D I P L O M A R B E I T

Eine Variante des Backpropagation-Lernverfahrens für Künstlich Neuronale Netze (KNN)

Conny Dethloff

FB Mathematik
Universität Rostock

25. Mai 1999

Betreuer: Prof.Dr. K.Engel
Dipl.Phys. W.Wustlich

Vorwort

Der Einsatz von Künstlich Neuronalen Netzen ist inzwischen sehr stark verbreitet. Anwendungsgebiete sind beispielsweise die Medizin, die Elektrotechnik und die Bildverarbeitung. Sie steuern Roboter, Fahrzeuge und werden zur Meßdatenanalyse benutzt. Großen Anklang finden die KNN auch in der Muster- und Zeichenerkennung.
Eine spezielle Anwendung erfahren die Netze in dem vollautomatischen Auswertesystem ArgusSelect der Firma Planet GmbH Schwerin, in der ich auch meine Diplomarbeit geschrieben habe. Dieses System wertet Negativfilme aus, die im Verlaufe von Geschwindigkeitskontrollen im Straßenverkehr angefertigt wurden. Mein Lernverfahren, welches ich in Kapitel 3 vorstellen werde, wurde in diesem System getestet.
In der vorliegenden Arbeit beschäftige ich mich im 1. Kapitel mit dem Aufbau und der Arbeitsweise von KNN, bevor ich dann im Anschluß zu der interessantesten Komponente von Neuronalen Netzen komme, zu den Lernverfahren. Ich behandle den wohl bekanntesten Lernalgorithmus für Feed-Forward-Netze, das Backpropagation-Lernverfahren. Wie schon gesagt, folgt im 3. Kapitel eine Variante vom Backpropagation-Lernen, das qcg-Verfahren. In den beiden darauffolgenden Kapiteln 4 und 5 werde ich mich thematisch ein wenig von den KNN entfernen und das qcg-Verfahren als allgemeines Minimierungsverfahren für quadratische Fehlerfunktionen betrachten. Außerdem wird ein anderes bekanntes Optimierungsverfahren für quadratische Minimierungsprobleme, das cg-Verfahren, vorgestellt. Im 6. Kapitel werde ich mich mit Fragen der Implementierung des neuen Verfahrens auseinandersetzen, bevor ich im naechsten und gleichzeitig letzten Kapitel auf Vergleiche des qcg-Verfahrens mit anderen Lernverfahren eingehe.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einführung in KNN

Als erstes möchte ich bemerken, daß ich mich in allen folgenden Bezeichnungen und Konventionen an [zell] orientiert habe.

„Neuronale Netze (NN),..., sind informationsverarbeitende Systeme, die aus einer großen Anzahl einfacher Einheiten (Zellen, Neuronen) bestehen, die sich Information in Form der Aktivierung der Zellen über gerichtete Verbindungen (connections, links) zusenden.“ [1]

Man unterscheidet Biologisch Neuronale Netze (BNN) und Künstlich Neuronale Netze (KNN). Ich gehe aber nur auf die zweiten Netze ein. Zu den BNN findet man etwas in [zell]. Auch zu Eigenschaften der Neuronalen Netze und zu ihrer Geschichte kann man in [zell] nachlesen.

1.1 Aufbau von KNN

Ein KNN besteht aus einer Eingabeschicht (*input layer*), einer Ausgabeschicht (*output layer*) und 0 bis k verdeckte Schichten (*hidden layers*). Die Neuronen der Eingabeschicht leiten die Signale in das Netz und die Ausgabeneuronen geben das vom Netz verarbeitete Signal nach außen. Die verdeckten Neuronen dienen der Informationsverarbeitung. Ein allgemeines Feed-Forward-Netz ist in Abbildung [ffn] 1.1 dargestellt.

[1] Andreas Zell „Simulation neuronaler Netze“ S.23

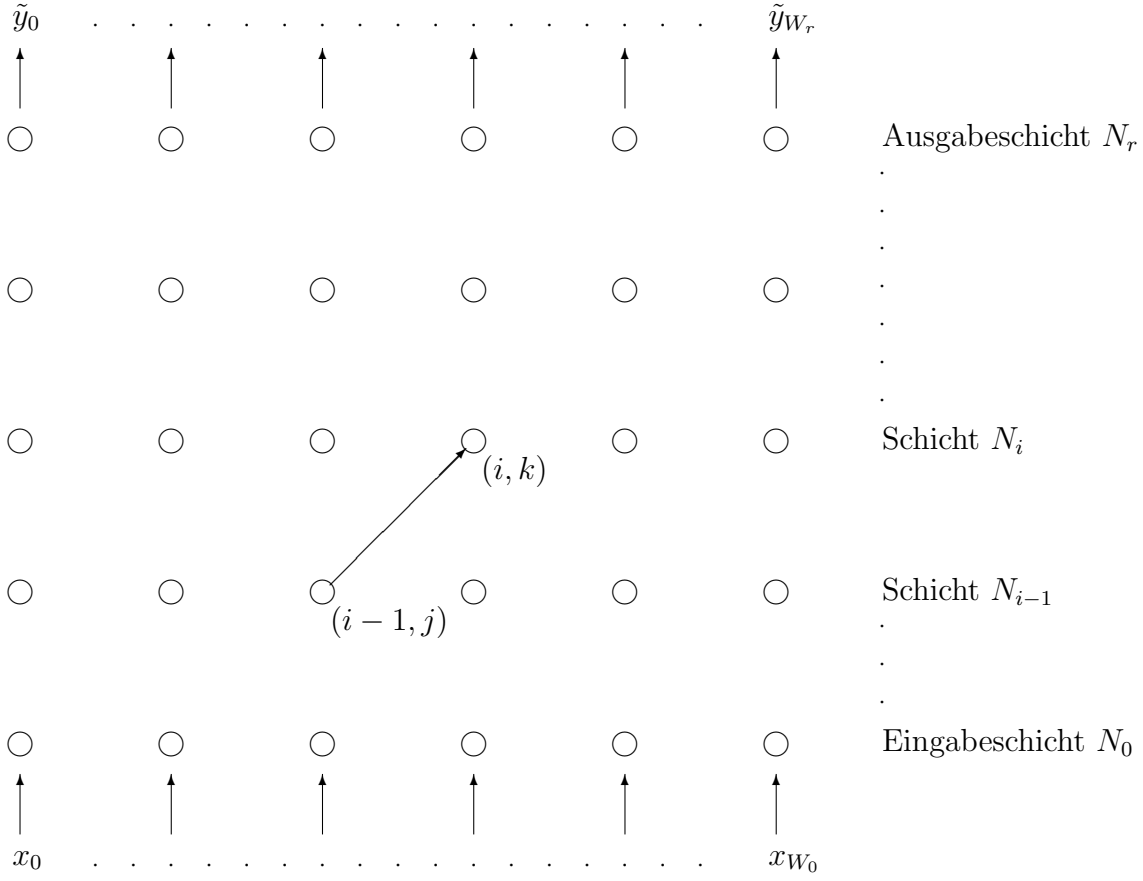

Abbildung 1.1: **allgemeines Feed-Forward-Netz,** *der Übersichtlichkeit halber nur 1 Verbindung eingezeichnet* `ffn`

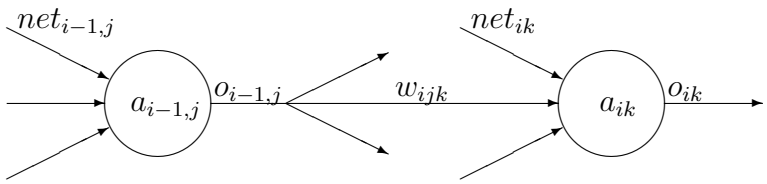

Abbildung 1.2: **Neuronen eines KNN und ihr Zusammenwirken** 2neuron

Komponenenten von KNN: siehe Abbildung 2neuron 1.2

1. *Neuronen.* Diese haben folgende Bestandteile:

   - *Aktivierungszustand* $a_{ij}(t)$
   - *Aktivierungsfunktion* f_{act}.
     Sie gibt an, wie sich ein neuer Aktivierungszustand $a_{ij}(t+1)$ des Neurons (i, j) aus der alten Aktivierung $a_{ij}(t)$ und der *Netzeingabe* $net_{ij}(t)$ ergibt.

     $$a_{ij}(t+1) = f_{act}(a_{ij}(t), net_{ij}(t), \theta_{ij}(t)) \tag{1.1}$$ activation

   - *Ausgabefunktion* f_{out}.
     Die Ausgabe des Neurons (i, j) ergibt sich aus der Aktivierung des Neurons (i, j) wie folgt

     $$o_{ij} = f_{out}(a_{ij}) \tag{1.2}$$

     Meistens ist aber die Ausgabefunktion schon in der Aktivierungsfunktion enthalten. Dann wird die Ausgabefunktion der Form halber als Identität angenommen. Nachfolgend ist bei mir die Aktivierungsfunktion gleich Aktivierungsfunktion und Ausgabefunktion zusammen.
   - *Schwellwert* θ_{ij} von Neuron (i, j).
     Der Schwellwert gibt die Schwelle an, ab der ein Neuron stark aktiv ist. Mathematisch gesehen ist dies die Stelle der größten Steigung monoton wachsender Aktivierungsfunktionen. Im biologischen Sinne ist θ die Reizschwelle ab der ein Neuron feuert.

2. *Netztopologie.*
   Ein KNN kann man als gerichteten, gewichteten Graph ansehen, wobei die Kanten die gewichteten Verbindungen zwischen den Neuronen darstellen. w_{ijk} ist also das Gewicht der Verbindung von Neuron $(i-1, j)$ nach Neuron (i, k).

3. *Propagierungsfunktion.*
   Sie gibt an, wie sich die Netzeingabe eines Neurons aus den Ausgaben der anderen Neuronen und den Verbindungsgewichten berechnet. Die Netzeingabe $net_{ik}(t)$ von Neuron (i, k) berechnet sich wie folgt

$$net_{ik}(t) = \sum_j o_{i-1,j}(t) w_{ijk}(t) \tag{1.3}$$ `netin`

4. *Lernregel.*
   Lernverfahren sind die interessanteste Komponente von KNN.
   Diese sind Algorithmen, nach denen die Netze lernen, für eine vorgegebene Eingabe eine erwünschte Ausgabe zu erzielen. Folgende Möglichkeiten des Lernens bestehen:

   - Entwicklung neuer Verbindungen
   - Löschen existierender Verbindungen
   - Modifikation der Gewichte w_{ijk} von Verbindungen
   - Modifikation des Schwellwertes θ_{ij} von Neuronen
   - Modifikation der Aktivierungs- und Propagierungsfunktion
   - Entwicklung neuer Zellen
   - Löschen von Zellen

1.2 Komponenten von KNN

1.2.1 Aktivierungsfunktion

Ich werde in diesem Abschnitt etwas über sigmoide Aktivierungsfunktionen sagen. Diese Funktionen sind in der Praxis am häufigsten gebräuchlich. Diese Funktionenklasse löst das Problem, daß ein Netzwerk oft sowohl auf Signale kleiner als auch sehr großer Amplitude reagieren muß. Bei kleiner Amplitude muß es dabei sensibler sein. Daher haben sigmoide Aktivierungsfunktionen ihre höchste Sensibilität (größte Steigung) um den Bereich des Arbeitspunktes (Schwellwert) θ. Außerdem sind sie überall stetig und differenzierbar.

2 Vertreter dieser sigmoiden Funktionen sind:

1. **Logistische Aktivierungsfunktion** (Abbildung 1.3)
   Diese Funktionenklasse hat folgende Form:

$$f_{log}(x) = \frac{1}{1+e^{-x}} \tag{1.4}$$

   Ganz besonders wichtig im Zusammenhang mit dem Benutzen der logistischen Aktivierungsfunktion ist die einfache Ableitung der Funktion:

$$\begin{aligned} \frac{d}{dx} f_{log}(x) &= -\left(\frac{1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right)(-e^{-x}) \\ &= \left(\frac{1}{1+e^{-x}}\right)\left(\frac{e^{-x}}{1+e^{-x}}\right) \\ &= \left(\frac{1}{1+e^{-x}}\right)\left(\frac{1+e^{-x}-1}{1+e^{-x}}\right) \\ &= f_{log}(x)(1-f_{log}(x)) \end{aligned} \tag{1.5}$$

   Die Ableitung ist besonders für das Backpropagation - Lernverfahren interessant, worauf ich aber noch später zu sprechen komme.
   Durch Einführung eines großen Parameterwertes λ kann man die sigmoide Funktion $f_{log}(-\lambda x)$ beliebig genau an die binäre Schrittfunktion annähern. Dabei bewirken kleine Werte von λ eine Annäherung an eine lineare Funktion. Dieses Phänomen spielt bei der schlechten Generalisierungsleistung von Resilient Propagation eine große Rolle, worauf ich im Kapitel 4.5 genauer eingehe.

