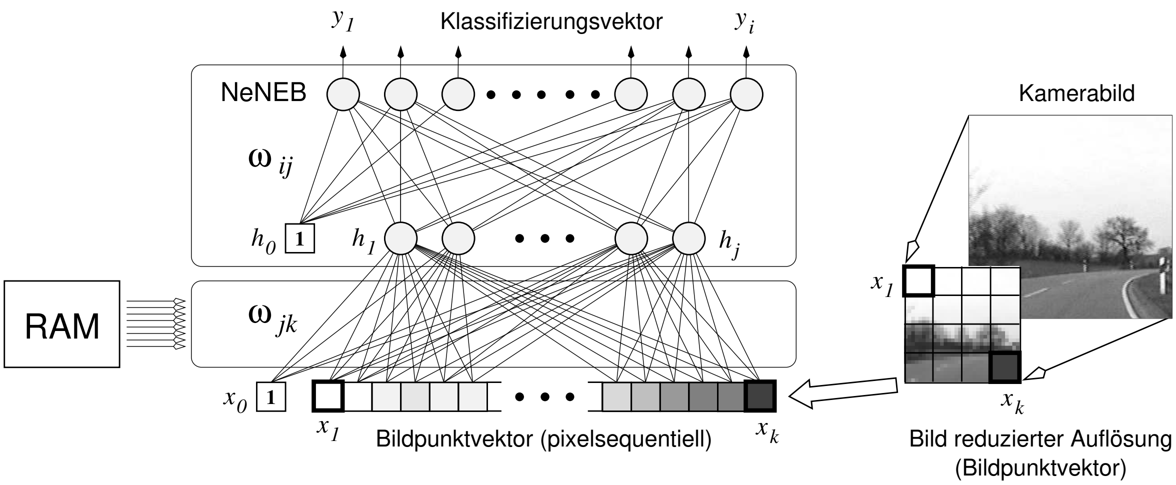


Abbildung 1: **Verarbeitungsschema** *von NeNEB.*

Bei der vorliegenden Arbeit stand die Abschätzung des Implementationsaufwands sowie der erreichbaren Verarbeitungsgeschwindigkeit im Vordergrund, um eine Basis für eine VLSI-Realisierung des Backpropagation-Algorithmus zum *Lernen* von Bildklassifizierungen in Echtzeit zu schaffen. Dabei zielt die Implementation zunächst auf die Verarbeitung von Einzelbildern ab, die Verarbeitung von Bildsequenzen (reduzierter Auflösung) wurde bei der Konzeption mit einbezogen. Die Erweiterung des bestehenden Konzepts durch Eigenschaftsextraktion ist denkbar. In einem endgültigen System ist die Integration digitaler Videosignalvorverarbeitung vorgesehen.

Die *System Level Simulation* unter VHDL sowie auf EPLDs (*Electrically Programmable Logic Devices*) basierendes Prototyping spielen für das Projekt eine wichtige Rolle. So sind anfallende Ergebnisse unter Umständen für ganz andere Anwendungsbereiche von Bedeutung, die mit neuronalen Netzwerken nicht in unmittelbarem Zusammenhang stehen.

# 2 Systemskizze

Die vorliegende Hardware-Implementation umfaßt den *Feed Forward Path* eines Backpropagation (BPG) Netzwerks. Die Bestimmung eines Satzes synaptischer Gewichte zur Lösung einer vorgegebenen Klassifizierungsaufgabe erfolgt (off-line) auf Workstations oder PCs unter Verwendung des von D. E. Rumelhart, J. L. McCelland, G. E. Hinton und R. J. Williams eingeführten Backpropagation-Algorithmus [3, 4, 5] zum Training mehrschichtiger, rückkopplungsfreier neuronaler Netzwerke.

Es können Netzwerke [k-j-i] mit bis zu $i = 32$ Ausgangsneuronen $y_i$, $j = 8$ Hidden-Neuronen $h_j$ und bis zu $k = 65536$ Eingängen $x_k$ realisiert werden, was etwa $512K$ Synapsen (für ein [65536-8-32] Netzwerk) entspricht (Abb. 1). Bei einer vorgesehenen Taktfrequenz von 60 $MHz$ ist NeNEB in der Lage, von $55K$ Mustern pro Sekunde bei einer Mustergröße von 66 Punkten bis zu 50 Graustufenbildern einer Größe von $256 \times 256$ Bildpunkten pro Sekunde zu klassifizieren, was einem Spitzendurchsatz von

$\approx 25$ *Mcps* (*connections per second*) entspricht. Dazu wurden 8 Hidden-Neuronen und 32 Ausgangsneuronen als parallel arbeitende Einheiten realisiert.

