Prüfer: Prof. Dr. Andreas Reuter

Betreuer: Dr. Andreas Zell

begonnen am: 1. Januar 1992

beendet am: 28. April 1992

CR-Klassifikation: C.1.3, G.1.2, I.2.6, I.5.1

Implementierung und Anwendung von 'Generalized Radial Basis Functions' in einem Simulator neuronaler Netze

Michael Vogt
Diplomarbeit Nr. 875

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

Zusammenfassung

Die Anwendung von Radial Basis Functions ist ein neuerer Ansatz für die mathematische Approximation von Funktionen. Hierbei werden an vorgegebenen Stützstellen rotationssymmetrische Funktionen (Zentrumsfunktionen) aufgespannt, deren additive Überlagerung die zu approximierende Funktion ergibt. Für bestimmte Klassen von Zentrumsfunktionen kann gezeigt werden, daß die freien Parameter der Approximation direkt durch die Lösung eines linearen Gleichungssystems bestimmt werden können. Radial Basis Functions lassen sich sehr einfach auf neuronale Netze mit einer versteckten Ebene abbilden. Hierbei repräsentiert jedes Neuron der versteckten Ebene genau eine Stützstelle des Approximationsproblems. Der wesentliche Unterschied zu herkömmlichen feedforward–Netzen besteht in der Aktivierungsfunktion der mittleren Schicht. Zur Aktivierung wird im Gegensatz zu bisher gebräuchlichen Netzen nicht das Skalarprodukt zwischen der Ausgabe der Eingangsneuronen und den Verbindungsgewichten berechnet, sondern der euklidische Abstand zwischen diesen beiden Vektoren. Die Aktivierung selbst erfolgt durch eine Zentrumsfunktion, die für positive Argumente definiert ist. Oft verwendet man hierfür den positiven Bereich einer Gaußschen Glockenkurve. Die weiteren Gewichte im neuronalen Netz können in einem umfassenden Initialisierungsschritt direkt berechnet und bei Bedarf nachtrainiert werden. Die Integration der Radial Basis Functions in den Stuttgarter Neuronale Netze Simulator (SNNS) umfaßt drei verschiedene Zentrumsfunktionen sowie die Verfahren zur Durchführung des Initialisierungsschritts und des anschließenden Trainings. Da die Wahl der Stützstellen entscheidend für die Qualität der Approximation ist, wurde zusätzlich das selbstorganisierende Verfahren von Kohonen implementiert. Hierdurch konnte das Ergebnis der Initialisierung wesentlich verbessert werden. Die Leistungsfähigkeit der Radial Basis Functions wurde an drei Beispielen geprüft. Die Probleme der Erkennung handgeschriebener Ziffern und gesprochener englischer Vokale konnten befriedigend gelöst werden, wobei es sich zeigte, daß gegenüber anderen Verfahren (*Backpropagation*) kein wesentlicher Vorteil zu verzeichnen ist. Bei der Approximation einer Funktion konnten dagegen sehr gute Ergebnisse erzielt werden.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Die Anwendung und die Untersuchung des Verhaltens künstlicher neuronaler Netze ist in den letzten Jahren immer weiter in den Mittelpunkt der Erforschung neuer Modelle der Informationsverarbeitung gerückt. Der Anwendungsbereich neuronaler Netze erstreckt sich von Aufgaben der Mustererkennung über die assoziative Speicherung von Informationen bis hin zu allgemeinen Problemen der Signalverarbeitung.

Die Fülle an unterschiedlichen Netztypen, Aktivierungsmodellen und Lernverfahren wächst durch die aktive Forschung ständig an. Ein neuer, vielversprechender Ansatz ist die Anwendung von Radial Basis Functions für die Aktivierung der Neuronen. Die Radial Basis Functions stammen aus der Theorie der klassischen Approximationstechniken. Sie eignen sich besonders zur Approximation mehrdimensionaler Funktionen. Diese Fähigkeit läßt sich durch die Abbildung auf neuronale Netze hervorragend für Aufgaben der Musterklassifikation nutzen. Der große Vorteil eines solchen Netzes liegt in seiner Fähigkeit, einen großen Teil des für das Netztraining notwendigen Lernaufwands in einem einzigen, umfassenden Initialisierungsschritt zu bewältigen. Hierbei wird die Möglichkeit genutzt, daß Approximationsaufgaben, die mit Radial Basis Functions angegangen werden, durch die Lösung eines einfachen linearen Gleichungssystems bewerkstelligt werden können.

Die vorliegende Arbeit beschreibt die Implementierung und Anwendung verallgemeinerter (generalized) Radial Basis Functions in einem Simulator für neuronale Netze.

Im ersten Teil der Arbeit werden die mathematischen Grundlagen der Approximation mehrdimensionaler Funktionen mit Radial Basis Functions besprochen. Hierbei wird sowohl die Herleitung der direkten Lösung des Approximationsproblems als auch der allgemeine Lösungsweg über den Ansatz eines Problems der Variationsrechnung angesprochen.

Anschließend erfolgt die Beschreibung der Abbildung dieses Verfahrens auf neuronale Netze sowie eine Gegenüberstellung mit herkömmlichen feedforward–Netzen. Die Initialisierung des Netzes bzw. die Lösung des Approximationsproblems ist an gewisse Anfangsbedingungen geknüpft. Da diese im allgemeinen nicht optimal sind, wird ein Trainingsverfahren zur Verbesserung des Ergebnisses hergeleitet. Hierzu wird das bekannte Verfahren des Gradientenabstiegs im Fehlerraum herangezogen.

Im weiteren Verlauf wird die Implementierung des Verfahrens innerhalb des vorhandenen Simulators für neuronale Netze, des SNNS (Stuttgarter Neuronale Netze Simulator), dargelegt. Der SNNS ist ein universell einsetzbarer Simulator für neuronale Netze. Er umfaßt einen graphischen Netzwerk–Editor sowie eine Fülle von Aktivierungsmodellen und Lernverfahren für neuronale Netze. Die neuen Funktionen, die durch die Integration der

Radial Basis Functions in den Simulator zugänglich geworden sind, werden ausführlich beschrieben. Hierbei wird auch die Vorgehensweise bei der Anwendung des neuen Verfahrens umfassend erläutert.

Anhand von drei unterschiedlichen Problemstellungen werden anschließend die Ergebnisse der Anwendung von Radial Basis Functions sowie ihre Probleme aufgezeigt. Die umfangreichste der drei beschriebenen Anwendungen ist die Erkennung handgeschriebener Ziffern. Die beiden weiteren Anwendungen sind die Klassifikation gesprochener englischer Vokale sowie die Approximation einer einfachen mathematischen Funktion.

Eine Zusammenfassung der Eigenschaften der Radial Basis Functions sowie der Vor- und Nachteile bilden den Abschluß der Arbeit. Hierbei werden auch interessante Punkte für weitere Entwicklungsmöglichkeiten im Bereich der Radial Basis Functions aufgezeigt.

Kapitel 2

Mathematische Grundlagen

In diesem Abschnitt wird die Anwendung der Methode der Radial Basis Functions für die Approximation und Interpolation vorgestellt. Es wird hier jedoch nur ein kurzer Überblick über das Verfahren und seine Anwendung gegeben. Für die ausführliche mathematische Herleitung sowie für den Beweis verschiedener Sätze möchte ich auf die jeweils angegebene Literatur verweisen.

2.1 Interpolation mit Radial Basis Functions

Gegeben seien N Punkte $\vec{x_i} \in \Re^n$ sowie N Funktionswerte $y_i \in \Re$ mit $i \in \{1, \ldots, N\}$. Gesucht ist eine Funktion f mit den Eigenschaften:

$$f \in \Re^n \to \Re \quad \text{und} \quad f(\vec{x_i}) = y_i \quad \text{für alle} \quad i \in \{1, \ldots, N\}$$

Für die Interpolation mit Radial Basis Functions gilt weiterhin die Bedingung, daß f einen *Punkt* in einem Funktionsraum der Dimension N darstellt, wobei die elementaren Basisfunktionen dieses Raumes von der Wahl der Stützstellen $\vec{x_i}$ abhängen. Die Funktion f ist somit eine Linearkombination dieser Basis.
Die Basis des Raumes ist definiert als:

$$\mathcal{B} = \{h(|\vec{x} - \vec{x_i}|) \quad | \quad i = 1, \ldots, N\}$$

$h \in \Re^+ \to \Re$ ist die eigentliche radiale Basisfunktion, die der Methode ihren Namen gegeben hat. Sie ist für positive, skalare Argumente definiert. Im weiteren Verlauf dieser Arbeit soll h die „Zentrumsfunktion“ heißen. Außerdem wird die folgende abkürzende Schreibweise eingeführt:

$$h_i(\vec{x}) :\Longleftrightarrow h(|\vec{x} - \vec{x_i}|)$$

Der Betrag $|\cdot|$, der in der Definition der Zentrumsfunktion auftaucht, ist eine beliebige Norm. Üblicherweise wird die $\mathcal{L}_2$–Norm verwendet, die dem euklidischen Abstand der Punkte $\vec{x}$ und $\vec{x_i}$ im Vektorraum $\Re^n$ entspricht.

Die obigen Bedingungen für die Funktion f führen auf die folgende Darstellung:

$$f(\vec{x}) = \sum_{i=1}^{N} c_i h_i(\vec{x}) \tag{2.1}$$

Im allgemeinsten Fall werden bei der Interpolation mit Radial Basis Functions Polynome zur Funktion f addiert. Hieraus ergibt sich die allgemeine Darstellung:

$$f(\vec{x}) = \sum_{i=1}^{N} c_i h_i(\vec{x}) + \sum_{i=1}^{m} d_i p_i(\vec{x}) \qquad m \leq n \tag{2.2}$$

Die Funktionen p_i bilden hierbei eine Basis des Polynomraums $\pi_{k-1}(\Re^n)$, d.h. des Raums der Polynome aus $\Re^n \rightarrow \Re$ mit maximalem Grad $k-1$ bei vorgegebenem k. Für bestimmte Zentrumsfunktionen h ist diese Ergänzung notwendig, um das Interpolationsproblem lösen zu können. Außerdem kann die Funktion f durch die Addition der Polynome insgesamt geglättet werden, so daß die Amplitude unerwünschter Schwingungen zwischen den einzelnen Stützstellen $\vec{x_i}$ möglichst gering wird. Wählt man als Zentrumsfunktion h den positiven Bereich einer Gaußschen Glockenkurve, so kann gezeigt werden, daß das Interpolationsproblem ohne zusätzliche Polynome lösbar ist.

Für die Lösung des Problems müssen die Koeffizienten c_i und d_i bestimmt werden. Es sind also $N + m$ Unbekannte zu bestimmen. Aus den Interpolationsbedingungen folgen direkt die ersten N Bestimmungsgleichungen:

$$\sum_{i=1}^{N} c_i h_i(\vec{x_j}) + \sum_{i=1}^{m} d_i p_i(\vec{x_j}) = y_j \quad \text{für} \quad j = 1, \ldots, N \tag{2.3}$$

Weitere m Bedingungen können frei gewählt werden. Um den weiteren Rechenweg so einfach wie möglich zu halten, wählt man:

$$\sum_{i=1}^{m} d_i p_j(\vec{x_i}) = 0 \quad \text{für} \quad j = 1, \ldots, m \tag{2.4}$$

Ein bekanntes Ergebnis aus der Approximationstheorie besagt, daß für dieses Gleichungssystem eine Lösung existiert, falls die Zentrumsfunktion h positiv definit vom Grad k auf $\Re^n$ ist ($h \in \mathcal{P}_k(\Re^n)$, siehe auch [PG89]).

Schönberg hat gezeigt, daß das lineare Gleichungssystem mit N Gleichungen, das aus der Definition 2.1 für f folgt, gelöst werden kann, falls h *strikt* positiv definit ist. Dies ist jedoch eine hinreichende, nicht aber notwendige Bedingung. Mitchelli erweiterte den Bereich für erlaubte Zentrumsfunktionen h, indem er zeigte, daß die Bedingung „h' ist vollständig monoton“ ausreichend für die Lösbarkeit dieses Gleichungssystems ist (siehe [PG89]). (Vollständige Monotonie einer Funktion g auf einem Intervall $\mathcal{I}$ bedeutet, daß g beliebig oft stetig differenzierbar ist und es gilt: $g'(x) \leq 0$, $g''(x) \geq 0$, $g'''(x) \leq 0$, ... für $x \in \mathcal{I}$.)

Hierdurch ergibt sich eine große Auswahl an möglichen Zentrumsfunktionen $h(r)$. Einige typische Beispiele werden kurz aufgeführt:

1. $e^{-\left(\frac{r}{\alpha}\right)^2}$ Gaußsche Glockenkurve

2. $\dfrac{1}{(c^2+r^2)^\alpha}$, $\alpha > 0$

3. $\sqrt{(c^2+r^2)}$ multiquadratische Funktion

Die Vielzahl der möglichen Funktionen für h sagt leider noch nichts über die Zahl der wirklich brauchbaren Zentrumsfunktionen aus.

Ein großes Problem besteht z.B. darin, daß der Funktionsverlauf zwischen den Stützstellen beliebig oszillieren kann, ohne dabei die Interpolationsbedingungen zu verletzen.

Außerdem stellt sich die Frage, ob eine exakte Interpolation überhaupt sinnvoll ist. Oft dienen Meßwerte, die naturgemäß verrauscht sind, als Stützstellen. Anstelle einer exakten Interpolation besteht der eigentliche Wunsch also eher darin, eine möglichst glatte Approximation der Stützstellen zu finden.

Weiterhin fällt die Tatsache auf, daß die Größe des linearen Gleichungssystems direkt von der Zahl der Stützstellen abhängig ist (N). Wünschenswert wäre dagegen, daß die Zahl der Gleichungen unabhängig bleibt; nicht zuletzt deshalb, da sonst der Aufwand für die Lösung des Systems ab einer bestimmten Zahl von Meßwerten einfach nicht mehr akzeptabel ist.

Alle diese Gründe führen dazu, daß ein allgemeinerer Lösungsansatz gewählt wird, der auf einer festen Anzahl von Gleichungen aufbaut. Er wird im nächsten Abschnitt besprochen.

2.2 Verallgemeinerte Radial Basis Functions

Unabhängig von der Zahl der Stützstellen wird nun eine feste Anzahl K von Zentren bzw. Knoten vorgegeben, die als Mittelpunkte für die Zentrumsfunktion dienen sollen. Diese Zentren können eine Untermenge der Stützstellen oder aber unabhängig von ihnen sein. Aus diesem Grunde werden die Zentren allgemein mit $\vec{t_i}$ bezeichnet ($i \in \{1, \ldots, K\}$). Die gesuchte Funktion f hat dann die allgemeine Form:

$$f(\vec{x}) = \sum_{i=1}^{K} c_i h(|\vec{x} - \vec{t_i}|) \tag{2.5}$$

Für die Berechnung der K Koeffizienten c_i stehen N Bestimmungsgleichungen zur Verfügung, die sich wiederum direkt aus den Interpolationsbedingungen ergeben:

$$\sum_{i=1}^{K} c_i h(|\vec{x_j} - \vec{t_i}|) = y_j \quad \text{für} \quad j = 1, \ldots, N$$

Durch die folgende Einführung der Vektoren $\vec{c}$ und $\vec{y}$ sowie der Matrix H als

$$\vec{c} = \begin{pmatrix} c_1 \\ \vdots \\ c_K \end{pmatrix}, \quad \vec{y} = \begin{pmatrix} y_1 \\ \vdots \\ y_N \end{pmatrix}, \quad H = \begin{pmatrix} h(|\vec{x_1} - \vec{t_1}|) & \cdots & h(|\vec{x_1} - \vec{t_K}|) \\ \vdots & \ddots & \vdots \\ h(|\vec{x_N} - \vec{t_1}|) & \cdots & h(|\vec{x_N} - \vec{t_K}|) \end{pmatrix}$$

läßt sich das Gleichungssystem in Vektorschreibweise angeben:

$$H \cdot \vec{c} = \vec{y}$$

Für den üblichen Fall $N > K$ ist dieses lineare Gleichungssystem überbestimmt und kann nicht exakt gelöst werden. Es ist deshalb notwendig, eine geeignete Näherungslösung zu finden. Der Ansatz, die Summe der Quadrate der Fehler in den N Stützstellen zu minimieren, führt auf die Lösung:

$$\vec{c} = H^+ \cdot \vec{y} \tag{2.6}$$

H^+ ist hierbei die sogenannte „Moore–Penrose Pseudoinverse“ der Matrix H, die sich ergibt zu:

$$H^+ = \left(H^T \cdot H\right)^{-1} \cdot H^T \tag{2.7}$$

Aus einem Theorem von Mitchelli folgt, daß die notwendige Matrixinversion durchgeführt werden kann, falls die Ableitung der Zentrumsfunktion h' vollständig monoton ist und die Zentren $\vec{t_i}$ eine Untermenge der Stützstellen $\vec{x_j}$ bilden (siehe [PG89]). Broomhead zeigte in speziellen Fällen, daß dieses Verfahren auch bei unabhängig gewählten $\vec{t_i}$ anwendbar ist ([BL88]).

Ein anderer Weg, eine Lösung für das Approximationsproblem zu finden, besteht in der Anwendung der Variationsrechnung. Bei dieser Methode ist u.a. auch die Möglichkeit gegeben, die Glattheit der gefundenen Lösung zu beeinflussen. Der Ansatz für das Variationsproblem lautet:

$$H[f] = \sum_{i=1}^{N} \left(y_i - f(x_i)\right)^2 + \lambda \|Pf\|^2 \tag{2.8}$$

Gesucht ist nun eine Funktion f, die das Funktional $H[f]$ minimiert. Die Summe, die in diesem Funktional auftaucht, beinhaltet die Interpolationsbedingungen. Je kleiner der Wert dieser Summe ist, desto näher liegt f an den vorgegebenen Stützstellen. P ist ein Differential–Operator und wirkt als Stabilisator für das Variationsproblem. Durch P werden Eigenschaften der gesuchten Funktion bestimmt, die *a priori* bekannt sind. Im vorliegenden Fall soll durch P die gewünschte Glattheit der Funktion ausgedrückt werden. $\|\cdot\|$ ist eine Norm des Funktionsraums, in welchem f liegt, und λ stellt den Variationsparameter dar, der den Einfluß des Stabilisators gegenüber den Interpolationsbedingungen reguliert.

Poggio und Girosi beschreiben in [PG89] ausführlich die Lösung des Variationsproblems. Für die vorliegende Arbeit ist der Weg der Herleitung jedoch weniger relevant. Das Ergebnis des Variationsproblems ist dagegen sehr interessant. Ich möchte mich deshalb bei der Darstellung auf die abschließenden Schritte der Herleitung und auf das Endergebnis beschränken:

Die Wahl eines geeigneten Operators P und die Lösung der Euler–Lagrangeschen Differentialgleichungen führt auf einen Lösungsansatz für die Funktion f von folgender Form:

$$f(\vec{x}) = \sum_{i=1}^{K} c_i g(\vec{x}, \vec{t_i}) \tag{2.9}$$

Diese Form entspricht dem Ansatz 2.5, der bei der direkten Rechnung verwendet wurde. Die Funktion g, die aus der Lösung des Variationsproblems folgt, ist abhängig von der Wahl des Operators P. Wählt man P derart, daß der Stabilisator $\|Pf\|^2$ translations- und rotationsinvariant wird, so besitzt g die Form einer Zentrumsfunktion, deren Wert nur noch vom euklidischen Abstand der Vektoren $\vec{x}$ und $\vec{t_i}$ im $\Re^n$ abhängt. Für g können durch spezielle Wahl von P z.B. Funktionen gefunden werden, die zu den in Abschnitt 2.1 aufgeführten Beispielen für Zentrumsfunktionen h äquivalent sind.

Im Ansatz 2.9 ist jedoch noch die Wahl der Koeffizienten c_i offen, die nun gemäß der gewünschten Glattheit bestimmt werden sollen. Für den Stabilisatorterm ergibt sich, ebenfalls durch die Wahl eines geeigneten Operators P, die folgende, relativ einfache Form:

$$\|Pf\|^2 = \sum_{i=1}^{K} \sum_{j=1}^{K} c_i c_j g(\vec{t_i}, \vec{t_j}) \tag{2.10}$$

Führt man nun die beiden Matrizen G und $G_\square$ wie folgt ein, so läßt sich das Funktional 2.8 sehr einfach in Vektorschreibweise angeben.

$$G = \begin{pmatrix} g(\vec{x_1}, \vec{t_1}) & \cdots & g(\vec{x_1}, \vec{t_K}) \\ \vdots & \ddots & \vdots \\ g(\vec{x_N}, \vec{t_1}) & \cdots & g(\vec{x_N}, \vec{t_K}) \end{pmatrix}, \quad G_\square = \begin{pmatrix} g(\vec{t_1}, \vec{t_1}) & \cdots & g(\vec{t_1}, \vec{t_K}) \\ \vdots & \ddots & \vdots \\ g(\vec{t_K}, \vec{t_1}) & \cdots & g(\vec{t_K}, \vec{t_K}) \end{pmatrix}$$

Setzt man den gefundenen Ansatz 2.9 für f sowie den Stabilisator aus Gleichung 2.10 in das Funktional 2.8 ein und bedient sich dabei der Vektorschreibweise, so erhält man:

$$\begin{aligned}
H[f] &= \sum_{i=1}^{N} (y_i - f(x_i))^2 + \lambda \|Pf\|^2 \\
&= \sum_{i=1}^{N} \left(y_i - \sum_{j=1}^{K} c_j g(\vec{x_i}, \vec{t_j}) \right)^2 + \lambda \sum_{i=1}^{K} \sum_{j=1}^{K} c_i c_j g(\vec{t_i}, \vec{t_j}) \\
&= \sum_{i=1}^{N} \left(y_i^2 - 2y_i \sum_{j=1}^{K} c_j g(\vec{x_i}, \vec{t_j}) + \left(\sum_{j=1}^{K} c_j g(\vec{x_i}, \vec{t_j}) \right)^2 \right) + \lambda \sum_{i=1}^{K} \sum_{j=1}^{K} c_i c_j g(\vec{t_i}, \vec{t_j}) \\
&= \sum_{i=1}^{N} y_i^2 - 2 \sum_{i=1}^{N} \left(y_i \sum_{j=1}^{K} c_j g(\vec{x_i}, \vec{t_j}) \right) + \sum_{i=1}^{N} \left(\sum_{j=1}^{K} c_j g(\vec{x_i}, \vec{t_j}) \right)^2 + \lambda \sum_{i=1}^{K} \sum_{j=1}^{K} c_i c_j g(\vec{t_i}, \vec{t_j}) \\
&= \vec{y} \cdot \vec{y} \quad - \quad 2\vec{c} \cdot G^T \cdot \vec{y} \quad + \quad \vec{c} \cdot (G^T \cdot G) \cdot \vec{c} \quad + \quad \lambda \vec{c} \cdot G_\square \cdot \vec{c} \\
&= \vec{c} \cdot (G^T \cdot G + \lambda G_\square) \cdot \vec{c} - 2\vec{c} \cdot G^T \cdot \vec{y} + \vec{y} \cdot \vec{y}
\end{aligned}$$

Gesucht ist nun ein Minimum dieses Funktionals in Abhängigkeit des Koeffizientenvektors $\vec{c}$. Durch Ableiten des Funktionals nach $\vec{c}$, Nullsetzen der Ableitung und Auflösen nach $\vec{c}$ erhält man schließlich als Endergebnis:

$$\frac{\partial H[f]}{\partial \vec{c}} = 2\vec{c} \cdot (G^T \cdot G + \lambda G_\Box) - 2G^T \cdot \vec{y} \stackrel{!}{=} 0$$

$$\Longleftrightarrow \qquad \vec{c} = (G^T \cdot G + \lambda G_\Box)^{-1} \cdot G^T \cdot \vec{y} \tag{2.11}$$

Vergleicht man diese Lösung mit den Gleichungen 2.6 und 2.7 und beachtet hierbei die Äquivalenz der Matrizen G und H, so unterscheiden sich die beiden Lösungen nur durch den zusätzlichen Term $\lambda G_\Box$. Wählt man $\lambda = 0$, so erhält man exakt die gleiche Lösung wie bei der direkten Rechnung. Der Parameter λ übernimmt in der neuen Lösung die Einstellung der Glattheit von f. Da bei der Lösung des Variationsproblems keine Annahmen über die Wahl und die Anzahl der Zentren gemacht wurden (außer $N \geq K$), ist hierdurch auch das spezielle Ergebnis von Broomhead bestätigt.

Die endgültige Lösung des Approximationsproblems mit verallgemeinerten Radial Basis Functions läßt sich also wie folgt zusammenfassen:

Gegeben seien N Stützstellen $\vec{x_i} \in \Re^n$ sowie die dazugehörigen Funktionswerte $y_i \in \Re$. Außerdem seien K Zentren $\vec{t_i} \in \Re^n$ gegeben, die beliebig gewählt werden können, sowie der Grad für die Glattheit λ.
Die gesuchte Funktion f lautet dann:

$$f(\vec{x}) = \sum_{i=1}^{K} c_i g(\vec{x}, \vec{t_i})$$

Hierbei gilt:

$$g(\vec{x}, \vec{t_i}) = h(|\vec{x} - \vec{t_i}|)$$

$$\vec{c} = (G^T \cdot G + \lambda G_\Box)^{-1} \cdot G^T \cdot \vec{y}$$

Die Matrizen G und $G_\Box$ sowie die Vektoren $\vec{c}$ und $\vec{y}$ sind dabei wie oben angegeben definiert.

Bei diesem Verfahren ist nach wie vor die Wahl der Zentren t_i offen. Wie sich später bei der Anwendung des Verfahrens zeigen wird, ist eine günstige Wahl der Zentren jedoch entscheidend für die Qualität der Approximation. Um auch bei ungünstig gewählten Zentren ein befriedigendes Ergebnis zu erhalten, ist es notwendig, die Zentren um kleine Schritte zu verschieben. Hierbei müssen die Koeffizienten entsprechend angepaßt werden. Wie diese Aufgabe auf das Trainieren eines neuronalen Netzes abgebildet werden kann, wird im nächsten Kapitel besprochen.

Kapitel 3

Abbildung auf neuronale Netze

In diesem Kapitel wird die Abbildung von Radial Basis Functions auf neuronale Netze besprochen. Hierzu wird zunächst ein spezielles Neuron definiert, das jeweils einem der Zentren der Approximationsfunktion entspricht. Anschließend wird unter Verwendung dieses Neuronentyps der Aufbau von dreistufigen feedforward–Netzen demonstriert. Diese Netze entsprechen einer direkten Abbildung der Approximationsfunktion bzw. einer Schar von Funktionen. Die Beschreibung der Trainingsmöglichkeiten des neuronalen Netzes und ein Vergleich mit anderen mehrstufigen feedforward-Netzen bilden den Abschluß dieses Kapitels.

3.1 Ein spezielles Neuron

Typisch für Radial Basis Functions ist eine Zentrumsfunktion h, die auf den Abstand zweier Ortsvektoren in einem mehrdimensionalen Vektorraum angewendet wird. Es muß nun also eine Möglichkeit gefunden werden, diese Abstandsberechnung sowie die Zentrumsfunktion h effizient auf ein neuronales Netz abzubilden. Hierbei sollen die allgemeinen Eigenschaften eines neuronalen Netzes erhalten bleiben:

- keine Verwendung von globalen Daten oder Methoden
- Beschränkung des Informationsflusses auf die Verbindung zwischen einzelnen Neuronen
- Möglichkeit der parallelen Aktivierung von Neuronen und der parallelen Weitergabe ihrer Aktivierung

Geht man von diesen Bedingungen aus, so ist die Abbildung auf ein neuronales Netz naheliegend: Jedes Zentrum wird durch genau ein Neuron repräsentiert. Die Aktivierung dieses Neurons entspricht dem Wert der Zentrumsfunktion h, die durch das Neuron definiert und berechnet wird.