2. **Tangens hyperbolicus** (Abbildung 1.4) ist definiert durch:

$$f_{tanh}(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{1.6}$$

   Sie läßt sich aus f_{log} wie folgt darstellen:

$$\begin{aligned} \tanh(x) &= \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{e^x(1-e^{-2x})}{e^x(1+e^{-2x})} = \frac{1-e^{-2x}}{1+e^{-2x}} \\ &= \frac{2-(1+e^{-2x})}{1+e^{-2x}} \\ &= 2f_{log}(2x) - 1 \end{aligned} \tag{1.7}$$

   Die Ableitung ist ebenso leicht zu berechnen

$$\begin{aligned} \frac{d}{dx} &= \frac{(e^x+e^{-x})(e^x+e^{-x}) - (e^x-e^{-x})(e^x-e^{-x})}{(e^x+e^{-x})^2} \\ &= 1 - \tanh^2(x) \end{aligned} \tag{1.8}$$

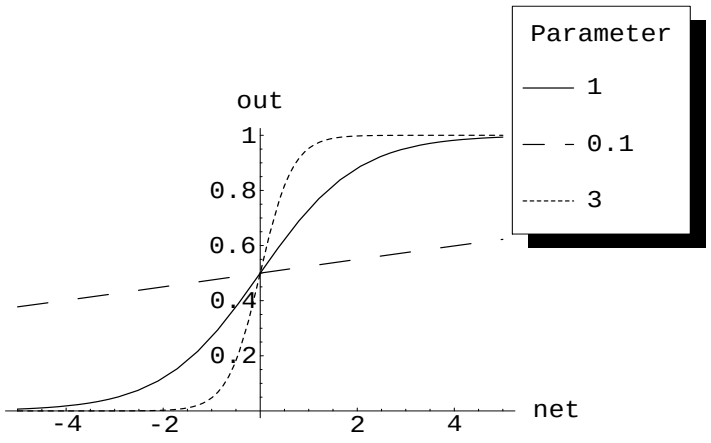

Abbildung 1.3: **Logistische Aktivierungsfunktion $f_{log}(\lambda x)$** `logakt`

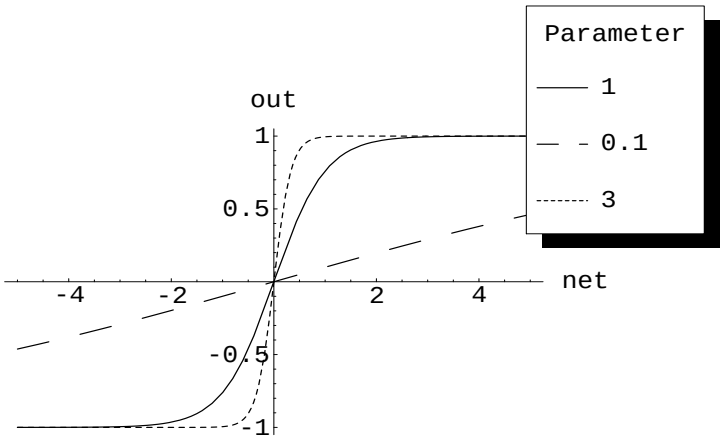

Abbildung 1.4: **Tangens Hyperbolicus $f_{tanh}(\lambda x)$** `tanhyp`

1.2.2 Lernen in KNN

Unter Lernen in KNN versteht man meist die Abänderung der Verbindungsgewichte und Schwellwerte, um eine bessere Übereinstimmung zwischen erwünschter und tatsächlicher Ausgabe zu erhalten.
Man unterscheidet 3 Arten des Lernens: aus [1] entnommen

- *Überwachtes Lernen (supervised learning)* Ein externer Lehrer gibt zu jedem Eingabemuster der Trainingsmenge das korrekte Ausgabemuster an. Das bedeutet, dem Netz liegen vollständige Trainingspattern (Eingabe- und Ausgabemuster) vor. Aufgabe des Lernverfahrens ist es, nach wiederholter Präsentation der Trainingspattern die Angleichung von Ein- und Ausgabe selbständig vorzunehmen. Ganz wichtig ist die Generalisierungsfähigkeit, d.h., das Netz erkennt unbekannte ähnliche Eingabemuster (Testpattern). Diese Arten von Lernverfahren sind üblicherweise die schnellsten, jedoch sind sie am biologisch unplausibelsten. Ein wichtiges Lernverfahren dieser Klasse ist das Backpropagation-Lernverfahren mit seinen Varianten.

- *Bestärkendes Lernen (reinforcement learning)* Der externe Lehrer gibt zu jedem Eingabemuster der Trainingsmenge nur an, ob es richtig oder falsch klassifiziert wurde. Die korrekte Ausgabe jedoch gibt er nicht an, diese muß das Netz selber finden. Diese Art des Lernens ist sehr viel langsamer als die erste Art, jedoch biologisch plausibler, weil man einfache Rückkopplungsmechanismen der Umwelt beobachten kann, wie z.B. Bestrafung bei falschen und Belohnung bei richtigen Entscheidungen.

- *Unüberwachtes Lernen (unsupervised learning)* Der große Unterschied zu den beiden anderen Arten ist, daß wir hier keinen externen Lehrer haben. Das Lernen erfolgt durch Selbstorganisation (*self-organized learning*). Die Trainingspattern bestehen nur aus Eingabemustern. Durch das Lernverfahren versucht das Netz, ähnliche Eingabemuster in ähnliche Kategorien zu packen. Es existieren aber keine erwünschten Angaben, ob das KNN Trainingspattern richtig oder falsch klassifiziert hat, wie noch beim *reinforcement learning*. Diese Lernverfahren sind am biologisch plausibelsten, jedoch nicht auf so viele Probleme anwendbar wie die anderen Verfahren.

Auf zwei unterschiedliche Arten des überwachten Lernens werde ich jetzt noch eingehen. Da gibt es zum einen das online-Training und das offline- oder batch-Training. Bei den online-Verfahren wird nach jeder Präsentation eines Trainingspatterns ein Lernschritt, also eine Änderung der Gewichte

und Schwellwerte, durchgeführt. Bei der offline-Variante wird erst nach einmaliger Zuführung jedes Trainingspatterns ein Lernschritt vollführt. Daraus ergeben sich verschiedene Probleme und Vorteile. Die offline-Algorithmen sind unempfindlich gegenüber ungemischten Trainingsdatensätzen, ganz im Gegensatz zu den online-Verfahren, machen aber Sorgen bei großen Trainingspatternsätzen.
Man muß aber bedenken, daß die online-Variante optimierungstechnisch kein einwandfreies Verfahren ist, da man nicht über die gesamte Fehlerfläche minimiert, sondern nur über einen Teilfehler.

1.2.3 Netztopologie

Ich möchte mich in meiner Diplomarbeit nur auf Feed-Forward-Netze (Netze ohne Rückkopplung) beschränken. Bei diesen Netzen existiert kein Pfad, der von einem Neuron direkt oder über zwischengeschaltete Neuronen wieder zu dem Neuron zurückführt. Mathematisch gesehen ist die Topologie des Netzes also ein azyklischer Graph.
Man unterscheidet zusätzlich noch 2 Arten von Feed-Forward-Netzen:

- Ebenenweise verbundene Feed-Forward-Netze:
  Die Netze bestehen aus mehreren Schichten. Es gibt nur Verbindungen von einer Schicht zur nächsten. siehe Abbildung 1.5
- Allgemeine Feed-Forward-Netze (mit shortcut connections):
  Diese Netze bestehen auch aus Verbindungen, die Ebenen überspringen, d.h. die direkt von einem Neuron in Schicht k zu einem Neuron in Schicht $k+i$ mit $i > 1$ verlaufen. siehe Abbildung 1.6

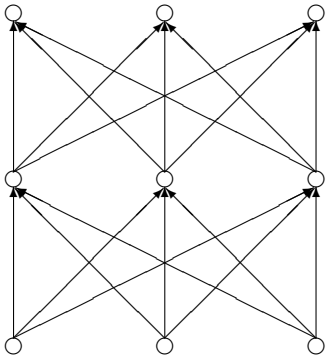

Abbildung 1.5: **Feed-Forward-Netz ohne shortcut connections** nnosc

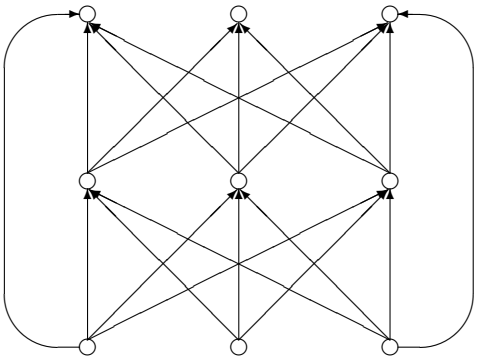

Abbildung 1.6: **Feed-Forward-Netz mit 2 shortcut connections** nnmsc

Die andere Klasse von Netzen, die man unterscheidet sind die *Netze mit Rückkopplungen*. Ich möchte aber, wie vorher schon erwähnt, nicht auf diese Klasse eingehen, weil sie für Bachpropagation und seine Varianten uninteressant ist. Näheres zu diesen KNN findet man in [1].

Kapitel 2

Das Backpropagation-Lernverfahren

In diesem Kapitel wende ich mich nun dem populärsten Lernalgorithmus für vorwärtsgerichtete Netze zu.
Backpropagation ist ein Gradientenabstiegsverfahren (Methode des steilsten Abstiegs). Mit diesem Verfahren versucht man möglichst schnell das globale Minimum der Fehlerfläche, aufgetragen über die Gewichte und Schwellwerte des KNN, zu finden.

2.1 Herleitung von Backpropagation

Ich werde am Anfang erst einmal einige Größen definieren, die ich im weiteren Verlauf dieser Arbeit benutze: (bezogen auf Abbildung 1.1)

- $W_i = |N_i|$ mit $0 \leq i \leq r$ und $r \geq 1$
- $\underline{x} = (x_0, \ldots, x_{W_0})^T$
- $\underline{y} = (y_0, \ldots, y_{W_r})^T$
- $\underline{\tilde{y}} = (\tilde{y}_0, \ldots, \tilde{y}_{W_r})^T$
- w_{ijk} ist das Gewicht von $((i-1,j),(i,k))$
- θ_{ij} ist der Schwellwert von Neuron (i,j)
- $\left(\underline{x}^{(1)}, \underline{y}^{(1)}\right), \ldots, \left(\underline{x}^{(t)}, \underline{y}^{(t)}\right)$ sind die Lernpattern

Ziel des Lernverfahrens: $\underline{\tilde{y}}^{(1)} \approx \underline{y}^{(1)}, \ldots, \underline{\tilde{y}}^{(t)} \approx \underline{y}^{(t)}$

Der erste Schritt des Backpropagation-Verfahrens ist der *Feed-Forward-Schritt*, in dem die Eingabewerte bis zur Ausgabeschicht vorwärtspropagiert werden. Für die Herleitung benutze ich als Aktivierungsfunktion die logistische Funktion (siehe Formel (2.3)).

$$o_{0j}^{(s)} = net_{0j}^{(s)} = x_j^{(s)} \quad \text{mit} \quad \begin{array}{l} s = 1, \ldots, t \\ j = 0, \ldots, W_0 \end{array} \tag{2.1}$$

Wir nehmen jetzt an, wir hätten $o_{i-1,j}^{(s)}$ und $net_{i-1,j}^{(s)}$ schon berechnet für $1 \leq i < r$. Daraus ergibt sich folgendes

$$net_{ik}^{(s)} = \sum_{j=1}^{W_{i-1}} o_{i-1,j}^{(s)} w_{ijk} \tag{2.2}$$ net

$$o_{ik}^{(s)} = \frac{1}{1 + e^{-(net_{ik}^{(s)} - \theta_{ik})}} \tag{2.3}$$ log

Damit erhalten wir als Ausgabe in der Ausgabeschicht:

$$\tilde{y}_j^{(s)} = o_{rj}^{(s)} \quad \text{mit} \quad j = 0, \ldots, W_r \tag{2.4}$$ ytilde

An diesen *Feed-Forward-Schritt* schließt sich nun der *Backpropagation-Schritt* des Fehlers an, in dem der Fehler von der Ausgabeschicht in die Eingabeschicht zurückpropagiert wird.
Als Fehlerfunktion F haben wir folgende:

$$F(\underline{w}, \underline{\theta}) = 1/2 \sum_{s=1}^{t} \sum_{j=1}^{W_r} \left(\tilde{y}_j^{(s)} - y_j^{(s)} \right)^2 \to \text{Min} \tag{2.5}$$ fehler

Den Faktor $1/2$ nehmen wir deshalb hinzu, weil er sich später zusammen mit dem Faktor 2 vom Differenzieren herauskürzt.
Bei diesem zweiten Schritt stoßen wir nun auf ein Problem, welches nur für Feed-Forward-Netze mit größer oder gleich einer verdeckten Schicht auftritt. Man benötigt aber nur solche KNN in der Praxis, da die anderen nur sehr wenige Probleme lösen können. Deshalb ist dieses Problem im Umgang mit Backpropagation immer präsent.

Problem 2.1. Wie kann man $\nabla_{\underline{w},\underline{\theta}} F(\underline{w},\underline{\theta})$ geschickt berechnen ?

Idee. Setze

$$\begin{aligned} \alpha_{ij}^{(s)} &:= \frac{\partial F}{\partial o_{ij}^{(s)}} \\ \beta_{ij}^{(s)} &:= \frac{\partial F}{\partial net_{ij}^{(s)}} \end{aligned} \tag{2.6}$$

alpha_beta

Wir fangen also im zweiten Schritt bei der Ausgabeschicht N_r an.
Wegen den Formeln (2.4),(2.5) und (2.6) wissen wir folgendes:

$$\alpha_{rj}^{(s)} = \frac{\partial F}{\partial o_{rj}^{(s)}} = \tilde{y}_j^{(s)} - y_j^{(s)} = o_{rj}^{(s)} - y_j^{(s)} \tag{2.7}$$

Dann haben wir:

$$F(o_{ij}^{(s)}) = F(net_{i+1,1}^{(s)}(o_{ij}^{(s)}), \ldots, net_{i+1,W_{i+1}}^{(s)}(o_{ij}^{(s)})) \tag{2.8}$$

fvono

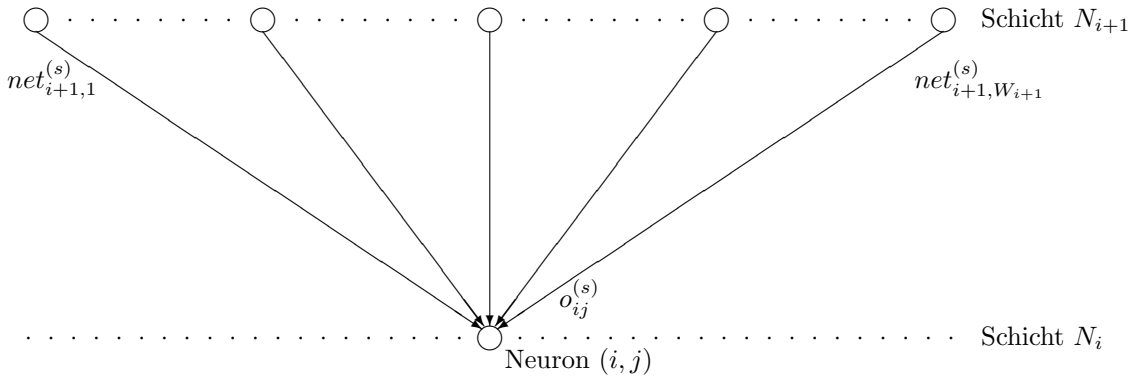

Abbildung 2.1: **Erläuterung von Formel** (2.8)

ffs

Dann berechnen wir mit Hilfe der Kettenregel $\alpha_{ij}^{(s)}$ gemäß folgender Formel:

$$\alpha_{ij}^{(s)} = \frac{\partial F}{\partial o_{ij}^{(s)}} = \frac{\partial F}{\partial net_{i+1,1}^{(s)}} * \frac{\partial net_{i+1,1}^{(s)}}{\partial o_{ij}^{(s)}} + \cdots + \frac{\partial F}{\partial net_{i+1,W_{i+1}}^{(s)}} * \frac{\partial net_{i+1,W_{i+1}}^{(s)}}{\partial o_{ij}^{(s)}}$$

und erhalten mit Hilfe von Formel (2.6) und (2.2):

$$\alpha_{ij}^{(s)} = \sum_{k=1}^{W_{i+1}} \beta_{i+1,k}^{(s)} w_{i+1,j,k} \tag{2.9}$$

alpha_herl alpha_beta net

Wir haben also:

$$\boxed{\alpha_{ij}^{(s)} = \sum_{k=1}^{W_{i+1}} \beta_{i+1,k}^{(s)} w_{i+1,j,k}} \tag{2.10}$$

alpha

Nun müssen wir noch $\beta_{ij}^{(s)}$ berechnen, was in Analogie zur Berechnung von $\alpha_{ij}^{(s)}$ gemacht wird.

$$F(net_{ij}^{(s)}) = F(o_{ij}^{(s)}(net_{ij}^{(s)})) \tag{2.11}$$

fvonnet

Wiederum mit Hilfe der Kettenregel erhalten wir:

$$\beta_{ij}^{(s)} = \frac{\partial F}{\partial net_{ij}^{(s)}} = \frac{\partial F}{\partial o_{ij}^{(s)}} * \frac{\partial o_{ij}^{(s)}}{\partial net_{ij}^{(s)}}$$

Nun kommt die relativ leichte Differenzierbarkeit der logistischen Aktivierungsfunktion zum Tragen, worauf ich schon in Kapitel 1.2.1 eingegangen bin, denn:

$$\frac{\partial o_{ij}^{(s)}}{\partial net_{ij}^{(s)}} = o_{ij}^{(s)}(1 - o_{ij}^{(s)}) \tag{2.12}$$

Wir bekommen letztendlich mit Formel (2.6):

alpha_beta

$$\boxed{\beta_{ij}^{(s)} = \alpha_{ij}^{(s)} * o_{ij}^{(s)}(1 - o_{ij}^{(s)})} \tag{2.13}$$

beta

Das bedeutet, im zweiten Schritt des Backpropagation-Verfahrens muß man schichtenweise von oben nach unten erst $\alpha_{ij}^{(s)}$ und dann $\beta_{ij}^{(s)}$ berechnen.