Die große Anzahl synaptischer Gewichte der Hidden-Neuronen wird in einem externen Speicher abgelegt. Ein besonderes Speicherschema für diese Gewichte begrenzt die Zahl der für die Kommunikation benötigten Pads. Die (verhältnismäßig wenigen) synaptischen Gewichte der Ausgangsneuronen werden auf dem Chip gespeichert, um ebenfalls die Zahl der benötigten Pads möglichst gering halten zu können. NeNEB mischt gleichsam den *pixelsequentiellen* Datenstrom eines in der Auflösung reduzierten Videobildes mit dem *datenbitsequentiellen* Strom der synaptischen Gewichte der Hidden-Neuronen, um dadurch einem Bild einen Klassifizierungsvektor zuzuordnen. Die Sigmoidfunktion der Neuronen wird ähnlich, wie in [6] vorgeschlagen, stückweise linear approximiert.

### Implementationsrandbedingungen

Software- und Hardware-Implementation neuronaler Netzwerke implizieren unterschiedliche Randbedingungen. Bei der Hardware-Implementation muß durch geeignete Modifikation der betreffenden Algorithmen ein Kompromiß zwischen algorithmischen und VLSI-bestimmten Randbedingungen gefunden werden. Die Implementation als *stand-alone* System erfordert die Integration einiger zusätzlicher Komponenten, die Koprozessoren zur Beschleunigung neuronaler Algorithmen, wie MA16 oder CNAPS [7, 8] extern benötigen.

Eine eigene Implementation des Backpropagation-Algorithmus [9] auf massiv parallelen Rechnern (MasPar MP-1216 / MP-2216 ($16K$ Prozessoren)) mit einer SIMD-Architektur (*single instruction multiple data*) ergab, unter Vernachlässigung der Übertragungszeit der Mustervektoren, eine Verarbeitungsgeschwindigkeit in dem Bereich ($10\ Mcps \ldots 30\ Mcps$), wie in der Literatur (etwa in [10]) zu finden ist.

Für eine (flächen-)effiziente Hardware-Implementation sind Approximationen entscheidend. Erfordert ein Algorithmus jedoch präzise Fließkommaarithmetik, so ist ein Standardprozessor (mit Fließkommaeinheit) im allgemeinen die bessere Wahl.

## 3 Bildvorverarbeitung

Bei unserem Ansatz wird die Bildauflösung zunächst reduziert. Die Reduktion der Bildauflösung hat keinen Nachteil, wenn Details zu klassifizierender Bilder von untergeordneter Bedeutung für die Klassifizierung sind, wie bei ALVINN [1, 2]. Unser Reduzierungsschema weicht jedoch von dem in [1, 2] beschriebenen Schema ab. Der Vorteil einer variablen Auflösungsreduzierung liegt in der damit einstellbaren Bildpunktrate. Somit kann die Bildpunktrate an eine für das neuronale Netz verarbeitbare Rate angepaßt werden, wenn der erreichte Durchsatz von dem durch Simulation vorhergesagten abweichen sollte. Die Verwendung einer Kamera mit geringer Auflösung würde dagegen die Klassifizierung bei höherer Auflösung von vornherein ausschließen.