Als Argument für die Berechnung von h ist der euklidische Abstand zwischen dem Eingabevektor $\vec{x}$ und dem Zentrumsvektor $\vec{t_j}$ notwendig. Der Zentrumsvektor ist spezifisch für das Zentrum und sollte demnach ebenfalls durch das Neuron definiert werden. Dieses Modell ist jedoch einerseits wenig anschaulich, da hierdurch die *Größe* des Neurons (der notwendige Speicherplatz) von der Dimension des Eingabevektors abhängig ist. Andererseits ist das Modell auch ineffizient, da die gleichzeitige, parallele Aktivierung des Neurons nicht unterstützt wird.

Aus diesen Gründen wird der Zentrumsvektor $\vec{t_j}$ in die Verbindungen verlegt, die auf das Neuron zuführen. Die Zahl dieser Verbindungen entspricht genau der Dimension der Vektoren $\vec{x}$ und $\vec{t_j}$. Die einzelnen Komponenten des Eingangsvektors $\vec{x}$ dienen als Eingangsaktivierungen der Verbindungen. Da bei Verbindungen im allgemeinen von Verbindungsgewichten gesprochen wird, sollen auch hier die einzelnen Komponenten des Zentrumsvektors als Gewichte bezeichnet werden, weil sie durch die Verbindung selbst repräsentiert werden. Der euklidische Abstand der beiden Vektoren wird nun einfach dadurch berechnet, daß jeweils das Quadrat der Differenz zwischen Eingangsaktivierung und Verbindungsgewicht berechnet wird. Diese Quadrate werden durch das Neuron aufsummiert. Die Wurzel der Summe ist dann gleich dem gewünschten Abstand und dient als Argument für die Zentrumsfunktion. Abbildung 3.1 veranschaulicht die Funktion eines solchen Neurons:

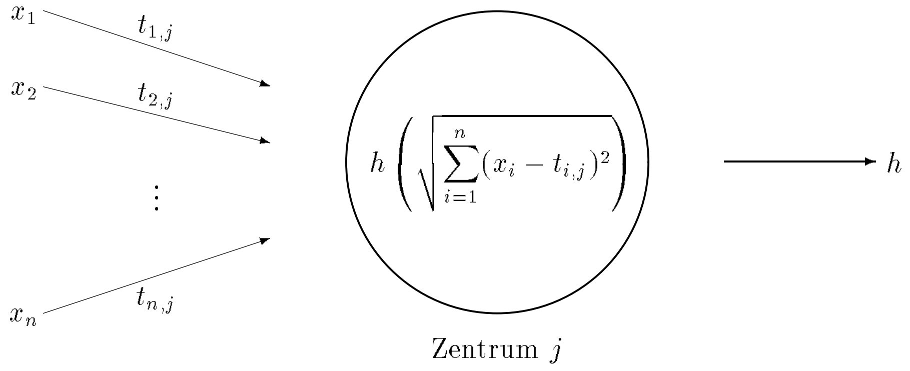

Abbildung 3.1: Berechnung der Zentrumsfunktion durch ein Neuron

Die in der Abbildung auftretenden Werte x_i entsprechen jeweils der i–ten Komponente des Eingangsvektors $\vec{x}$. Ebenso entsprechen die Werte $t_{i,j}$ den Komponenten des Zentrumsvektors $\vec{t_j}$.

In der endgültigen Implementierung des Netzes kann das Ziehen der Quadratwurzel in die eigentliche Zentrumsfunktion h verlegt werden. Dies kann für bestimmte Zentrumsfunktionen sehr vorteilhaft sein. Verwendet man z.B. eine Gaußsche Glockenkurve, so wird die Wurzel und ein Quadrat eingespart.

3.2 Der Aufbau von Netzen

3.2.1 Einfache, reellwertige Funktionen

Nachdem nun ein Neuron definiert wurde, welches genau einem Zentrum der approximierenden Funktion entspricht, ist die Konstruktion eines vollständigen neuronalen Netzes naheliegend. Abbildung 3.2 zeigt ein Netz, das die Funktion $f \in \Re^n \rightarrow \Re$ definiert, wobei K Zentren verwendet werden:

$$f(\vec{x}) = \sum_{i=1}^{K} c_i h(|\vec{x} - \vec{t_i}|)$$

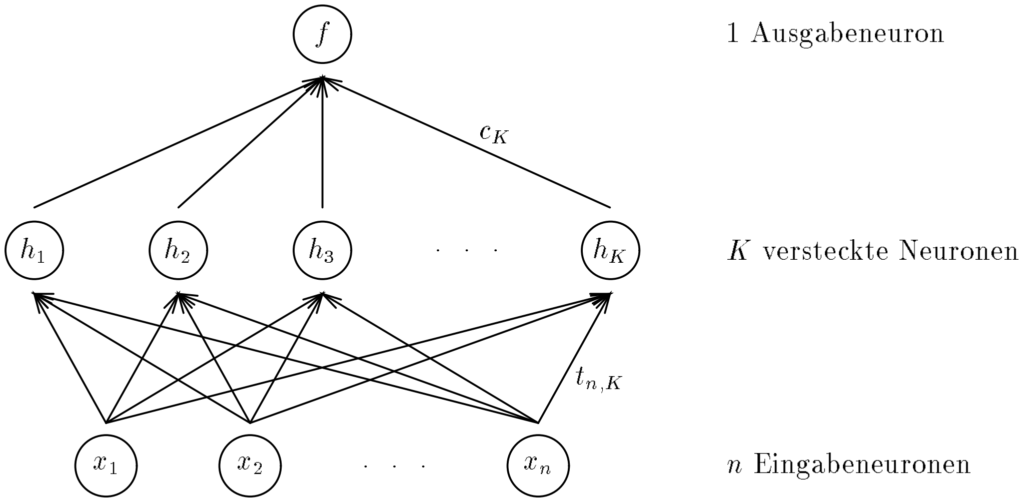

Abbildung 3.2: Abbildung von f auf ein neuronales Netz

Die Neuronen sind durch Kreise dargestellt, deren Beschriftung der Ausgabe des jeweiligen Neurons entspricht. Die Pfeile geben die Verbindungen zwischen den Neuronen an. Aus Gründen der Übersichtlichkeit sind nicht alle Verbindungen mit ihrem Gewicht beschriftet.

Die einzige Aufgabe der Eingabeschicht (Eingabeneuronen) besteht darin, den angelegten Vektor $\vec{x}$ an alle Neuronen der nächsten Ebene weiterzugeben. Die erste und die zweite Ebene sind also vollständig miteinander verbunden. Die mittlere Schicht (versteckte Neuronen) besteht aus Neuronen von dem Typ, wie sie im letzten Abschnitt definiert wurden. Hier werden die einzelnen Zentrumsfunktionen h berechnet. Die Verbindungsgewichte zwischen der mittleren und der Ausgabeschicht entsprechen den Koeffizienten c_i, die in der Definition von f auftauchen. Das einzige Ausgabeneuron (Ausgabeneuron) summiert die mit c_i gewichteten Zentren auf und bildet damit den Funktionswert von f.

3.2.2 Scharen von Funktionen

Das Verfahren der Radial Basis Functions läßt sich ohne Probleme auf Funktionen $\vec{f} \in \Re^n \rightarrow \Re^m$ übertragen. Dies ist äquivalent mit der Approximation einer Funktionenschar, die aus m einzelnen Funktionen $f_i \in \Re^n \rightarrow \Re$ besteht. Hierbei wird vorausgesetzt, daß die Stützstellen $\vec{x_i}$ sowie die Position der Zentren $\vec{t_i}$ für alle Funktionen der Schar dieselben sind. Betrachtet man die Formel 2.11 aus dem letzten Kapitel, so sieht man, daß die Berechnung der Koeffizienten c_i für verschiedene Funktionen f_i unter diesen Voraussetzungen nur noch vom jeweiligen Vektor $\vec{y}$ abhängig ist. Der komplizierte Matrizenausdruck, der ebenfalls in dieser Formel auftaucht, ist für alle Funktionen der Schar identisch und braucht deshalb nur ein einziges Mal berechnet zu werden.

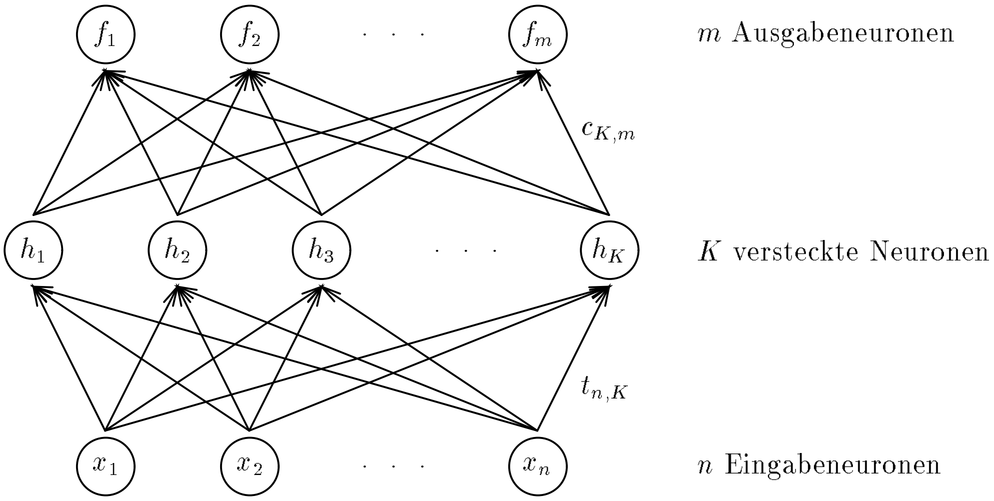

Abbildung 3.3: Abbildung von mehreren Funktionen auf ein neuronales Netz

3.2.3 Ergänzung um linearen Anteil

In der direkten mathematischen Herleitung der Approximation mit Radial Basis Functions wurde erwähnt, daß es für bestimmte Zentrumsfunktionen h notwendig ist, einen weiteren Term zur Funktion f zu addieren, um optimale Ergebnisse zu erhalten. Dieser Term sollte ein Polynom in $\vec{x}$ vom Grad „kleiner oder gleich k" sein. Für $k = 1$ erhält f folgende spezielle Form:

$$f(\vec{x}) = \sum_{i=1}^{K} c_i h_i(\vec{x}) + \sum_{i=1}^{n} d_i x_i + b$$

Diese einfache Form der Funktion f mit zusätzlichem linearen Anteil des Eingabevektors läßt sich nun ebenfalls problemlos auf ein Netz abbilden. Abbildung 3.4 zeigt ein solches Netz. Die Erweiterung auf eine Schar von Funktionen ist hier ebenfalls möglich. Ihre Darstellung als Netz wäre jedoch sehr unübersichtlich und wird deshalb nicht in einer Abbildung wiedergegeben. Die anschließende Implementierung in den Simulator für neuronale Netze, SNNS, umfaßt dagegen alle Kombinationsmöglichkeiten.

Die Terme $d_i x_i$ entstehen durch direkte Verbindungen zwischen der Eingabeschicht und dem Ausgabeneuron. Die Gewichte dieser Verbindungen entsprechen den Koeffizienten d_i.

Der konstante Anteil b wird durch ein spezielles Neuron realisiert, dessen Ausgabewert konstant 1 beträgt. Das Verbindungsgewicht zwischen diesem Neuron und dem Ausgabeneuron entspricht dem Wert von b. In der tatsächlichen Realisierung wird dieses spezielle Neuron jedoch eingespart. Seine Funktion wird dann durch einen *Bias* in der Aktivierungsfunktion des Ausgabeneurons übernommen.

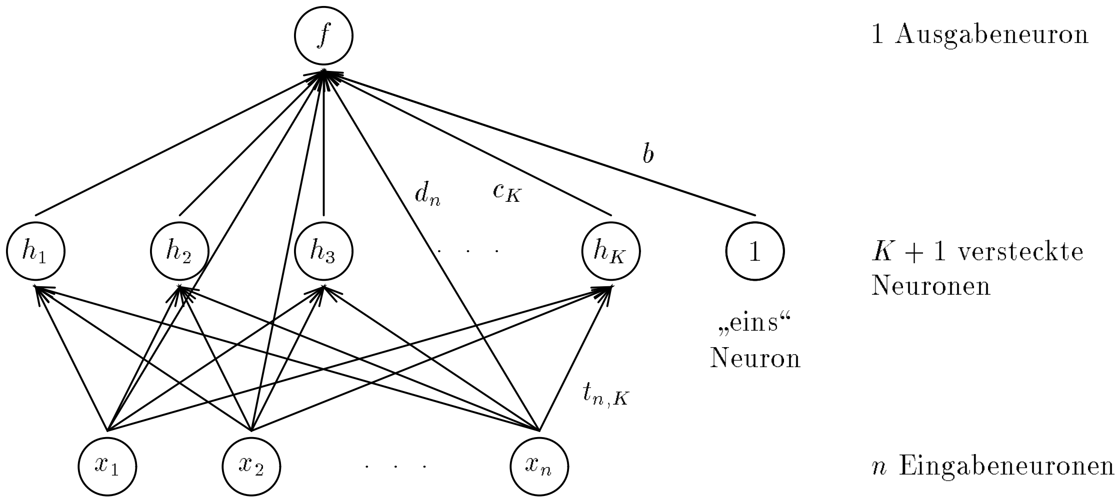

Abbildung 3.4: Funktion f mit linearem Anteil

3.2.4 Weitere Erweiterungen des Modells

Wird ein neuronales Netz zur Musterklassifikation verwendet, so werden die Ausgabeneuronen in der Regel über eine sigmoide Funktion aktiviert. Bei der Anwendung von Radial Basis Functions für Approximationsaufgaben ist es hingegen üblich, direkt den Funktionswert von f als Ausgabe zu verwenden. Dies entspricht einer linearen Aktivierungsfunktion im Ausgabeneuron. Für Klassifikationsaufgaben erhält man — auch bei Verwendung von Radial Basis Functions — bessere Ergebnisse, wenn die Aktivierung der Ausgabeneuronen sigmoid verläuft. Im allgemeinen Fall soll also auf $f(\vec{x})$ eine Funktion $\sigma \in \Re \rightarrow \Re$ angewendet werden.

Die sogenannten *Hyper Basis Functions* stellen eine weitere Erweiterung der verallgemeinerten Radial Basis Functions dar. Hier wird durch eine Gewichtung des Arguments $|\vec{x} - \vec{t_j}|$ der Zentrumsfunktion h jedes Zentrum individuell beeinflußt. Diese Gewichtung kann auf verschiedene Arten erfolgen:

- individuelle Gewichtung der einzelnen Terme $(x_i - t_{i,j})$ für jede der i Komponenten des Zentrumsvektors
- Gewichtung des euklidischen Abstands $|\vec{x} - \vec{t_j}|$ durch einen einzelnen Parameter
- Veränderung eines weiteren Parameters p, der in die Zentrumsfunktion h aufgenommen wird.

Die erste Möglichkeit erfordert einen zusätzlichen Parameter für jede Verbindung zwischen der ersten und zweiten Ebene des Netzes. Die zweite und die dritte Möglichkeit sind nahezu gleichbedeutend, abhängig davon, in welcher Weise ein zusätzlicher Parameter p in die Funktion h eingebaut wird. Im dritten Fall ist es z.B. möglich, das Argument von h mit p

zu multiplizieren. Hierdurch wird der gleiche Effekt erreicht, der durch die zweite angegebene Möglichkeit erzielt wird. In die vorliegende Arbeit wurde deshalb die Möglichkeit der Veränderung eines Parameters p der Funktion h mit aufgenommen. Wählt man für p in allen Zentren den gleichen Wert, so erhält man wieder die Funktionalität der verallgemeinerten Radial Basis Functions .

Faßt man nun alle aufgeführten Erweiterungen zusammen (Schar von Funktionen f, linearer Anteil von $\vec{x}$ in f, sigmoide Aktivierung der Ausgabeneuronen durch σ und Parameter p für h), so erhält man ein neuronales Netz mit folgender Funktionalität:

$$o_k(\vec{x}) = \sigma\left(\sum_{j=1}^{K} c_{j,k} h\left(|\vec{x}-\vec{t_j}|, p_j\right) + \sum_{i=1}^{n} d_{i,k} x_i + b_k\right) = \sigma\left(f_k(\vec{x})\right) \ , \quad k = 1, \ldots, m$$

Diese Formel beschreibt das Verhalten eines dreistufigen neuronalen Netzes mit n Eingabeneuronen, K Neuronen in der mittleren Ebene und m Neuronen in der Ausgabeschicht. Es besteht eine vollständige Verbindung zwischen erster und zweiter, zweiter und dritter sowie zwischen erster und dritter Schicht des Netzes. Die Aktivierung des i–ten Ausgabeneurons wird durch den Funktionswert von $o_i(\vec{x})$ bestimmt, wobei $\vec{x}$ der Vektor der Eingangsaktivierungen $x_i, \ldots, x_n$ ist.

3.3 Training des fertigen Netzes

Für ein weiteres Training des vollständig aufgebauten neuronalen Netzes existieren die folgenden Gründe:

- Die Wahl der Zentren $\vec{t_j}$ ist im allgemeinen nicht optimal. Indem die Vektoren $\vec{t_j}$ verschoben werden, wobei die Koeffizienten $c_{j,k}$ ebenfalls verändert werden, kann sich das Ergebnis wesentlich verbessern.

- Die Gewichte der Verbindungen zwischen Eingabe- und Ausgabeschicht werden nicht direkt berechnet. Sie müssen, beginnend mit sehr kleinen Werten, nachträglich gelernt werden.

- Bei der Berechnung der inversen Matrix treten bei großen Matrizen numerische Probleme auf. Die Gewichte zwischen mittlerer und Ausgabeschicht sind deshalb nicht optimal und können durch nachträgliches Training verbessert werden. Ebenso gehen die speziellen Eigenschaften einer eventuell vorhandenen Funktion σ nicht in die direkte Berechnung dieser Gewichte mit ein. Stattdessen wird ein sehr einfaches Modell von σ für die Berechnung der $c_{j,k}$ angewendet (siehe Abschnitt 4.1.2).

- Die Wahl des Parameters p_j in den Zentrumsfunktionen ist ebenso wie die Wahl der Zentren $\vec{t_j}$ nicht zwangsläufig optimal. Auch die Parameter p können also gegebenenfalls durch Lernen verbessert werden.

Ziel des Trainings ist es, den Fehler des Netzes möglichst klein zu machen. Um die hierzu notwendigen Schritte zu beschreiben, wird zunächst eine Fehlerfunktion E definiert, die den Gesamtfehler des Netzes angibt:

$$E = \sum_{k=1}^{m} \left(\sum_{i=1}^{N} (y_{i,k} - o_k(\vec{x_i}))^2 \right) \tag{3.1}$$

E ist der Fehler des Netzes mit m Ausgabeneuronen unter Verwendung von N Trainingsmustern (Stützstellen). $o_k(\vec{x_i})$ stellt die Ausgabe des Neurons k bei Anlegen des Trainingsmusters i dar. Die Differenz der Ausgabe $o_k(\vec{x_i})$ zur vorgegebenen Trainingsausgabe $y_{i,k}$ wird berechnet und quadriert. Diese Quadrate werden für alle Trainingsmuster und alle Ausgabeneuronen aufsummiert.

Das Training des Netzes erfolgt durch ein Gradientenabstiegsverfahren. Hierzu müssen alle notwendigen partiellen Ableitungen der Fehlerfunktion bestimmt werden. Die veränderlichen Parameter ($c_{j,k}$, $d_{i,k}$, b_k, p_j, $\vec{t_j}$) werden dann abhängig von ihren Ableitungen um einen kleinen Betrag verändert, so daß E insgesamt kleiner wird. Für eine übersichtliche Darstellung der partiellen Ableitungen wird zunächst folgende Schreibweise eingeführt:

$$\Delta_{i,k} := y_{i,k} - o_k(\vec{x_i}) \quad \text{Fehler bei Trainingsmuster } i \text{ am Ausgabeneuron } k$$

Außerdem wird ab sofort das Quadrat des Abstandes der Ortsvektoren $\vec{x}$ und $\vec{t_j}$ als Argument für die Zentrumsfunktion h verwendet. Dies geschieht aus zwei Gründen:

- Oft werden Zentrumsfunktionen verwendet, die ohnehin das Quadrat des Abstandes verwenden (z.B. Gaußsche Glockenkurve).
- Die Ableitungen von h lassen sich einfacher angeben.

Damit keine Inkonsistenzen in der Bezeichnung der Funktionen auftreten, wird eine neue Zentrumsfunktion z wie folgt eingeführt:

$$h\left(|\vec{x} - \vec{t}|, p\right) =: z\left(|\vec{x} - \vec{t}|^2, p\right) = z(q, p)$$

Zunächst wird jetzt allgemein die partielle Ableitung der Zentrumsfunktion z nach $\vec{t}$ berechnet, und zwar unter der Voraussetzung, daß p unabhängig von $\vec{t}$ ist:

$$\frac{\partial z(q,p)}{\partial \vec{t}} = \frac{\partial z(q,p)}{\partial q}\frac{\partial q}{\partial \vec{t}} = \frac{\partial |\vec{x} - \vec{t}|^2}{\partial \vec{t}}\frac{\partial z(q,p)}{\partial q} = -2(\vec{x} - \vec{t})\frac{\partial z(q,p)}{\partial q} = -2(\vec{x} - \vec{t})z_q$$

$$\text{wobei} \quad \frac{\partial |\vec{x} - \vec{t}|^2}{\partial \vec{t}} = \frac{\partial}{\partial \vec{t}}\sum_{i=1}^{n}(x_i - t_i)^2 = \sum_{i=1}^{n} -2(x_i - t_i)\vec{e_i} = -2(\vec{x} - \vec{t})$$

Die partiellen Ableitungen von z nach p und q (z_p und z_q) können nicht allgemein angegeben werden. Alle anderen partiellen Ableitungen von z (nach $c_{j,k}$, $d_{i,k}$ und b_k) sind gleich null. Nun können alle, für das Gradientenabstiegsverfahren notwendigen Ableitungen der Fehlerfunktion E angegeben werden:

$$\frac{\partial E}{\partial c_{j,k}} = \sum_{i=1}^{N} 2\Delta_{i,k}\frac{\partial \Delta_{i,k}}{\partial c_{j,k}} = -2\sum_{i=1}^{N} \Delta_{i,k}\,\sigma'(f_k(\vec{x_i}))\,z\left(|\vec{x_i} - \vec{t_j}|^2, p_j\right) \tag{3.2}$$

$$\frac{\partial E}{\partial d_{j,k}} = -2\sum_{i=1}^{N} \Delta_{i,k}\, \sigma'(f_k(\vec{x_i}))\, x_{j,i} \tag{3.3}$$

$$\frac{\partial E}{\partial b_k} = -2\sum_{i=1}^{N} \Delta_{i,k}\, \sigma'(f_k(\vec{x_i})) \tag{3.4}$$

$$\begin{aligned}\frac{\partial E}{\partial \vec{t_j}} &= 4\sum_{k=1}^{m}\left(c_{j,k}\sum_{i=1}^{N} \Delta_{i,k}\, \sigma'(f_k(\vec{x_i}))\, z_q\left(|\vec{x_i}-\vec{t_j}|^2, p_j\right)(\vec{x_i}-\vec{t_j})\right)\\ &= 4\sum_{i=1}^{N}\left(z_q\left(|\vec{x_i}-\vec{t_j}|^2, p_j\right)(\vec{x_i}-\vec{t_j})\sum_{k=1}^{m} c_{j,k}\Delta_{i,k}\, \sigma'(f_k(\vec{x_i}))\right)\end{aligned} \tag{3.5}$$

$$\begin{aligned}\frac{\partial E}{\partial p_j} &= -2\sum_{k=1}^{m}\left(c_{j,k}\sum_{i=1}^{N} \Delta_{i,k}\, \sigma'(f_k(\vec{x_i}))\, z_p\left(|\vec{x_i}-\vec{t_j}|^2, p_j\right)\right)\\ &= -2\sum_{i=1}^{N}\left(z_p\left(|\vec{x_i}-\vec{t_j}|^2, p_j\right)\sum_{k=1}^{m} c_{j,k}\Delta_{i,k}\, \sigma'(f_k(\vec{x_i}))\right)\end{aligned} \tag{3.6}$$

Für die Anwendung des Gradientenverfahrens werden die veränderbaren Parameter abhängig von ihren partiellen Ableitungen angepaßt. Hierzu muß aber noch die sogenannte Lernrate η für die einzelnen Parameter definiert werden. Da die Gewichte $c_{j,k}$, $d_{i,k}$, und b_k qualitativ die gleiche Funktion haben, wird für diese drei Parameter eine gemeinsame Lernrate verwendet. Die Lernraten von p_j und $\vec{t_j}$ werden dagegen individuell festgelegt:

$$\Delta\vec{t_j} = -\eta_1\frac{\partial E}{\partial \vec{t_j}} \quad , \quad \Delta p_j = -\eta_2\frac{\partial E}{\partial p_j}$$

$$\Delta c_{j,k} = -\eta_3\frac{\partial E}{\partial c_{j,k}} \quad , \quad \Delta d_{i,k} = -\eta_3\frac{\partial E}{\partial d_{i,k}} \quad , \quad \Delta b_k = -\eta_3\frac{\partial E}{\partial b_k}$$

Beim Trainieren von neuronalen Netzen ist es oft üblich, die veränderbaren Gewichte nach der Präsentation jedes einzelnen Trainingsmusters anzupassen. Die hier angegebenen Formeln für das Training summieren dagegen die Änderungen über die gesamte Trainingsmenge auf und führen eine Gewichtsanpassung erst nach der Präsentation aller Trainingsmuster durch. In der vorliegenden Arbeit wurden beide Verfahren implementiert und getestet. Hierbei zeigte es sich, daß die Anpassung der Gewichte nach jedem einzelnen Trainingsmuster zu einem sehr instabilen Lernverhalten führt, obwohl bei anderen Lernverfahren (wie z.B. *Backpropagation*) diese Art des Trainings vorteilhaft ist.

Das hier vorgestellte Lernverfahren kann noch verbessert werden, indem ein sogenannter *Momentum–Term* mit in das Verfahren aufgenommen wird. Hierbei wird zu jeder Gewichtsänderung die mit einem Faktor versehene Gewichtsänderung des vorhergehenden Lernschritts addiert. Der gewünschte Effekt ist, daß die Geschwindigkeit der Veränderung insgesamt dann zunimmt, wenn die Änderung jeweils das gleiche Vorzeichen (bzw. die gleiche Richtung) hat. Bei unterschiedlichen Vorzeichen von Lernschritt zu Lernschritt wirkt der Momentum–Term dämpfend und vermindert somit die Änderungsgeschwindigkeit. Für die Änderung eines allgemeinen Parameters g im Lernschritt $t+1$ gilt:

$$\Delta g_{t+1} = -\eta\frac{\partial E}{\partial g} + \mu\Delta g_t$$

Eine weitere Verbesserung des Lernverhaltens kann erzielt werden, indem ein maximaler Fehler definiert wird, der noch nicht zu einer Änderung von Gewichten führt. Hierdurch kann verhindert werden, daß das Netz übertrainiert wird und seine Fähigkeit zur Generalisierung verliert. Besonders bei der Klassifikation von Mustern ist dieser Aspekt von Bedeutung, wohingegen bei einer reinen Interpolationsaufgabe der Fehler tatsächlich so gering wie möglich sein sollte.

In der vorliegenden Arbeit wurde sowohl das Verfahren des Momentum–Terms, als auch die beschriebene Fehlerbegrenzung realisiert.

3.4 Vergleich mit herkömmlichen feedforward–Netzen

Die Architektur von Netzen, die auf Radial Basis Functions basieren, unterscheidet sich auf den ersten Blick nicht wesentlich von anderen dreistufigen feedforward–Netzen. Erst bei näherer Betrachtung der einzelnen Neuronen fallen signifikante Unterschiede auf.

Den auffälligsten Unterschied bildet die Aktivierungsfunktion der Neuronen in der mittleren Schicht. Tabelle 3.1 zeigt einen Vergleich zwischen sigmoider Aktivierung und Aktivierung mit Radial Basis Functions .