Nach diesem Schritt kommt das *Aufsummieren der Gradienten*. Hier summiert man die Gradienten des Fehlers F bezüglich der Gewichte und der Schwellwerte auf.
Erst einmal betrachten wir den Gradienten bezüglich der Gewichte.

$$F(w_{ijk}) = F(net_{ik}^{(s)}(w_{ijk}) : s \in S)$$

wobei $S \in \mathcal{P}(1, \ldots, t)$

Mit Hilfe der Kettenregel berechnen wir wieder

$$\frac{\partial F}{\partial w_{ijk}} = \sum_{s \in S} \frac{\partial F}{\partial net_{ik}^{(s)}} * \frac{\partial net_{ik}^{(s)}}{\partial w_{ijk}} \tag{2.14}$$

Nach Formel (2.2) und (2.6) ergibt sich

$$= \sum_{s \in S} \beta_{ik}^{(s)} o_{i-1,j}^{(s)} \tag{2.15}$$

Damit erhalten wir also:

$$\boxed{\frac{\partial F}{\partial w_{ijk}} = \sum_{s \in S} \beta_{ik}^{(s)} o_{i-1,j}^{(s)}} \tag{2.16}$$

Auf ähnliche Art und Weise berechnen wir den Gradienten bezüglich der Schwellwerte.

$$F(\theta_{ij}) = F(o_{ij}^{(s)}(\theta_{ij}) : s \in S)$$

Wegen der Kettenregel folgt

$$\frac{\partial F}{\partial \theta_{ij}} = \sum_{s \in S} \frac{\partial F}{\partial o_{ij}^{(s)}} * \frac{\partial o_{ij}^{(s)}}{\partial \theta_{ij}} \tag{2.17}$$

Außerdem wissen wir folgendes

$$\begin{aligned}
\frac{\partial o_{ij}^{(s)}}{\partial \theta_{ij}} &= \frac{\partial}{\partial \theta_{ij}} \frac{1}{1 + e^{-(net_{ij}^{(s)} - \theta_{ij})}} \\
&= -\frac{e^{-(net_{ij}^{(s)} - \theta_{ij})}}{\left(1 + e^{-(net_{ij}^{(s)} - \theta_{ij})}\right)^2} \\
&= -\frac{1}{1 + e^{-(net_{ij}^{(s)} - \theta_{ij})}} \left(1 - \frac{1}{1 + e^{-(net_{ij}^{(s)} - \theta_{ij})}}\right) \\
&= -o_{ij}^{(s)}(1 - o_{ij}^{(s)}) \qquad (2.18) \quad \boxed{\texttt{fnachtheta}}
\end{aligned}$$

Formel (2.6) und (2.18) ergibt

$$\frac{\partial F}{\partial \theta_{ij}} = -\sum_{s \in S} \alpha_{ij}^{(s)} * o_{ij}^{(s)}(1 - o_{ij}^{(s)}) \tag{2.19}$$

Aus Formel (2.13) folgt schließlich:

$$\boxed{\frac{\partial F}{\partial \theta_{ij}} = -\sum_{s \in S} \beta_{ij}^{(s)}} \qquad (2.20) \quad \boxed{\texttt{sum\_fnachtheta}}$$

Als letztes schließt sich dann der eigentliche *Lernschritt* an. Jetzt werden die Gewichte und Schwellwerte modifiziert nach der Arbeitsweise eines Gradientenabstiegsverfahrens, man bewegt sich auf der Fehlerfläche in Richtung des negativen Gradienten (steilster Abstieg ist die Suchrichtung z):

$$\begin{aligned} z_{w_{ijk}}^{neu} &= -\varepsilon \frac{\partial F}{\partial w_{ijk}^{alt}} \\ z_{\theta_{ij}}^{neu} &= -\varepsilon \frac{\partial F}{\partial \theta_{ij}^{alt}} \end{aligned} \tag{2.21}$$

such

Hierbei wird $\varepsilon \in \mathbb{R}$ als Lernrate oder Schrittweite bezeichnet. Diese wird heuristisch festgelegt.

Der Lernschritt lautet:

$$\begin{aligned} w_{ijk}^{neu} &= w_{ijk}^{alt} + z_{w_{ijk}}^{neu} \\ \theta_{ij}^{neu} &= \theta_{ij}^{alt} + z_{\theta_{ij}}^{neu} \end{aligned} \tag{2.22}$$

lernschritt

Zwei Verfahren des Backpropagation-Verfahrens unterscheidet man noch, zum einen ist es online-Backpropagation, zum anderen offline- (oder batch-) Backpropagation. Vergleiche dazu auch Kapitel 1.2.2. Da bei offline erst ein Lernschritt durchgeführt wird, wenn alle Trainingspattern dem Netz einmal zugeführt wurden, ist in den Formeln (2.16) und (2.20) die Menge S gleich die Menge aller Trainingspattern, also $S = (1, \ldots, t)$. Bei der online-Variante hingegen ist S einelementig, das bedeutet, wir führen keine Aufsummation der Gradienten durch, sondern machen gleich den Lernschritt.

sum_fnachw sum_fnachtheta

Der Backpropagation-Lernalgorithmus besitzt eine Menge von Problemen, auf die ich im Detail im nächsten Abschnitt eingehen möchte. Trotz allem ist dieses Lernverfahren sehr bekannt und populär.

2.2 Probleme von Backpropagation

In diesem Abschnitt werde ich einige grundlegende Probleme vom Backpropagation-Lernverfahren näher diskutieren.
Wie für alle Gradientenabstiegsverfahren üblich, resultieren die Probleme daraus, daß Backpropagation ein lokales Verfahren ist, welches keine Information über die gesamte Fehlerfläche besitzt, sondern nur aus der unmittelbaren Umgebung des Gradienten.
Die folgenden Probleme habe ich aus [1] entnommen.

1. **Symmetry breaking**

   Die ist ein Problem der Initialisierung der Anfangsgewichte und Anfangsschwellwerte für vollständig ebenenweise verbundene Feed-Forward-Netze.
   Wenn nämlich alle Anfangswerte des Netzes gleich groß gesetzt werden, kann Backpropagation in den der Ausgabeschicht vorgelagerten Schichten keine unterschiedlichen Gewichte und Schwellwerte mehr erzeugen. Dadurch schränkt man natürlich das Lernen erheblich ein.
   Die Behebung dieses Problems ist aber sehr einfach. Man nimmt als Initialisierung der Anfangswerte einfach kleine zufällige Werte.

2. **Lokale Minima der Fehlerfläche**

   Typisch für alle Gradientenverfahren ist, daß sie sich in lokalen Minima verfangen.
   Speziell auf KNN bezogen muß man noch sagen, daß je höher die Dimension des Netzes ist, das heißt je mehr Verbindungen das Netz besitzt, um so zerklüfteter ist auch die Fehlerfläche F. Das bedeutet wiederum, die Wahrscheinlichkeit, daß man in einem lokalen Minimum landet, wird größer. Durch Regulierung der Schrittweiten (siehe Formel (2.21)) während des Lernens kann man diesem Problem ein wenig entgegenwirken, jedoch ganz auslöschen kann man es nicht.

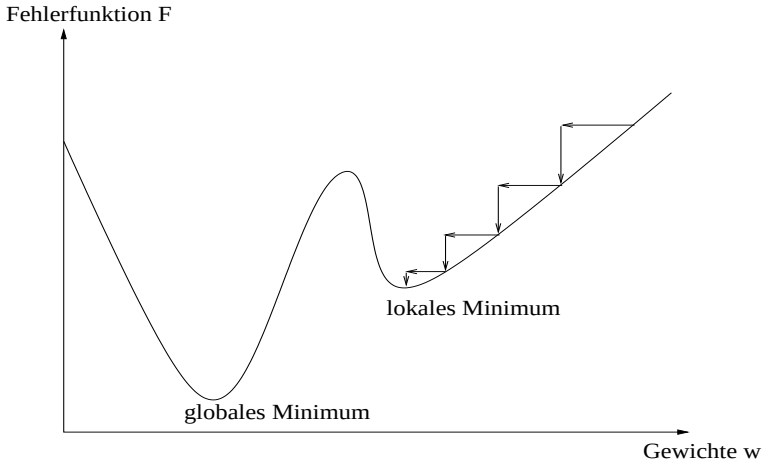

Abbildung 2.2: **lokale Minima der Fehlerfläche** lokmin

3. Flache Plateaus

Die Größe der Gewichts- und Schwellwertänderung hängt bei Backpropagation, wie wir ja schon wissen, vom Betrag des Gradienten ab. Das Verfahren braucht also zur Überwindung dieses Plateaus sehr viele Lernschritte, da die Größe des Gradienten hier relativ klein ist. Eine Erhöhung der Schrittweite wäre hier angebracht.
Ganz schlecht sieht es bei ebenen Plateaus aus. Da der Gradient hier der Nullvektor ist, führt Backpropagation keinen Lernschritt mehr aus. Zusätzlich kann das Verfahren nicht erkennen, ob man auf solch einem Plateau stagniert oder ob man sich in einem lokalen oder globalen Minimum befindet.
Eine Variante des Backpropagation zur Behebung dieses Problems ist der *Momentum-Term*, welcher sich an den *cg-Verfahren für nichtquadratische Probleme* anlehnt. Diese Modifizierung von Backpropagation wird in [Zell] vorgestellt.

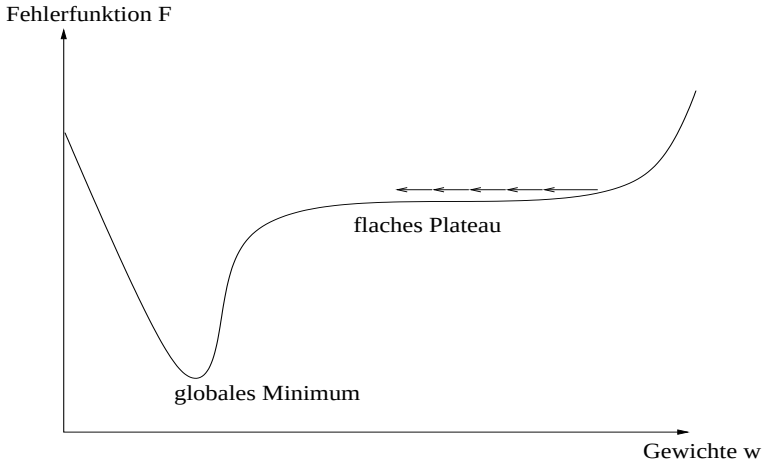

Abbildung 2.3: **flaches Plateau der Fehlerfläche** `flapla`

4. Oszillation in steilen Schluchten

Dieses Problem tritt auf, wenn der Gradient am Rande einer Schlucht so groß ist, daß man durch die Änderung der Gewichte und Schwellwerte auf die gegenüberliegende Seite der Schlucht gelangt. Ist die Schlucht auf dieser Seite genauso steil tritt dieses Phänomen erneut auf, und wir erhalten eine Oszillation.
Auch dieses Problem läßt sich mit dem *Momentum-Term* abschwächen.

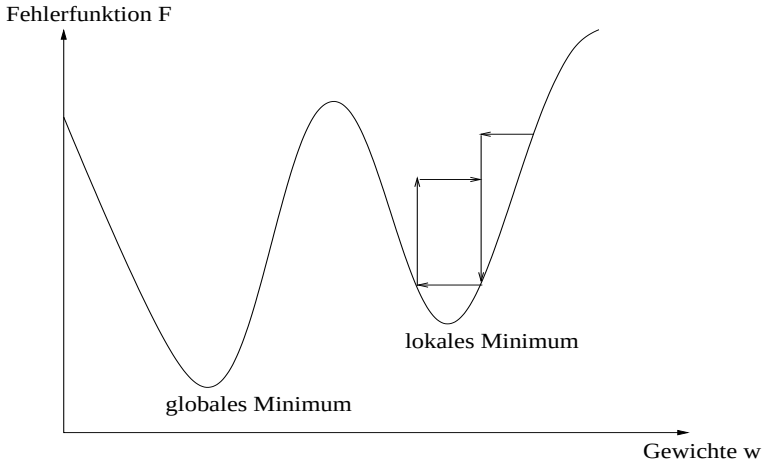

Abbildung 2.4: **Oszillation in steilen Schluchten** `oszill`

5. Verlassen guter Minima

Dieses Problem kann bei sehr engen Tälern von F auftreten, nämlich dann, wenn der Betrag des Gradienten so groß ist, daß man durch den Lernschritt das lokale oder gar globale Minimum verläßt.
Unglücklich ist, daß man durch Einsatz des *Momentum-Terms* oder Erhöhung der Schrittweiten, die Wahrscheinlichkeit des Auftretens dieses Problems entscheidend erhöht. Wodurch man sieht, daß der Versuch, ein Problem des Backpropagation-Lernverfahrens zu beheben, andere Probleme nach sich ziehen kann.