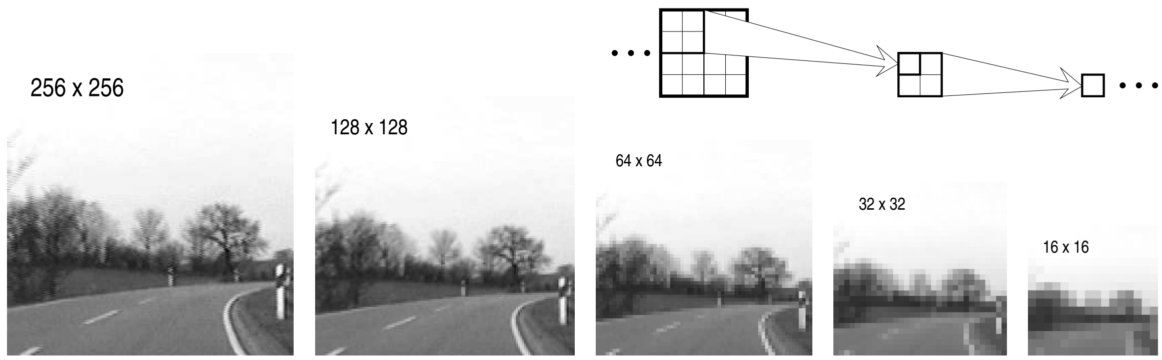


Abbildung 2: **Bildvorverarbeitungsschema.** *Die Auflösungsreduktion ermöglicht die Variation der Bildpunktfrequenz zur Anpassung an die Verarbeitungsgeschwindigkeit des neuronalen Netzwerks in einen weiten Bereich möglicher Systemfrequenzen. Dazu werden jeweils vier benachbarte Bildpunkte zu einem Bildpunkt gemittelt. Die Bilder bleiben erkennbar, sofern Details eine untergeordnete Rolle spielen.*

Zudem sind Standardvideokameras mit einer Auflösung, die unter Umständen höher als die benötigte ist, kostengünstig verfügbar. Um die Auflösung um eine Stufe zu reduzieren, werden jeweils vier benachbarte Bildpunkte gemittelt und zu einem reduzierten Bildpunkt zusammengefaßt. Dieses einfache Schema läßt sich als vergleichsweise lokale Operation realisieren und wurde zunächst testweise mit Hilfe der *Altera Hardware Description Language* (AHDL) [11] auf zwei *Altera EPM7128LC84 EPLDs* [12] abgebildet [13]. Die Reduktion um eine Stufe reduziert die Bildpunktfrequenz um ein Viertel.

Das Schema wurde gewählt, da ein so reduziertes Bild für den Menschen vergleichsweise gut erkennbar bleibt, wenn die Auflösung beispielsweise von $256 \times 256$ auf $128 \times 128$ oder $64 \times 64$ Bildpunkte reduziert wird (Abb. 2). Jedes in der Auflösung reduzierte Bild repräsentiert einen zu klassifizierenden Eingangsvektor des neuronalen Netzwerks.

Im vorgesehenen Gesamtsystem soll das Videosignal einer einfachen CCD-Kamera mit einem 8-Bit-Flash-ADC (etwa MC10319 [15]) digitalisiert werden, um die Bildinformation rein digital zu extrahieren. Ein VHDL-Modell eines rein digitalen *Frame Grabbers* wurde auf zwei *Altera EPM7128LC84 EPLDs* abgebildet und als Prototyp realisiert. Die rein digitale Signalverarbeitung eines mit 8-Bit Genauigkeit und einer Abtastrate von 20 $MHz$ digitalisierten PAL-Videosignals einer Standardfarbkamera wird derzeit untersucht. Die dabei erreichbare Bildqualität konnte mit einem EPLD-basierten Transientenrekorder bereits demonstriert werden [16]. Die Verwendung von Farbbildern anstelle von Graubildern kann ein leistungsfähiges Hilfsmittel zur Unterdrückung von durch Schattenwurf hervorgerufene Störungen sein, wie im Zusammenhang mit ALVINN in [2] beschrieben.

Wegen der Kompatibilität mit digitalen VLSI-Prozessen stellt die digitale Videosignalverarbeitung gegenüber der analogen eine interessante Alternative dar. Ein validiertes VHDL-Modul könnte später in bildverarbeitende Chips integriert werden.