Vergleich	Radial Basis Functions	Standard sigmoid
Netzeingabe	$net_j = \sqrt{\sum_{i=1}^{n}(x_i - w_i)^2}$ (Abstand)	$net_j = \sum_{i=1}^{n} x_i w_i$ (Skalarprodukt)
Aktivierung	$e^{-p_j net_j^2}$	$\frac{1}{1+e^{-(net_j+p_j)}}$
Ausgabebereich	$(0,1]$	$(0,1)$

Tabelle 3.1: Vergleich: Radial Basis Function — sigmoide Aktivierung

Bei Radial Basis Functions ist es normalerweise üblich, eine Zentrumsfunktion zu wählen, die ihr Maximum bei einem Abstand von Null hat (also für den Fall, daß Eingabevektor $\vec{x}$ und Zentrumsvektor $\vec{t_j}$ identisch sind) und für größer werdende Abstände gegen Null geht. In der Tabelle wurde die Gaußfunktion verwendet, die am häufigsten als Zentrumsfunktion eingesetzt wird.

Hieraus folgt sofort, daß Netzeingaben, die sich von den gespeicherten Zentrumsvektoren stark unterscheiden, zu einer sehr geringen Aktivierung der Neuronen in der mittleren Schicht führen. Dies bedeutet wiederum, daß sich das Verhalten des Netzes bei Eingabemustern, die stark von den gelernten Mustern abweichen, relativ gut vorhersagen läßt.

Bei sigmoider Aktivierung hingegen kann eine solche allgemeine Aussage nicht gemacht werden. Abhängig davon, in welcher Richtung der Eingabevektor vom Vektor der Verbindungsgewichte abweicht, ist das Skalarprodukt positiv oder negativ. Die Ausgabe eines solchen Neurons kann also den gesamten Wertebereich der sigmoiden Funktion abdecken, was zu einem nicht vorhersagbaren Verhalten der Netzausgabe führt. In diesem Punkt ist also ein Vorteil auf Seiten der Radial Basis Functions zu sehen.

Ein weiterer Vorteil liegt in der Möglichkeit, die Verbindungsgewichte in einem einmaligen, umfangreichen Lernschritt so weit zu initialisieren, daß das Netz ohne zusätzlichen Lernaufwand bereits vernünftige Ergebnisse liefert.

In diesem Initialisierungsschritt liegt jedoch auch ein Nachteil der Radial Basis Functions verborgen: Die Initialisierung geht von einer linearen Aktivierung der Ausgabeneuronen aus. Soll hier jedoch eine sigmoide Funktion verwendet werden, wie es für Klassifikationsaufgaben sinnvoll und üblich ist, so wird die Initialisierung schwierig: Bei einer gewünschten Netzausgabe von 0.0 sollte die Netzeingabe in das Ausgabeneuron (also die Summe der gewichteten Zentrumsfunktionen) möglichst stark negativ sein. Für eine gewünschte Ausgabe von 1.0 sollte dieser Wert hingegen stark positiv sein. Es muß daher bei der Berechnung der Gewichte eine Solleingabe für die Ausgabeschicht festgelegt werden. Ideal wäre es, wenn hierfür die Umkehrfunktion der Aktivierungsfunktion der Ausgabeneuronen zur Verfügung stände. Dies kann aber im allgemeinen Fall nicht gewährleistet werden. Aus diesem Grund muß mit einem Modell gearbeitet werden, wodurch die Qualität der Initialisierung leidet. Auch kann es bei ungeschickter Wahl des Modells dazu kommen, daß die Verbindungsgewichte sehr große Werte annehmen, was im anschließenden Training zu einem sehr instabilen Lernverhalten führen kann. Die Beschreibung des implementierten Modells befindet sich in Abschnitt 4.1.2.

Mehrstufige feedforward–Netze besitzen oft mehr als eine versteckte Ebene von Neuronen. Diese Architektur ist jedoch bei der Anwendung von Radial Basis Functions wenig anschaulich, obgleich sie nicht unmöglich ist. Allerdings unterstützt die direkte Gewichtsberechnung nur eine einzige Schicht spezieller Neuronen.

Eine weitere Eigenschaft, die Radial Basis Functions von anderen feedforward–Netzen unterscheidet, ist die Tatsache, daß aufeinanderfolgende Schichten von Neuronen jeweils vollständig miteinander verbunden sind. Rezeptive Felder, also die Verbindung von kleinen Neuronengruppen miteinander, werden dagegen ebenfalls nicht vom Initialisierungsverfahren der Radial Basis Functions unterstützt. Hierin ist ein interessanter Aspekt für die weitere Erforschung dieses Verfahrens zu sehen.

Trotz der genannten Nachteile gibt es eine Reihe von Anwendungen, die mit Radial Basis Functions hervorragend bearbeitet werden können. Kapitel 5 stellt einige dieser Anwendungen vor.

Kapitel 4

Integration in SNNS

Das nun folgende Kapitel beschreibt die Integration des Verfahrens der Radial Basis Functions in den Stuttgarter Neuronale Netze Simulator, SNNS. SNNS ist ein universell einsetzbarer Simulator auf Software–Ebene. Der Aufbau des Simulators ermöglicht die problemlose Ergänzung von neuen Aktivierungsfunktionen, Lernverfahren und Initialisierungsmethoden. Eine ausführliche Beschreibung von SNNS bietet das Benutzerhandbuch ([ZMH+92]). An dieser Stelle werden deshalb nur die für das neue Verfahren spezifischen Erweiterungen vorgestellt.
Diese umfassen die folgenden Punkte:

- drei neue Aktivierungsfunktionen, die unterschiedliche Zentrumsfunktionen für die Neuronen der mittleren Schicht realisieren
- eine neue Aktivierungsfunktion für Neuronen der Ausgabeschicht
- drei unterschiedliche Initialisierungsverfahren für die direkte Berechnung von Verbindungsgewichten
- ein neues Lernverfahren, das die nachträgliche Verbesserung der Netzleistung ermöglicht

Im folgenden werden nun zunächst die Neuerungen und ihre Verwendung der Reihe nach vorgestellt. Anschließend wird die Vorgehensweise bei der Anwendung des neuen Verfahrens besprochen. Den Abschluß dieses Kapitels bildet eine Betrachtung der Implementierungsdetails.

4.1 Neuerungen im SNNS

4.1.1 Aktivierungsfunktionen

Für die Anwendung der Radial Basis Functions wurden drei verschiedene Zentrumsfunktionen implementiert. Als Argumente dieser Funktionen wurden einheitlich das Quadrat des Abstandes $r^2 = |\vec{x} - \vec{t}|^2$ sowie ein weiterer Parameter p gewählt. Die Vektoren $\vec{x}$ und $\vec{t}$ ergeben sich aus den Aktivierungen und den Gewichten der auf das entsprechende Neuron zuführenden Verbindungen. Der Parameter p wird durch den Bias (Schwellwert) des Neurons übernommen.

Die folgenden Zentrumsfunktionen wurden implementiert:

- `Act_RBF_Gaussian` — die Gaußsche Glockenkurve

$$z(r^2, p) = z(q, p) = e^{-pq} \quad \text{mit} \quad q = |\vec{x} - \vec{t}|^2$$

- `Act_RBF_MultiQuadratic` — die multiquadratische Funktion

$$z(r^2, p) = z(q, p) = \sqrt{p + q} \quad \text{mit} \quad q = |\vec{x} - \vec{t}|^2$$

- `Act_RBF_ThinPlateSpline` – die Funktion der *thin plate splines*

$$\begin{aligned} z(r^2, p) = z(q, p) &= p^2 q \ln(p\sqrt{q}) \quad \text{mit} \quad q = |\vec{x} - \vec{t}|^2 \\ &= (pr)^2 \ln(pr) \quad \text{mit} \quad r = |\vec{x} - \vec{t}| \end{aligned}$$

Abbildung 4.1 zeigt das Auswahlmenü der Aktivierungsfunktionen des SNNS. Die am häufigsten verwendete Zentrumsfunktion wurde ausgewählt und erscheint in inverser Darstellung.

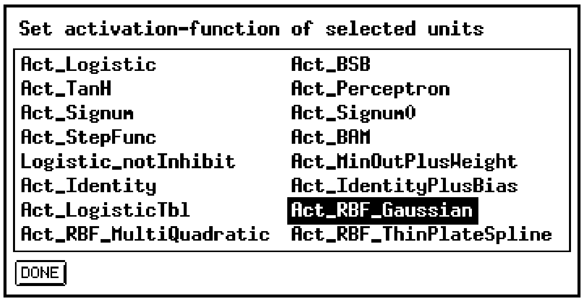

Abbildung 4.1: Auswahl der Aktivierungsfunktion

Bei der Konstruktion von dreistufigen feedforward–Netzen, die das Verfahren der Radial Basis Functions anwenden, ist unbedingt darauf zu achten, daß die drei genannten Aktivierungsfunktionen ausschließlich für Neuronen der mittleren Neuronenschicht verwendet werden.

In Abbildung 4.1 ist außerdem bereits eine neue Aktivierungsfunktion zu finden, die für Neuronen der Ausgabeschicht verwendet werden soll:

- `Act_IdentityPlusBias`

Diese Funktion aktiviert das entsprechende Neuron mit der Summe der über die Verbindungen gewichteten Eingangsaktivierungen, wobei zusätzlich der Bias des Neurons addiert wird. (Die Funktion `Act_Identity` leistet, abgesehen von der Addition des Bias, das gleiche.)

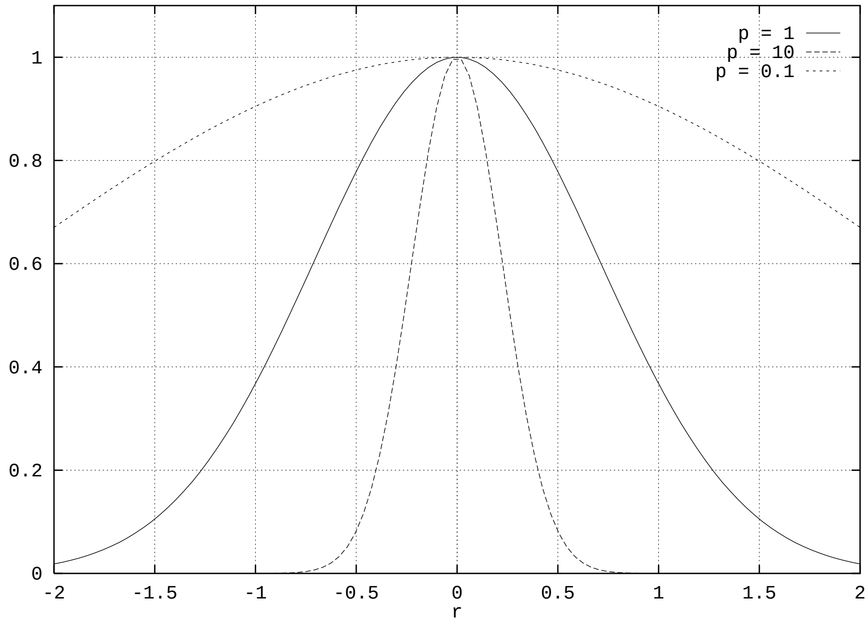

Abbildung 4.2: Einfluß des Parameters p auf die Funktion e^{-pr^2}

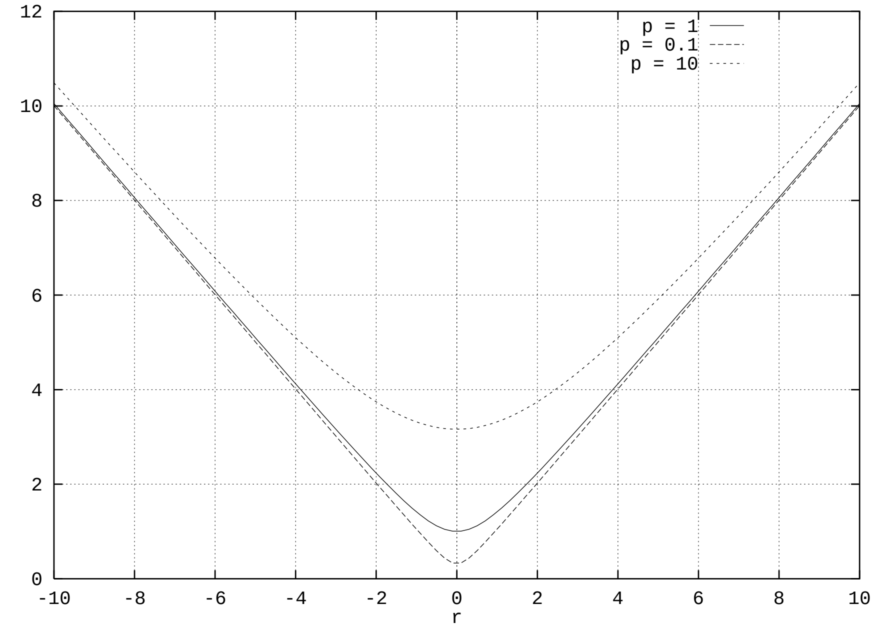

Abbildung 4.3: Einfluß des Parameters p auf die Funktion $\sqrt{p + r^2}$

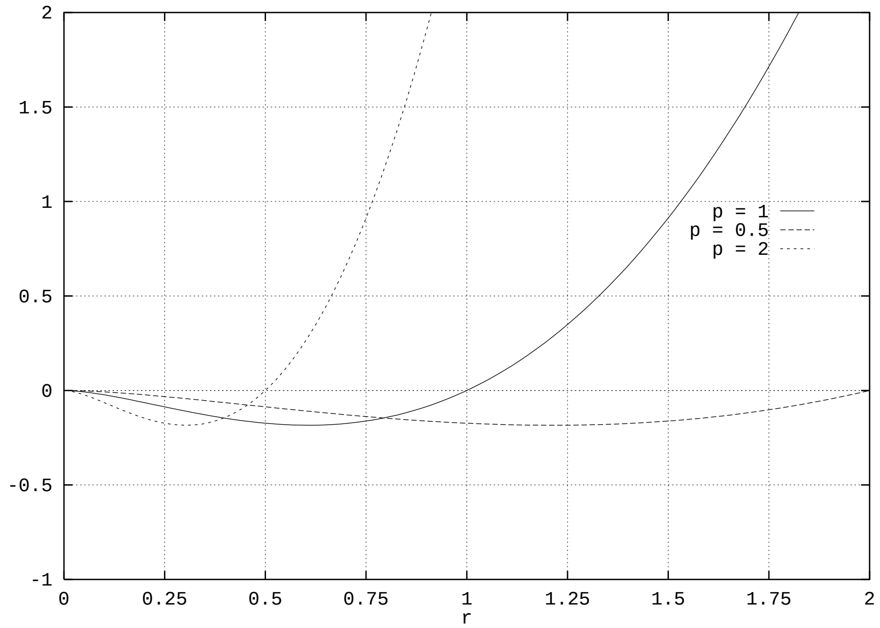

Abbildung 4.4: Einfluß des Parameters p auf die Funktion $(pr)^2 \ln(pr)$

Die vorhergehenden drei Abbildungen veranschaulichen die drei implementierten Zentrumsfunktionen. Die Aktivierung eines Neurons der mittleren Ebene ist jeweils über dem Abstand $r = |\vec{x} - \vec{t}\,|$ aufgetragen. Jede Zentrumsfunktion wurde für drei verschiedene Parameter p (Bias des Neurons) dargestellt.

Bei zwei der drei Zentrumsfunktionen fällt auf, daß sie für größer werdende Abstände r gegen Unendlich gehen. Dennoch eignen sich diese Funktionen für die Anwendung als Zentrumsfunktion. Dies wird einerseits durch die mathematischen Bedingungen gestützt, die durch die Funktionen erfüllt werden müssen (siehe Abschnitt 2.1), und andererseits durch praktische Versuche bestätigt (siehe Abschnitt 5.3.1). Die beiden Funktionen wurden in den SNNS aufgenommen, da ihre Verwendung in der Literatur über Radial Basis Functions eine gewisse Popularität besitzt.

Weit bekannter und üblicher ist jedoch die Gaußfunktion. Für große Abstände r geht sie gegen 0.0, wodurch das Verhalten des Netzes für Eingabemuster, die stark von den Lernmustern abweichen, vorhersagbar wird. Ein weiterer Vorteil der Gaußfunktion liegt darin, daß das neuronale Netz ohne Verbindungen zwischen Eingabe- und Ausgabeschicht auskommt, um für Approximationsprobleme gute Ergebnisse zu liefern.

4.1.2 Initialisierungsverfahren

Bei der Initialisierung eines Netzes, welches Radial Basis Functions verwendet, sollen die Gewichte zwischen mittlerer und Ausgabeschicht optimal berechnet werden. Hierbei tritt jedoch das Problem auf, daß sowohl die Zentrumsvektoren $\vec{t_j}$, also die Verbindungsgewichte zwischen Eingabe- und mittlerer Schicht, als auch die Parameter p, also die Schwellwerte der mittleren Neuronen sinnvoll gesetzt sein müssen. Aus diesem Grunde wurden drei Initialisierungsverfahren implementiert, die Unterschiedliches leisten:

- `RBF_Weights`

  Dieses Verfahren wählt zunächst gleichmäßig Zentrumsvektoren aus den geladenen Lernmustern aus und ordnet sie den Verbindungsgewichten zwischen Eingangs- und mittlerer Schicht zu. Anschließend wird der Bias aller Neuronen der mittleren Schicht auf einen einstellbaren Wert gesetzt (Parameter p), und die restlichen Gewichte werden direkt berechnet.

- `RBF_Weights_Redo`

  Gegenüber dem vorhergehenden Verfahren werden ausschließlich die Verbindungsgewichte zwischen mittlerer- und Ausgabeschicht berechnet und gesetzt. Alle anderen Gewichte bleiben unverändert.

- `RBF_Weights_Kohonen`

  Durch ein selbstorganisierendes Kohonen–Training werden aus den vorhandenen Lernmustern geeignete Zentrumsvektoren bestimmt. Diese Vektoren werden anschließend in die entsprechenden Verbindungsgewichte eingetragen. Ansonsten werden keine Gewichte verändert.

Abbildung 4.5 zeigt das Auswahlmenü der Initialisierungsfunktion mit den drei neuen Einträgen.

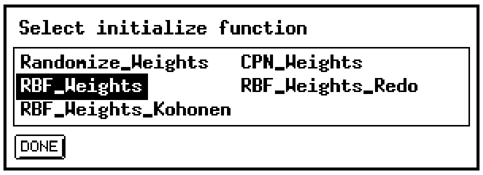

Abbildung 4.5: Auswahl des Initialisierungsverfahrens

Wie bereits aus der kurzen Beschreibung hervorgeht, ist die Voraussetzung für die Anwendung der drei Verfahren, daß gültige Lernmuster im SNNS geladen sind. Sollte dies nicht der Fall sein, so erscheint beim Aufruf des Verfahrens eine Alarmbox, die auf diesen Bedienungsfehler hinweist. Die genaue Funktion der einzelnen Verfahren sowie die Bedeutung der Parameter werden nun im einzelnen beschrieben.

`RBF_Weights`

Das umfassendste der drei Verfahren ist `RBF_Weights`. Hier können in einem Schritt alle Gewichte und der Bias aller Neuronen eines vollständig verbundenen dreistufigen feedforward–Netzes (ohne Verbindungen zwischen Eingabe- und Ausgabeschicht) initialisiert werden. Entsprechend einfach fällt die Wahl der Zentrumsvektoren, also der Verbindungsgewichte zwischen Eingabe- und mittlerer Schicht, aus: Die Zentrumsvektoren werden gleichmäßig aus den geladenen Lernmustern ausgewählt und den mittleren Neuronen zugeordnet. Die Auswahl erfolgt derart, daß das erste Lernmuster dem ersten Neuron zugeordnet wird und das letzte Lernmuster dem letzten Neuron. (Die Neuronen besitzen eine interne Reihenfolge, die aber nichts mit ihrer Anordnung in einem Display–Fenster zu tun hat.) Die übrigen Neuronen erhalten Zentrumsvektoren, die gleichmäßig aus den Lernmustern herausgegriffen werden. Dies geschieht derart, daß die Abstände der laufenden Nummern der ausgewählten Lernmuster möglichst gleich sind. Sind z.B. 13 Lernmuster geladen und die mittlere Schicht enthält 5 Neuronen, so werden die Lernmuster 1, 4, 7, 10 und 13 als Zentrumsvektoren ausgewählt.

Bevor ein ausgewähltes Lernmuster nun auf die entsprechenden Verbindungsgewichte verteilt wird, kann es geringfügig modifiziert werden. Dies geschieht zufällig. Zu diesem Zweck wird bei der Initialisierung ein Parameter angegeben (siehe Abbildung 4.6, Parameter *Streubreite*), der die maximale prozentuale Abweichung bestimmt, die zufällig erfolgen darf. Bei der Berechnung der Abweichungen wird durch eine Arcus–Tangens–Funktion eine Normalverteilung angenähert, so daß geringere Abweichungen wahrscheinlicher sind als größere Abweichungen. Wird als *Streubreite* 1.0 angegeben, so beträgt die maximale Abweichung 100%. Für eine *Streubreite* von 0 werden die Lernmuster unverändert in die Verbindungsgewichte übertragen.

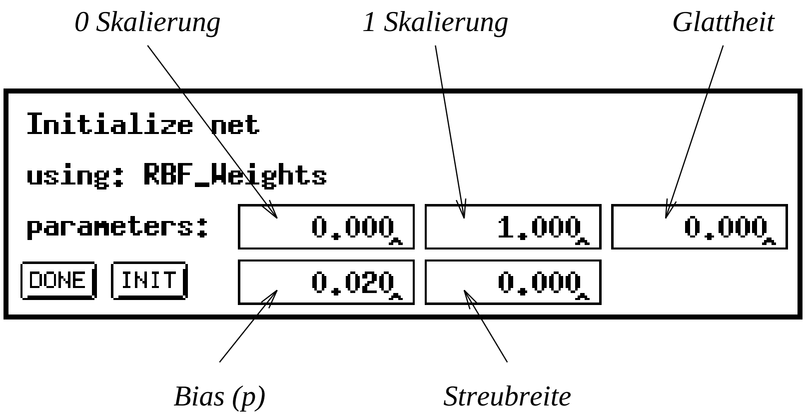

Abbildung 4.6: Vollständige Initialisierung aller Parameter

Für eine geringe Modifikation der Verbindungsgewichte sprechen vor allem zwei Gründe: Zum einen kann der Fall eintreten, daß die Zahl der mittleren Neuronen die Zahl der Lernmuster übersteigt. Dann ist es erforderlich, die Symmetrien aufzubrechen, die ohne Modifikation entständen. Diese Symmetrien würden die Berechnung der inversen Matrix bei der weiteren Initialisierung unmöglich machen. Der zweite Grund liegt darin, daß in der Menge der Lernmuster oft sogenannte Ausreißer vorkommen. Diese Ausreißer können zu schlechten Ergebnissen führen, falls sie zufälligerweise als Zentrumsvektoren gewählt werden. Durch die gleichmäßige Überlagerung eines Rauschanteils kann der negative Effekt der Ausreißer vermindert werden. Soll jedoch ein exaktes Interpolationsproblem gelöst werden, so empfiehlt es sich, eine Modifikation der Zentrumsvektoren nicht durchzuführen.

Der nächste Schritt der Inititalisierung besteht darin, den freien Parameter p der Zentrumsfunktion, also den Bias der mittleren Neuronen, zu bestimmen. Hierzu wird der Inititalisierungsparameter *Bias (p)* (siehe Abbildung 4.6) direkt als Bias aller Neuronen der mittleren Ebene eingesetzt. Die Wahl des Bias hängt stark von der verwendeten Zentrumsfunktion und von den Eigenschaften der Lernmuster ab. Wird als Zentrumsfunktion die Gaußfunktion verwendet, so kann nur die allgemeine Empfehlung gegeben werden, den Bias derart zu wählen, daß durch das Anlegen eines Lernmusters etwa 5–10% der mittleren Neuronen aktiviert werden. Ist der Bias zu gering gewählt, so werden fast alle Neuronen der mittleren Ebene gleichmäßig aktiviert. Ist der Bias zu hoch, so wird nur das spezielle Neuron aktiviert, dessen Zentrumsvektor dem aktuell anliegenden Lernmuster entspricht. Beide Fälle führen dazu, daß die weiteren Gewichte nicht sinnvoll berechnet werden können.

Nun erst erfolgt die eigentliche, aufwendige Inititalisierung der Verbindungsgewichte zwischen mittlerer- und Ausgabeschicht. Hierzu wird die Formel 2.11 von Seite 8 angewendet:

$$\vec{c} = (G^T \cdot G + \lambda G_{\square})^{-1} \cdot G^T \cdot \vec{y}$$

Der Inititalisierungsparameter *Glattheit* (siehe Abbildung 4.6) entspricht dem Wert von λ in der Berechnungsvorschrift. Die Matrizen wurden derart erweitert, daß der Wert eines konstanten Anteils jeweils mitberechnet wird. Die Funktion, die durch das Netz gebildet wird, lautet also:

$$f(\vec{x}) = \sum_{i=1}^{K} c_i h_i(\vec{x}) + b$$

Enthält die Ausgabeschicht aber mehr als ein Neuron ergeben sich die folgenden Funktionen:

$$f_j(\vec{x}) = \sum_{i=1}^{K} c_{i,j} h_i(\vec{x}) + b_j$$

Der berechnete Wert von b bzw. b_j wird direkt als Bias in den Neuronen der Ausgabeschicht eingetragen. Hierzu ist es jedoch unbedingt notwendig, für diese Neuronen eine Aktivierungsfunktion zu wählen, die den Bias berücksichtigt. In der momentanen Version des SNNS sind dies die Funktion `Act_Logistic` und die neue Funktion `Act_IdentityPlusBias`.

Die Aktivierungsfunktion der Ausgabeneuronen führt auf die beiden verbleibenden Parameter der Inititalisierungsfunktion. Wie bereits in Kapitel 3 gesagt wurde, geht das Berechnungsverfahren von einer linearen Aktivierung der Ausgabeneuronen aus. Die Verbindungsgewichte werden also derart berechnet, daß die gewichtete Summe der Zentrumsfunktionen die gewünschte Ausgabe ergibt. Wird nun jedoch eine sigmoide Aktivierungsfunktion gewählt, die für die Musterklassifikation besonders geeignet ist, so muß diese Funktion bei der Gewichtsberechnung berücksichtigt werden. Idealerweise müßte über die Umkehrfunktion der jeweiligen Aktivierungsfunktion die Solleingabe für die Aktivierungsfunktion, abhängig von der Sollausgabe des Lernmusters, berechnet werden. Diese Solleingabe wird dann bei der Berechnung der Gewichte mit dem Vektor $\vec{y}$ assoziiert. Leider ist jedoch die notwendige Umkehrfunktion im allgemeinen Fall nicht bekannt. Es ist auch zweifelhaft, ob sich die derartig berechnete Solleingabe sinnvoll verwenden läßt. Wird z.B. die logistische Aktivierungsfunktion verwendet, so ist für eine Aktivierung von 0 die Solleingabe $-\infty$ notwendig. Für eine Aktivierung von 1 wird dagegen eine Eingabe von $+\infty$ gebraucht. Es ist jedoch wenig sinnvoll Verbindungsgewichte so zu berechnen, daß nachfolgende Neuronen eine unendlich hohe oder zumindest eine sehr hohe Eingabe erhalten.

Abhilfe aus diesem Dilemma schaffen die beiden Initialisierungsparameter *0 Skalierung* und *1 Skalierung* (siehe Abbildung 4.6). Durch diese beiden Parameter wird die Solleingabe in die Ausgabeneuronen für die beiden Sollausgaben 0 und 1 angegeben. Wird zum Beispiel die lineare Aktivierungsfunktion `Act_IdentityPlusBias` verwendet, so müssen hier die Werte 0.0 und 1.0 eingetragen werden, wie es in Abbildung 4.6 dargestellt ist. Für die logistische Aktivierungsfunktion `Act_Logistic` empfehlen sich z.B. die Werte -4.0 und 4.0, die bei einem Bias von 0 auf die Aktivierungen 0.018 und 0.982 führen. Dies sind relativ gute Näherungen der gewünschten Sollausgaben 0 und 1. Die tatsächliche Implementierung interpoliert linear zwischen den eingetragenen Werten für *0 Skalierung* und *1 Skalierung*. So werden auch Sollausgaben, die von 0 und 1 verschieden sind, auf einen entsprechenden Solleingabewert abgebildet.