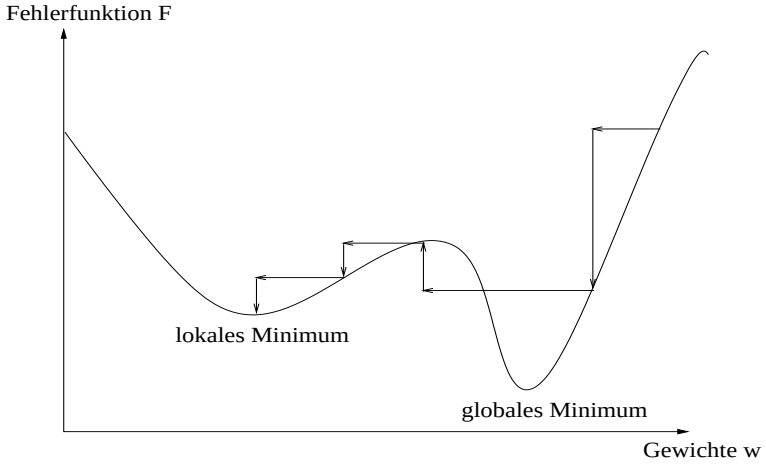

Abbildung 2.5: **Verlassen guter Minima der Fehlerfläche** `vergumi`

6. Wahl der Schrittweite

Die Wahl der Schrittweite ε ist sehr bedeutend für das Konvergenzverhalten von Backpropagation.
Ist sie zu groß besteht die Gefahr, daß das Lernverfahren enge Täler nicht findet oder sogar aus ihnen wieder herausspringt oder ins Oszillieren kommt, wie schon vorgestellt.
Wählt man die Schrittweite jedoch zu klein, wird dadurch der Zeitaufwand unakzeptabel groß, weil man die Anzahl der Lernschritte erhöht.
Die Schrittweitenwahl ist abhängig vom gestellten Problem, welches das KNN lösen soll, von den Trainingspattern und von der Größe und der Topologie des Netzes.
Bei praktischen Anwendungen liegt die Schrittweite zwischen 0.1 und 0.9.

7. Wahl des Dynamikbereichs

Bezogen auf die logistische Aktivierungsfunktion macht man hier folgendes: Man ändert ihren Dynamikbereich von $[0, 1]$ auf $[-1, 1]$.
Dieses beruht auf der Feststellung, daß die Gewichtsänderung Δw_{ijk} und die Schwellwertänderung $\Delta\theta_{ij}$ abhängig ist von der Ausgabe o_{ij} des Neurons (i, j). Im Falle von $o_{ij} = 0$ erfolgt sogar keine Gewichts- und Schwellwertänderung.
Eine Möglichkeit wäre die Änderung des Eingabebereichs zu $[-1/2, 1/2]$ und das Hinzufügen eines Termes, der den Wertebereich der logistischen Aktivierungsfunktion auf $[-1/2, 1/2]$ zurückschraubt:

$$o_{ij} = \frac{1}{1 + e^{-net_{ij}}} - \frac{1}{2} \tag{2.23}$$

Auf Grund dieser Probleme hat man im Laufe der Zeit einige Modifikationen und Varianten von Backpropagation entwickelt. Diese werden ausführlich in [Zell] vorgestellt.
Eine andere Variante stelle ich im anschließenden Kapitel vor.

Kapitel 3

Das qcg-Verfahren

In diesem Kapitel werde ich eine Modifikation des Backpropagation-Lernverfahrens herleiten und seine Arbeitsweise beschreiben. Dieses Verfahren lehnt sich an das cg-Verfahren für nichtquadratische Probleme an, wobei man die Fehlerfläche F durch eine quadratische Funktion approximiert, daher auch der Name *qcg-Verfahren*.

Dieses Verfahren wurde als Lernverfahren im System *ArgusSelect* der Firma Planet GmbH Schwerin getestet und gleichzeitig mit den bisher dort eingesetzten Lernverfahren verglichen. Auf die Ergebnisse des Vergleichs gehe ich im 7. Kapitel näher ein.

3.1 Herleitung des qcg-Verfahrens

Ich möchte am Anfang noch einmal an den Lernschritt beim Standard-Backpropagation erinnern:

$$\begin{aligned} w_{ijk}^{neu} &= w_{ijk}^{alt} + z_{w_{ijk}}^{neu} \\ \theta_{ij}^{neu} &= \theta_{ij}^{alt} + z_{\theta_{ij}}^{neu} \end{aligned} \tag{3.1}$$

Idee. Wir merken uns zusätzlich zu der Abstiegsrichtung des Standard-Backpropagation-Verfahrens $\binom{z_w^{neu}}{z_\theta^{neu}}$ die alte Suchrichtung $\binom{z_w^{alt}}{z_\theta^{alt}}$

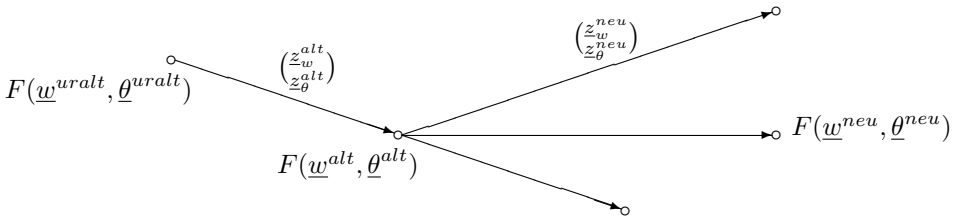

Abbildung 3.1: **Linearkombination der Suchrichtungen** linko

wobei hier folgendes gilt

$$\begin{pmatrix} \underline{z}_w^{alt} \\ \underline{z}_\theta^{alt} \end{pmatrix} = \text{ Kombination der Suchrichtungen aus dem vorherigen Lernschritt}$$

$$\begin{pmatrix} \underline{z}_w^{neu} \\ \underline{z}_\theta^{neu} \end{pmatrix} = -\nabla_{\underline{w},\underline{\theta}} F(\underline{w}^{alt}, \underline{\theta}^{alt}) = -\begin{pmatrix} \nabla_{\underline{w}} F(\underline{w}^{alt}, \underline{\theta}^{alt}) \\ \nabla_{\underline{\theta}} F(\underline{w}^{alt}, \underline{\theta}^{alt}) \end{pmatrix} \tag{3.2}$$

Dann führen wir eine Linearkombination der Suchrichtungen durch und erhalten somit folgenden Lernschritt:

$$\boxed{\begin{pmatrix} \underline{w}^{neu} \\ \underline{\theta}^{neu} \end{pmatrix} = \begin{pmatrix} \underline{w}^{alt} \\ \underline{\theta}^{alt} \end{pmatrix} + \mu^* \begin{pmatrix} \underline{z}_w^{neu} \\ \underline{z}_\theta^{neu} \end{pmatrix} + \lambda^* \begin{pmatrix} \underline{z}_w^{alt} \\ \underline{z}_\theta^{alt} \end{pmatrix}} \tag{3.3}$$ lern

Die Aufgabe besteht darin, die Koeffizienten μ^* und λ^* zu bestimmen.
Wir interpolieren hierfür die Fehlerfunktion F durch eine quadratische Funktion in λ und μ wie folgt, wobei ψ hier der Ansatz ist:

$$\varphi(\lambda, \mu) = F(\underline{w}^{neu}, \underline{\theta}^{neu}) = F\left(\begin{pmatrix} \underline{w}^{alt} \\ \underline{\theta}^{alt} \end{pmatrix} + \mu \begin{pmatrix} \underline{z}_w^{neu} \\ \underline{z}_\theta^{neu} \end{pmatrix} + \lambda \begin{pmatrix} \underline{z}_w^{alt} \\ \underline{z}_\theta^{alt} \end{pmatrix}\right) \tag{3.4}$$

$$\psi(\lambda, \mu) = a\lambda^2 + b\lambda\mu + c\mu^2 + d\lambda + e\mu + f \tag{3.5}$$

Ein sinnvolles Ziel ist es, an ausgewählten Stellen folgendes zu fordern:

$$\begin{aligned} \varphi(\lambda,\mu) &= \psi(\lambda,\mu) \\ \text{Ableitung von } \varphi(\lambda,\mu) &= \text{Ableitung von } \psi(\lambda,\mu) \end{aligned}$$

Dazu müssen wir die Koeffizienten $a,\ldots,f$ bestimmen.
Dafür legen wir uns Bedingungen fest, so daß wir relativ leicht diese Berechnungen durchführen und gleichzeitig so viel wie möglich schon berechnete Werte benutzen können.
Außerdem sieht man, daß wir 6 Bedingungen benötigen, um die Koeffizienten eindeutig bestimmen zu können.

1. Bedingung:

$$\frac{\partial\psi}{\partial\lambda}\Big|_{\substack{\lambda=0\\ \mu=0}} = \frac{\partial\varphi}{\partial\lambda}\Big|_{\substack{\lambda=0\\ \mu=0}} \tag{3.6}$$

$$\begin{aligned} \frac{\partial\psi}{\partial\lambda}\Big|_{\substack{\lambda=0\\ \mu=0}} &= 2a\lambda + b\mu + d\Big|_{\substack{\lambda=0\\ \mu=0}} \\ &= d \end{aligned} \tag{3.7}$$

$$\begin{aligned} \frac{\partial\varphi}{\partial\lambda}\Big|_{\substack{\lambda=0\\ \mu=0}} &= (\underline{z}_w^{alt}, \underline{z}_\theta^{alt})^T \nabla_{\underline{w},\underline{\theta}} F\left(\begin{pmatrix}\underline{w}^{alt}\\ \underline{\theta}^{alt}\end{pmatrix} + \mu\begin{pmatrix}\underline{z}_w^{neu}\\ \underline{z}_\theta^{neu}\end{pmatrix} + \lambda\begin{pmatrix}\underline{z}_w^{alt}\\ \underline{z}_\theta^{alt}\end{pmatrix}\right)\Big|_{\substack{\lambda=0\\ \mu=0}} \\ &= -\left(\underline{z}_w^{alt}, \underline{z}_\theta^{alt}\right)^T \begin{pmatrix}\underline{z}_w^{neu}\\ \underline{z}_\theta^{neu}\end{pmatrix} \end{aligned} \tag{3.8}$$

$$\boxed{d = -\left(\underline{z}_w^{alt}, \underline{z}_\theta^{alt}\right)^T \begin{pmatrix}\underline{z}_w^{neu}\\ \underline{z}_\theta^{neu}\end{pmatrix}} \tag{3.9}$$ d

2. Bedingung:

$$\frac{\partial\psi}{\partial\mu}\Big|_{\substack{\lambda=0\\ \mu=0}} = \frac{\partial\varphi}{\partial\mu}\Big|_{\substack{\lambda=0\\ \mu=0}} \tag{3.10}$$

$$\begin{aligned} \frac{\partial\psi}{\partial\mu}\Big|_{\substack{\lambda=0\\ \mu=0}} &= 2c\mu + b\lambda + e\Big|_{\substack{\lambda=0\\ \mu=0}} \\ &= e \end{aligned} \tag{3.11}$$

$$\begin{aligned}\frac{\partial\varphi}{\partial\mu}\Big|_{\substack{\lambda=0\\ \mu=0}} &= (\underline{z}_w^{neu}, \underline{z}_\theta^{neu})^T \nabla_{\underline{w},\underline{\theta}} F\left(\begin{pmatrix}\underline{w}^{alt}\\ \underline{\theta}^{alt}\end{pmatrix} + \mu\begin{pmatrix}\underline{z}_w^{neu}\\ \underline{z}_\theta^{neu}\end{pmatrix} + \lambda\begin{pmatrix}\underline{z}_w^{alt}\\ \underline{z}_\theta^{alt}\end{pmatrix}\right)\Big|_{\substack{\lambda=0\\ \mu=0}} \\ &= -(\underline{z}_w^{neu}, \underline{z}_\theta^{neu})^T \begin{pmatrix}\underline{z}_w^{neu}\\ \underline{z}_\theta^{neu}\end{pmatrix} \qquad (3.12)\end{aligned}$$

$$\boxed{e = -(\underline{z}_w^{neu}, \underline{z}_\theta^{neu})^T \begin{pmatrix}\underline{z}_w^{neu}\\ \underline{z}_\theta^{neu}\end{pmatrix}} \qquad (3.13) \quad \boxed{\texttt{e}}$$

3. Bedingung:

$$\varphi(\lambda,\mu)\Big|_{\substack{\lambda=0\\ \mu=0}} = \psi(\lambda,\mu)\Big|_{\substack{\lambda=0\\ \mu=0}} \qquad (3.14)$$

$$\varphi(\lambda,\mu)\Big|_{\substack{\lambda=0\\ \mu=0}} = \varphi(0,0) = F\left(\underline{w}^{alt}, \underline{\theta}^{alt}\right) \qquad (3.15)$$

$$\psi(\lambda,\mu)\Big|_{\substack{\lambda=0\\ \mu=0}} = \psi(0,0) = f \qquad (3.16)$$

$$\boxed{f = F\left(\underline{w}^{alt}, \underline{\theta}^{alt}\right)} \qquad (3.17) \quad \boxed{\texttt{f}}$$

4. Bedingung:

$$\varphi(\lambda,\mu)\Big|_{\substack{\lambda=-1\\ \mu=0}} = \psi(\lambda,\mu)\Big|_{\substack{\lambda=-1\\ \mu=0}} \qquad (3.18)$$

$$\begin{aligned}\varphi(\lambda,\mu)\Big|_{\substack{\lambda=-1\\ \mu=0}} &= \varphi(-1,0) = F\left(\begin{pmatrix}\underline{w}^{alt}\\ \underline{\theta}^{alt}\end{pmatrix} - \begin{pmatrix}\underline{z}_w^{alt}\\ \underline{z}_\theta^{alt}\end{pmatrix}\right) \\ &= F\left(\underline{w}^{uralt}, \underline{\theta}^{uralt}\right) \qquad (3.19)\end{aligned}$$

$$\psi(\lambda,\mu)\Big|_{\substack{\lambda=-1\\ \mu=0}} = \psi(-1,0) = a - d + f \qquad (3.20)$$

$$\boxed{a = F\left(\underline{w}^{uralt}, \underline{\theta}^{uralt}\right) + d - f} \qquad (3.21) \quad \boxed{\texttt{a}}$$

5. Bedingung:

$$\frac{\partial\psi}{\partial\mu}\Big|_{\substack{\lambda=-1\\ \mu=0}} = \frac{\partial\varphi}{\partial\mu}\Big|_{\substack{\lambda=-1\\ \mu=0}} \qquad (3.22)$$

$$\begin{aligned}\frac{\partial\psi}{\partial\mu}\Big|_{\substack{\lambda=-1\\ \mu=0}} &= 2c\mu + b\lambda + e\Big|_{\substack{\lambda=-1\\ \mu=0}} \\ &= -b + e \end{aligned} \tag{3.23}$$

$$\begin{aligned}\frac{\partial\varphi}{\partial\mu}\Big|_{\substack{\lambda=-1\\ \mu=0}} &= (\underline{z}_w^{neu}, \underline{z}_\theta^{neu})^T \nabla_{\underline{w},\underline{\theta}} F \left(\begin{pmatrix} \underline{w}^{alt} \\ \underline{\theta}^{alt} \end{pmatrix} + \mu \begin{pmatrix} \underline{z}_w^{neu} \\ \underline{z}_\theta^{neu} \end{pmatrix} + \lambda \begin{pmatrix} \underline{z}_w^{alt} \\ \underline{z}_\theta^{alt} \end{pmatrix} \right) \Big|_{\substack{\lambda=-1\\ \mu=0}} \\ &= (\underline{z}_w^{neu}, \underline{z}_\theta^{neu})^T \nabla_{\underline{w},\underline{\theta}} F \left(\begin{pmatrix} \underline{w}^{alt} \\ \underline{\theta}^{alt} \end{pmatrix} - \begin{pmatrix} \underline{z}_w^{alt} \\ \underline{z}_\theta^{alt} \end{pmatrix} \right) \\ &= (\underline{z}_w^{neu}, \underline{z}_\theta^{neu})^T \nabla_{\underline{w},\underline{\theta}} F \left(\underline{w}^{uralt}, \underline{\theta}^{uralt} \right) \end{aligned} \tag{3.24}$$

$$\boxed{b = e - (\underline{z}_w^{neu}, \underline{z}_\theta^{neu})^T \nabla_{\underline{w},\underline{\theta}} F \left(\underline{w}^{uralt}, \underline{\theta}^{uralt} \right)} \tag{3.25}$$ b