# 4 Backpropagation Netzwerk

Dieser Abschnitt faßt das Backpropagation Netzwerkmodell kurz zusammen, wie es für die Implementation von NeNEB und zum (off-line) Training verwendet wurde. Jedes Neuron $i$ mit Schwelle $\theta_i$ berechnet die gewichtete Summe seiner Eingänge $x_j$. Die Ausgangsaktivität $y_i$ des Neurons $i$ ergibt sich mit Hilfe einer nicht linearen Aktivierungsfunktion (4) als

$$y_i = g(\theta_i + \sum_{j=1}^{n} \omega_{ij} \cdot x_j). \tag{1}$$

Durch Erweiterung der Gleichung (1) mit $\omega_{i0} := \theta_i$, $x_0 := 1$ und $j = 0$ erhält man die Form mit *Bias*. Ein Netzwerk mit $k$ Eingängen, $j$ Hidden Neuronen und $i$ Ausgangsneuronen klassifiziert einen Bildvektor $\vec{x}$ mit $k$ Bildpunkten gemäß

$$\vec{y} = F(\Omega_{ij}, \Omega_{jk}, \vec{x}), \tag{2}$$

wobei $\Omega_{ij}$ und $\Omega_{jk}$ die Kopplungsmatrizen der synaptischen Gewichte sind (Abb. 1). Die Vektoren $\vec{v} \in \{\vec{x}, \vec{h}\}$ in (3) sind um eine Dimension $v_0$, zur Aufnahme des *Bias* $v_0 := 1$, zu erweitern und die synaptischen Kopplungsmatrizen $\Omega$ sind um eine Spalte zu erweitern. Diese zusätzliche Spalte dient zur Aufnahme der Schwellenvektoren $\vec{\theta} = \vec{\Omega}_0$. Die vom Netzwerk berechnete Klassifizierung (Feed Forward) gemäß (2) ergibt sich aus

$$\vec{y} = g(\vec{o}), \quad \vec{o} = \Omega_{ij} \cdot \vec{\gamma}, \quad \vec{\gamma} = g(\vec{h}), \quad \vec{h} = \Omega_{jk} \cdot \vec{x}, \tag{3}$$

mit Spaltenvektoren $\vec{x}$, $\vec{h}$, $\vec{\gamma}$, $\vec{o}$, $\vec{y}$. Als Aktivierungsfunktion $g(x)$ wurde die Sigmoidfunktion

$$\begin{aligned} g(a_i) &= 1/(1 + e^{-a_i/\tau}) \\ g'(a_i) &= g(a_i) \cdot g(-a_i)/\tau \end{aligned} \tag{4}$$

mit einem Temperaturparameter $\tau$ (*typisch* $0.1 < \tau < 10$) gewählt. Gelernt werden $M$ Muster $(\vec{x}; \vec{t})_M$ ($\vec{x} \approx$ Eingangsbildvektor; $\vec{t} \approx$ Zielklassifikation), bis der Fehler

$$E = \frac{1}{2} \sum_M (\vec{y} - \vec{t})^2_M \tag{5}$$

minimal ist oder unterhalb einer Grenze liegt. Dazu wird für jedes Muster $M$

$$\begin{aligned} &\vec{d} = \vec{y} - \vec{t}, & &\vec{\delta} = g'(\vec{o}), \\ &\vec{e} =^t \Omega_{ij} \cdot (\vec{d} \otimes \vec{\delta}), & &\vec{\epsilon} = g'(\vec{e}), \end{aligned} \tag{6}$$

berechnet, wobei ${}^t\Omega_{ij}$ die transponierte Matrix $\Omega_{ij}$ darstellt. Der Operator $\otimes$ ist hier als komponentenweises Vektor-Vektor-Produkt

$$\vec{c} = \vec{a} \otimes \vec{b} \Leftrightarrow c_i = a_i \cdot b_i \tag{7}$$

definiert. Beim Lernen erfolgt die schrittweise Modifikation der Gewichtsmatrizen

$$\begin{array}{rcl} \Omega_{ij} & := & \Omega_{ij} - \eta \cdot \vec{\delta} \cdot^t \vec{\gamma} \quad + \quad \alpha \cdot \Delta\Omega_{ij} \\ \Omega_{jk} & := & \Omega_{jk} - \eta \cdot \vec{\epsilon} \cdot^t \vec{x} \quad + \quad \alpha \cdot \Delta\Omega_{jk} \end{array} \tag{8}$$

in Richtung des negativen Gradienten der Fehlerfunktion $E$ mit einer Lernrate $\eta$ (*typisch* $0.001 < \eta < 0.5$). Die zusätzliche Addition des Impulsparameters $\alpha$ (*typisch* $0.1 < \alpha < 0.9$) reduziert Oszillationen während des Lernens.