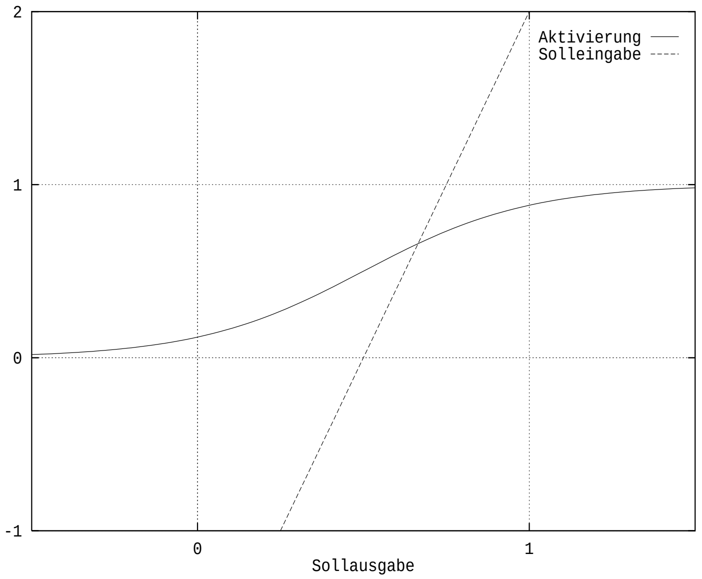

Abbildung 4.7: Zusammenhang zwischen Sollausgabe, Solleingabe und logistischer Aktivierung

Die Abbildung 4.7 zeigt die Aktivierung eines Ausgabeneurons unter Verwendung der logistischen Aktivierungsfunktion. Die Skalierung wurde so gewählt, daß die Sollausgaben 0 und 1 auf eine Solleingabe von -2 und 2 führen.

Die richtigen Werte für *0 Skalierung* und *1 Skalierung* können im allgemeinen Fall nicht angegeben werden. Hier muß ebenso wie beim Eintrag *Bias (p)* experimentiert werden. Wird die logistische Aktivierungsfunktion verwendet, so gilt: Betragsmäßig große Werte führen zu guten Ergebnissen, behindern aber das anschließende Training, da die logistische Funktion hauptsächlich in ihren sehr flachen Bereichen angewendet wird. Zu kleine Werte führen dagegen zu schlechten Initialisierungsergebnissen, bieten aber dafür eine gute Grundlage für das weitere Training.

RBF_Weights_Kohonen

Ein Nachteil des soeben besprochenen Initialisierungsverfahrens ist vor allem die sehr einfache Auswahl der Zentrumsvektoren aus den vorhandenen Lernmustern. Diese Wahl kann u.U. sehr unglücklich verlaufen. Günstig wäre es, wenn die Zentrumsvektoren möglichst gleichmäßig den Raum der Lernmuster abdecken. Für sehr viele ähnliche Lernmuster sollten mehrere unterschiedliche Zentrumsvektoren gewählt werden, für Ausreißer in den Lernmustern sollte dagegen kein Zentrumsvektor verbraucht werden. Ideal wäre eine Menge von Zentrumsvektoren, die möglichst die gleichen statistischen Eigenschaften besitzt wie die Lernmuster.

Das Verfahren RBF_Weights_Kohonen ermöglicht ein selbstorganisierendes Training der Zentrumsvektoren. Wie der Name des Verfahrens bereits andeutet, wird die Methode von Kohonen angewendet (siehe z.B. [Was89]). Es wurde die einfachste Form dieser Methode realisiert, die nun kurz beschrieben wird:

Voraussetzung für das Verfahren von Kohonen ist, daß die Lernmuster in normierter Form vorliegen. Dies bedeutet, daß sie als Vektoren, deren Länge jeweils 1 beträgt, aufgefaßt werden. Aus den n Lernmustern werden K Muster ausgewählt, die als Startwerte für die Zentrumsvektoren eingesetzt werden. Nun wird jeweils das Skalarprodukt zwischen einem Lernvektor und jedem einzelnen Zentrumsvektor berechnet. Sind die Vektoren auf 1 normiert, so ist das Skalarprodukt näherungsweise ein Maß für den Abstand der beiden multiplizierten Vektoren: Je größer das Skalarprodukt, desto geringer ist der Winkel den die beiden Vektoren aufspannen und desto geringer ist der Abstand der Punkte, die durch diese Vektoren repräsentiert werden (unter der Voraussetzung, daß es sich um Einheitsvektoren handelt). Es wird nun derjenige Zentrumsvektor bestimmt, dessen Abstand zum aktuellen Lernvektor am geringsten ist, d.h. dessen Skalarprodukt am größten ist. Dieser Zentrumsvektor wird um einen kleinen Betrag in Richtung auf den aktuellen Lernvektor verschoben:

$$\vec{z}_{\text{neu}} = \vec{z}_{\text{alt}} + \alpha(\vec{l} - \vec{z}_{\text{alt}})$$

Die gesamte Prozedur wird für alle Lernmuster mehrmals wiederholt. Der Effekt ist, daß die Zentrumsvektoren allmählich die statistischen Eigenschaften der Lernvektoren übernehmen.

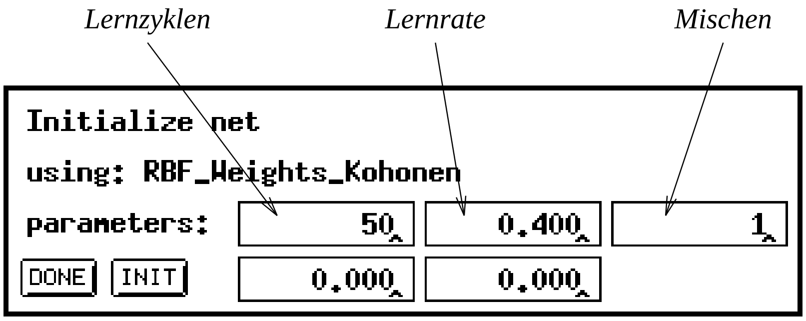

Abbildung 4.8: Initialisierung der Zentrumsvektoren

Abbildung 4.8 zeigt die Bedeutung und die Position der Inititalisierungsparameter im Initialisierungsfenster. *Lernzyklen* gibt die Anzahl an, wie oft jedes einzelne Lernmuster für das Kohonen–Training verwendet wird. Hierbei werden alle gespeicherten Lernmuster der Reihe nach angewendet und dieser Zyklus wird so oft wiederholt, wie der Parameter angibt. Es ist auch erlaubt, 0 Lernzyklen anzugeben, wodurch die Zentrumsvektoren nur gesetzt, aber nicht mehr nachtrainiert werden.

Der zweite Parameter bestimmt die *Lernrate* α. Sie sollte zwischen 0.0 und 1.0 gewählt werden. Eine Lernrate von 0.0 läßt die Zentrumsvektoren unverändert. Bei einer Lernrate von 1.0 wird dagegen der ausgewählte Zentrumsvektor durch den aktuellen Lernvektor ersetzt.

Der dritte Parameter beeinflußt die Auswahl der Zentrumsvektoren zu Beginn des Verfahrens. Ein Wert von 0.0 führt dazu, daß die Zentrumsvektoren in gleicher Weise gleichmäßig aus den Lernmustern ausgewählt werden, wie es für das Inititalisierungsverfahren `RBF_Weights` beschrieben wurde. Ein Wert ungleich 0.0 bewirkt, daß die Zentrumsvektoren zufällig aus den Lernmustern ausgewählt werden.

Es sei hier nochmals gesagt, daß dieses Initialisierungsverfahren ausschließlich die Zentrumsvektoren, also die Verbindungsgewichte zwischen Eingabe- und mittlerer Schicht, initialisiert. Der Bias der Neuronen kann über den graphischen Editor des SNNS von Hand gesetzt werden. Um nun die noch, fehlenden Gewichte zu initialisieren, wurde ein weiteres Verfahren implementiert.

`RBF_Weights_Redo`

Dieses Initialisierungsverfahren beeinflußt ausschließlich die Verbindungsgewichte zwischen mittlerer und Ausgabeschicht. Es dient dazu, das Netz möglichst gut zu initialisieren, wobei die Zentrumsvektoren und die Schwellwerte der mittleren Neuronen als Ausgangsbasis genommen werden. Die Zentrumsvektoren können z.B. durch das zuvor beschriebene Verfahren gesetzt werden. Eine andere Möglichkeit besteht darin, Zentrumsvektoren durch ein externes Verfahren zu erzeugen, diese Vektoren in die Form einer SNNS Musterdatei zu bringen und dann mit dem vorhergehenden Verfahren in die entsprechenden Verbindungsgewichte des Netzes zu kopieren. Hierbei sollte natürlich nicht mehr gelernt werden!

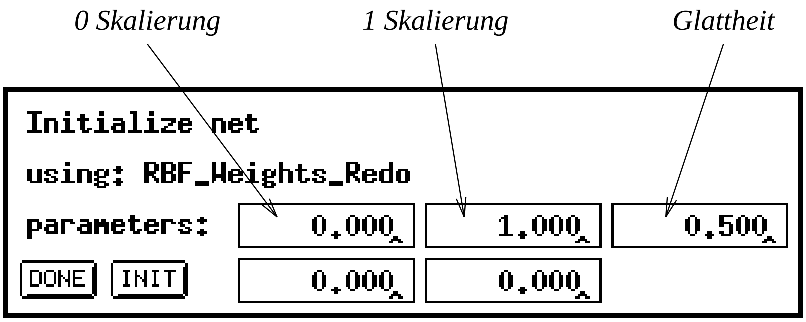

Abbildung 4.9: Initialisierung der zweiten Gewichtsebene

Die Funktion des Verfahrens `RBF_Weights_Redo` unterscheidet sich vom ersten der drei Verfahren also nur darin, daß die Zentrumsvektoren und der Bias der mittleren Neuronen nicht verändert werden. Dementsprechend entfallen die beiden letzten Parameter des Initialisierungsverfahrens. Die Bedeutung und Funktion der restlichen drei Parameter ist identisch mit denen des ersten der drei vorgestellten Verfahren und wird deshalb nicht weiter besprochen.

4.1.3 Lernverfahren

Für die speziellen Aktivierungsfunktionen der Radial Basis Functions ist ein eigenes Lernverfahren notwendig. Es ist nicht möglich, Netze, die die Aktivierungsfunktionen `Act_RBF_...` verwenden, mit Backpropagation zu trainieren. Auch das neu implementierte Lernverfahren ist nur für den Fall anwendbar, daß Neuronen, die die speziellen Aktivierungsfunktionen verwenden, die mittlere Ebene eines dreistufigen feedforward–Netzes bilden. Außerdem müssen die Neuronen der Ausgabeschicht bei ihrer Aktivierung den Bias berücksichtigen.

Das neue spezielle Lernverfahren hat den Namen `RadialBasisLearning`. Die Abbildung 4.10 zeigt die Auswahl dieses Verfahrens im Menü der Lernfunktionen.

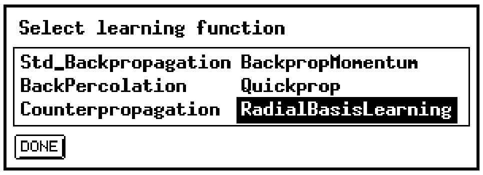

Abbildung 4.10: Auswahl des Lernverfahrens

Die Formeln 3.2–3.6 aus dem Abschnitt 3.3 bilden die Grundlage für das implementierte Lernverfahren. Für die Modifikation der Zentrumsvektoren, der Schwellwerte der mittleren Neuronen sowie aller übrigen Verbindungsgewichte und Schwellwerte wurden jeweils unterschiedliche Lernraten eingeführt. Abbildung 4.11 zeigt die Namen und die Anordnung der insgesamt fünf Lernparameter:

- *Zentren*

  Dieser Eintrag bestimmt den Lernparameter η_1 für die Veränderung der Zentren gemäß der Formel:

  $$\Delta \vec{t_j} = -\eta_1 \frac{\partial E}{\partial \vec{t_j}}$$

  $\Delta \vec{t_j}$ ist die Gewichtsänderung der Zentrumsvektoren (der Verbindungsgewichte zwischen Eingabe- und mittlerer Schicht) im aktuellen Lernschritt.

- *Bias (p)*

  Die Lernrate η_2 für die Anpassung der Parameter p der Zentrumsfunktionen wird durch diesen Eintrag bestimmt. p wird im Bias der mittleren Neuronen gespeichert

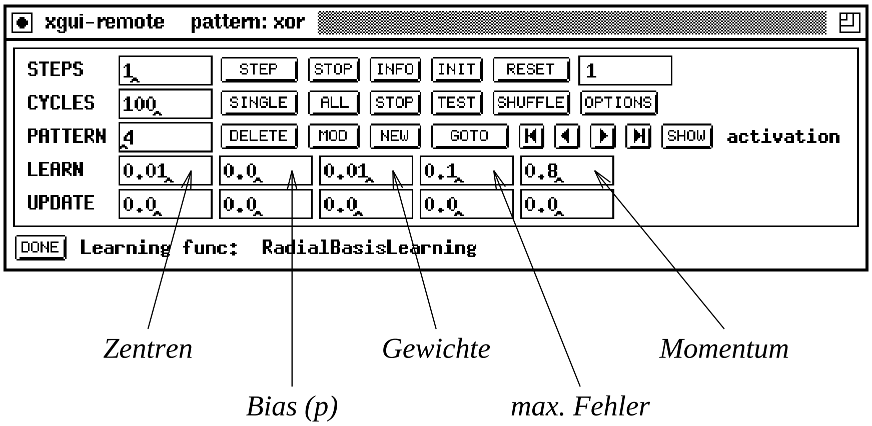

Abbildung 4.11: Bedeutung der Lernparameter

und durch die folgende Formel trainiert:

$$\Delta p_j = -\eta_2 \frac{\partial E}{\partial p_j}$$

- *Gewichte*

  Die dritte Lernrate η_3 beeinflußt das Training aller Verbindungsgewichte, die auf die Ausgabeschicht zuführen, sowie die Schwellwerte der Ausgabeneuronen:

  $$\Delta c_{j,k} = -\eta_3 \frac{\partial E}{\partial c_{j,k}} \quad , \quad \Delta d_{i,k} = -\eta_3 \frac{\partial E}{\partial d_{i,k}} \quad , \quad \Delta b_k = -\eta_3 \frac{\partial E}{\partial b_k}$$

- *max. Fehler*

  Um zu verhindern, daß das Netz übertrainiert wird, kann der maximal zulässige Fehler der Ausgabeneuronen definiert werden, der noch nicht zu einer Veränderung der betroffenen Gewichte führen soll. Falls der Wert von $\Delta_{i,k}$ (die Differenz zwischen Aktivierung des Ausgabeneurons k und der Lernvorgabe i) betragsmäßig kleiner als *max. Fehler* ist, so wird $\Delta_{i,k}$ gleich 0.0 gesetzt. Dies führt in den Formeln 3.2–3.6 zu einer Nichtbeachtung des entsprechenden Fehlers.

- *Momentum*

  Dieser Parameter beeinflußt die Stärke des Momentum–Terms beim Training. In der Formel

  $$\Delta g_{t+1} = -\eta \frac{\partial E}{\partial g} + \mu \Delta g_t \quad ,$$

  die die allgemeine Anpassung von Gewichten und Biassen beschreibt, wird der Parameter μ durch den fünften Eintrag der Lernparameter bestimmt.

Wie bereits die Abbildung 4.11 zeigt, müssen die Lernraten η_1 bis η_3 sehr vorsichtig eingesetzt werden. Werden zu große Werte gewählt (wie sie z.B. für Backpropagation üblich sind), so sind die Gewichtsänderungen zu stark, und das Lernverfahren wird instabil. Während der Tests hat sich herausgestellt, daß das Verfahren stabiler ist, wenn nur eine der drei Lernraten auf einen Wert größer 0.0 gesetzt wird. Besonders anfällig ist der Parameter *Bias (p)* (η_2). Dies liegt daran, daß er die Zentrumsfunktion grundlegend verändert und somit einen sehr großen Einfluß hat.

Übliche Werte für den Parameter *max. Fehler* liegen im Bereich von 0.0 bis 0.3. Die Wahl hängt von den Anforderungen an das Netz ab. Für den Momentum–Term werden in der Regel Werte im Bereich 0.8 bis 0.9 verwendet. Durch einen Wert von 0.0 kann der Einfluß des Momentums ausgeschaltet werden.

Bei den Tests des Lernverfahrens zeigte es sich, daß das Training wesentlich stabiler bleibt, wenn im *Batch*–Betrieb gearbeitet wird. *Batch*–Betrieb bedeutet, daß sämtliche Änderungen erst aktiv werden, sobald alle Lernmuster genau einmal präsentiert wurden. Dies ist auch die Form des Trainings, die in der Literatur für Radial Basis Functions vorgeschlagen wird. Das Gegenteil zum *Batch*–Betrieb bildet das *Online*–Verfahren, bei dem nach Präsentation jedes einzelnen Lernmusters Gewichte verändert werden. Welches Verfahren angewendet werden soll, kann zum Übersetzungszeitpunkt des Simulators ausgewählt werden. Standardeinstellung ist der *Batch*–Betrieb. Der *Online*–Betrieb kann aktiviert werden, indem während der Übersetzung des Simulator–Kerns die Konstante `RBF_INCR_LEARNING` definiert wird.

4.2 Vorgehensweise bei der Anwendung

Es wird nun kurz die Vorgehensweise bei der Anwendung der Radial Basis Functions mit SNNS angesprochen. Im einzelnen werden die Schritte aufgeführt, die für die Entwicklung einer allgemeinen Anwendung notwendig sind.

Zunächst wird ein dreistufiges feedforward–Netz konstruiert, dessen erste und zweite Neuronenebene sowie die zweite und dritte Ebene vollständig miteinander verbunden sind. Hierzu kann der graphische Editor des SNNS oder das Werkzeug `BIGNET` des SNNS verwendet werden.

Als Ausgabefunktionen (`output function`) aller Neuronen wird die Funktion `Out_Identity` gewählt. Hierdurch wird die Aktivierung der Neuronen unverändert an die nachfolgende Schicht weitergegeben. Für die Neuronen der mittleren Ebene wird eine der drei neuen Aktivierungsfunktionen `Act_RBF_...` gewählt (vorzugsweise `Act_RBF_Gaussian`). Für die Aktivierung der Ausgabeneuronen muß eine Funktion gewählt werden, die den Bias des Neurons berücksichtigt. Dies sind die Funktionen `Act_Logistic` und `Act_IdentityPlusBias`.

Der nächste Schritt besteht in der Generierung der Lernmuster. Diese können entweder ebenfalls mit dem graphischen Editor von Hand generiert oder aber aus vorhandenen Daten durch ein entsprechendes Filterprogramm direkt in die Form einer SNNS–Musterdatei gebracht werden (siehe auch [ZMH+92]). Soll später das Inititalisierungsverfahren `RBF_Weights_Kohonen` verwendet werden, so sollten die Lernmuster Vektoren der Länge 1.0 darstellen oder zumindest alle eine ähnliche Länge besitzen.

Bevor nun eine Initialisierung des Netzes durchgeführt wird, ist es notwendig, einen sinnvollen Bias für die Neuronen der mittleren Ebene zu bestimmen. Hierzu werden zunächst mit dem Verfahren `RBF_Weights_Kohonen` die Verbindungsgewichte zwischen Eingabe- und mittlerer Schicht so gesetzt, daß die Zentrumsvektoren, die durch diese Gewichte repräsentiert werden, eine Auswahl aus den vorgegebenen Lernmustern darstellen. Die notwendigen Initialisierungsparameter lauten: *Lernzyklen* = 0, *Lernrate* = 0.0 und *Mischen* = 0.0. Hierdurch werden Lernmuster unverändert als Zentrumsvektoren übernommen.

Für die Wahl des Bias wird nun mit dem Knopf `TEST` des SNNS–remote–panel die Aktivierung der Neuronen der mittleren Ebene für verschiedene Lernmuster betrachtet. Hierbei wird der Bias dieser Neuronen so eingestellt, daß die Aktivierung der Neuronen möglichst vielfältig ist. Verwendet man die Gaußkurve als Zentrumsfunktion, so sind bei einem zu kleinen Bias alle Neuronen gleichmäßig erregt (Extremfall: Bias = 0.0 führt zur Aktivierung 1.0 aller Neuronen). Bei einem zu großen Bias wird im Extremfall immer nur dasjenige Neuron aktiviert, dessen Zentrumsvektor dem aktuell anliegenden Lernmuster entspricht. Eine sinnvolle Vorgehensweise besteht darin, zunächst den Bias auf 1.0 zu setzen und ihn dann abhängig vom Netzverhalten gleichmäßig zu verändern. Hierbei ist darauf zu achten, daß der Bias auf keinen Fall negativ wird, da dies für keine der implementierten Zentrumsfunktionen definiert ist. Eine allgemeine Empfehlung für den Bias kann nicht gegeben werden, da die richtige Einstellung von der Dimension der Eingabeschicht und von der Ähnlichkeit der einzelnen Lernmuster zueinander abhängt.

Nachdem nun ein günstiger Bias für die Neuronen der mittleren Ebene bestimmt wurde, kann z.B. das Inititalisierungsverfahren `RBF_Weights` gestartet werden. Abhängig von der Wahl der Aktivierungsfunktion in der Ausgabeschicht, werden die beiden *Skalierungsparameter* der Initialisierung gesetzt (siehe Seite 26). Für die Funktion `Act_IdentityPlusBias` sollten die Werte 0.0 und 1.0 gewählt werden. Bei der logistischen Aktivierungsfunktion `Act_Logistic` empfehlen sich die Werte −4.0 und 4.0 (siehe auch Abbildung 4.7). Die Parameter *Glattheit* und *Streubreite* sollten zunächst auf 0.0 gesetzt werden. Für den *Bias* wird der zuvor experimentell ermittelte günstige Wert eingetragen. Die Initialisierung kann, abhängig von der Zahl der Lernmuster und der Zahl der Neuronen in der mittleren Ebene, eine längere Zeit in Anspruch nehmen. Während der Initialisierung erscheinen deshalb Kontrollausgaben auf dem Terminal, von dem aus SNNS gestartet wurde.

Nach Abschluß der Initialisierung kann mit dem Knopf `TEST` das Ergebnis überprüft werden. Der genaue Fehler des Netzes kann jedoch nur durch das Lernverfahren bestimmt werden. Hierzu wird zunächst das Lernverfahren `RadialBasisLearning` ausgewählt. Alle Lernparameter werden auf 0.0 gesetzt, die Zahl der Lernzyklen (`CYCLES`) erhält den Wert 1. Durch den Knopf `ALL` startet das Lernverfahren. Da die Lernparameter gleich 0.0 sind, wird keine Änderung am Netz vorgenommen. Im Terminal–Fenster erscheint nach Durchlauf aller Lernmuster der aktuelle Fehler des Netzes. Der Fehler ist wie üblich definiert als die Summe der Fehlerquadrate in den Ausgabeneuronen (vergleiche Formel 3.1). Unter bestimmten Umständen kann es vorkommen, daß der Fehler des Netzes sehr, sehr groß wird. Dies ist vor allem auf numerische Probleme zurückzuführen. Als problematische Ausgangssituation für die Initialisierung hat sich z.B. ein schlecht gewählter Bias herausgestellt. Auch ist der Fall, daß die Zahl der Lernmuster mit der Zahl der Neuronen in der mittleren Ebene übereinstimmt, problematisch. In diesem Fall ist die Zahl der unbekannten Verbindungsgewichte pro Ausgabeneuron plus dem zu berechnenden Bias genau um eins höher als die Zahl der Lernmuster. Es sind also mehr Unbekannte zu berechnen als Gleichungen verfügbar sind. Ein Neuron weniger in der mittleren Schicht verbessert das Ergebnis bedeutend.

Nach der ersten Initialisierung empfiehlt es sich, das aktuelle Netz zu speichern, um anschließend die Möglichkeiten des Lernverfahrens zu testen. Die Parameter des Lernverfahrens wurden bereits im vorhergehenden Abschnitt beschrieben. Es hat sich herausgestellt, daß zu große Lernraten schnell dazu führen, daß das Lernverfahren instabil wird. Die Lernraten sind daher mit äußerster Vorsicht einzustellen. Es empfiehlt sich, zunächst nur eine der drei Lernraten (*Zentren*, *Bias (p)* und *Gewichte*) auf einen Wert ungleich 0.0 zu setzen und so die Empfindlichkeit des Verfahrens auf diesen einen Parameter zu überprüfen. Besonders kritisch ist der Parameter *Bias (p)*, da seine Anwendung sehr starke Änderungen der Zentrumsfunktionen nach sich zieht. Sollte der Bias eines der mittleren Neuronen während des Lernvorgangs einen negativen Wert annehmen, so erscheint eine entsprechende Meldung im Terminal–Fenster. In diesem Fall ist weiteres sinnvolles Training nicht möglich. Das Netz sollte dann neu initialisiert werden.

Direkt nach der Initialisierung ist es oft gut, nur die Gewichte zwischen mittlerer und Ausgabeschicht zu trainieren. Hierdurch werden die numerischen Ungenauigkeiten ausgeglichen, die bei der direkten Gewichtsberechnung aufgetreten sind. Ein optimales Gesamtergebnis läßt sich allerdings nur dann erzielen, wenn auch die Zentrumsvektoren nachtrainiert werden, da ihre Auswahl unter Umständen ungünstig war.

Das Initialisierungsverfahren für die direkte Gewichtsberechnung kann nicht die Verbindungsgewichte zwischen der Eingabe- und der Ausgabeschicht berechnen. Sind solche Verbindungen definiert, so empfiehlt sich das folgende Vorgehen:

Noch bevor die Zentrumsvektoren durch `RBF_Weights_Kohonen` gesetzt werden und ein günstiger Bias gesucht wird, sollten alle Verbindungsgewichte durch das Verfahren `Randomize_Weights` auf zufällige Werte zwischen -0.1 und 0.1 gesetzt werden. Hierdurch erhalten die Verbindungen zwischen Eingabe- und Ausgabeebene eine Vorinitialisierung. Nachdem dann das eigentliche Verfahren `RBF_Weights` durchlaufen wurde, ist der Fehler des Netzes noch relativ groß, da die genannten Verbindungen nicht berücksichtigt wurden. Sie können nun aber recht schnell nachtrainiert werden, indem für das Training lediglich der Lernparameter *Gewichte* verwendet wird.

4.3 Details der Implementierung

Bei der Implementierung aller neuen Funktionen und Verfahren wurde vollständig das modulare Konzept des SNNS ausgenutzt. Die neuen Aktivierungsfunktionen finden sich in der Datei `trans_f.c`. Das neue Lernverfahren wurde in die Datei `learn_f.c` aufgenommen. Die Initialisierungsverfahren befinden sich in `init_f.c`. Zusätzlich zu diesen Änderungen wurde das Modul `matrix.c` implementiert, das allgemein verwendbare Matrizenoperationen realisiert.

Besonders bei der Implementierung des Lernverfahrens wurde die Effizienz der Ausführung in den Vordergrund gestellt. Dies wirkt sich bis auf die Realisierung der neuen Aktivierungsfunktionen aus: Während der Vorwärts–Propagierung werden bereits Daten gesammelt, die für den darauffolgenden Lernschritt ebenfalls benötigt werden. Im einzelnen werden die speziellen Felder der Neuronen und Verbindungen folgendermaßen verwendet:

- Neuronen der Ausgabeschicht
  - `value_a`: Aktueller Fehler des Ausgabeneurons

    In diesem Feld wird die Differenz zwischen der Aktivierung des Neurons und der aktuellen Lernvorgabe gespeichert ($\Delta_{i,k}$). Sollte dieser Wert betragsmäßig kleiner als der Parameter *max. Fehler* des Lernverfahrens sein, so wird er auf 0.0 gesetzt.

- `value_b`: Summe der Änderungen des Bias

  Hier werden während des aktuellen Lernzyklus (im *Batch*–Betrieb) alle anfallenden Änderungen des Bias dieses Neurons aufsummiert. Nach Durchlauf aller Lernmuster wird diese Summe mit der Lernrate für Gewichte (dritter Lernparameter) multipliziert und zum aktuellen Bias addiert. Der Momentum–Term wird für die Änderung des Bias der Ausgabeneuronen nicht berücksichtigt.