6. Bedingung:

   Als letztes muß noch Koeffizient c berechnet werden. Hier ergibt sich ein kleines Problem.
   c steht vor μ^2 in Formel (3.5), und μ ist gleichzeitig auch der Koeffizient vor dem neuen negativen Gradienten (Suchrichtung bei Standard-Backpropagation) $(\underline{z}_w^{neu}, \underline{z}_\theta^{neu})^T$, den man gerade erst berechnet hat.
   Daraus folgt, man kennt in Richtung dieser Suchrichtung noch keine Werte auf der Fehlerlandschaft F, die man zur Berechnung der Koeffizienten verwenden kann.
   Daher muß man μ in Formel (3.5) oder in den partiellen Ableitungen von ψ immer 0 setzen. Dadurch fällt der Koeffizient c bei den Berechnungen gleich heraus und kann nicht so elegant berechnet werden, wie die anderen 5 Koeffizienten.
   Man muß sich also einen Hilfswert bestimmen, um c berechnen zu können:

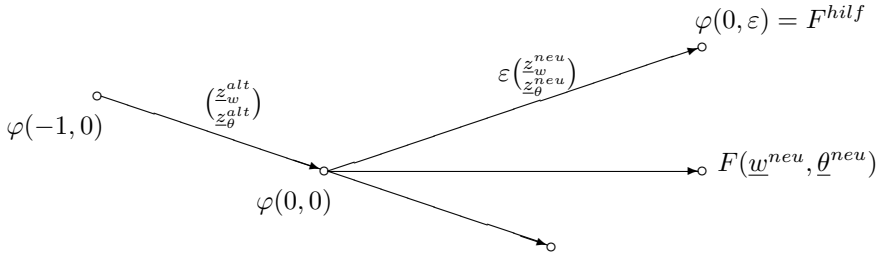

Abbildung 3.2: **Berechnung von F^{hilf}** fhilf

Wir befinden uns am Punkt $\varphi(0,0)$ und berechnen von hier aus in Richtung $\varepsilon\begin{pmatrix}\underline{z}_w^{neu}\\ \underline{z}_\theta^{neu}\end{pmatrix}$, mit ε als Schrittweite (vergleiche Formel (2.21)), den neuen Wert F^{hilf} (siehe Abb. 3.2).
Folgendes gilt:

$$F^{hilf} = F\left(\underline{w}^{hilf}, \underline{\theta}^{hilf}\right) = F\left(\begin{pmatrix}\underline{w}^{alt}\\ \underline{\theta}^{alt}\end{pmatrix} + \varepsilon\begin{pmatrix}\underline{z}_w^{neu}\\ \underline{z}_\theta^{neu}\end{pmatrix}\right) \tag{3.26}$$

Dann verfahren wir in Analogie zur Berechnung der anderen 5 Koeffizienten.

$$\varphi(\lambda,\mu)\Big|_{\substack{\lambda=0\\ \mu=\varepsilon}} = \psi(\lambda,\mu)\Big|_{\substack{\lambda=0\\ \mu=\varepsilon}} \tag{3.27}$$

$$\varphi(\lambda,\mu)\Big|_{\substack{\lambda=0\\ \mu=\varepsilon}} = \varphi(0,\varepsilon) = F\left(\begin{pmatrix}\underline{w}^{alt}\\ \underline{\theta}^{alt}\end{pmatrix} + \varepsilon\begin{pmatrix}\underline{z}_w^{neu}\\ \underline{z}_\theta^{neu}\end{pmatrix}\right) \tag{3.28}$$

$$\psi(\lambda,\mu)\Big|_{\substack{\lambda=0\\ \mu=\varepsilon}} = \psi(0,\varepsilon) = \varepsilon^2 c + \varepsilon e + f \tag{3.29}$$

$$\boxed{c = \frac{1}{\varepsilon^2}\left[F\left(\begin{pmatrix}\underline{w}^{alt}\\ \underline{\theta}^{alt}\end{pmatrix} + \varepsilon\begin{pmatrix}\underline{z}_w^{neu}\\ \underline{z}_\theta^{neu}\end{pmatrix}\right) - \varepsilon e - f\right]} \tag{3.30}$$ c

Wir haben die Koeffizienten $a, \ldots, f$ nun bestimmt und können ψ eindeutig angeben.
Die Funktion ψ hat, da sie ja durch den Ansatz quadratisch ist, genau ein Optimum. Dieses bestimmen wir, aber nur dann wenn dieses Optimum ein Minimum ist (siehe hierzu Kapitel 3.2.3), und berechnen dann die Koeffizienten λ^* und μ^* wie folgt:

$$\nabla\psi = 0 \Longleftrightarrow \begin{cases} \frac{\partial\psi}{\partial\lambda} = 2a\lambda + b\mu + d = 0 \\ \frac{\partial\psi}{\partial\mu} = 2c\mu + b\lambda + e = 0 \end{cases} \tag{3.31}$$

gradpsi

Das bedeutet wir erhalten ein Gleichungssystem mit 2 Gleichungen und 2 Unbekannten λ^* und μ^*. Dieses läßt sich eindeutig lösen mit Hilfe eines Gauß-Schrittes:

$$\Rightarrow \qquad \lambda^* = \frac{be - 2cd}{4ac - b^2} \tag{3.32}$$

lambda

$$\mu^* = \frac{bd - 2ae}{4ac - b^2} \tag{3.33}$$

mu

Zum Schluß führen wir den Lernschritt durch nach Formel (3.3)

lern

3.2 Arbeitsweise des qcg-Verfahrens

3.2.1 Veränderung der Patternpakete

Ich habe mich im Kapitel 2 schon über online- und offline- Varianten des Backpropagation-Verfahrens geäußert. Da aber das qcg-Verfahren ein reines offline-Verfahren ist, wie beispielsweise auch Resilient-Propagation oder Quickprop (siehe [1]), kommt hier nur die offline-Variante zum Einsatz. Ich habe Tests durchgeführt mit qcg-online und qcg-offline. Die Ergebnisse kann man im Kapitel 7 nachlesen.

zell

Man kann aber trotzdem die Pakete der Lernpatternsätze variieren und dann Lernschritte durchführen. Auch diese Ergebnisse kann man im Kapitel 7 finden.
Man muß ganz klar feststellen, daß die optimale Größe der Pakete von vielen Faktoren abhängt. Zum einen wäre da das gestellte Problem, welches das KNN lösen soll, das einen sehr starken Einfluß auf die Wahl der Paketgröße besitzt. Aber auch die Art der Lernpatternsätze spielt eine große Rolle.
Näher werde ich hierauf aber im letzten Kapitel eingehen.

3.2.2 Arbeitsweise pro Lernschritt

Zur Ermittlung des Koeffizienten c, siehe auch Bedingung 6 auf Seite b31, benötigt man die Berechnung von F^{hilf}.
Dafür benötigen wir einen „Hilfsschritt“, das heißt, ein Lernschritt des qcg-Verfahrens ist unterteilt in einen Hilfsschritt und den eigentlichen Lernschritt.

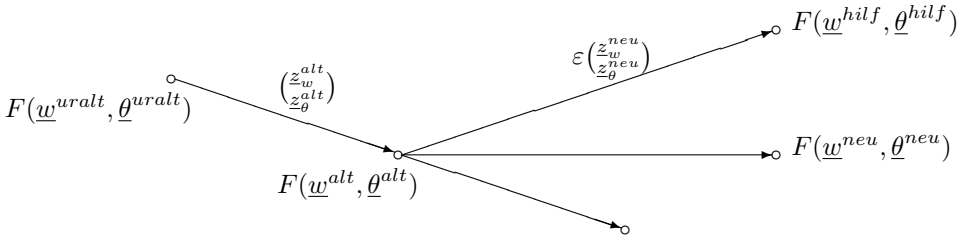

Abbildung 3.3: **Arbeitsweise pro Lernschritt** arprolern

1. Teilschritt (Hilfsschritt):

   - Berechnung von $F(\underline{w}^{alt}, \underline{\theta}^{alt})$ gemäß Formel (fehler2.5) auf Seite fehler16
   - Berechnung von $\binom{\underline{z}_w^{neu}}{\underline{z}_\theta^{neu}}$ gemäß Formel (such2.21) auf Seite such21
   - Berechnung von $\binom{\underline{w}^{hilf}}{\underline{\theta}^{hilf}}$ gemäß Formel (lernschritt2.22) auf Seite lernschritt21
   - Skalarproduktberechnungen, die nötig sind zur Ermittlung der Koeffizienten b (Formel (b3.25) Seite b31), d (Formel (d3.9) Seite d29) und e (Formel (e3.13) Seite e30)

2. Teilschritt (eigentlicher Lernschritt):

   - Berechnung von $F(\underline{w}^{hilf}, \underline{\theta}^{hilf})$ gemäß Formel (fehler2.5) auf Seite fehler16
   - Berechnung von den Koeffizienten $a, \ldots, f$
   - Berechnung von λ^* und μ^*
   - Lernschritt gemäß Formel (lern3.3) auf Seite lern28

3.2.3 Besonderheiten der Arbeitsweise

Zu Beginn des Verfahrens muß man einen Standard-Backpropagation-Schritt als Lernschritt ausführen, weil die Größen $F(\underline{w}^{uralt}, \underline{\theta}^{uralt})$, $\nabla F(\underline{w}^{uralt}, \underline{\theta}^{uralt})$ und $\binom{z_w^{alt}}{z_\theta^{alt}}$ noch unbekannt sind.

Im 2.Teilschritt muß man eine Fallunterscheidung durchführen. In den Formeln (3.32) und (3.33) werden Quotienten berechnet. Wenn in den Berechnungen der Nenner gleich 0 wird, ist der unmittelbar vorher durchgeführte Hilfsschritt der „richtige“ Lernschritt. In diesem Fall wird also ebenfalls ein Standard-Backpropagation-Schritt durchgeführt. Diese Überprüfung wird aber auch schon bei der Kontrolle der positiven Definitheit der Hesse-Matrix durchgeführt (siehe Formel (3.40)).

Wir machen nur dann eine Linearkombination der Suchrichtungen, wenn die quadratische Ansatzfunktion $\psi(\lambda, \mu)$ für den entsprechenden Punkt $\binom{\underline{w}^{alt}}{\underline{\theta}^{alt}}$ ein Minimum besitzt.
Dazu betrachten wir die Hesse-Matrix von $\psi(\lambda, \mu)$ im Punkt $\binom{\underline{w}^{alt}}{\underline{\theta}^{alt}}$, die wir mit $H_\psi(\underline{w}^{alt}, \underline{\theta}^{alt})$ bezeichnen.

$$H_\psi(\underline{w}^{alt}, \underline{\theta}^{alt}) = \begin{pmatrix} 2a & b \\ b & 2c \end{pmatrix} \tag{3.34}$$

Definition 3.1. Eine Matrix $A = A^T \in \mathbb{R}^{n \times n}$ heißt positiv definit (spd), wenn für alle $\underline{h} \in \mathbb{R}^n > 0$ die quadratische Form $\underline{h}^T A \underline{h}$ nur positive Werte annehmen kann.

Satz 3.1. *Die Funktion $f : U \to \mathbb{R}$, mit $U \subseteq \mathbb{R}^n$, sei zweimal stetig differenzierbar und $x^* \in U$ sei lokale Optimumstelle von f. Ist dann die $H_f(x^*)$ positiv definit, so ist x^* lokale Minimalstelle von f.*

Beweis. Ich möchte für den Beweis dieses Satzes auf [9] Seite 134 ff. verweisen. □

Aus dem Satz 3.1 läßt sich also folgern, wenn $H_\psi(\underline{w}^{alt}, \underline{\theta}^{alt})$ positiv definit ist, dann ist das Optimum von $\psi(\lambda, \mu)$ ein lokales Minimum. Da wir $\psi(\lambda, \mu)$ als quadratisch vorausgesetzt haben, ist dies sogar ein globales Minimum.
Nun müssen wir noch ein leicht nachprüfbares Kriterium für positive Definitheit aufstellen.

Satz 3.2. *Eine Matrix $A = A^T \in \mathbb{R}^{n\times n}$ ist genau dann positiv definit, wenn die Determinante jeder führenden Hauptuntermatrix*

$$A_k := \begin{pmatrix} a_{11} & \cdots & a_{1k} \\ \vdots & & \vdots \\ a_{k1} & \cdots & a_{kk} \end{pmatrix} \in \mathbb{R}^{n\times n} \quad , \quad k \in 1, \ldots, n \tag{3.35}$$

von A positiv ist.

Beweis. Für den Beweis verweise ich auf [11] Seite 315 ff. □

Satz 3.3. *Die Matrix $A = A^T \in \mathbb{R}^{n\times n}$ ist positiv definit. Daraus folgt, daß die Hauptdiagonalelemente von A größer als 0 sein müssen.*

Beweis. Nach Voraussetzung ist die Matrix A positiv definit. Nach Definition 3.1 wissen wir daher, daß für jeden Vektor $\underline{h} \in \mathbb{R}^n > 0$ folgendes gilt

$$\underline{h}^T A \underline{h} > 0 \tag{3.36}$$

Wir ersetzen $\underline{h}$ durch die Vektoren $\underline{e}_i$ für $i = 1, \ldots, n$, die an der i-ten Stelle eine 1 und sonst überall eine 0 haben. Daraus folgt

$$0 < \underline{e}_i^T A \underline{e}_i = a_{ii} \quad , \quad i = 1, \ldots, n \tag{3.37}$$

□

Bemerkung. Die Umkehrung des letzten Satzes gilt im allgemeinen nicht. Die Positivität der Hauptdiagonalelemente ist also für die Eigenschaft der positiven Definitheit einer Matrix hinreichend.

Wir müssen also folgendes nachprüfen (vergleiche Formel (3.34))

$$2c > 0 \quad \Rightarrow \quad c > 0 \tag{3.38}$$

$$2a > 0 \quad \Rightarrow \quad a > 0 \tag{3.39}$$

$$4ac - b^2 > 0 \quad \Rightarrow \quad \text{Nenner aus (3.32) und (3.33)} > 0 \tag{3.40}$$

Wenn die Bedingungen (3.39) und (3.40) notwendig und hinreichend und die Bedingung (3.38) hinreichend erfüllt sind, machen wir also eine Linearkombination der Suchrichtungen.
Ansonsten führen wir einen Standard-Backpropagation-Schritt durch.