# 5 Implementation

Zur Implementation des BPG-Algorithmus sind die Grundoperationen zu realisieren. Dies sind die Matrixmultiplikation (3), Sigmoidfunktion $g(a_i)$ sowie deren Ableitung $g'(a_i)$ (4), Vektordifferenz, die Multiplikation mit transponierter Matrix (6), das Vektor-Vektor-Produkt (7) sowie die Modifikation der Gewichtsmatrizen (8).

NeNEB umfaßt die Implementation des Vorwärtspfades (*Feed Forward Path*) des Backpropagation-Algorithmus. Dazu sind nur die Matrixmultiplikation (3) und die Sigmoidfunktion (4) zu realisieren. Implementiert wurden synthetisierbare VHDL-Module [13], die funktional äquivalent zur Gleichung (1) der Neuronen sind und die Klassifizierung gemäß Gleichung (2), basierend auf Festpunktarithmetik, berechnen. Die Sigmoidfunktion (4) wird stückweise linear approximiert, wobei die Steigung der Geradenstücke nur diskrete Werte (bestimmte Zweierpotenzen) annehmen kann. Interne Wortbreiten sind derart gewählt, daß ein Überlauf von Zwischenergebnissen vermieden wird [14].

Backpropagation-Lernen gemäß der Gleichungen (6), (7) und (8) erfolgt mit einem in C geschriebenen Programm. Acht parallel arbeitende Hidden Neuronen berechnen – Bildpunkt für Bildpunkt – gewichtete Summen (1). Zweiunddreißig Ausgangsneuronen ermitteln die gewichteten Summen über die Ausgaben der Hidden Neuronen.