• Neuronen der mittleren Schicht (Zentrumsfunktionen)

- `value_a`: Aktuelles Abstandsquadrat

  Bei der Vorwärts–Propagierung des aktuellen Lernmusters wird in diesem Feld das Quadrat des berechneten Abstands abgelegt ($|\vec{x} - \vec{t_j}|^2$). Dieser Wert wird anschließend bei der Berechnung der Gewichtsänderung weiter verwendet.

- `value_b`: Summe der Änderungen des Bias

  Ebenso wie bei Neuronen der Ausgabeschicht werden hier die Änderungen des Bias für dieses Neuron aufsummiert. Da der Bias aber die spezielle Rolle des Parameters p der Zentrumsfunktion übernimmt, wird als Lernrate der zweite Lernparameter verwendet (*Bias (p)*). Der Momentum–Term wird hier ebenfalls nicht aktiv.

- `value_c`: Gewichtete Fehlersumme

  In diesem Eintrag werden die mit den Verbindungsgewichten und mit der Ableitung der Aktivierungsfunktion der Ausgabeneuronen gewichteten Fehler der Ausgabeschicht addiert. Der Wert entspricht dem Anteil $\sum_{k=1}^{m} c_{j,k}\Delta_{i,k}\sigma'(f_k(\vec{x_i}))$ der Formeln 3.5 und 3.6. Der Eintrag wird während der Vorwärts–Propagierung des aktuellen Lernmusters auf 0.0 gesetzt und anschließend bei der Berechnung der Fehler in der Ausgabeschicht aufsummiert.

• Verbindungen, die auf die Ausgabeschicht führen

- `value_a`: Summe der Gewichtsänderungen

  Ebenso wie bei den Biassen der Neuronen werden in diesem Eintrag die Änderungen des Verbindungsgewichts während des aktuellen Lernzyklus summiert. Als Lernrate wird am Ende des Zyklus der dritte Lernparameter verwendet (*Gewichte*). Zu dieser Änderung wird dann noch der Anteil des Momentum–Terms addiert, und das Verbindungsgewicht wird endgültig angepaßt.

- `value_b`: Letzte Änderung

  Dieses Feld enthält den Wert der letzten Änderung des Verbindungsgewichts. Er wird für die Berechnung des Momentum–Anteils benötigt. Sobald die neue Änderung berechnet wurde, wird sie in diesem Feld abgespeichert. Beim ersten Aufruf des Lernverfahrens wird der Eintrag dieses Feldes mit 0.0 initialisiert.

- Verbindungen zwischen Eingabe- und mittlerer Schicht (Zentrumsvektoren)
  - `value_a`: Summe der Gewichtsänderungen

    Wie auch bei den übrigen Verbindungen wird hier die Summe der zukünftigen Änderungen berechnet. Als Lernrate wird jedoch der erste Lernparameter (*Zentren*) herangezogen, da es sich bei diesen Gewichten um die Repräsentation der Zentrumsvektoren handelt. Ebenso wie bei den anderen Verbindungsgewichten wird hier der Momentum–Term berücksichtigt.
  - `value_b`: Letzte Änderung

    Die Bedeutung dieses Feldes ist identisch mit dem entsprechenden Feld der Verbindungen, die auf die Ausgabeschicht führen(s.o.).

Die Ableitungen der Aktivierungsfunktionen für Neuronen der mittleren Ebene sind, ebenso wie die Aktivierungsfunktionen selbst, in der Datei `trans_f.c` abgelegt. Normalerweise kann hier aber nur genau eine Ableitungsfunktion für jede Aktivierungsfunktion stehen. Da aber bei der Zentrumsfunktion eine partielle Ableitung nach dem Abstandsquadrat und zusätzlich nach dem Parameter p notwendig ist, mußte eine gemeinsame Funktion für beide Ableitungen entwickelt werden. Die notwendige Unterscheidung wird beim Aufruf der Funktion über den Inhalt des Feldes `Aux.int_no` im betreffenden Neuron getroffen:

- `Aux.int_no` = 0

  partielle Ableitung nach dem Abstandsquadrat
- `Aux.int_no` = 1

  partielle Ableitung nach dem Parameter p
- `Aux.int_no` = 2

  partielle Ableitung nach dem Abstandsquadrat, wobei das aktuelle Abstandsquadrat im Feld `value_a` des Neurons gespeichert ist und deshalb nicht neu berechnet wird
- `Aux.int_no` = 3

  partielle Ableitung nach dem Parameter p, wobei hier ebenfalls das aktuelle Abstandsquadrat im Feld `value_a` des Neurons gespeichert ist

Im Lernverfahren, wie es derzeitig implementiert ist, werden ausschließlich die beiden letzten Fälle verwendet.

In bestimmten Fällen ist es sinnvoll, während des Trainings nicht alle Gewichte gleichzeitig zu trainieren (siehe vorhergehender Abschnitt). In diesem Fall führt das Lernverfahren nur die Lernschritte aus, die tatsächlich notwendig sind und eine Änderung bewirken. Die Ausführungsgeschwindigkeit wird hierdurch wesentlich gesteigert.

Für die direkte Gewichtsberechnung beim Initialisierungsverfahren `RBF_Weights` muß eine Matrixinversion durchgeführt werden. Dafür wird die Matrix in eine obere und eine untere Dreiecksmatrix zerlegt. Für die Zerlegung wurde der Algorithmus aus [PFTV90] herangezogen:

Gesucht ist die Inverse der Matrix A. Um die Inversion durchzuführen, wird das Gleichungssystem

$$A \cdot \vec{x} = \vec{b}$$

für eine allgemeine rechte Seite $\vec{b}$ gelöst. Setzt man für den Vektor $\vec{b}$ einen Spaltenvektor der Einheitsmatrix ein, so erhält man für $\vec{x}$ den entsprechenden Spaltenvektor der inversen Matrix A^{-1}. Dies wird für alle Spaltenvektoren der Einheitsmatrix wiederholt, wodurch nacheinander die Spaltenvektoren der inversen Matrix bestimmt werden.
Für die Lösung des Gleichungssystems wird A in folgender Weise zerlegt:

$$A = L \cdot U$$

Hierbei ist U eine obere Dreiecksmatrix mit besetzten Diagonalelementen. L ist eine untere Dreiecksmatrix, deren Diagonalelemente den Wert 1.0 besitzen. Nach der Zerlegung von A kann das Gleichungssystem in die Form

$$L \cdot (U \cdot \vec{x}) = \vec{b} \iff L \cdot \vec{y} = \vec{b}$$

gebracht werden. Da L eine untere Dreiecksmatrix ist, kann der Vektor $\vec{y}$ ohne Probleme direkt berechnet werden. Anschließend wird das System

$$U \cdot \vec{x} = \vec{y}$$

in gleicher, einfacher Weise berechnet.

Um die Inversion durchzuführen, muß die LU–Zerlegung nur ein einziges Mal durchgeführt werden. Anschließend können alle Spalten der Inversen A^{-1} schnell bestimmt werden.

Während der Implementierung wurde zunächst der Gauß–Jordan Algorithmus zur Matrixinversion (siehe [Sto89]) verwendet. Dieser Algorithmus ist jedoch numerisch weniger stabil, was sich durch probeweises Berechnen der Einheitsmatrix aus $A \cdot A^{-1}$ bestätigen ließ.

Kapitel 5

Anwendungen

Im folgenden Kapitel wird die Anwendung der Radial Basis Functions anhand von drei Beispielen demonstriert. Die umfangreichste der drei Anwendungen ist die Erkennung handgeschriebener Ziffern. Es handelt sich hierbei um ein typisches Problem der Musterklassifikation. Die zweite Anwendung behandelt die Erkennung von gesprochenen englischen Vokalen und ist ebenfalls ein Problem der Mustererkennung. Als drittes Beispiel wurde die Approximation einer mathematischen Funktion gewählt. Hier soll die Fähigkeit der Radial Basis Functions zur allgemeinen Approximation aufgezeigt werden.

5.1 Erkennung handgeschriebener Ziffern

Die erste der drei Anwendungen ist die Erkennung handgeschriebener Ziffern. In umfangreichen Tests wird der Einfluß jedes einzelnen Parameters untersucht, der für die Initialisierung eines neuronalen Netzes mit Radial Basis Functions benötigt wird. Anschließend wird das Lernverhalten eines initialisierten Netzes betrachtet.

Zunächst wird jedoch kurz die Ausgangsbasis der Versuchsreihen beschrieben: Zwei unabhängige Mengen von jeweils 10000 handgeschriebenen Ziffern bilden die Ausgangsbasis für das Problem der Ziffernerkennung. Die einzelnen Ziffern sind jeweils in Höhe und Breite normiert, so daß jede Ziffer eine Matrix von 16×16 „Bildpunkten" bis zu den Rändern ausfüllt. Die insgesamt 256 Bildpunkte, durch die eine Ziffer gebildet wird, können jeweils 256 verschiedene Grauwerte zwischen weiß und schwarz annehmen.

Für die Verwendung im neuronalen Netz wurden die Graustufen auf den Wertebereich 0.0 (weiß) bis 1.0 (schwarz) skaliert. Diese Werte dienten als Aktivierungswerte für die Neuronen der Eingabeschicht des Netzes. Die Abbildung 5.1 zeigt eine kleine Auswahl der verwendeten Ziffern.

Für die Versuche der Ziffernerkennung mit dem Simulator SNNS wurde die Zahl der Lern- sowie der Testziffern jeweils auf 1000 beschränkt. Dies geschah deshalb, da ansonsten die notwendige Zeit für das Netztraining nicht akzeptabel gewesen wäre.

5.1.1 Initialisierung des Netzes

Zunächst wurden die Auswirkungen unterschiedlicher Initialisierungsparameter auf die Qualität der Netzinitialisierung untersucht. Hierzu wurde ein dreistufiges Netz, bestehend

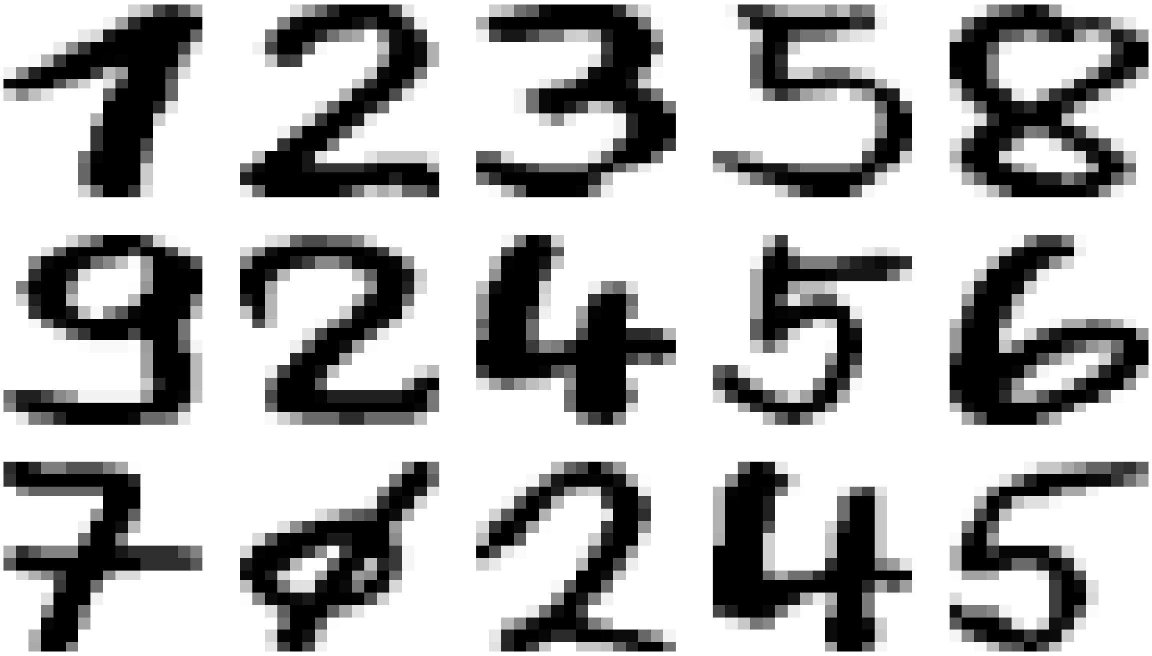

Abbildung 5.1: Eine Auswahl von Ziffern

aus 256 Eingabeneuronen, 100 Neuronen in der mittleren Schicht und 10 Ausgabeneuronen, definiert. Für die Neuronen der mittleren Ebene wurde die Aktivierungsfunktion `Act_RBF_Gaussian` gewählt. Die Ausgabeneuronen bekamen die sigmoide Aktivierungsfunktion `Act_Logistic` zugewiesen.

Entscheidend für die Verwendung der Radial Basis Functions ist die Wahl des richtigen Bias für die Neuronen der mittleren Schicht, da hierdurch die Form der Zentrumsfunktion festgelegt wird (siehe Abbildung 4.2). Durch mehrere Tests (wie in Abschnitt 4.2 beschrieben) konnte der Wertebereich für einen günstigen Bias auf den Bereich 0.1 bis 0.01 eingeschränkt werden. Für die weiteren Parameter wurden zunächst die folgenden Werte gewählt:

$$\textit{0 Skalierung} = -4, \quad \textit{1 Skalierung} = 4, \quad \textit{Glattheit} = 0, \quad \textit{Streubreite} = 0$$

Die Tabelle 5.1 verdeutlicht den Fehler des Initialisierungsverfahrens `RBF_Weights` in Abhängigkeit des Initialisierungsparameters *Bias*.

Der Initialisierungsfehler E_l ergibt sich aus der Summe der Fehlerquadrate in den Ausgabeneuronen über die verwendeten 1000 Lernmuster. Er wurde jeweils nach der Initialisierung durch einen einzigen Lernschritt des Lernverfahrens `RadialBasisLearning` berechnet. Bei diesem Lernschritt wurden alle Lernparameter auf 0.0 gesetzt, wodurch die Gewichte des Netzes unverändert blieben. Durch Laden der Testziffern in den Simulator wurde in gleicher Weise der Fehler E_t bestimmt. Der Eintrag q_E in der Tabelle zeigt das Verhältnis der beiden Fehler zueinander. Dieses Verhältnis kann als Maß für die Generalisierungsfähigkeit des Netzes gesehen werden: Je größer der Wert von q_E ist, desto spezialisierter ist das Netz auf die Lernziffern.

Bias	Initialisierungs-fehler E_l	Testfehler E_t	$q_E = \frac{E_t}{E_l}$
0.04	174.4	203.7	1.17
0.03	140.9	171.2	1.22
0.02	120.5	150.6	1.25
0.01	110.4	141.2	1.28
0.005	111.1	145.4	1.31
0.004	117.2	152.9	1.30
0.003	1468.3	—	—
0.002	8224.4	—	—

Tabelle 5.1: Initialisierungsfehler in Abhängigkeit des Bias

Aus der Tabelle ergibt sich weiterhin, daß der Initialisierungsfehler geringer wird, je kleiner der Bias gewählt wird. Der Testfehler verhält sich ebenso, jedoch wird er nicht im gleichen Maß geringer wie der Initialisierungsfehler. Wird der Bias zu klein gewählt, so zeigen sich numerische Probleme: In den untersten beiden Zeilen der Tabelle wurde der Bias so klein gewählt, daß die Neuronen der mittleren Ebene einheitlich stark aktiviert wurden, wodurch die Verbindungsgewichte nicht mehr sinnvoll berechnet werden konnten. Für die anderen Fälle führt die Initialisierung zu einem vernünftigen Ergebnis, wenn man die rein zufällige Auswahl der Zentrumsvektoren aus der Lernmenge berücksichtigt.

Aus dem Initialisierungsfehler E_l läßt sich der durchschnittliche Fehler für jedes Ausgabeneuron (die durchschnittliche Differenz zwischen Aktivierung und Lernvorgabe) wie folgt berechnen:

$$\overline{\Delta} = \sqrt{\frac{E_l}{m \cdot N}}$$

In dieser Formel steht m für die Zahl der Neuronen in der Ausgabeschicht. N gibt die Zahl der Lernmuster an. Für 1000 Lernmuster und 10 Ausgabeneuronen ergibt sich z.B. bei einem Initialisierungsfehler von 120.5 ein durchschnittlicher Fehler von $\overline{\Delta} = 0.11$ in den Ausgabeneuronen.

Für die weiteren Versuche wurde der Bias auf 0.02 festgelegt, wodurch ein Kompromiß zwischen Lernfehler und Spezialisierung des Netzes getroffen wurde.

Die Abbildung 5.2 stellt den Aufruf des Initialisierungsverfahrens im Simulator SNNS dar. Die notwendigen Parameter wurden in das kleine Initialisierungsfenster eingetragen (vergleiche Abbildung 4.6, Seite 24). Die drei Fenster in der Mitte der Abbildung zeigen die Aktivierung der Neuronen des Netzes. Von links nach rechts sind die Neuronen der Eingabe-, der mittleren und der Ausgabeschicht dargestellt. Die Kantenlänge der Quadrate, die die einzelnen Neuronen repräsentieren, ist jeweils proportional zur Aktivierung des Neurons. In der Ausgabeschicht wird zusätzlich der Name des Neurons und der numerische Wert der Aktivierung angezeigt. In der Abbildung wurde bereits zuvor eine identische Initialisierung durchgeführt, so daß die angezeigte Aktivierung der Neuronen realistisch ist.

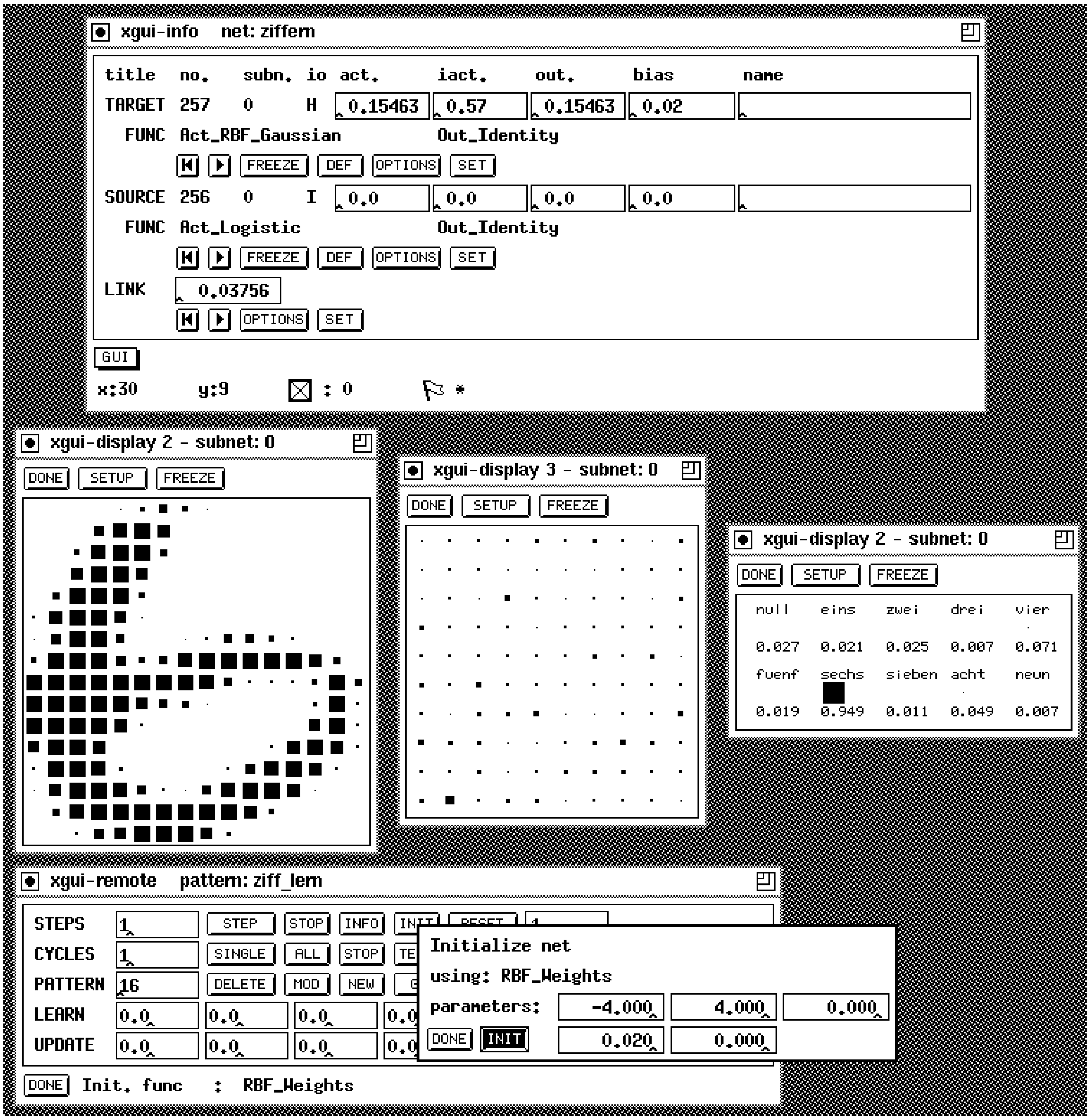

Abbildung 5.2: Initialisieren des Netzes mit SNNS

Im nächsten Schritt der Versuchsreihe wird die Abhängigkeit des Initialisierungsergebnisses von den beiden Parametern *0 Skalierung* und *1 Skalierung* demonstriert. Aus der Tabelle 5.2 wird der Fehler der Initialisierung für unterschiedliche Skalierungen ersichtlich, wobei ein Bias von 0.02 gewählt wurde. Die übrigen Parameter wurden wie bei den ersten Versuchen auf 0.0 gesetzt.

Die Bedeutung der Tabelleneinträge entspricht der Bedeutung der vorhergehenden Tabelle. Zusätzlich wurde die Spalte $\overline{\Delta_l}$ aufgenommen, die den durchschnittlichen Fehler der Ausgabeneuronen gemäß obiger Formel angibt. Berechnungsgrundlage für diese Spalte war

0 Skalierung	*1 Skalierung*	Initialisierungs-fehler E_l	Testfehler E_t	$q_E = \frac{E_t}{E_l}$	$\overline{\Delta_l}$
-1.0	1.0	865.4	891,8	1.03	0.29
-2.0	2.0	299.9	328.6	1.09	0.17
-3.0	3.0	155.4	184.3	1.18	0.12
-4.0	4.0	120.5	150.6	1.25	0.11
-5.0	5.0	112.1	143.9	1.28	0.11
-6.0	6.0	110.8	144.3	1.30	0.11
-8.0	8.0	113.2	149.0	1.32	0.11
-10.0	10.0	116.7	153.9	1.32	0.11

Tabelle 5.2: Initialisierungsfehler in Abhängigkeit der Skalierung

der Lernfehler des Netzes (E_l).

In der Tabelle ist wiederum ein deutlicher Zusammenhang ablesbar: Je größer die Skalierungsparameter gewählt werden, desto stärker wird der flache Bereich der sigmoiden Aktivierungsfunktion in den Ausgabeneuronen angesprochen (vergleiche Abbildung 4.7, Seite 26) und desto geringer ist der Initialisierungsfehler. Für zu große Skalierungswerte wächst der Fehler jedoch wieder. Der Grund hierfür ist, daß die Netzeingabe in die Ausgabeneuronen bei unterschiedlichen Lernvorgaben zu stark variiert. In der untersten Zeile der Tabelle sollen die Neuronen der Ausgabeschicht z.B. eine Eingabe von -10.0 erhalten, um eine Ausgabe von 0 zu erzeugen sowie eine Eingabe von 10.0 für die Ausgabe einer 1. Durch diese großen Gegensätze leidet die direkte Gewichtsberechnung, da die Ungenauigkeiten der Matrixinversion durch die Skalierung ebenfalls linear vergrößert werden.

Für sehr kleine Skalierungswerte wird die sigmoide Aktivierungsfunktion in ihrem mittleren Bereich angeregt. Dies führt dazu, daß die Ausgabe der Neuronen nicht mehr 0 bzw. 1 betragen kann, wie es die Lernvorgabe vorschreibt, sondern z.B. im Bereich von 0.27 bis 0.73 liegt (bei Skalierung -1, 1). Dadurch steigt der Initialisierungsfehler beträchtlich. Beachtenswert ist hierbei jedoch die Tatsache, daß der durchschnittliche Fehler in den Ausgabeneuronen $\overline{\Delta_l}$ (0.29) der durch die Initialisierung *gewollten* Abweichung von 0.27 sehr nahe kommt.

Aus der Tabelle kann außerdem abgelesen werden, daß die Fähigkeit der Generalisierung des Netzes für höhere Skalierungswerte abnimmt. Diese Tatsache spricht dagegen, daß bei der Initialisierung sehr große Skalierungswerte verwendet werden. Für die weiteren Versuche wurde daher die Skalierung auf die Werte -4.0 und 4.0 festgelegt.

Wie die nächste Tabelle veranschaulicht, ist die Modifikation der *Streubreite* der Zentrumsvektoren bei der Initialisierung nicht relevant. Dies begründet sich durch das gesamte Prinzip des Initialisierungsverfahrens. Werden 100 Lernmuster ohne Veränderung als Zentrumsvektoren eingesetzt, so überwiegen dennoch die übrigen 900 Lernmuster den Initialisierungsvorgang. Eine geringfügige Veränderung der Zentrumsvektoren muß deshalb ebenfalls zu guten Ergebnissen führen. Tabelle 5.3 zeigt den Initialisierungsfehler für unterschiedliche Streubreiten. Die übrigen Parameter wurden wie oben gewählt (Skalierung -4, 4).

Streubreite	Initialisierungs-fehler E_l	Testfehler E_t
0.0	120.5	150.6
0.1	120.3	150.5
0.2	120.4	150.7
0.3	120.4	151.3
0.5	119.9	150.9

Tabelle 5.3: Initialisierungsfehler in Abhängigkeit der Streubreite

Der letzte Inititialisierungsparameter, dessen Einfluß zu untersuchen ist, ist die *Glattheit* λ. Dieser Parameter bestimmt, wie genau die einzelnen Funktionen, die durch die Ausgabeneuronen repräsentiert werden, approximiert werden sollen bzw., wie stark der Wunsch berücksichtigt werden soll, eine möglichst glatte Funktion zu erhalten. Für die folgende Versuchsreihe wurde die Streubreite auf 0.0 festgelegt. Die übrigen Parameter wurden wie bei den vorhergehenden Versuchen gewählt.

Glattheit	Initialisierungs-fehler E_l	Testfehler E_t	$q_E = \frac{E_t}{E_l}$
0.0	120.5	150.6	1.25
0.1	125.9	155.8	1.24
0.2	133.8	163.8	1.22
0.3	219.9	249.3	1.13
0.4	219.3	293.1	1.37
0.5	152.2	183.1	1.20
0.6	150.1	178.1	1.19

Tabelle 5.4: Initialisierungsfehler in Abhängigkeit der Glattheit

Die Ergebnisse der Versuchsreihe sind in der Tabelle 5.4 veranschaulicht. Es ist zu erkennen, daß der Initialisierungsfehler größer wird, sobald für die *Glattheit* ein Wert größer als 0.0 eingesetzt wird. Dies entspricht genau dem Einfluß des Parameters, denn durch ihn wird die Approximation weniger exakt durchgeführt. Die Generalisierungseigenschaften des Netzes ändern sich unter dem Einfluß des Parameters nur minimal.

Nachdem nun der Einfluß aller Parameter auf den Initialisierungsschritt untersucht wurde, wird zuletzt die Größe des Netzes variiert. Konkret wird die Zahl der Neuronen in der mittleren Ebene verändert. Die anschließende Initialisierung wird jeweils mit den folgenden Parametern durchgeführt: *Skalierung* $-4, 4$; *Glattheit* 0.0; *Bias* 0.02; *Streubreite* 0.3. Die Tabelle 5.5 zeigt das Ergebnis der durchgeführten Testreihe.