3.3 Die Approximationsfunktion $\psi(\lambda, \mu)$

Ich werde in diesem Abschnitt untersuchen, wie sich die quadratische Approximationsfunktion $\psi(\lambda, \mu)$ in Abhängigkeit der Koeffizienten a, b, c, d, e und f verhält.
Dazu setzen wir $\psi(\lambda, \mu)$ gleich ζ (vergleiche Formel (3.5))

$$\zeta := \psi(\lambda, \mu) = a\lambda^2 + b\lambda\mu + c\mu^2 + d\lambda + e\mu + f \tag{3.41}$$

In Matrixschreibweise hat diese Formel folgende Gestalt

$$(\lambda, \mu, \zeta)^T \underbrace{\begin{pmatrix} a & \frac{b}{2} & 0 \\ \frac{b}{2} & c & 0 \\ 0 & 0 & 0 \end{pmatrix}}_{=:\Psi} \begin{pmatrix} \lambda \\ \mu \\ \zeta \end{pmatrix} + (d, e, -1) \begin{pmatrix} \lambda \\ \mu \\ \zeta \end{pmatrix} + f = 0 \tag{3.42}$$

In Abhängigkeit der Koeffizienten a, b, c, d, e und f wollen wir nun untersuchen, welche Gestalt $\psi(\lambda, \mu)$ annehmen kann.
Die Kriterien dieser Untersuchung habe ich aus [10] genommen.
Wir sehen erst einmal, daß der Rang von Ψ ($r(\Psi)$) kleiner ist als 3. Deshalb benutzen wir zur Flächenbestimmung die Rangbetrachtung.
Berechnen wir also $r(\Psi)$. Nach Anwendung eines Gauß-Eliminationsschrittes erhält man folgende Matrix

$$\tilde{\Psi} = \begin{pmatrix} a & \frac{b}{2} & 0 \\ 0 & \frac{-b^2}{4a} + c & 0 \\ 0 & 0 & 0 \end{pmatrix} \tag{3.43}$$

1. **Fall:** $r(\Psi) = r(\tilde{\Psi}) = 0$
   Daraus ergibt sich dann $a = b = c = 0$. Das bedeutet für die Approximationsfunktion $\psi(\lambda, \mu)$, daß sie entartet ist.
   Sie stellt eine Ebene dar. Das heißt in diesem Fall werden wir keine Linearkombination der Suchrichtungen durchführen (vergleiche Formeln (3.38) und (3.39))

2. **Fall:** $r(\Psi) = r(\tilde{\Psi}) = 1$
   Hieraus ergibt sich (siehe Formel (3.43)) erst einmal $a \neq 0$ und $b, c \in \mathbb{R}$ mit

$$\frac{-b^2}{4a} + c = 0 \quad \Leftrightarrow \quad 4ac - b^2 = 0 \tag{3.44}$$

In diesem Fall stellt $\psi(\lambda, \mu)$ einen parabolischen Zylinder dar. Auch hier wäre eine Linearkombination der Suchrichtungen sinnlos (vergleiche Formel (3.40)).

3. Fall: $r(\Psi) = r(\tilde{\Psi}) = 2$
Nun werden wir uns mit dem interessantesten Fall befassen. Diesen Fall müssen wir wiederum unterteilen, denn jetzt kann $\psi(\lambda, \mu)$ entweder ein elliptisches oder ein hyperbolisches Paraboloid sein. Dazu müssen wir die Vorzeichen der Eigenwerte der Matrix Ψ untersuchen.
Als erstes werden die Eigenwerte der Matrix Ψ mit der *Sarrusschen Regel* (siehe [10]) bestimmt.

$$\begin{aligned} \left|\Psi - \rho I\right| &= \begin{vmatrix} a-\rho & \frac{b}{2} & 0 \\ \frac{b}{2} & c-\rho & 0 \\ 0 & 0 & -\rho \end{vmatrix} \\ &= \rho\left(\rho^2 - (a+c)\rho + ac - \frac{b^2}{4}\right) = 0 \end{aligned} \qquad (3.45) \quad \boxed{\texttt{eig1}}$$

Aus Formel (3.45) sieht man, daß $\rho_1 = 0$ ist, was aber nicht weiter interessant für uns ist, da dies wegen $r(\Psi) < 3$ zu erwarten war. Betrachten wir also die beiden anderen Eigenwerte.
Aus Formel (3.45) folgt

$$\rho^2 - (a+c)\rho - \frac{b^2}{4} = 0 \qquad (3.46) \quad \boxed{\texttt{eig23}}$$

Aus der quadratischen Lösungsformel ergibt sich

$$\rho_{2,3} = \frac{a+c}{2} \pm \frac{1}{2}\sqrt{(a+c)^2 - 4ac + b^2} \qquad (3.47) \quad \boxed{\texttt{quadloes}}$$

Die Formel (3.47) muß nun diskutiert werden

(a) *beide Eigenwerte positiv*
Daraus folgt aus Formel (3.47)

$$\frac{a+c}{2} > 0 \qquad \text{und} \qquad (3.48) \quad \boxed{\texttt{ac1}}$$

$$\frac{a+c}{2} > \frac{1}{2}\sqrt{(a+c)^2 - 4ac + b^2} \qquad (3.49) \quad \boxed{\texttt{wurz1}}$$

Daraus ergibt sich aus Formel (3.48)

$$a + c > 0 \tag{3.50}$$ folg11

und aus Formel (3.49)

$$4ac - b^2 > 0 \tag{3.51}$$ folg12

Aus den Formeln (3.50) und (3.51) ergeben sich

$$a > 0 \quad \text{und} \quad c > 0 \tag{3.52}$$

In diesem Fall stellt $\psi(\lambda, \mu)$ ein elliptisches Paraboloid mit einem globalen Minimum dar. Das bedeutet, wir machen eine Linearkombination der Suchrichtungen.

(b) *ein positiver und ein negativer Eigenwert*
Diesen Fall muß man unterteilen. Aus Formel (3.47) ergeben sich folgende 2 Unterfälle

i.

$$\frac{a+c}{2} \leq 0 \qquad \text{und} \tag{3.53}$$ ac21

$$\frac{a+c}{2} > -\frac{1}{2}\sqrt{(a+c)^2 - 4ac + b^2} \tag{3.54}$$ wurz21

Daraus ergibt sich aus Formel (3.53)

$$a + c \leq 0 \tag{3.55}$$

und aus Formel (3.54)

$$4ac - b^2 < 0 \tag{3.56}$$

ii.

$$\frac{a+c}{2} \geq 0 \qquad \text{und} \tag{3.57}$$ ac22

$$\frac{a+c}{2} < \frac{1}{2}\sqrt{(a+c)^2 - 4ac + b^2} \tag{3.58}$$ wurz22

Daraus ergibt sich aus Formel (3.57)

$$a + c \geq 0 \tag{3.59}$$

und aus Formel (3.58)

$$4ac - b^2 < 0 \tag{3.60}$$

Hier stellt $\psi(\lambda, \mu)$ ein hyperbolisches Paraboloid dar, das bedeutet, es liegt kein Extrempunkt, sondern ein Sattelpunkt vor. Demzufolge führen wir hier keine Linearkombination der Suchrichtungen durch.

(c) *beide Eigenwerte negativ*
Daraus folgt aus Formel (3.47)

$$\frac{a+c}{2} < 0 \qquad \text{und} \tag{3.61}$$

ac3

$$\frac{a+c}{2} < -\frac{1}{2}\sqrt{(a+c)^2 - 4ac + b^2} \tag{3.62}$$

wurz3

Daraus ergibt sich aus Formel (3.61)

$$a + c < 0 \tag{3.63}$$

folg31

und aus Formel (3.62)

$$4ac - b^2 > 0 \tag{3.64}$$

folg32

Aus den Formeln (3.63) und (3.64) ergeben sich

$$a < 0 \quad \text{und} \quad c < 0 \tag{3.65}$$

In diesem Fall stellt $\psi(\lambda, \mu)$ ein elliptisches Paraboloid mit einem globalen Maximum dar. Hier wäre es nicht sinnvoll eine Linearkombination der Suchrichtungen zu machen (vergleiche hierzu Kapitel 3.2.3).

Zusammenfassung. Man kann abschließend feststellen, daß nur dann eine Linearkombination der Suchrichtungen durchgeführt wird (vergleiche Formel (3.3) auf Seite 28), wenn die quadratische Approximationsfunktion $\psi(\lambda, \mu)$ ein elliptisches Paraboloid mit einem globalen Minimum darstellt.

Kapitel 4

Quadratische Minimierung

In diesem Kapitel der Diplomarbeit wird untersucht, wie sich das qcg-Verfahren bei der Minimierung quadratischer Funktionen verhält, d.h., wir setzen folgendes voraus

$$F(\underline{x}) = \frac{1}{2}\underline{x}^T C \underline{x} + \underline{p}^T \underline{x} \tag{4.1}$$

mit

$$\begin{aligned} C &\in \mathbb{R}^{n\times n} \quad \text{ist eine spd-Matrix} \\ \underline{p}, \underline{x} &\in \mathbb{R}^n \end{aligned} \tag{4.2}$$

spdc

Aus (4.2) folgt, daß die Fehlerfunktion F ein Minimum haben muß (vergleiche Satz 3.1 auf Seite 35).

Problem 4.1. Ist die Hesse-Matrix H_ψ der quadratischen Approximationsfunktion $\psi(\lambda, \mu)$ immer positiv definit ?

Dazu werde ich jetzt die Koeffizienten a, b und c unter Berücksichtigung, daß F quadratisch ist, neu darstellen. Vergleiche hierzu die Formeln (3.21) auf Seite 30, (3.25) auf Seite 31 und (3.30) auf Seite 32.

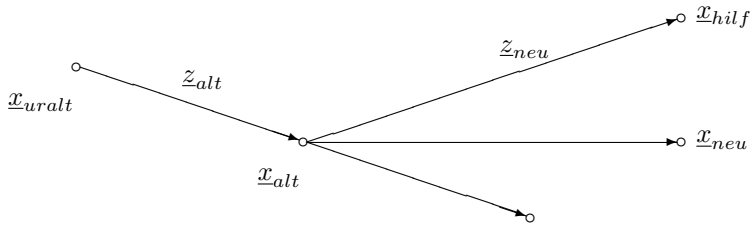

Abbildung 4.1: **Erläuterung der Suchschritte** problem1

- *Berechnung von a:*

$$a = F(\underline{x}_{uralt}) + d - f \tag{4.3}$$

Wegen

$$F(\underline{x}_{uralt}) = \frac{1}{2}\underline{x}_{uralt}^T C \underline{x}_{uralt} + \underline{p}^T \underline{x}_{uralt} \tag{4.4}$$

$$d = -\underline{z}_{alt}^T \underline{z}_{neu} = -\underline{z}_{alt}^T(-C\underline{x}_{alt} - \underline{p}) \tag{4.5}$$

$$f = F(\underline{x}_{alt}) = \frac{1}{2}\underline{x}_{alt}^T C \underline{x}_{alt} + \underline{p}^T \underline{x}_{alt} \tag{4.6}$$

gilt

$$a = \frac{1}{2}\underline{x}_{uralt}^T C \underline{x}_{uralt} + \underline{p}^T \underline{x}_{uralt} - \underline{z}_{alt}^T(-C\underline{x}_{alt} - \underline{p}) - \frac{1}{2}\underline{x}_{alt}^T C \underline{x}_{alt} - \underline{p}^T \underline{x}_{alt} \tag{4.7}$$

Wir setzen $\underline{x}_{uralt} = \underline{x}_{alt} - \underline{z}_{alt}$ (siehe Abbildung 4.1) problem1
und klammern aus. Daraus ergibt sich

$$\boxed{a = \frac{1}{2}\underline{z}_{alt}^T C \underline{z}_{alt}} \tag{4.8}$$ aprob2

Da wir C als symmetrisch positiv definite Matrix vorausgesetzt haben, gilt (vergleiche Definition 3.1 auf Seite 35) spd spd

$$a > 0 \tag{4.9}$$ aprob1

- *Berechnung von b:*

$$b = e - \underline{z}_{neu}^T \nabla F(\underline{x}_{uralt}) \tag{4.10}$$

Wegen

$$e = -\underline{z}_{neu}^T \underline{z}_{neu} \tag{4.11}$$
$$\nabla F(\underline{x}_{uralt}) = C\underline{x}_{uralt} + \underline{p} \tag{4.12}$$

gilt

$$b = -\underline{z}_{neu}^T \underline{z}_{neu} - \underline{z}_{neu}^T (C\underline{x}_{uralt} + \underline{p}) \tag{4.13}$$

Wir setzen $\underline{x}_{uralt} = \underline{x}_{alt} - \underline{z}_{alt}$ und erhalten

$$b = -\underline{z}_{neu}^T \underline{z}_{neu} - \underline{z}_{neu}^T \left(C(\underline{x}_{alt} - \underline{z}_{alt}) + \underline{p}\right) \tag{4.14}$$
$$= -\underline{z}_{neu}^T (\underline{z}_{neu} + C\underline{x}_{alt} + \underline{p} - C\underline{z}_{alt}) \tag{4.15}$$

Wir wissen $\underline{z}_{neu} = -C\underline{x}_{alt} - \underline{p}$. Daraus ergibt sich

$$\boxed{b = \underline{z}_{neu}^T C \underline{z}_{alt}} \tag{4.16}$$ bprob2

- *Berechnung von c:*

$$c = F(\underline{x}_{hilf}) - e - f \tag{4.17}$$

Wegen

$$F(\underline{x}_{hilf}) = \frac{1}{2}\underline{x}_{hilf}^T C \underline{x}_{hilf} + \underline{p}^T \underline{x}_{hilf} \tag{4.18}$$

$$e = -\underline{z}_{neu}^T \underline{z}_{neu} \tag{4.19}$$

$$f = F(\underline{x}_{alt}) = \frac{1}{2}\underline{x}_{alt}^T C \underline{x}_{alt} + \underline{p}^T \underline{x}_{alt} \tag{4.20}$$

gilt

$$c = \frac{1}{2}\underline{x}_{hilf}^T C \underline{x}_{hilf} + \underline{p}^T \underline{x}_{hilf} + \underline{z}_{neu}^T \underline{z}_{neu} - \frac{1}{2}\underline{x}_{alt}^T C \underline{x}_{alt} - \underline{p}^T \underline{x}_{alt} \tag{4.21}$$

Wir setzen $\underline{x}_{hilf} = \underline{x}_{alt} + \underline{z}_{neu}$ und multiplizieren aus

$$c = \underline{z}_{neu}^T C \underline{x}_{alt} + \frac{1}{2}\underline{z}_{neu}^T C \underline{z}_{neu} + \underline{p}^T \underline{z}_{neu} + \underline{z}_{neu}^T \underline{z}_{neu} \tag{4.22}$$

Da $\underline{z}_{neu} = -C\underline{x}_{alt} - \underline{p}$ ist, gilt nach Ausmultiplikation

$$\boxed{c = \frac{1}{2}\underline{z}_{neu}^T C \underline{z}_{neu}} \tag{4.23}$$ cprob2