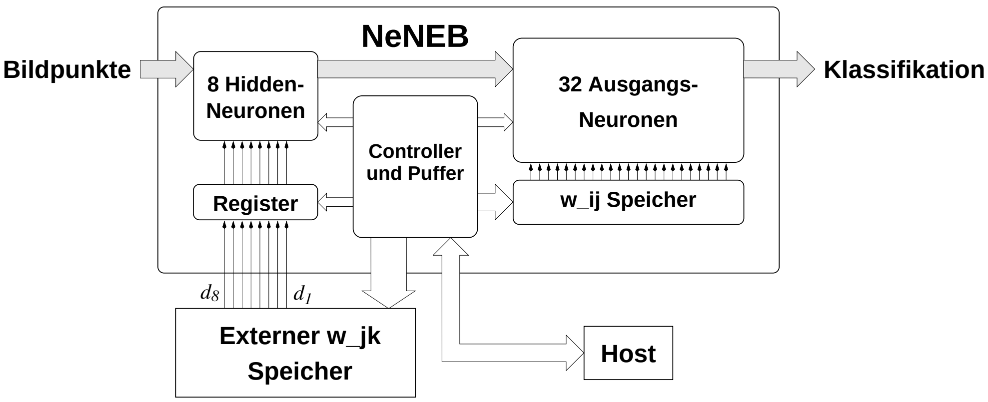


Abbildung 3: **NeNEB Blockschema.**

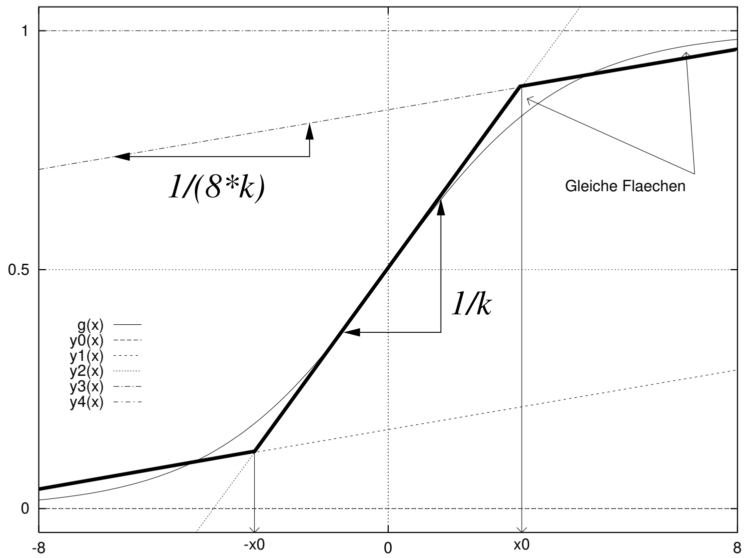


Abbildung 4: **Stückweise lineare Approximation der Sigmoidfunktion.** *Die Sigmoidfunktion $g(x)$ wird durch fünf Geradenstücke $y_0(x)$, $y_1(x)$, $y_2(x)$, $y_3(x)$, $y_4(x)$ approximiert, wobei sich $y_0(x)$ und $y_4(x)$ durch Bereichsbegrenzung ergeben. Zu berechnen sind nur drei Geradenstücke $y_1(x), y_2(x), y_3(x)$ mit zwei verschiedenen Steigungen $1/k$ und $1/(8 \cdot k)$. Die Bestimmung des reziproken Steigungsparameters $k = \{0.5, 1, 2, .., 64\}$ erfolgt in zwei Schritten. Zu erst wird das $k$ gewählt, welches $1/g'(0)$ am nächsten ist. Anschließend werden die additiven Konstanten für $y_1(x), y_2(x), y_3(x)$ derart gewählt, daß die Flächenstücke gleich sind.*

# 6 Architektur

Die synaptischen Gewichte der acht Hidden-Neuronen werden extern gespeichert. Die Zuführung zu den acht Hidden-Neuronen erfolgt bitseriell über acht Datenleitungen. Die Multiplikation ist mit sequentiellen Multiplizierern [17] realisiert. Die stückweise lineare Approximation (9) der Sigmoidfunktion (4) erfolgt mit Hilfe von Division durch Zweierpotenz (*Barrel Shifter*) und Konstantenaddition in Abhängig des Funktionsarguments.

Der in NeNEB enthaltene Controller beinhaltet Adresszähler für den externen Speicher sowie interne Pufferregister und steuert neben der Kommunikation mit der Außenwelt alle inneren Abläufe, so daß NeNEB eine Klassifizierung (2) völlig selbständig durchführen kann.

## Neuron

Ausgangsneuronen und Hidden-Neuronen sind ähnlich realisiert, nur daß die synaptischen Gewichte der Ausgangsneuronen auf dem Chip gespeichert werden. Die Multiplikation $\omega \cdot x$ ist bitsequentiell. Die synaptischen Gewichte werden als vorzeichenbehaftete

Festpunktzahlen mit drei Vorkommastellen und zwölf Nachkommastellen gespeichert. Korrespondierende Bits der acht Hidden-Neuronen bilden ein Byte. Die acht Datenbits $d_j$ der synaptischen Gewichte $\omega_{jk}[n]$ der $(j = 1, \ldots, 8)$ Hidden Neuronen (Abb. 3) werden in sechzehn aufeinander folgenden Adressen $(16 \times k + n, n = 0, \ldots, 15)$ des externen Speichers abgelegt. Es wird vor Beginn der Multiplikation ein Taktzyklus zur Initialisierung benötigt. Danach erfolgt die vorzeichenlose Multiplikation $\omega \cdot x$ plus einem abschließenden Taktzyklus zur Vorzeichenbehandlung. Diese 17 Taktzyklen bilden den Flaschenhals unserer Implementation. Die Taktfrequenz muß 17 $\times$ Bildpunktfrequenz betragen. Bei einer Taktfrequenz von 60 $MHz$ wird der vorgesehene Datendurchsatz jedoch erreicht.

## Sigmoidfunktion

Um den Verlauf der Sigmoidfunktion einzustellen, kann ein reziproker Temperaturparameter $c = 1/\tau$ von acht möglichen $c \in \{0,0625, 0,125, 0,25, 0.5, 1, 2, 4, 8\}$ durch Programmierung eines Parameters $\mathsf{s} \in \{0, 1, 2, 3, 4, 5, 6, 7\}$ gewählt werden (Abb. 4). Die additiven Konstanten $(o_-,\ o_+)$ der Geradenstücke $y_1$, $y_2$, $y_3$ werden in Registern auf dem Chip gespeichert. Die Approximation der Sigmoidfunktion erfolgt so durch Selektion einer Geradengleichung in Abhängigkeit vom Argument der Funktion gemäß