Wie zu erwarten war, wird der Inititialisierungsfehler kleiner, je weiter sich die Größe der mittleren Schicht der Anzahl der Lernmuster annähert. Durch diese Annäherung wird das Netz immer mehr zu einer *lookup*–Tabelle, in der für jedes Lernmuster direkt die

Neuronen der mittleren Ebene	Initialisierungs-fehler E_l	Testfehler E_t	$q_E = \frac{E_t}{E_l}$
10	627.0	602.8	0.96
18	455.9	447.8	0.98
32	282.6	277.0	0.98
56	194.2	206.4	1.06
100	120.9	151.5	1.25
178	84.2	118.7	1.41
316	60.2	101.6	1.69
562	18.5	80.9	4.37
900	5.6	70.7	12.63

Tabelle 5.5: Initialisierungsfehler in Abhängigkeit der Netzgröße

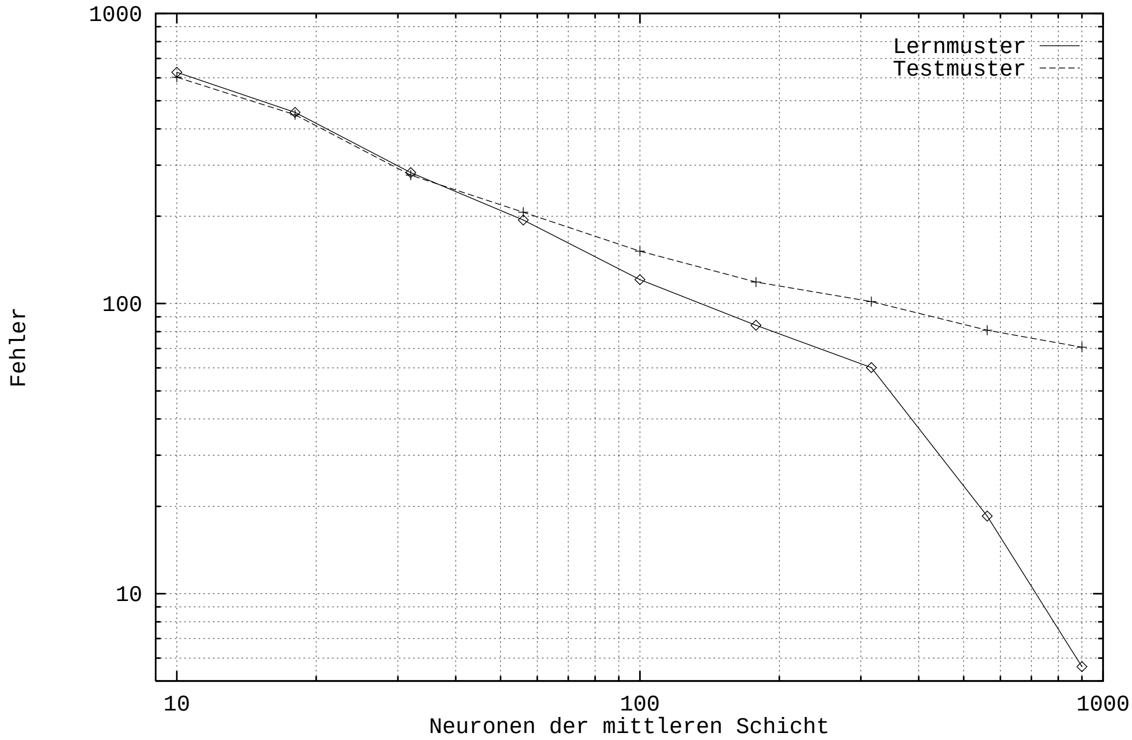

Abbildung 5.3: Initialisierungsfehler bei unterschiedlicher Netzgröße

zugehörige Netzausgabe gespeichert ist. Dies zeigt sich auch daran, daß der Fehler für die Testmenge nicht in gleicher Weise sinkt.

Abbildung 5.3 veranschaulicht den Zusammenhang noch einmal in einer graphischen Darstellung. Für beide Achsen des Graphen wurde ein logarithmischer Maßstab gewählt. Dadurch verläuft in der Darstellung der Zusammenhang zwischen Netzgröße und Initialisierungsfehler nahezu linear, soweit die Zahl der Neuronen in der mittleren Ebene nicht über 300 steigt. Aus der graphischen Darstellung ist auch gut die Spezialisierung des Netzes auf die Lernmuster bei wachsender Netzgröße zu erkennen, da die beiden Kurven immer weiter auseinander laufen.

Bei den bisherigen Versuchen wurden die Zentrumsvektoren für die Initialisierung jeweils direkt aus den vorhandenen Lernmustern ausgewählt. Bei den theoretischen Betrachtungen des Verfahrens wurde jedoch bereits erwähnt, daß diese Auswahl keineswegs optimal ist. Deshalb wird nun noch eine andere Möglichkeit untersucht, die Zentrumsvektoren zu bestimmen. Diese Möglichkeit wird durch das Initialisierungsverfahren `RBF_Weights_Kohonen` realisiert.

Um gute Ergebnisse bei der Anwendung des Verfahrens von Kohonen zu erhalten, müssen die Lernmuster zunächst normiert werden. Jede Ziffer wird als Vektor im 256–dimensionalem Raum aufgefaßt, dessen Länge den Wert 1 beträgt. Durch die Normierung der Eingabemuster verändert sich die Empfindlichkeit der Zentrumsfunktionen, so daß der Bias der mittleren Neuronen auf 0.8 eingestellt werden muß.

Zahl der Lernschritte	Initialisierungsfehler E_l	Testfehler E_t
0	107.2	137.3
10	87.3	125.4
20	82.6	122.6
40	81.7	121.1

Tabelle 5.6: Initialisierungsfehler in Abhängigkeit des Kohonen–Trainings

In Tabelle 5.6 sind die Initialisierungsfehler aufgelistet. Die erste Spalte gibt die Anzahl der Lernschritte des Kohonen–Trainings an. Die Lernrate für das Kohonen–Training wurde jeweils auf den Wert 0.4 gesetzt (vergleiche Abbildung 4.8, Seite 27). Bei der nachträglichen Initialisierung des Netzes mit dem Verfahren `RBF_Weights_Redo` wurden den Werten für die Skalierung die üblichen -4.0 und 4.0 zugewiesen (siehe Abbildung 4.9, Seite 28).

Es fällt sofort auf, daß bereits durch die Normierung der Muster eine Verbesserung der Initialisierung zu erzielen ist (oberste Zeile der Tabelle). Durch das selbstorganisierende Training der Zentrumsvektoren kann der Fehler weiter vermindert werden. Es zeichnet sich jedoch auch hier sehr bald, das der Initialisierungsfehler eine bestimmte Grenze nicht unterschreitet.

Durch die Versuchsreihe wird deutlich, daß das Initialisierungsergebnis sehr stark von der Wahl der Zentrumsvektoren abhängig ist. Andere Verfahren könnten unter Umständen noch weitaus bessere Ergebnisse liefern. Die richtige Bestimmung der Zentrumsvektoren ist deshalb ein vielversprechendes Gebiet weiterer Forschung.

5.1.2 Training des initialisierten Netzes

Bisher wurde ausschließlich die Initialisierung des Netzes betrachtet. Im folgenden wird nun das Lernverhalten des Netzes untersucht. Ausgangsbasis für die einzelnen Versuchsreihen ist jeweils ein Netz mit 100 Neuronen in der mittleren Ebene, das mit folgenden Parametern initialisiert wurde: *Skalierung* $-4, 4$; *Glattheit* 0.0; *Bias* 0.02; *Streubreite* 0.3. Als Lernmuster werden, wie bisher, 1000 handgeschriebene Ziffern verwendet.

Zu Anfang der Versuche wurde jeweils nur eine der drei Lernraten (*Zentren*, *Bias (p)*, *Gewichte*) auf einen von 0.0 verschiedenen Wert gesetzt. Hierdurch konnte die Stabilität des Verfahrens gezielt untersucht werden.

Die Abbildungen 5.4 und 5.5 veranschaulichen den Verlauf des Lernfehlers für die ersten 10 Lernschritte bei unterschiedlicher Wahl des Lernparameters *Gewichte*. Alle anderen Lernparameter (auch *max. Fehler* und *Momentum*) wurden auf 0.0 gesetzt.

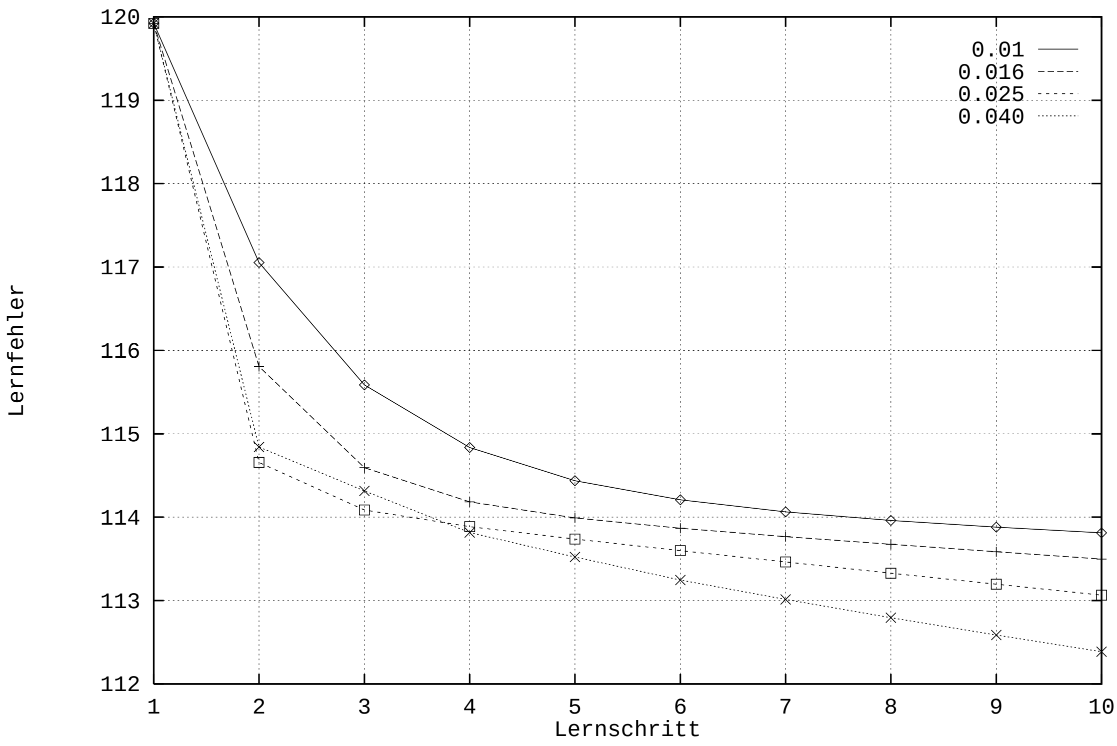

Abbildung 5.4: Lernfehler für Gewichtslernrate kleiner 0.04

Unter einem Lernschritt versteht man, daß alle 1000 Lernmuster genau einmal dem Netz präsentiert werden (eine andere übliche Bezeichnung hierfür ist *Lernepoche*). Bei jeder Präsentation werden die Aktivierungen aller Neuronen berechnet (das Muster wird *durch–propagiert*), und die Ausgabe des Netzes wird mit der Lernvorgabe verglichen. Daraufhin werden notwendige Gewichtsänderungen berechnet, die aber zunächst nicht durchgeführt werden. Nach Präsentation aller 1000 Lernmuster werden alle Gewichte gleichzeitig

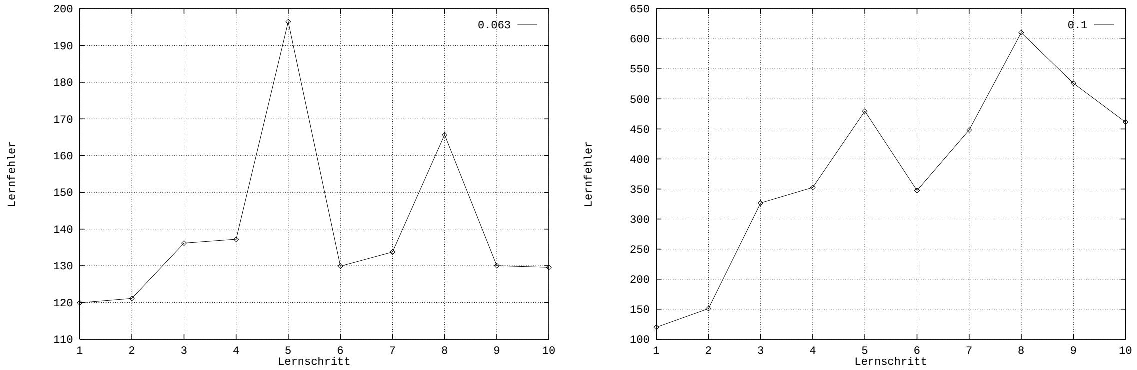

Abbildung 5.5: Lernfehler für Gewichtslernrate größer 0.04

verändert, wodurch der aktuelle Lernschritt beendet wird.

Die Fehler der Ausgabeneuronen, die während eines Lernschritts quadratisch aufsummiert wurden, ergeben den Fehler des Netzes. Dieser Fehler ist im ersten Lernschritt unabhängig von den Lernraten, was in den graphischen Darstellungen gut erkennbar ist.

Die Graphiken zeigen, daß ein stabiles Lernverhalten nur dann gewährleistet werden kann, wenn die Lernrate für die Gewichte kleiner als 0.04 gewählt wird. Andernfalls sind die Änderungen, die pro Lernschritt durchgeführt werden, so groß, daß der Netzfehler nicht minimiert werden kann.

Im nächsten Schritt der Untersuchung wurde die Versuchsreihe wiederholt, wobei nun jedoch die Lernrate für die *Zentren* variiert wurde. Auch bei diesen Versuchen wurden jeweils alle anderen Lernparameter auf 0.0 gesetzt, so daß ausschließlich die Zentrumsvektoren trainiert wurden.

Ebenso wie in der ersten Versuchsreihe zeigt sich in den Abbildungen 5.6 und 5.7, daß die Lernrate für die Zentren einen Wert von 0.04 nicht übersteigen darf, um ein stabiles Lernverhalten zu garantieren.

In einer dritten Versuchsreihe sollte der Einfluß des Parameters p der Zentrumsfunktionen (*Bias* (p)) auf die Lernrate untersucht werden. Es zeigte sich jedoch, daß eine Lernrate, deren Wert größer als 10^{-7} ist, zu so starken Veränderungen der Parameter p führt, daß kein stabiles Lernen möglich ist. Zumindest für den Fall der Erkennung handgeschriebener Ziffern gilt also, daß der Bias der Neuronen in der mittleren Ebene nicht sinnvoll trainiert werden kann. Hierfür gibt es mindestens zwei Gründe: Zum einen wurde der Bias der Neuronen bei der Initialisierung auf einen Wert von 0.02 eingestellt. Geringe Veränderungen haben also bereits eine große Wirkung. Zweitens sind die Lernmuster untereinander relativ ähnlich (jeweils etwa 100 Ziffern der gleichen Klasse). Diese Ähnlichkeit führt dazu, daß die Veränderung des Parameters p einer Zentrumsfunktion Einfluß auf sehr viele Trainingsmuster nimmt. Im Abschnitt 5.3.2 wird jedoch gezeigt, daß dieser Parameter bei anderen Anwendungen durchaus wirkungsvoll trainiert werden kann.

Abschließend wurde nun ein initialisiertes Netz so gut wie möglich trainiert, wobei sowohl die Zentrumsvektoren als auch die übrigen Gewichte verändert wurden. Der Bias der Neuronen der mittleren Schicht wurde jedoch nicht trainiert.

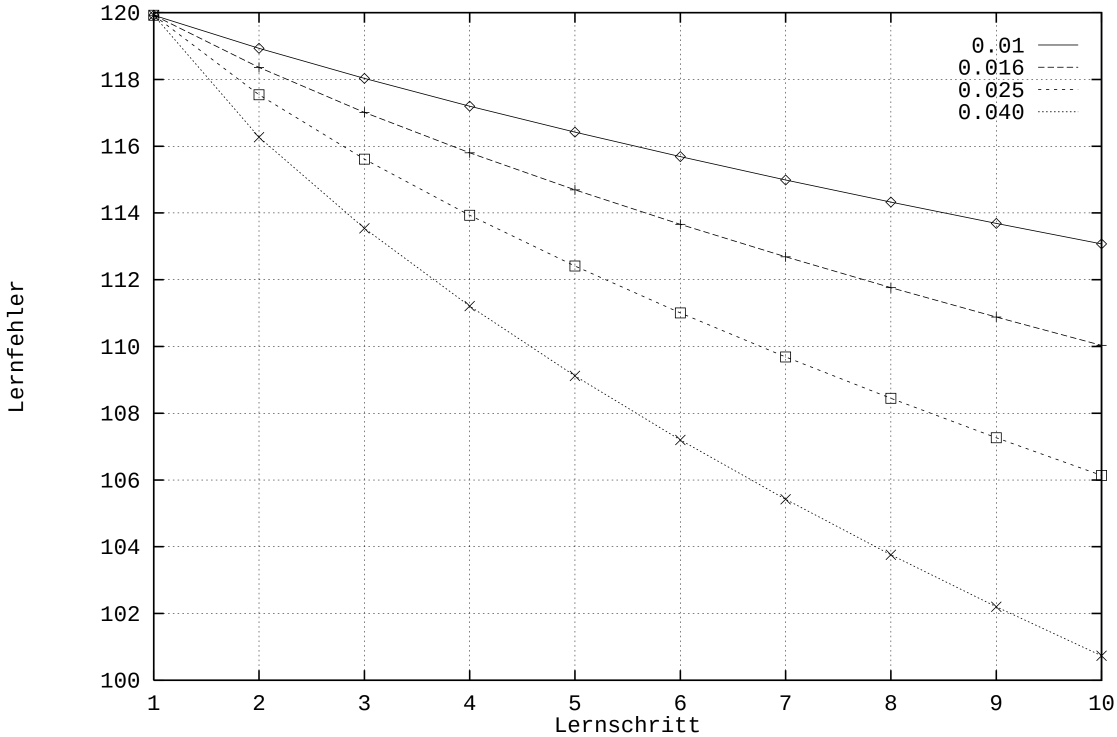

Abbildung 5.6: Lernfehler für Zentrumslernrate kleiner 0.04

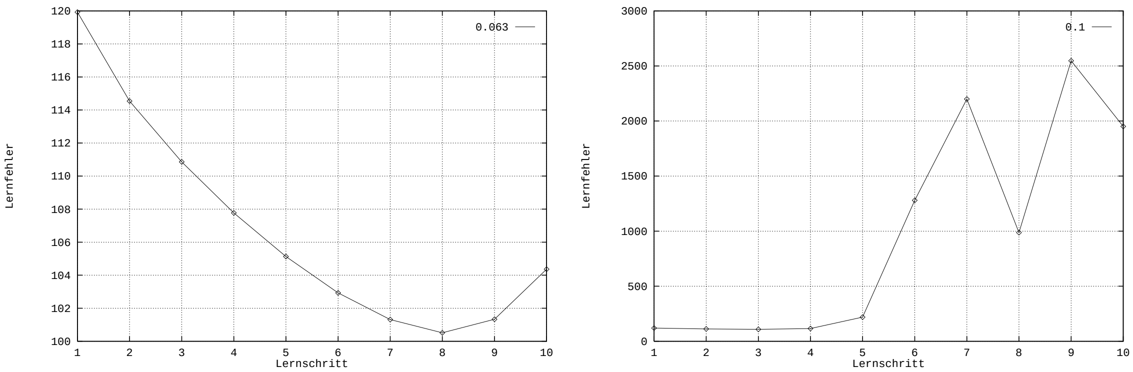

Abbildung 5.7: Lernfehler für Zentrumslernrate größer 0.04

Das Netz wurde zunächst mit folgenden Parametern initialisiert: *Skalierung* −4, 4; *Glattheit* 0.0; *Bias* 0.02; *Streubreite* 0.3. Anschließend wurde das Lernverfahren mit relativ geringen Lernraten gestartet: *Zentren*: 0.01; *Bias (p)*: 0.0; *Gewichte*: 0.01 *max. Fehler*: 0.1 *Momentum*: 0.8.

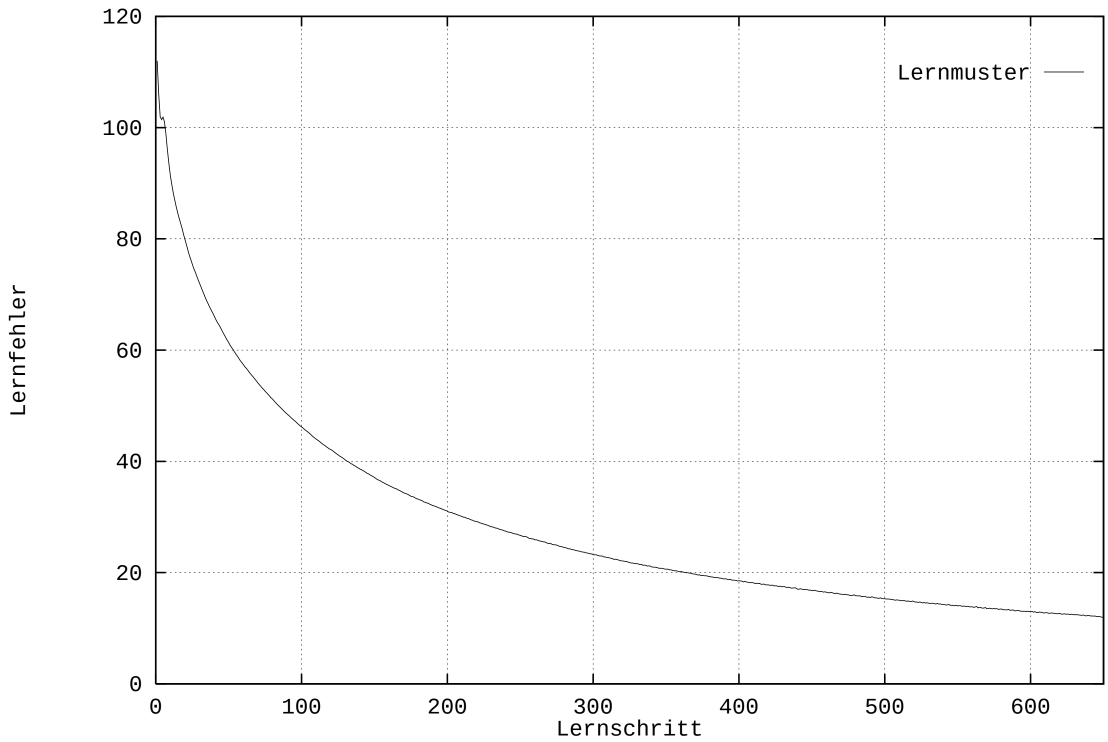

Abbildung 5.8: Verhalten des Lernfehlers bei defensivem Lernen

Das Ergebnis des Trainings ist in Abbildung 5.8 wiedergegeben. Zu beachten ist, daß während des Lernvorgangs der Lernparameter *max. Fehler* auf 0.1 gesetzt wurde. Das bedeutet, daß nur diejenigen Fehler der Ausgabeneuronen in den Gesamtfehler eingingen, deren Betrag größer als 0.1 war.

Im Anschluß an das Training wurde deshalb der Gesamtfehler für unterschiedliche Werte von *max. Fehler* bestimmt. Hierbei wurden gleichzeitig die Eingabemuster gezählt, die bei mindestens einem der Ausgabeneuronen zu einer Abweichung führen, die *max. Fehler* übersteigt. Durch diese Zählung kann die Erkennungsrate des Netzes bei unterschiedlichen Anforderungen an die Ausgabegenauigkeit bestimmt werden. Die Untersuchungen wurden sowohl für die Lernmuster, die für das Training des Netzes verwendet wurden, als auch für die unabhängigen Testmuster durchgeführt.

In Tabelle 5.7 sind die Fehler des Netzes für unterschiedliche Werte von *max. Fehler* dargestellt. Die Bedeutung der einzelnen Spalten ist wie folgt:

Δ_{max} steht für den Lernparameter *max. Fehler* und bestimmt die geforderte Genauigkeit der Netzausgabe. E_l und E_t geben wie üblich die Fehler des Netzes für die Lernmuster und

Δ_{max}	Trainingsmuster			Testmuster		
	E_l	$\sum_l$	R_l	E_t	$\sum_t$	R_t
0.0	19.0	1000	0%	97.4	1000	0%
0.1	12.1	357	64.3%	91.6	453	54.7%
0.2	4.3	50	95.0%	84.1	270	73.0%
0.3	1.9	8	99.2%	75.9	178	82.2%
0.4	1.4	3	99.7%	68.6	136	86.4%
0.5	1.2	2	99.8%	59.5	104	89.6%
0.6	0.9	1	99.9%	49.4	74	92.6%
0.7	0.9	1	99.9%	37.1	48	95.2%
0.8	0.9	1	99.9%	25.6	31	96.9%
0.9	0.9	1	99.9%	13.8	15	98.5%
1.0	0.0	0	100%	0.0	0	100%

Tabelle 5.7: Verteilung des Fehlers nach dem Training

die Testmuster an. Die Spalten $\sum_l$ und $\sum_t$ enthalten jeweils die Zahl der Muster, die bei mindestens einem der Ausgabeneuronen zu einem Fehler, dessen Wert größer als Δ_{max} ist, führen. Entsprechend verdeutlichen R_l und R_t den Prozentsatz der Ziffern, die gemäß Δ_{max} fehlerfrei erkannt werden konnten.

Die untere Hälfte der Tabelle ist von geringerer Bedeutung, da hier der erlaubte Fehler Δ_{max} größer als 50% des möglichen Ausgabebereichs ist. Die Daten wurden dennoch aus Gründen der Vollständigkeit angegeben.

Interessant ist folgende Überlegung über eine konkrete Anwendung des fertig trainierten Netzes: Eine Ziffer soll als „erkannt“ gelten, falls die Aktivierung des zugehörigen Ausgabeneurons größer als 0.6 und gleichzeitig die Aktivierung aller anderen Augabeneuronen kleiner als 0.4 ist. Für diese Regel werden 86.4% aller unbekannten Ziffern richtig klassifiziert.

In höchstens 7.4% der Fälle ist eine Fehlklassifikation möglich (Fehlerrate für Δ_{max} gleich 0.6). Eine Fehlklassifikation tritt jedoch nur dann auf, wenn die Aktivierung des *richtigen* Neurons unter 0.4 liegt, wobei gleichzeitig die Aktivierung genau eines anderen Ausgabneurons größer als 0.6 ist. Das Netz besitzt jedoch eine größere Tendenz, das *richtige* Neuron nicht zu aktivieren als ein *falsches* Neuron zu aktivieren, da in 9 von 10 Fällen eine Nichtaktivierung richtig ist.

Der Anteil der falsch klassifizierten Ziffern wird also weit unterhalb des Wertes von 7.4% liegen. Hieraus folgt wiederum, daß das Netz in mindestens 6.2%, aber höchstens 13.6% der Fälle keine Klassifikation gemäß obiger Regel durchführen kann.

In einer letzten Untersuchung wird nun nochmals das Lernverhalten des Netzes betrachtet. Ausgangssituation für diese Untersuchung ist wiederum ein initialisiertes Netz, wie es auch für den vorhergehenden Trainingslauf verwendet wurde. Nun werden jedoch größere Lernraten für die *Zentren* und *Gewichte* gewählt. Außerdem wird parallel zum Training immer wieder der aktuelle Fehler der Testmenge bestimmt. Hierdurch kann das Training abgebrochen werden, sobald sich der Fehler für die Testmenge nicht mehr entscheidend ändert, bzw. bevor sich das Netz zu sehr auf die Lernmuster spezialisiert und seine Generalisierungseigenschaften verliert.