Nach äquivalenter Schlußfolgerung wie beim Koeffizienten a gilt

$$c > 0 \tag{4.24}$$ cprob1

Die Hesse-Matrix H_ψ hat folgende Gestalt (vergleiche Formel (3.34) auf Seite 35)

$$H_\psi = \begin{pmatrix} 2a & b \\ b & 2c \end{pmatrix} \tag{4.25}$$

Zur Überprüfung der positiven Definitheit von H_ψ müssen wir also folgendes kontrollieren (vergleiche Formeln (3.38), (3.39) und (3.40))

$$c > 0 \tag{4.26}$$ cdef1

$$a > 0 \tag{4.27}$$ adef1

$$4ac - b^2 > 0 \tag{4.28}$$ quotdef1

Wegen den Formeln (4.24) und (4.9) sind (4.26) und (4.27) schon erfüllt.
Bleibt nur noch die Kontrolle von Formel (4.28)

$$\begin{aligned} \left|H_\psi\right| &= 4ac - b^2 \\ &= \left(\underline{z}_{alt}^T C \underline{z}_{alt}\right)\left(\underline{z}_{neu}^T C \underline{z}_{neu}\right) - \left(\underline{z}_{neu}^T C \underline{z}_{alt}\right)^2 \end{aligned} \tag{4.29}$$

Nach Definition 5.1 auf Seite 47 erhalten wir

$$\left|H_\psi\right| = \left(\underline{z}_{alt}, \underline{z}_{alt}\right)_C \left(\underline{z}_{neu}, \underline{z}_{neu}\right)_C - \left(\underline{z}_{alt}, \underline{z}_{neu}\right)_C^2 \tag{4.30}$$

Wegen der *Cauchy-Schwarz-Ungleichung* (siehe [10])

$$\left|\left(\underline{z}_{alt}, \underline{z}_{neu}\right)_C\right| < \sqrt{\left(\underline{z}_{alt}, \underline{z}_{alt}\right)_C}\sqrt{\left(\underline{z}_{neu}, \underline{z}_{neu}\right)_C} \tag{4.31}$$

gilt also auch die Formel (4.28).
Demzufolge ist die Hesse-Matrix H_ψ immer positiv definit, wenn die Fehlerfunktion F quadratisch ist, d.h., wir führen also bei der quadratischen Optimierung mit dem qcg-Verfahren nur Linearkombinationen der Suchrichtungen durch.

Problem 4.2. Wenn $C \in \mathbb{R}^{2\times 2}$ gilt, ist dann $\psi(\lambda, \mu) = \varphi(\lambda, \mu)$, d.h., entspricht die approximierte Funktion der zu approximierenden Funktion ?

Wir müssen erst einmal die Koeffizienten d, e und f neu darstellen

- *Berechnung von d:*

$$d = -\underline{z}_{alt}^T \underline{z}_{neu} \tag{4.32}$$

Da $\underline{z}_{neu} = -C\underline{x}_{alt} - \underline{p}$ ist, gilt

$$\boxed{d = \underline{z}_{alt}^T C \underline{x}_{alt} + \underline{z}_{alt}^T \underline{p}} \tag{4.33}$$ dprob2

- *Berechnung von e:*

$$e = -\underline{z}_{neu}^T \underline{z}_{neu} \tag{4.34}$$

Da $\underline{z}_{neu} = -C\underline{x}_{alt} - \underline{p}$ ist, gilt

$$\boxed{e = \underline{z}_{neu}^T C \underline{x}_{alt} + \underline{z}_{neu}^T \underline{p}} \tag{4.35}$$ eprob2

- *Berechnung von f:*

$$f = F(\underline{x}_{alt}) \tag{4.36}$$

Daraus folgt

$$\boxed{f = \frac{1}{2}\underline{x}_{alt}^T C \underline{x}_{alt} + \underline{p}^T \underline{x}_{alt}} \tag{4.37}$$ fprob2

Wir wollen nun $\varphi(\lambda, \mu)$ in Abhängigkeit der Koeffizienten a, b, c, d, e und f darstellen

$$\begin{aligned} \varphi(\lambda,\mu) &= F(\underline{x}_{alt} + \lambda \underline{z}_{alt} + \mu \underline{z}_{neu}) \\ &= \frac{1}{2}\left(\underline{x}_{alt} + \lambda \underline{z}_{alt} + \mu \underline{z}_{neu}\right)^T C \left(\underline{x}_{alt} + \lambda \underline{z}_{alt} + \mu \underline{z}_{neu}\right) \\ &\quad + \underline{p}^T \left(\underline{x}_{alt} + \lambda \underline{z}_{alt} + \mu \underline{z}_{neu}\right) \end{aligned} \tag{4.38}$$

Nachdem man ausmultipliziert und die Glieder geordnet hat, erhält man folgendes

$$\begin{aligned} \varphi(\lambda,\mu) = &\left\{\frac{1}{2}\underline{z}_{alt}^T C \underline{z}_{alt}\right\}\lambda^2 + \left\{\underline{z}_{alt}^T C \underline{z}_{neu}\right\}\lambda\mu + \left\{\frac{1}{2}\underline{z}_{neu}^T C \underline{z}_{neu}\right\}\mu^2 \\ &+ \left\{\underline{x}_{alt}^T C \underline{z}_{alt} + \underline{p}^T \underline{z}_{alt}\right\}\lambda + \left\{\underline{x}_{alt}^T C \underline{z}_{neu} + \underline{p}^T \underline{z}_{neu}\right\}\mu \\ &+ \frac{1}{2}\underline{x}_{alt}^T C \underline{x}_{alt} + \underline{p}^T \underline{x}_{alt} \end{aligned} \tag{4.39}$$

Wegen den Formeln (4.8), (4.16), (4.23), (4.33), (4.35) und (4.37) aprob2 bprob2 cprob2 dprob2 eprob2 fprob2 ergibt sich

$$\varphi(\lambda,\mu) = a\lambda^2 + b\lambda\mu + c\mu^2 + d\lambda + e\mu + f = \psi(\lambda,\mu) \tag{4.40}$$

Man sieht also, daß in diesem Falle $\psi(\lambda, \mu)$ und $\varphi(\lambda, \mu)$ übereinstimmen.

Kapitel 5

Das cg-Verfahren

In diesem Kapitel werde ich mich mit einem anderen bekannten Verfahren zur Minimierung quadratischer Funktionen befassen, dem cg-Verfahren. Dieses Verfahren wird beschrieben und hergeleitet.
Ich werde theoretisch nachweisen, daß das qcg-Verfahren und das cg-Verfahren bei der Minimierung quadratischer Funktionen identisch arbeiten. Dies wird außerdem noch an einem Beispiel illustriert.

5.1 Herleitung des cg-Verfahrens

Voraussetzung. Die Fehlerfunktion hat folgende Gestalt

$$F(\underline{x}) = \frac{1}{2}\underline{x}^T C\underline{x} + \underline{p}^T\underline{x} + q \tag{5.1}$$ vor1

und ist somit quadratisch. Außerdem gilt:

$$\begin{aligned} C &\in \mathbb{R}^{n\times n} \quad \text{ist eine spd-Matrix} \\ \underline{p}, \underline{x} &\in \mathbb{R}^n \\ q &\in \mathbb{R} \end{aligned} \tag{5.2}$$ vor2

Ich werde jetzt die Orthogonalität von Vektoren einführen, die zum Verständnis des cg-Verfahrens unerläßlich ist.

skalar **Definition 5.1.** Das Standardskalarprodukt zweier Vektoren $\underline{v}$ und $\underline{w}$ ist definiert durch:

$$(\underline{v}, \underline{w}) = \underline{v}^T\underline{w}.$$

Es wird ein zweites Skalarprodukt definiert:

$$(\underline{v}, \underline{w})_C = \underline{v}^T C\underline{w}$$

Definition 5.2. Die beiden Vektoren $\underline{v}$ und $\underline{w}$ sind orthogonal, genau dann wenn $(\underline{v}, \underline{w}) = 0$. Die beiden Vektoren $\underline{v}$ und $\underline{w}$ sind C-orthogonal (C-konjugiert), genau dann wenn $(\underline{v}, \underline{w})_C = 0$.

Im Verlaufe des cg-Verfahrens werden Suchrichtungen $\underline{z}_0, \underline{z}_1, \ldots$ so berechnet, daß sie alle C-konjugiert sind. Daher kommt auch der Name cg-Verfahren (*conjugate gradient method*). Dieses Verfahren wurde von Hestenes und Stiefel im Jahre 1952 vorgestellt.
Wir wollen nun F minimieren. Dazu benötigen wir erst einmal den Gradienten von F:

$$\nabla F(\underline{x}) = C\underline{x} + \underline{p} = 0 \tag{5.3}$$ gradient

Daran sieht man also, daß die Minimierung von $F(\underline{x})$ äquivalent ist mit der Lösung des Gleichungssystems $C\underline{x} + \underline{p} = 0$.

Angenommen, wir hätten $\underline{z}_0, \ldots, \underline{z}_{k-1}$ mit $\underline{z}_{k-1} \neq 0$ schon berechnet. Unsere Aufgabe ist es nun, die neue Suchrichtung $\underline{z}_k$ und die optimale Schrittweite γ_k zu bestimmen.

Der Vorteil bei der Minimierung quadratischer Funktionen mit Gradientenverfahren ist, daß man die Schrittweite γ_k explizit berechnen kann.
Wir setzen

$$\vartheta(\gamma) := F(\underline{x}_k + \gamma \underline{z}_k) \tag{5.4}$$

Dann berechnen wir das Minimum von $\vartheta(\gamma)$

$$\vartheta'(\gamma) = \underline{z}_k^T \nabla F(\underline{x}_k + \gamma \underline{z}_k) = 0 \tag{5.5}$$

Wegen Formel (5.3) gilt

$$\vartheta'(\gamma) = \underline{z}_k^T (C(\underline{x}_k + \gamma \underline{z}_k) + \underline{p}) \tag{5.6}$$

Daraus ergibt sich dann

$$\gamma_k = \frac{-\underline{z}_k^T (C(\underline{x}_k + \underline{p})}{\underline{z}_k^T C \underline{z}_k} \tag{5.7}$$

Bezeichnung. $\underline{g}_k = C\underline{x}_k + p$

Daraus ergibt sich für die optimale Schrittweite

$$\gamma_k = \frac{-\underline{z}_k^T \underline{g}_k}{\underline{z}_k^T C \underline{z}_k} \tag{5.8}$$

Nun bleibt nur noch offen, wie die Suchrichtung $\underline{z}_k$ bestimmt wird, die zu den anderen Suchrichtungen $\underline{z}_0, \ldots, \underline{z}_{k-1}$ C-konjugiert ist.
Wir berechnen $\underline{z}_k$ wie folgt

$$\underline{z}_k = -\underline{g}_k + \beta_{k-1} \underline{z}_{k-1} \tag{5.9}$$

cgsuch

Da alle Suchrichtungen $\underline{z}$ C-konjugiert sind, folgt speziell

$$\underline{z}_{k-1}^T C \underline{z}_k = 0 \tag{5.10}$$

Daraus ergibt sich

$$\begin{aligned} \underline{z}_{k-1}^T C \underline{z}_k &= \underline{z}_{k-1}^T C \left(-\underline{g}_k + \beta_{k-1} \underline{z}_{k-1} \right) && (5.11) \\ &= -\underline{z}_{k-1}^T C \underline{g}_k + \beta_{k-1} \underline{z}_{k-1}^T C \underline{z}_{k-1} && (5.12) \\ &= 0 && (5.13) \end{aligned}$$

Damit erhalten wir

$$\beta_{k-1} = \frac{\underline{z}_{k-1}^T C \underline{g}_k}{\underline{z}_{k-1}^T C \underline{z}_{k-1}} \tag{5.14}$$

Dann machen wir den Suchschritt

$$\underline{x}_{k+1} = \underline{x}_k + \gamma_k \underline{z}_k \tag{5.15}$$

cgschritt

exaktloes **Satz 5.1.** *Ist bei dem cg-Verfahren $k \leq n$ der kleinste Index, für den $\underline{g}_k = 0$ gilt, so folgt für dieses Verfahren, daß die berechneten Vektoren $\underline{g}_0, \ldots, \underline{g}_{k-1}$ paarweise orthogonal, also insbesondere linear unabhängig sind.*

Beweis. Ich möchte für den Beweis auf stoer2 [6] Seite 296 ff. verweisen. □

Aus diesem Satz folgt, daß bei exakter Rechnung spätestens $\underline{g}_n = 0$ folgt und damit $\underline{x}_n$ die gesuchte Lösung des Problems (gradient 5.3) ist. Aber wegen des Einflusses von Rundungsfehler ist das berechnete $\underline{g}_n$ in der Regel von 0 verschieden. Deshalb wird das cg-Verfahren in der Praxis über die Iteration $k = n$ hinaus fortgeführt.
Ausführlich ist das cg-Verfahren in stoer2 [6] und hesten [12] beschrieben.

Algorithmus des cg-Verfahrens

Wähle beliebiges $\underline{x}_0 \in \mathbb{R}^n$ und setze

$$-\underline{z}_0 = \underline{g}_0 := C\underline{x}_0 + \underline{p}$$

Solange $\underline{g}_k \neq 0$ berechne

$$\begin{aligned}
\gamma_k &= \frac{-\underline{z}_k^T \underline{g}_k}{\underline{z}_k^T C \underline{z}_k} \\
\underline{x}_{k+1} &= \underline{x}_k + \gamma_k \underline{z}_k \\
\underline{g}_{k+1} &= C\underline{x}_{k+1} + \underline{p} \\
\beta_k &= \frac{\underline{z}_k^T C \underline{g}_{k+1}}{\underline{z}_k^T C \underline{z}_k} \\
\underline{z}_{k+1} &= -\underline{g}_{k+1} + \beta_k \underline{z}_k \\
k &= k+1
\end{aligned}$$

5.2 Vergleich zum qcg-Verfahren

Es gelten die gleichen Voraussetzungen (vor1 5.1) und (vor2 5.2) wie im vorigen Abschnitt.
Ich werde im Verlaufe dieses Abschnittes nachweisen, daß das Verhältnis der Schrittweiten in negativer Gradientenrichtung $-\underline{g}_{neu}$ und in alter Suchrichtung $\underline{z}_{alt}$ und damit die Arbeitsweise beider Verfahren identisch ist.