$$y(x) = \begin{cases} 0 & \text{für} \quad \frac{\mathsf{x}}{(8*\mathsf{k})} + o_- \; < \; 0 \\ \frac{\mathsf{x}}{(8*\mathsf{k})} + o_- & \text{für} \quad \mathsf{x} \; \leq \; -\mathsf{x0} \\ \frac{\mathsf{x}}{\mathsf{k}} & \text{für} \quad -\mathsf{x0} < \mathsf{x} < +\mathsf{x0} \\ \frac{\mathsf{x}}{(8*\mathsf{k})} + o_+ & \text{für} \quad \mathsf{x} \; \geq \; +\mathsf{x0} \\ 1 & \text{für} \quad \frac{\mathsf{x}}{(8*\mathsf{k})} + o_+ \; > \; 1 \end{cases}, \tag{9}$$

wobei $\mathsf{k} = 2^{\mathsf{s}-1}$. Der Parameter $\mathsf{s}$ steuert einen *Barrel Shifter*. Die für die Approximation benötigten Parameter

$$\begin{array}{rcl} \mathsf{k} & = & \frac{1}{g'(0)} \quad \text{mit } \mathsf{k} \in \{\frac{1}{2}, 1, 2, 4, 8, 16, 32, 64\} \\ \mathsf{x0} & \approx & 0.3825 \cdot \mathsf{k} \\ a & = & \frac{7}{8 \cdot \mathsf{k}} \cdot \mathsf{x0} \qquad \text{und} \qquad b \; = \; g(0) \\ o_- & = & b - a \qquad \text{und} \quad o_+ \; = \; b + a \end{array} \tag{10}$$

ergeben sich aus der Bedingung, daß eingeschlossenen Flächen (Abb. 4) zwischen der Sigmoidfunktion und deren Näherung gleich sein sollen, also daß die Flächendifferenz

$$\begin{aligned} F(x_0) &= |G(x_m) - Y(x_0, x_m)| \; = \; |\textstyle\int_0^{x_m} g(x)dx - \int_0^{x_m} y(x_0, x)dx| \\ &\approx |(x_m + \tau \cdot \log_2(\tfrac{1}{2})) - (\textstyle\int_0^{x_0} y_1(x)dx + \int_{x_0}^{x_1} y_2(x)dx + \int_{x_1}^{x_m} y_3(x)dx)| \end{aligned} \tag{11}$$

verschwindet. Dies stellt nur eine analytische (ad hoc) Berechnungsmöglichkeit dar, die jedoch ein brauchbares Ergebnis liefert.

# 7 Ergebnisse

Die in VHDL erstellte Verhaltensbeschreibung wurde mit dem *Synopsys VHDL System Simulator (VSS)* simuliert und mit dem *Synopsys Design Compiler (DC)* [18] und dem *ES2 ecpd07* als Zielprozeß erfolgreich synthetisiert. Die Validation der reinen Verhaltensbeschreibung sowie einer Mischung aus Verhaltensbeschreibung und synthetisierten Teilkomponenten erfolgte durch Vergleich der Klassifizierungsergebnisse von Schaltungssimulation mit denen eines in C geschriebenen Backpropagation Programms. Dazu wurden kleinere Probleme (XOR [2-2-1], Dekoder [3-8-8] sowie Buchstabenerkennung [35-8-26]) verwendet. Die gemischte Simulation war notwendig, um die Simulationszeit zu begrenzen. Die erwartete Performanz von NeNEB wird bei einer erreichbaren Taktfrequenz von 60 $MHz$ etwa um einen Faktor 25 höher als bei einer Mittelklasse-Workstation ($\approx 1$ $Mcps$) liegen. Eine mit *Cadence DFW II* [19] erstellte Abschätzung der benötigten Chip-Fläche ergab 157 $mm^2$.

# 8 Diskussion

Durch Beschränkung auf maximal 50 Klassifizierungen pro Sekunde – also Einschränkung auf Anwendung im Bereich der Bildklassifizierung – würde sich die notwendige Chip-Fläche deutlich reduzieren lassen, da sich dann die 32 parallel arbeitenden Ausgangsneuronen, die etwa drei Viertel der Chip-Fläche ausmachen, durch ein (32-fach) gemultiplextes Ausgangsneuron ersetzen ließen. Außerdem könnte die approximative Berechnung der Sigmoidfunktionen durch einfache (externe) Look-Up-Tables (LUTs) ersetzt werden, da externer Speicher ohnehin vorgesehen ist. Ob der dafür notwendige, zusätzliche Steuerungsaufwand mit der approximativen Berechnung vergleichbar ist, muß gegebenenfalls durch Implementation bestimmt werden.