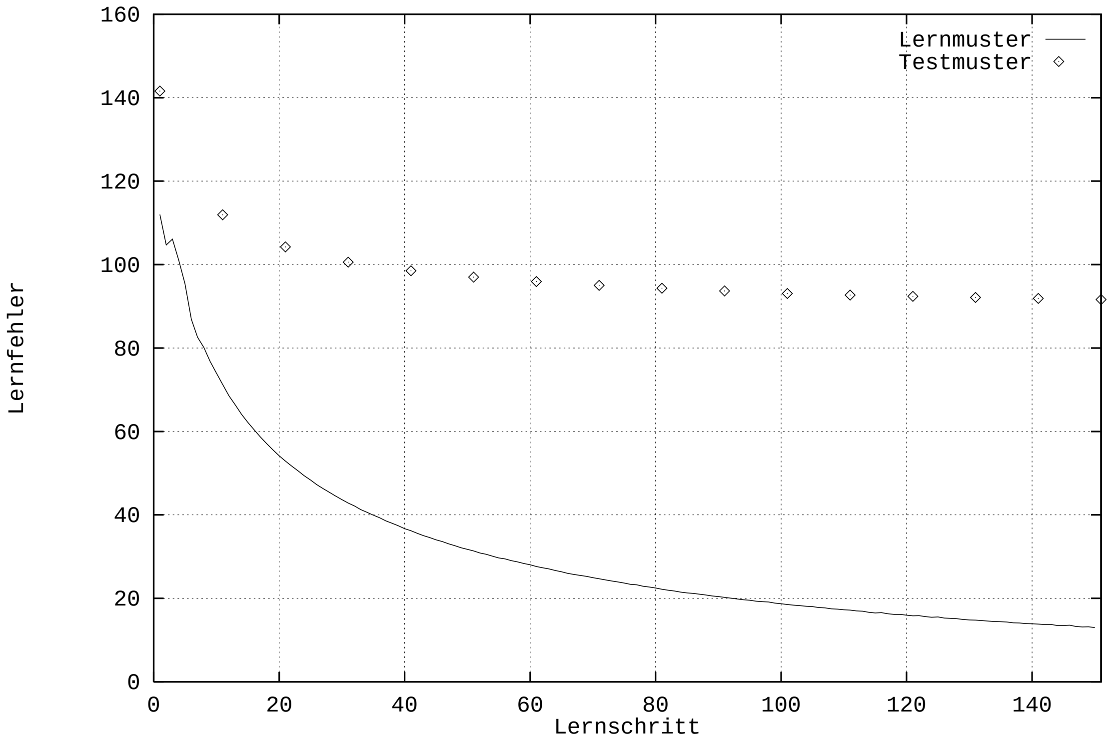

Abbildung 5.9: Verhalten des Fehlers für Lern- und Testmenge

In Abbildung 5.9 ist das Verhalten des Fehlers für die Lernmenge und die Testmenge dargestellt. Nach jeweils 10 Lernschritten wurde der Fehler für die Testmenge bestimmt. Die Lernparameter bei diesem Trainingslauf waren: *Zentren*: 0.04; *Bias (p)*: 0.0; *Gewichte*: 0.04 *max. Fehler*: 0.1 *Momentum*: 0.8.

Wie man aus dem Verlauf des Fehlers für die Testmenge ersieht, ist es nicht sinnvoll, mehr als 150 Lernschritte durchzuführen. Der Fehler veringert sich nach dieser Anzahl von Schritten nur noch minimal. Da die Lernraten für die *Zentren* und *Gewichte* um den Faktor 4 größer sind als beim vorhergehenden Trainingslauf, beträgt die Zahl der notwendigen Lernschritte nur etwa ein Viertel des obigen Aufwands (150 gegenüber 650 Lernschritte). Eine weitere Steigerung der Lerngeschwindigkeit ist leider nicht möglich, da das Lernverfahren für Lernraten größer als 0.04 instabil wird.

Um einen Vergleich der Qualität des Verfahrens der Radial Basis Functions mit anderen Netzmodellen anstellen zu können, wurde abschließend ein Versuch mit einem alternativen Verfahren durchgeführt.

Es wurde ein äquivalentes Netz mit 256 Eingabeneuronen, 100 Neuronen der mittleren Schicht sowie 10 Ausgabeneuronen definiert. Für alle Neuronen wurde die sigmoide Aktivierungsfunktion `Act_Logistic` ausgewählt. Anschließend wurden alle Verbindungsgewichte auf einen zufälligen Wert zwischen -1.0 und 1.0 gesetzt.

Das mit zufälligen Werten initialisierte Netz wurde mit dem bekannten Lernverfahren *Backpropagation* trainiert, wobei auch hier von einem Momentum–Term und von einem maximal zulässigen Fehler Gebrauch gemacht wurde. Für das Training wurden die folgenden Lernparameter gewählt: Lernrate η: 0.6; Faktor für Momentum–Term: 0.8; maximal zulässiger Fehler in den Ausgabeneuronen: 0.1.

Bereits nach 13 Lernschritten erreichte das Netz einen Fehler von 12.5 für die 1000 Lernmuster. Der Fehler wird in gleicher Weise berechnet wie beim Verfahren der Radial Basis Functions und kann deshalb direkt mit dem Lernfehler aus den obigen Versuchen verglichen werden.

Nach 13 Lernschritten hat das Netz also bereits den gleichen Fehler erreicht, der, aufgrund der geringeren Lernraten, bei den obigen Versuchen erst nach 150 bzw. 650 Schritten erreicht wurde. Der Fehler des Netzes für die unabhängigen Testmuster lag nach diesen 13 Lernschritten bereits bei 56.0 im Vergleich zu einem bestmöglichen Fehler von etwa 90 beim Verfahren der Radial Basis Functions .

Nach weiteren 20 Lernschritten mit Backpropagation erreichte der Fehler für die Lernmuster einen Wert von 3.0 und änderte sich im weiteren Verlauf des Trainings nur noch unwesentlich. Der entsprechende Fehler für die Testmuster lag zu diesem Zeitpunkt bei 44.2.

5.2 Vokale der englischen Sprache

Die zweite Anwendung, die vorgestellt wird, ist die Erkennung von gesprochenen Vokalen der englischen Sprache. Die notwendigen, vorverarbeiteten Daten stammen von D.H. Deterding ([Det89]). Sie wurden in folgender Weise gewonnen:

15 Personen (acht Männer und sieben Frauen) sprachen elf verschiedene englische Wörter jeweils sechsmal. Die zu sprechenden Wörter waren: *heed, hid, head, had, hard, hud, hod, hoard, hood, who'd, heard.* Diese Wörter enthalten alle elf Vokale, die sich phonetisch voneinander unterscheiden lassen. Für die Vorverarbeitung wurde das akustische Signal mit einem Tiefpaß gefiltert (Grenzfrequenz 4.7kHz) und mit einer Auflösung von 12 Bit bei einer Frequenz von 10kHz digitalisiert. Aus dem stimmhaften Bereich des digitalen Signals wurden durch ein Verfahren der Sprachverarbeitung jeweils zehn Parameter gewonnen, die das Phonem des Vokals repräsentieren.

T. Robinson hat umfangreiche Tests mit dem beschriebenen Datenmaterial durchgeführt ([RF88]). Er untersuchte unterschiedlichste Typen von neuronalen Netzen auf ihre Fähigkeit, die dargebotenen Vokale zu klassifizieren. Unabhängig vom Netztyp erreichte er eine bestmögliche Erkennungsrate von etwa 50% der Vokale, die nicht für das Training des Netzes herangezogen wurden. Er stufte daraufhin das Problem als „nicht mit konnektionistischem Ansatz lösbar“ ein.

Für meine Untersuchungen habe ich mich ebenfalls an die Versuchsbedingungen von Robinson gehalten, um die Ergebnisse vergleichen zu können:

Als Trainingsmenge für das neuronale Netz dienten die Daten der ersten acht Personen (vier Männer und vier Frauen). Bei elf Vokalen und sechs Sprechproben pro Person ergibt dies insgesamt 528 Trainingsmuster. Die übrigen 462 Muster der verbleibenden sieben Personen bildeten die Menge der Testmuster.

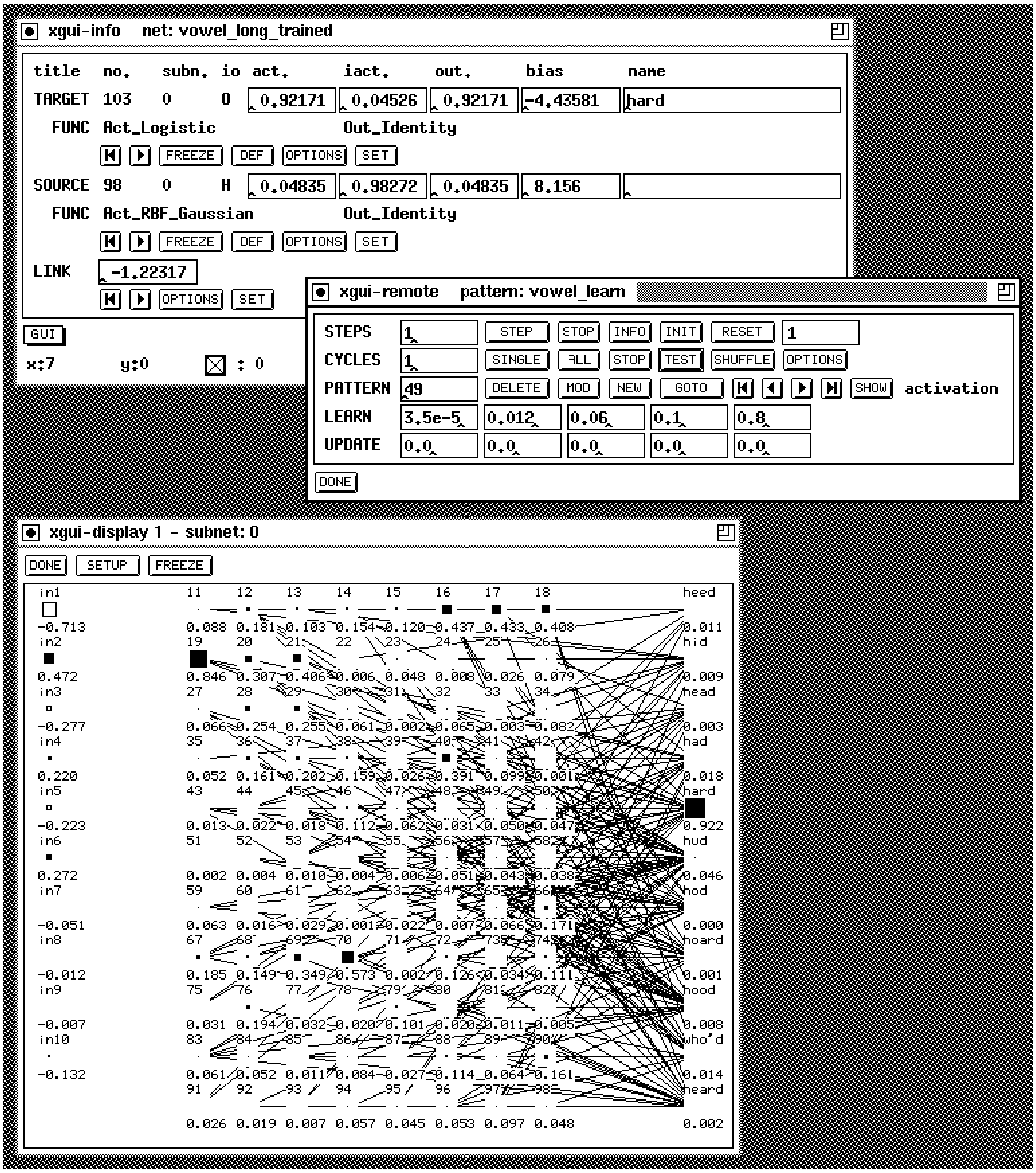

Abbildung 5.10: Erkennung von Vokalen mit SNNS

Für die Versuchsdurchführung wurden zwei dreistufige Netze definiert. Die Eingabeschicht, an die die aufbereiteten Daten angelegt wurden, bestand jeweils aus zehn Neuronen. In der mittleren Schicht wurden im ersten Netz 88 Neuronen und im zweiten Netz 527 Neuronen definiert. Diese Größen entsprechen den Vorgaben von Robinson. Als Aktivierungsfunktion für die mittleren Neuronen wurde, wie bei der Ziffernerkennung, die Gaußsche Zentrumsfunktion `Act_RBF_Gaussian` gewählt. Die Ausgabeschicht des Netzes bildeten jeweils elf Neuronen, deren Aktivierung das angelegte Muster klassifizieren sollten. Die Aktivierung dieser Neuronen erfolgte über die logistische Funktion `Act_Logistic`.

5.2.1 Initialisierung des Netzes

Für die Initialisierung der Netze wurde zunächst das Inititalisierungsverfahren `RBF_Weights` verwendet. Da die Ausgabeneuronen über die logistische Funktion aktiviert werden, ist es notwendig, für die Initialisierung geeignete *Skalierungs*parameter zu finden. Die Parameter -4.0 und 4.0 stellten sich als günstig heraus und wurden für alle folgenden Versuche verwendet.

Für die Parameter der *Glattheit* und der *Streubreite* wurde der Wert 0.0 eingesetzt. Versuche zeigten, daß das Ergebnis der Initialisierung nicht wesentlich beeinflußt wird, wenn für diese Parameter andere Werte gewählt werden (siehe auch Abschnitt 5.1.1).

Es verblieb demnach die Untersuchung des Initialisierungsergebnisses in Abhängigkeit des Bias der mittleren Neuronen. Hierzu wurden für die zwei betrachteten Netze zwei Versuchsreihen durchgeführt:

Bias	88er Netz		527er Netz	
	Initialisierungsfehler E_l	Testfehler E_t	Initialisierungsfehler E_l	Testfehler E_t
1	4929.6	4118.5	3706.2	3213.9
3	64.7	399.6	3709.9	3264.3
5	61.6	376.3	2942.1	2879.5
7	62.9	368.1	2894.8	2726.1
8	64.6	366.9	2829.8	2553.8
9	66.6	367.1	3065.4	3435.0
10	68.8	367.8	2730.2	2577.3
20	92.6	389.4	2873.9	2455.4
50	156.5	425.8	318.6	928.2
100	240.3	433.9	1.9	800.5

Tabelle 5.8: Initialisierungsfehler in Abhängigkeit des Bias

Aus der Tabelle 5.8 ergibt sich, daß das große Netz mit 527 Neuronen in der mittleren Ebene nur mit einem sehr hohen Bias initialisiert werden kann. Für kleinere Werte existieren offensichtlich numerische Probleme, da für die einzelnen Muster jeweils zu viele Neuronen in der mittleren Ebene aktiviert werden. Hierbei kommt es vor, daß die Aktivierungsmuster der mittleren Schicht für unterschiedliche Vokale sehr ähnlich sind. Das Verfahren der Initialisierung ist dann nicht mehr in der Lage, die Gewichte so zu berechnen, daß unterschiedliche Netzausgaben erzeugt werden können.

Ein sehr großer Bias führt dazu, daß jeweils nur genau das Neuron der mittleren Schicht aktiviert wird, dessen Zentrumsvektor dem aktuell anliegenden Muster entspricht. Hierdurch entartet das Netz aber zu einer *lookup*–Tabelle und entspricht nicht mehr der Grundidee der neuronalen Netze. Im weiteren wurde deshalb auf die Untersuchung des Netzes mit 527 Neuronen in der mittleren Schicht verzichtet.

Betrachtet man die Initialisierungsergebnisse für das Netz mit 88 Neuronen in der mittleren Ebene, so erkennt man, daß ein Bias von 8.0 zu den besten Ergebnissen bei der Erkennung der Testmuster führt. Im weiteren Verlauf wurde daher bei der Initialisierung des Netzes dieser Wert eingesetzt.

Aus der Tabelle ist ebenfalls abzulesen, daß die Generalisierungseigenschaften des Netzes sehr schlecht sind. Um die tatsächliche Erkennungsrate des Netzes zu bestimmen, mußte eine Regel definiert werden. Diese besagt, daß ein Vokal als erkannt gilt, wenn die Aktivierung des entsprechenden Ausgabeneurons größer ist als die Aktivierung aller anderen Ausgabeneuronen. Bei einem Bias von 8.0 erkennt das Netz mit 88 mittleren Neuronen unter Beachtung dieser Regel 183 der 462 dargebotenen Testmuster. Dies entspricht einer Erkennungsrate von 39.6%.

5.2.2 Training des initialisierten Netzes

Das Netz mit 88 Neuronen in der mittleren Schicht wurde nun trainiert. Hierzu mußten zunächst die größtmöglichen Lernraten bestimmt werden, die noch nicht zu einem instabilen Lernverhalten führen. Für jede der drei Lernraten (*Zentren, Bias (p), Gewichte*) wurden mehrere Versuche gemacht, wobei jeweils nur einer der drei Parameter auf einen von 0.0 verschiedenen Wert gesetzt wurde.

Es ergaben sich die folgenden drei Grenzwerte, die noch ein stabiles Training gewährleisten:

- *Zentren*: 0.000035

  Dieser Wert ist sehr klein. Der Grund hierfür ist, daß sich die Trainingsmuster untereinander sehr ähnlich sind. Hierdurch werden bei der Initialisierung für die Zentrumsvektoren ebenfalls sehr ähnliche Werte ausgewählt. Eine sehr geringe Änderung eines solchen Vektors beeinflußt deshalb sehr stark die gesamte Aktivierung der mittleren Schicht.

- *Bias (p)*: 0.012

  Obwohl dieser Wert ebenfalls sehr klein ist, kann durch ihn der Bias der mittleren Neuronen sinnvoll trainiert werden (im Gegensatz zur Erkennung der handgeschriebenen Ziffern im vorhergehenden Abschnitt).

- *Gewichte*: 0.06

Verwendet man nun alle drei Lernraten gleichzeitig, so kann es dennoch zu einem instabilen Trainingsverhalten kommen, obwohl jede Lernrate, für sich betrachtet, optimal gewählt ist.

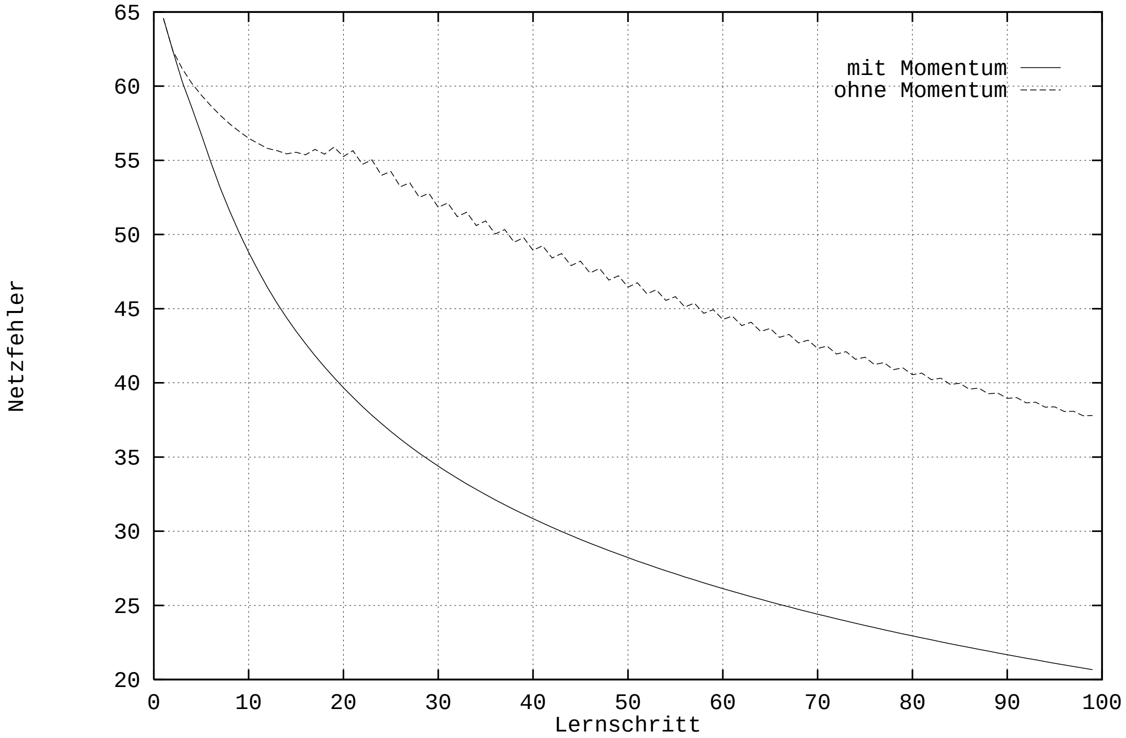

Abbildung 5.11: Einfluß des Momentum–Terms auf das Lernverhalten

In der Abbildung 5.11 sind zwei Lernkurven dargestellt, wobei jeweils die genannten Grenzwerte als Lernraten eingesetzt wurden. Bei einem der beiden Trainingsläufe wurde zusätzlich der Wert des Momentum–Terms auf 0.8 gesetzt. In der Graphik zeigt sich der doppelt positive Einfluß des Momentums: Zum einen wird die Lerngeschwindigkeit wesentlich gesteigert. Der zweite beobachtbare Effekt ist die dämpfenden Wirkung des Momentums. Das unruhige Lernverhalten, das beim Training ohne Momentum auftritt, wurde durch den Einsatz dieses Terms vollständig kompensiert.

Bereits bei der Initialisierung des Netzes wurde angesprochen, daß die Erkennungsrate der unbekannten Testmuster sehr gering ist. Es ist deshalb eine Untersuchung notwendig, inwieweit ein anschließender Trainingslauf überhaupt sinnvoll ist. Diese Untersuchung wurde in folgender Weise durchgeführt:

Ein initialisiertes Netz wurde mit oben angegebenen Lernraten und einem Momentum–Term von 0.8 schrittweise trainiert. Jeweils nach Abschluß eines Lernschrittes, d.h. nach Präsentation aller Trainingsmuster und anschließender Modifikation aller Gewichte, wurde das Netz in einen zweiten Simulator geladen. In diesem zweiten Simulator wurde der Fehler des Netzes für die Testmuster bestimmt, indem ein Lernschritt ausgeführt wurde. Die auftretenden Netzfehler wurden protokolliert und sind in den Abbildungen 5.12 und 5.13 wiedergegeben.

Obwohl der Fehler für die Trainingsmuster kontinuierlich abnimmt, ergibt sich keine

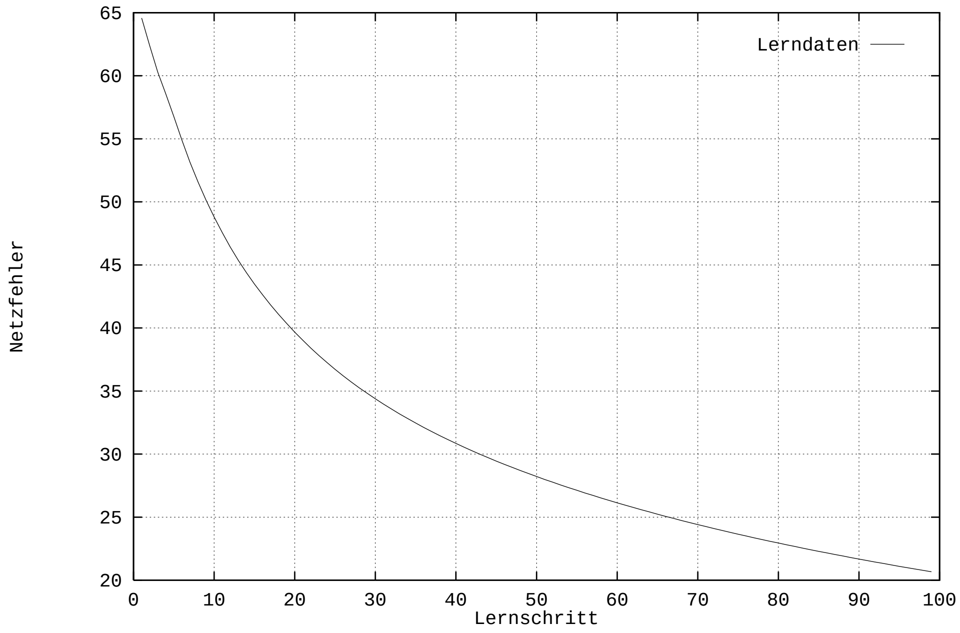

Abbildung 5.12: Verhalten des Netzfehlers für die Trainingsmuster

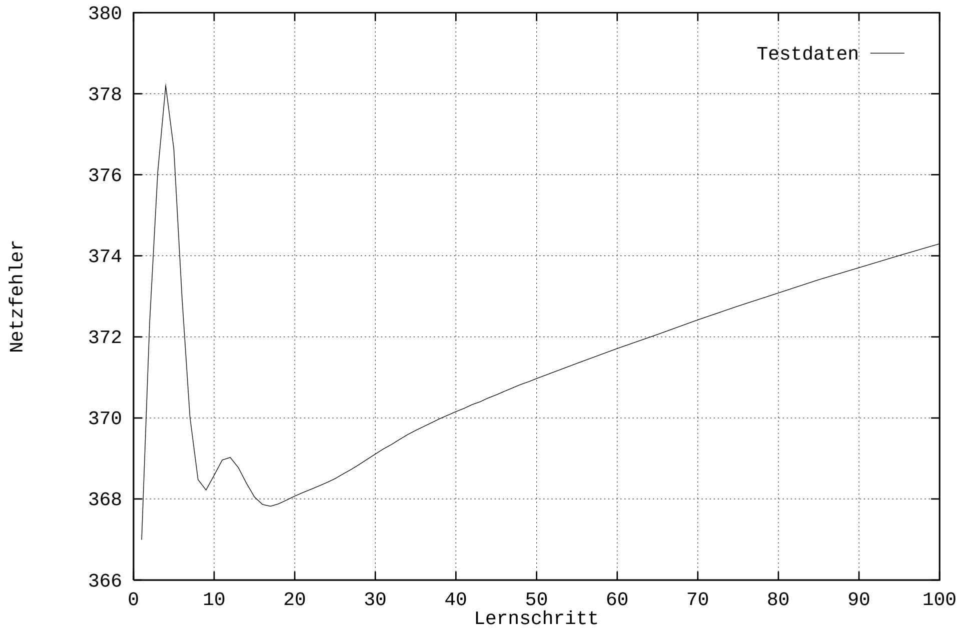

Abbildung 5.13: Verhalten des Netzfehlers für die Testmuster

Verbesserung des Netzes für die Testmuster. Der Grund für dieses Verhalten besteht darin, daß das Minimum der Fehlerfläche für die Trainingsdaten, das durch den Gradientenabstieg gesucht wird, nicht mit dem Minimum für die Testdaten übereinstimmt. Es ist deshalb grundsätzlich nicht möglich, durch das Training eine Verbesserung der Erkennungsraten für unbekannte Vokale zu erzielen. Dies entspricht dem Ergebnis von Robinson ([RF88]).

5.3 Approximation einer Funktion

Die letzte der drei untersuchten Anwendungen stellt die Approximation einer mathematischen Funktion dar. Sie wurde gewählt, da die Theorie der Radial Basis Functions aus dem Bereich der klassischen Approximationstechniken stammt. An drei einfachen Netzen wird gezeigt, daß neuronale Netze, die Radial Basis Functions verwenden, allgemeine Approximationsaufgaben lösen können.
Es soll folgende Funktion approximiert werden:

$$f(x) = e^{-5x^2} sin(3\pi x) + x^3 \quad , \quad x \in [-1, 1]$$

Insgesamt wurden hierfür 100 Stützstellen der Funktion bestimmt, die gleichmäßig über den Bereich $[-1, 1]$ verteilt sind. Um die Approximation einer Funktion zu simulieren, die auf Meßdaten beruht, wurden diese Stützdaten mit einem zufälligen Rauschanteil überlagert. Bei der Bestimmung dieses Rauschanteils wurde eine Normalverteilung angenähert, so daß geringe Abweichungen vom richtigen Funktionswert wahrscheinlicher waren als sehr große Abweichungen. Dies entspricht z.B. der Verteilung des Fehlers, der beim mechanischen Abtasten einer Oberfläche auftritt.