# Danksagung

Die *Synopsys Tools*, *Cadence DFW II* und das *Altera Max+Plus II* wurden uns durch die *Europäische Gemeinschaft* über *EUROCHIP* und *EUROPRACTICE* zugänglich gemacht. Besonderer Dank gilt unserem Kollegen Herrn A. Mäder für die Installation und ständige Wartung der CAD-Programme und seine Hilfe bei deren Handhabung.

# Literatur

[1] D. Pomerleau, ALVINN: An Autonomous Land Vehicle in a Neural Network, *Carnegie-Mellon Univ: reports; CMU-CS-89-107*, 1989.

[2] D. A. Pomerleau, Neural Network Perception for Mobile Robot Guidance, *Kluwer academic publishers*, 1993.

[3] D. E. Rumelhart, G. E. Hinton, R. J. Williams, Learning Internal Representations by Error Propagation, in J. A. Anderson and E. Rosenfeld (eds.) *Neurocomputing: Foundations of Research*, MIT Press 1988, S. 673-695

[4] D. E. Rumelhart, G. E. Hinton, R. J. Williams, Learning Representations by Back Propagating Errors, in J. A. Anderson and E. Rosenfeld (eds.) *Neurocomputing: Foundations of Research*, MIT Press 1988, S. 696-699

[5] D. E. Rumelhart, J. L. McCelland, *Parallel Distributed Processing*, Vol. 1&2, MIT Press, 1986.

[6] J. Cloutier, P. Y. Simard, Hardware Implementation of the Backpropagation without Multiplication, *Proc. of the 4th Int. Conf. on Microelectronics for Neural Networks and Fuzzy Systems*, Turin, Italy, September 26-28, 1994, S. 46-55, IEEE 1994.

[7] M. Glesner and W. Pöchmüller, *Neurocomputers, An Overview of Neural Networks in VLSI*, Chapman & Hall, 1994.

[8] B. Sheu, J. Choi, *Neural Information Processing and VLSI*, Kluwer, 1995.

[9] L. Larsson, Implementing the BPG Neural Network Algorithm on MP-1216 / MP-2216 Massive Parallel SIMD Machines, Report für MasPar, Sunnyvale, CA, USA, MasPar Challenge Project '94, May 1994.

[10] A. Zell, *Simulation Neuronaler Netzwerke*, Addison-Wesley, 1994.

[11] *Altera AHDL Reference Manual*, Altera Corporation, San Jose, CA, USA, 1994.

[12] *Altera Databook*, Altera Corporation, San Jose CA, USA, 1993.

[13] S. Krol, Hardware - Realisierung eines neuronalen Netzwerkes zur Echtzeit - Bildklassifikation – NeNEB-Chip, *Universität Hamburg, Fachbereich Informatik, Arbeitsbereich TECH, Diplomarbeit*, January 1996.

[14] L. Larsson, S. Krol, K. Lagemann, NeNEB - An Application Adjustable Single Chip Neural Processor for Mobile Real Time Image Processing, *Proc. 1996 Int. Workshop on Neural Networks for Identification, Control, Robotics, and Signal/Image Processing, NICROSP'96*, Venice, Italy, August 21-23, 1996, S. 154 - 162, IEEE 1996.

[15] *Analog/Digital and Digital/Analog Conversion Manual*, Motorola Inc., USA, 1984.

[16] L. Larsson, An EPLD Based Transient Recorder for Simulation of Video Signal Processing Devices in a VHDL Environment Close to System Level Conditions, *Proc. 6th Int. Workshop on Field Programmable Logic and Applications, FPL'96*, Darmstadt, 23-25 September 1996, in R. W. Hartenstein & M. Glesner, (Herausgeber), in *Lecture Notes in Computer Science*, Vol. 1142, S. 371 - 375, Springer, 1996.

[17] Amos R. Omondi, *Computer Arithmetic Systems, Algorithms, Architectures and Implementations*, Prentice Hall, 1994.

[18] *Synopsys VHDL Simulator Reference Manual*, Synopsys Inc., 1993.

[19] *Cadence Design Framework-II*, Cadence Inc., 1993.