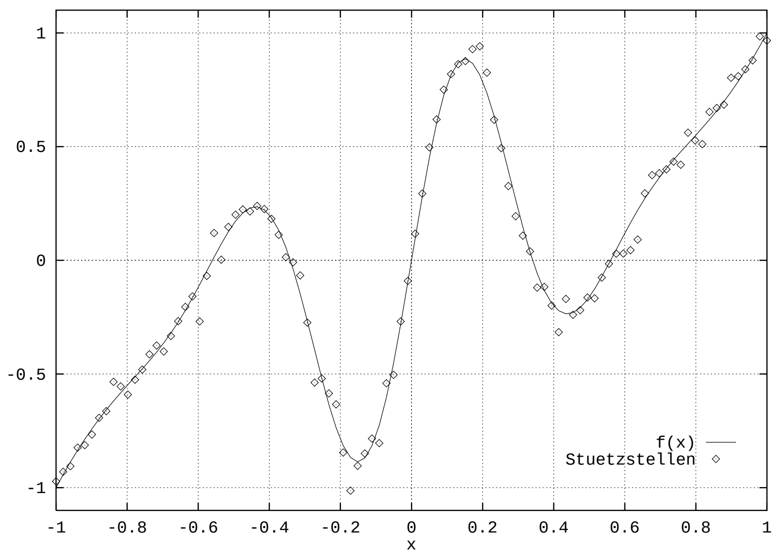

Abbildung 5.14: Funktion $f(x)$ und fehlerbehaftete Stützstellen

Die Funktion $f(x)$ sowie die ausgewählten, verrauschten Stützpunkte sind in der Abbildung 5.14 gemeinsam dargestellt.

Für die Durchführung der Approximation wurde drei sehr einfache Netze definiert. Die Eingabe- und Ausgabeschicht bestanden jeweils aus einem einzigen Neuron. Für das Ausgabeneuron wurde die Aktivierungsfunktion `Act_IdentityPlusBias` gewählt, da es sich bei der Aufgabe nicht um eine Klassifikation sondern um die Durchführung einer möglichst guten Approximation handelt. Die Größe der mittleren Schicht war für die drei Netze jeweils unterschiedlich. Die Aktivierungsfunktion ihrer Neuronen wurde im Verlauf der folgenden Tests variiert.

5.3.1 Initialisierung des Netzes

Zunächst wurden drei Netze betrachtet, deren Neuronen der mittleren Ebene wiederum durch die Funktion `Act_RBF_Gaussian` aktiviert wurden. Die Netze unterscheiden sich in der Größe der mittleren Schicht: Ein Netz wurde mit zehn, ein weiteres mit fünf und das dritte mit vier mittleren Neuronen definiert. Die Initialisierung geschah durch das Verfahren *RBF_Weights*. Da das Ausgabeneuron jeweils über eine lineare Funktion aktiviert wurde, mußten bei der Initialisierung die Werte 0.0 und 1.0 für die *Skalierung* angegeben werden. Für die *Streubreite* wurde der Wert 0.0 gewählt, da die ausgewählten Zentren möglichst gleichmäßig über den gesamten Definitionsbereich der zu approximierenden Funktion verteilt werden sollten. Der Parameter für die *Glattheit* der Funktion wurde zu Anfang ebenfalls auf 0.0 gesetzt. Es verblieb demnach noch die Wahl eines günstigen Bias für die Initialisierung.

In der Abbildung 5.15 ist die Abhängigkeit des Initialisierungsfehlers der drei Netze gegenüber dem gewählten Bias aufgetragen. Für das Netz mit zehn Neuronen in der mittleren Ebene ergibt sich das beste Ergebnis bei einem Bias von 11.0. Bei den beiden kleineren Netzen mit fünf bzw. vier mittleren Neuronen ist der Fehler nahezu unabhängig vom Bias. Das nicht sehr ausgeprägte Minimum ist hier für einen Bias von 14.0 zu finden.

Die weiteren Eigenschaften des Initialisierungsverfahrens wurden nur für das größte der drei Netze untersucht, da der Initialisierungsfehler für die beiden kleineren Netze zu hoch war.

Für das größte der drei Netze wurde nun der Einfluß des Parameters für die Glattheit untersucht. Da es sich um die Approximation einer Funktion einer einzigen Veränderlichen handelt, kann der Einfluß des Parameters sehr gut anschaulich dargestellt werden.

Die Graphen der Abbildung 5.16 geben direkt die Funktionen wieder, die durch das initialisierte Netz repräsentiert wurden. Es ist deutlich zu sehen, daß die Funktion um so flacher wird, je größer der Parameter für die Glattheit gewählt wird. Da in diesem Fall für die Approximation nur zehn Gaußkurven überlagert wurden (Zahl der mittleren Neuronen), ist der Einfluß des Parameters relativ global und weitreichend.

Verwendet man dagegen ein Netz mit annähernd 100 Neuronen in der mittleren Schicht und eine Glattheit von 0.0, so würde die Approximation ziemlich exakt durch die vorgegebenen Stützstellen verlaufen. Hierdurch würden sich alle negativen Einflüsse der fehlerbehafteten (verrauschten) Stützpunkte bemerkbar machen. Für den Fall dieses sehr großen Netzes würde ein positiver Wert für die Glattheit den Rauschanteil praktisch herausfiltern, wodurch das Netz eine Korrektur der Stützstellen realisieren würde.

Für das Netz mit zehn Neuronen in der mittleren Schicht ergibt sich das beste Initiali-

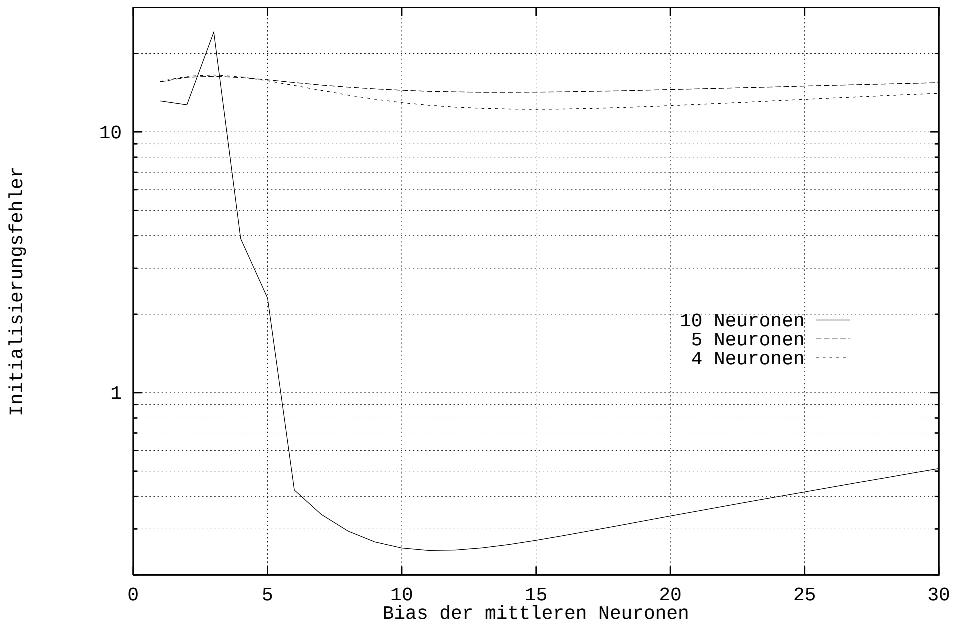

Abbildung 5.15: Fehler der Initialisierung in Abhängigkeit des Bias

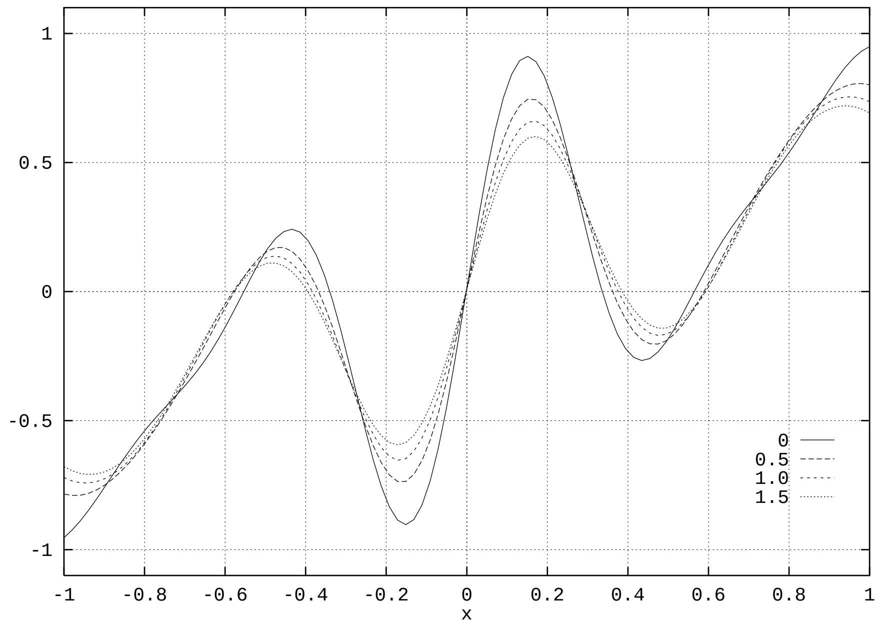

Abbildung 5.16: Funktion des großen Netzes in Abhängigkeit der Glattheit

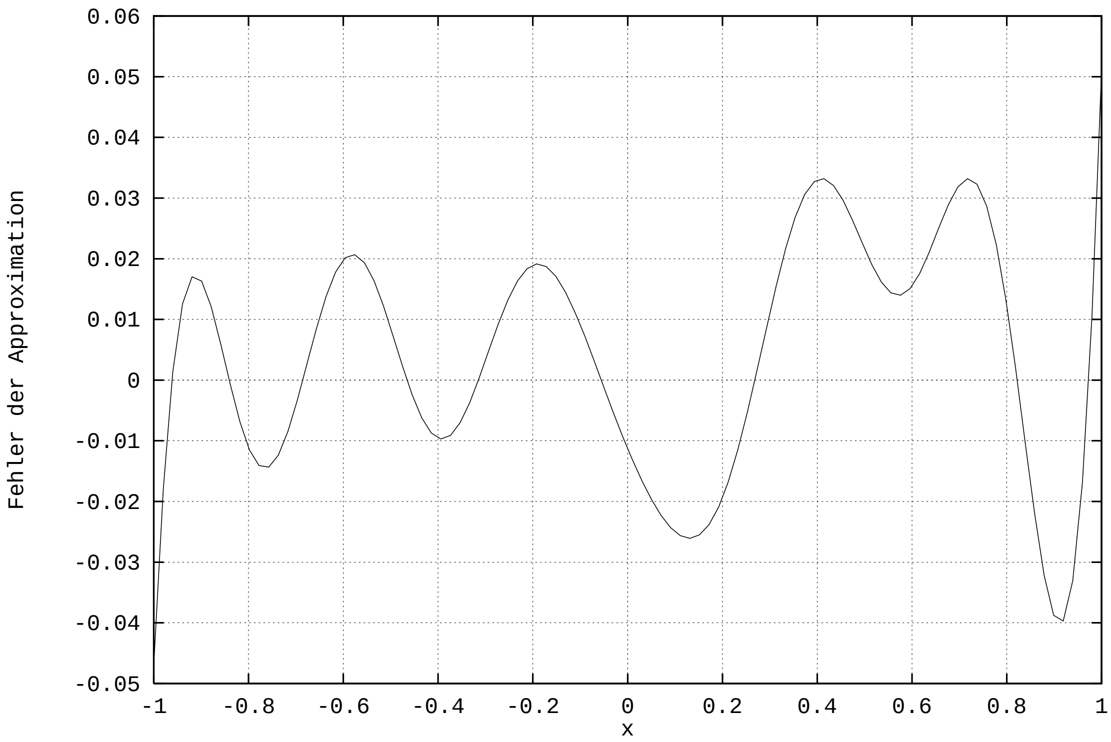

Abbildung 5.17: Fehler der Approximation

sierungsergebnis bei einer Glattheit von 0.0. Vergleicht man nun die Funktion, die das Netz realisiert, mit der Funktion $f(x)$, die den fehlerbehafteten Stützstellen zugrunde lag, so stimmen beide Funktionen erstaunlich gut überein. Die Differenz dieser beiden Funktionen ist so gering, daß sich in Abbildung 5.16 $f(x)$ mit der Funktion des Netzes nahezu überdecken würde. Deshalb wurde in der Abbildung 5.17 die Differenz dieser beiden Funktionen getrennt dargestellt.

Dieser sehr geringe Fehler ist durch das Initialisierungsverfahren zu begründen. Das Verfahren minimiert die Summe der Fehlerquadrate, wodurch der annähernd normalverteilte Rauschanteil optimal ausgeglichen wird.

In einer nächsten Untersuchung wurden in dem größten Netz die Aktivierungsfuntionen der mittleren Ebene durch die Funktionen `Act_RBF_MultiQuadratic` bzw. `Act_RBF_ThinPlateSpline` ersetzt (siehe Abbildungen 4.3 und 4.4, Seite 21 ff.).

Mit der Funktion `Act_RBF_ThinPlateSpline` ließen sich, ebenso wie mit den Gaußkurven von oben, sehr gute Ergebnisse erzielen. Der bestmögliche Initialisierungsfehler lag bei 0.51 (vergleiche Abbildung 5.15).

Die Funktion `Act_RBF_MultiQuadratic` führte dagegen nicht zu guten Ergebnissen. Der bestmögliche Initialisierungsfehler lag hier etwa bei 15.4. Auch durch nachfolgendes Training des Netzes konnte dieser Wert nicht wesentlich verbessert werden. Der Grund hierfür ist der feste Öffnungswinkel der Funktion (siehe Abbildungen 4.3), der durch die Wahl des

Bias nicht verändert werden kann. Hieraus ergibt sich die Schlußfolgerung, daß der Parameter p (der Bias) grundsätzlich in einer anderen Weise in die multiquadratische Funktion integriert werden sollte. Dies müßte derart geschehen, daß der Bias den Öffnungswinkel der Funktion variiert.

5.3.2 Training des initialisierten Netzes

Das größte der drei Netze lieferte nach der Initialisierung bereits so gute Ergebnisse, daß ein anschließendes Training keine Verbesserung erbrachte. Hier zeigt sich sehr deutlich der Vorteil des Verfahrens der direkten Gewichtsberechnung. Voraussetzungen für das gute Ergebnis des Verfahrens waren jedoch die günstig gewählten Zentrumsvektoren und eine dem Problem angepaßte Netzgröße.

Die beiden kleineren Netze konnten dagegen durch nachträgliches Training wesentlich verbessert werden. Hierbei wurden nicht nur die Verbindungsgewichte und die Zentrumsvektoren modifiziert, sondern auch der Bias der mittleren Neuronen. Da durch den Bias die *Breite* der Gaußschen Glockenkurve variiert wird, konnte durch das Training die optimale Approximation durch die Überlagerung von fünf bzw. vier unterschiedlichen Funktionen erreicht werden.

Für das Netz mit fünf Neuronen in der mittleren Ebene war es möglich, den Fehler durch das Training von 14.2 auf 3.68 zu vermindern. Hierbei wurden folgende Lernraten verwendet: *Zentren*: 0.01, *Bias (p)*: 0.025 und *Gewichte*: 0.005. Für den Momentum–Term wurde ein Wert von 0.8 bestimmt.

Der Fehler des trainierten Netzes ist noch immer relativ hoch. Er konnte nicht weiter verbessert werden, da das Lernverfahren bei der Fortsetzung des Trainings instabil wurde. Dies erklärt sich dadurch, daß die Fehlerfläche, durch die erlaubte Modifikation des Bias der mittleren Neuronen, sehr zerklüftet war.

Beim Training des kleinsten Netzes wurde das Lernverfahren noch früher instabil. Dennoch wurde das Training mit den gleichen Parametern wie oben fortgesetzt, da der Gesamtfehler des Netzes immer wieder ein neues Minimum erreichte. Das Training wurde fortgeführt, bis ein Fehler von 2.30 erreicht wurde. Dieser Fehler ist geringer als der Fehler für das Netz mit fünf Neuronen in der mittleren Schicht.

Das trainierte Netz mit vier mittleren Neuronen ist in Abbildung 5.18 wiedergegeben. Ebenso ist in dieser Abbildung das größte der drei Netze zu sehen. Bei beiden Netzen wurde in der Darstellung die gleiche Aktivierung für das Eingabeneuron gewählt. Da die Funktion, die durch das große Netz realisiert wird, sehr gut mit der zu approximierenden Funktion übereinstimmt, kann durch den Vergleich der Ausgaben der beiden gezeigten Netze der Fehler des kleineren Netzes erkannt werden.

In Abbildung 5.19 ist das Endergebnis des Trainings für das Netz mit vier mittleren Neuronen graphisch dargestellt. Es sind die vier Gaußkurven zu erkennen, die durch die mittleren Neuronen repräsentiert werden. Die Funktion $netz(x)$ entsteht durch die Überlagerung dieser Kurven, wobei zusätzlich der Bias des Ausgabeneurons addiert wurde. In der Abbildung sind außerdem die Originalfunktion $f(x)$ sowie Stützstellen wiedergegeben, die durch die Funktion $netz(x)$ möglichst gut angenähert werden sollten. Die Funktionen $h_1(x)$ bis $h_4(x)$ entsprechen der gewichteten Ausgabe der entsprechenden Neuronen aus der Abbildung 5.18.

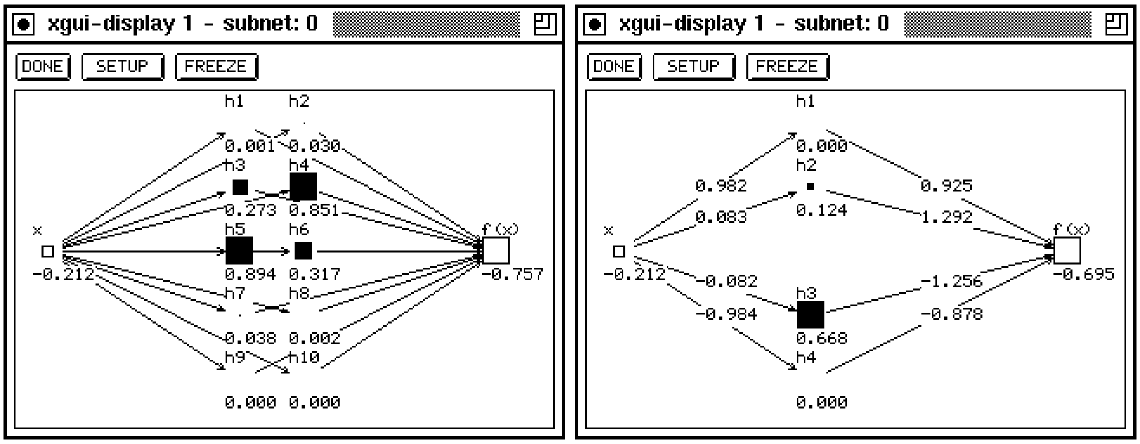

Abbildung 5.18: Initialisiertes Netz mit zehn und trainiertes Netz mit vier Neuronen

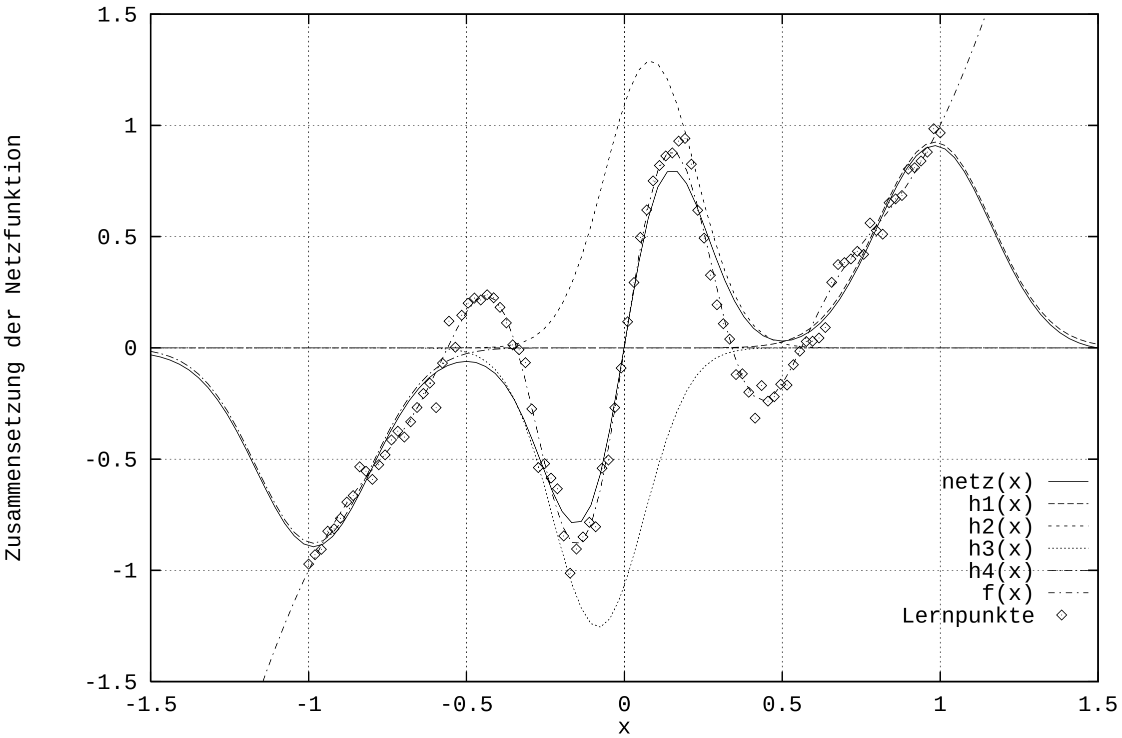

Abbildung 5.19: Überlagerung der Gaußkurven

Kapitel 6

Ausblick

Im letzten Abschnitt werden nun noch einmal alle gewonnenen Erkenntnisse über den Einsatz von Radial Basis Functions bei der Simulation neuronaler Netze zusammengefaßt.

Besonders charakteristisch für das Verfahren ist die vorgegebene Architektur des jeweiligen neuronalen Netzes: In jedem Fall kommen dreistufige feedforward–Netze zum Einsatz, deren Neuronen der mittleren Schicht in einer speziellen Weise aktiviert werden. Die Ebenen des Netzes sind jeweils vollständig miteinander verbunden. Interessant ist die Fragestellung, ob sich Radial Basis Functions auch in Netzen einsetzen lassen, deren Verbindungsstruktur nicht derartig festgelegt ist. Denkbar wäre z.B. die Realisierung von rezeptiven Feldern in der Eingabeschicht. Hierbei aktivieren jeweils kleine Gruppen von Eingabeneuronen ein Neuron der mittleren Schicht. Das Verfahren für die direkte Gewichtsberechnung ist jedoch nicht auf diesen Fall ausgelegt. Ebenso ist es nicht möglich, durch das Berechnungsverfahren Netze mit mehr als drei Schichten von Neuronen zu initialisieren. In diesen beiden Schwächen sind interessante Ansatzpunkte für weitere Entwicklungen zu sehen.

Trotz der Einschränkungen ist die Methode der direkten Gewichtsberechnung als großer Vorteil des Verfahrens anzusehen. Durch sie lassen sich sehr schnell funktionsfähige Netze konstruieren, die die gestellten Anforderungen bereits gut erfüllen.

Die Arbeit zeigte jedoch auch die Probleme auf, die durch das Initialisierungsverfahren bedingt sind. Das Verfahren versagt, sobald die Eigenschaften der Aktivierungsfunktionen für die mittlere Ebene ungünstig gewählt werden. In den vorgestellten Anwendungen machte sich dieses Problem durch die Schwierigkeit bemerkbar, einen günstigen Bias für die Neuronen der mittleren Schicht zu finden. Dieses Problem läßt sich grundsätzlich auf die Auswahl der verwendeten Zentrumsfunktion verallgemeinern. Es kann gesagt werden, daß das Initialisierungsverfahren nur dann befriedigend funktioniert, wenn die Zentrumsfunktion und ihr freier Parameter p (Bias) so gewählt werden, daß die Neuronen der mittleren Schicht nicht einseitig aktiviert werden. Es muß also gewährleistet sein, daß die mittlere Schicht für unterschiedliche Eingabemuster unterschiedlich angeregt wird.

Ein weiteres, interessantes Ergebnis besteht in der Tatsache, daß das Ergebnis der Initialisierung sehr stark von der Wahl der Zentrumsvektoren abhängig ist. Bei der Ziffernerkennung konnte der Initialisierungsfehler um ein Drittel vermindert werden, indem das selbstorganisierende Verfahren von Kohonen für die Bestimmung der Zentrumsvektoren verwendet wurde. Andere Auswahlverfahren könnten wahrscheinlich noch wesentlich bessere Ergebnisse liefern. Auch hierin ist ein Punkt für weitere Forschungen zu sehen.

Das Problem der Ziffernerkennung zeigte, daß das Verfahren der Radial Basis Functions nicht grundsätzlich besser als andere Verfahren ist. Es stellte sich z.B. heraus, daß das Lernverfahren *Backpropagation* bei einem äquivalenten Netz sehr viel schneller zu guten Ergebnissen führt als die Methode der Radial Basis Functions. Als Grund hierfür ist die stark zerklüftete Oberfläche des Lernfehlers zu sehen. Diese zeigt sich z.B. in der Tatsache, daß die Lernraten für das Training mit Radial Basis Functions sehr gering gewählt werden müssen, um ein stabiles Lernverhalten zu garantieren. Die eigentliche Ursache für die ungünstigen Eigenschaften der Fehlerfläche besteht wahrscheinlich darin, daß sich die beiden Ebenen der Verbindungsgewichte qualitativ sehr stark voneinander unterscheiden. Dies trifft auch für die Aktivierungsfunktionen der mittleren und der Ausgabeschicht zu.

Trotz der genannten Schwierigkeiten führt die Verwendung der Radial Basis Functions bei speziellen Anwendungen zu sehr guten Ergebnissen. Hier zeigt das Initialisierungsverfahren alle seine Vorteile. Durch einen einzigen Schritt kann ein Netz erzeugt werden, dessen Fehler so gering ist, daß es durch anschließendes Training nicht wesentlich verbessert werden kann.

Auch wenn Radial Basis Functions nicht bedenkenlos für die Simulation neuronaler Netze eingesetzt werden können, stellen sie doch eine sinnvolle Erweiterung der Modelle für neuronale Netze dar.

Anhang A

Literaturverzeichnis

[BL88] D.S. Brommhead, D. Lowe. Multivariable Functional Interpolation and Adaptive Networks. In *Complex Systems, Vol. 2*, S. 321–355, 1988.

[Det89] D.H. Deterding. Speaker Normalisation for Automatic Speech Recognition. University of Cambridge, 1989.

[HKK90] E.J. Hartman, J.D. Keeler, J.M. Kowalski. Layered Neural Networks with Gaussian Hidden Units as Universal Approximations. In *Neural Computation, Vol. 2*, S. 210–215. Massachusetts Institute of Technology, 1990.

[KSF91] U. Kreßel, J. Schürmann, J. Franke. Neuronale Netze für die Musterklassifikation. Institut für Informationstechnik, Daimler Benz AG, Ulm, 1991.

[PFTV90] William H. Press, Brian P. Flannery, Saul A. Teukolsky, William T. Vetterling. *Numerical recipes in C: the art of scientific computing.* Cambridge University Press, 1990.

[PG89] Thomas Poggio, Frederico Girosi. A Theory of Networks for Approximation and Learning. A.I. Memo No. 1140, Massachusetts Institute of Technology, 1989.

[PG90] Thomas Poggio, Frederico Girosi. Regularization Algorithms for Learning That Are Equivalent to Multilayer Networks. In *Science, Vol. 247*, S. 978–982, 1990.

[Pow85] M.J.D. Powell. Radial Basis Functions for Multivariable Interpolation: A Review. In *Conference on Algorithms for the Approximation of Functions and Data*, S. 143–167, RMCS Shrivenham, UK, 1985. Institute of Mathematics and its Applications (IMA).

[PS91] J. Park, I.W. Sandberg. Universal Approximation Using Radial-Basis-Function Networks. In *Neural Computation, Vol. 3*, S. 246–257. Massachusetts Institute of Technology, 1991.

[RF88] A.J. Robinson, F. Fallside. A Dynamic Connectionist Model for Phoneme Recognition. In *Proceedings of nEuro '88*, June 1988.

[Sto89] Josef Stoer. *Numerische Mathematik 1.* Springer Verlag, 1989.

[Was89] Philip D. Wasserman. *Neural Computing.* Van Nostrand Reinhold, New York, 1989.

[ZMH+92] A. Zell, N. Mache, R. Hübner, M. Schmalzl, T. Sommer, T. Korb. SNNS User Manual, Version 2.0. Report No. 3/92, Universität Stuttgart, IPVR, 1992.