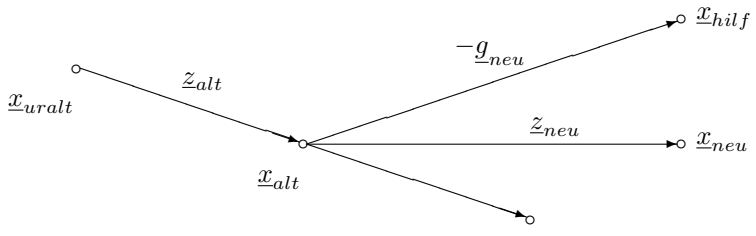

Abbildung 5.1: **Vergleich von cg- und qcg-Verfahren** cgqcg

- **cg-Verfahren:**
  Wegen den Formeln (5.9) und (5.15) wissen wir folgendes

$$\underline{z}_{neu} = (-\underline{g}_{neu} + \beta_{alt}\underline{z}_{alt})\gamma_{neu} \tag{5.16}$$
$$= \gamma_{neu}(-\underline{g}_{neu}) + \gamma_{neu}\beta_{alt}\underline{z}_{alt} \tag{5.17}$$

  Daraus ergibt sich das Verhältnis η_{cg} der Suchrichtungen

$$\eta_{cg} = \frac{\gamma_{neu}}{\gamma_{neu}\beta_{alt}} = \frac{1}{\beta_{alt}} \tag{5.18}$$

- **qcg-Verfahren:**
  Aus dem Kapitel 3.1 wissen wir

$$\underline{z}_{neu} = \mu^*(-\underline{g}_{neu}) + \lambda^*\underline{z}_{alt} \tag{5.19}$$

  Ich berechne nun das Verhältnis der Suchrichtungen $\eta_{qcg} = \frac{\mu^*}{\lambda^*}$
  Mit Hilfe der Formeln (4.8), (4.16), (4.23), (4.33) und (4.35) aus dem vorigen Kapitel stelle ich μ^* und λ^* neu dar

$$\mu^* = \frac{bd - 2ae}{4ac - b^2} = \frac{\left(-\underline{g}_{neu}^T C\underline{z}_{alt}\right)\left(\underline{z}_{alt}^T C\underline{x}_{alt} + \underline{z}_{alt}^T\underline{p}\right) - \left(\underline{z}_{alt}^T C\underline{z}_{alt}\right)\left(-\underline{g}_{neu}^T C\underline{x}_{alt} - \underline{g}_{neu}^T\underline{p}\right)}{4ac - b^2} \tag{5.20}$$

$$\lambda^* = \frac{be - 2cd}{4ac - b^2} = \frac{\left(-\underline{g}_{neu}^T C\underline{z}_{alt}\right)\left(-\underline{g}_{neu}^T C\underline{x}_{alt} - \underline{g}_{neu}^T\underline{p}\right) - \left(\underline{g}_{neu}^T C\underline{g}_{neu}\right)\left(\underline{z}_{alt}^T C\underline{x}_{alt} + \underline{z}_{alt}^T\underline{p}\right)}{4ac - b^2} \tag{5.21}$$

  Daraus ergibt sich

$$\eta_{qcg} = \frac{\left(-\underline{g}_{neu}^T C\underline{z}_{alt}\right)\left(\underline{z}_{alt}^T C\underline{x}_{alt} + \underline{z}_{alt}^T\underline{p}\right) - \left(\underline{z}_{alt}^T C\underline{z}_{alt}\right)\left(-\underline{g}_{neu}^T C\underline{x}_{alt} - \underline{g}_{neu}^T\underline{p}\right)}{\left(-\underline{g}_{neu}^T C\underline{z}_{alt}\right)\left(-\underline{g}_{neu}^T C\underline{x}_{alt} - \underline{g}_{neu}^T\underline{p}\right) - \left(\underline{g}_{neu}^T C\underline{g}_{neu}\right)\left(\underline{z}_{alt}^T C\underline{x}_{alt} + \underline{z}_{alt}^T\underline{p}\right)} \tag{5.22}$$

Wir setzen $\underline{g}_{neu} = C\underline{x}_{alt} + \underline{p}$. Dann folgt daraus

$$\eta_{qcg} = \frac{\left(-\underline{g}_{neu}^T C\underline{z}_{alt}\right)\left(\underline{z}_{alt}^T\underline{g}_{neu}\right) - \left(\underline{z}_{alt}^T C\underline{z}_{alt}\right)\left(-\underline{g}_{neu}^T\underline{g}_{neu}\right)}{\left(-\underline{g}_{neu}^T C\underline{z}_{alt}\right)\left(-\underline{g}_{neu}^T\underline{g}_{neu}\right) - \left(\underline{g}_{neu}^T C\underline{g}_{neu}\right)\left(\underline{z}_{alt}^T\underline{g}_{neu}\right)} \tag{5.23}$$

etaqcg1

Wir betrachten uns nun mal das Skalarprodukt $\underline{z}_{alt}^T\underline{g}_{neu}$. Vergleiche hierzu auch Abbildung 5.1.
cgqcg
Aus Kapitel 3.1 wissen wir, daß die quadratische Fehlerfunktion F durch eine quadratische Funktion ψ approximiert wurde.

$$\begin{aligned}\varphi(\tilde{\lambda}^*, \tilde{\mu}^*) &= F(\underline{x}_{alt}) \\ &= F(\underline{x}_{uralt} + \tilde{\mu}^*\underline{z}_{uralt} - \tilde{\lambda}^*\underline{g}_{alt}) \qquad (5.24)\\ \psi(\tilde{\lambda}^*, \tilde{\mu}^*) &= a\tilde{\lambda}^{*^2} + b\tilde{\lambda}^*\tilde{\mu}^* + c\tilde{\mu}^{*^2} + d\tilde{\lambda}^* + e\tilde{\mu}^* + f \qquad (5.25)\end{aligned}$$

mit

$$\underline{z}_{alt} = \tilde{\mu}^*\underline{z}_{uralt} - \tilde{\lambda}^*\underline{g}_{alt} \tag{5.26}$$

Wir betrachten unsere Fehlerfunktion nur in Suchrichtung $\underline{z}_{alt}$, das heißt, wir haben 2 quadratische Funktionen mit 2 Variablen $\tilde{\lambda}^*$ und $\tilde{\mu}^*$. Aus Kapitel 4 ist bekannt, daß in diesem Fall folgendes gilt

$$\varphi(\tilde{\lambda}^*, \tilde{\mu}^*) = \psi(\tilde{\lambda}^*, \tilde{\mu}^*) \tag{5.27}$$

Das bedeutet also, das Minimum der Approximationsfunktion ψ ist in $\underline{z}_{alt}-$ Richtung gleich dem Minimum der Fehlerfunktion F. Da wir $\tilde{\lambda}^*$ und $\tilde{\mu}^*$ gerade so bestimmt haben, daß wir uns im Minimum von ψ befinden (vergleiche Kapitel 3.1), folgt daraus, daß man an der Stelle $\underline{x}_{alt}$ in $\underline{z}_{alt}-$ Richtung ein Minimum von F erreicht hat.

Bezeichnung. $F(\underline{x}_{alt} + t\underline{z}_{alt}) = \rho(t)$

$$\rho'(t) = \underline{z}_{alt}^T \nabla F(\underline{x}_{alt} + t\underline{z}_{alt}) \tag{5.28}$$

Setzen wir $t = 0$, so folgt

$$\rho'(0) = \underline{z}_{alt}^T \nabla F(\underline{x}_{alt}) = 0 \tag{5.29}$$

$$= \underline{z}_{alt}^T \underline{g}_{neu} \tag{5.30}$$

deinf

Diese Erkenntnis benutzen wir nun und werten die Formel (5.23) weiter aus. Daraus ergibt sich

$$\eta_{qcg} = \frac{\underline{z}_{alt}^T C \underline{z}_{alt}}{\underline{g}_{neu}^T C \underline{z}_{alt}} \tag{5.31}$$

$$= \frac{1}{\beta_{alt}} \tag{5.32}$$

Wir haben also festgestellt, daß das cg-Verfahren und das qcg-Verfahren bei der Minimierung quadratischer Funktionen identisch arbeiten. Demzufolge ist der Satz 5.1 mit seiner Folgerung auch auf das qcg-Verfahren anwendbar. Für nichtquadratische Funktionen ist diese Aussage im allgemeinen falsch.

Bemerkung. Aus Formel (5.30) ergibt sich außerdem, daß sich das qcg-Verfahren für quadratische Fehlerfunktionen vereinfacht, da jetzt der Koeffizient d gleich 0 gesetzt werden kann. Die Formeln (3.21) auf Seite 30, (3.32) und (3.33) auf Seite 33 ändern sich wie folgt

$$a = F\left(\underline{w}^{uralt}, \underline{\theta}^{uralt}\right) - f \tag{5.33}$$

$$\lambda^* = \frac{be}{4ac - b^2} \tag{5.34}$$

$$\mu^* = -\frac{2ae}{4ac - b^2} \tag{5.35}$$

5.3 Das Beispiel

Ich habe mit dem mathematichen Softwarepaket MatLab ein Programm zum Vergleich der Arbeitsweise des cg- und qcg-Verfahrens geschrieben. Den

Quellcode dieses Programmes `beispiel.m` findet man im Anhang.

Mit diesem Programm wurde die Funktion

$$F(\underline{x}) = \frac{1}{2}\underline{x}^T C\underline{x} + \underline{p}^T\underline{x} \tag{5.36}$$

bsp

minimiert, wobei folgendes gegeben war

$$C = \begin{pmatrix} 4 & 1 \\ 1 & 4 \end{pmatrix}$$
$$\underline{p} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

Als Startvektor $\underline{x}_1$ wurde folgendes gesetzt

$$\underline{x}_1 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Durch Anwenden eines Gauß-Eliminationsschrittes auf das lineare System $C\underline{x} = -\underline{p}$ erhält man die exakte Lösung des Problems (5.36)bsp

$$\underline{x}_{exakt} = \begin{pmatrix} -0.1\bar{3} \\ -0.4\bar{6} \end{pmatrix}$$

Das cg-Verfahren und das qcg-Verfahren benötigten zur iterativen Lösung des Problems (5.36)bsp 2 Schritte, was man auf Grund des letzten Abschnittes auch erwarten konnte. Die letztendlich erhaltenen unterschiedlichen Fehler sind auf die Rundungsfehler beim Ablauf des Programmes zurückzuführen. Man sieht aber, daß sie sehr klein sind und sich unerheblich voneinander unterscheiden.

Schritte	cg-Verahren	qcg-Verfahren
0	85	85
1	0.0799049	0.0799049
2	$3.9443045 * 10^{-31}$	$1.2325952 * 10^{-32}$

Tabelle 5.1: **Vergleich von cg- und qcg-Verfahren**

Ich habe auch noch andere quadratische Minimierungsprobleme mit dem Programm `beispiel.m` getestet. Diese Beispiele bestätigten auch die Erkenntnisse des letzten Abschnittes.

Kapitel 6

Implementierung des qcg-Verfahrens

6.1 Der NN-Kernel

Der Neuronale-Netze-Kernel (NN-Kernel) ist ein Simulatorkernel für KNN. Neuronale Netze werden in dem Kernel intern als verkettete Strukturen verwaltet. Mit seiner Hilfe können Netze trainiert und angewendet werden. Gegenüber einem anderen sehr bekannten Netzwerksimulator, dem Stuutgarter Neuronale Netze Simulator (SNNS), besitzt der NN-Kernel mehrere Vorteile. Er arbeitet beispielsweise schneller, kann mehrere Netze gleichzeitig verwalten und läßt sich leichter in Programme einbinden. Detaillierte Auskunft über den SNNS erhält man in [1].
Das Lernen in diesem NN-Kernel ist hierarchisch aufgebaut:

Netzlernfunktion → Layerlernfunktion → Neuronenlernfunktion

6.2 Schwellwert als on-Neuron

Es gibt zwei Möglichkeiten den Schwellwert in Simulationen zu realisieren. Die eine habe ich bislang im Verlauf dieser Arbeit benutzt. Hier nämlich geht der Schwellwert als Parameter in den Neuronen ein. Vergleiche hierzu Abbildung 6.1.

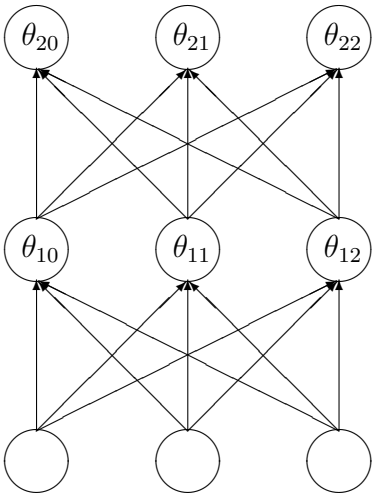

Abbildung 6.1: **Schwellwerte als Parameter in den Neuronen** param

Zur Implementierung des qcg-Verfahrens habe ich die Schwellwerte durch eine andere Variante realisiert. Hier fügt man ein einziges Neuron dem Netz hinzu. Dieses Neuron nennt man *on-Neuron*. Die Ausgabe dieses Neurons ist immer 1 und es ist mit jedem Neuron des Netzes verbunden, welches ein Schwellwert besitzt. Das negative Gewicht dieser Verbindung stellt dann den Schwellwert dar.
Die Formeln (7.1) und (7.3) auf den Seiten 8 und 9 ändern sich dadurch wie folgt:

$$a_{ij}(t+1) = f_{act}(a_{ij}(t), net_{ij}(t)) \tag{6.1}$$

$$net_{ik}(t) = \left(\sum_j o_{i-1,j}(t) w_{ijk}(t)\right) - \theta_{ij}(t) \tag{6.2}$$

Bei dieser Realisierung wird die Anzahl der Gewichte zwar erhöht, aber die Implementierung wird nicht so aufwendig durchzuführen sein, wie bei der ersten Variante, da man davon ausgeht, daß die Schwellwerte in ähnlicher Art und Weise abgeändert werden sollen wie die Gewichte. Ein KNN mit einem on-Neuron ist in Abbildung 6.2 dargestellt.

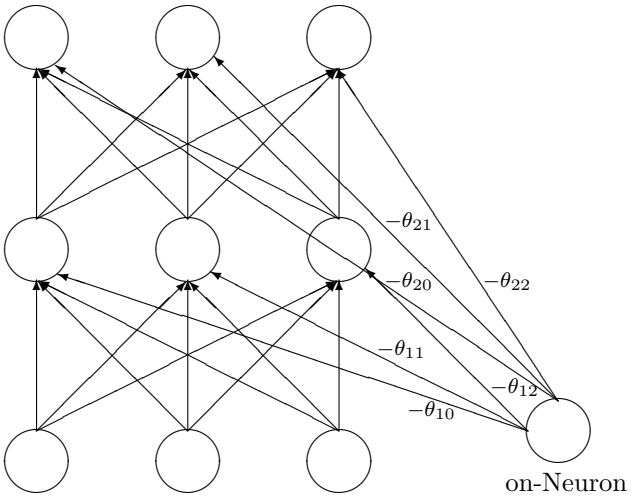

Abbildung 6.2: **Schwellwerte als on-Neuron** `onneuron`

6.3 Implementierungen in den NN-Kernel

Ich habe auf Neuronen- und Netzebene Funktionen in den NN-Kernel implementiert. Bei Backpropagation und seinen Varianten(also auch qcg-Verfahren) wird auf Layerebene die Funktion `nn_ll_dummy` aufgerufen, die von der Netz-ebene gleich auf die Neuronenebene schaltet, weil bei diesen Lernverfahren kein Layerlernen stattfindet.
Ich habe folgende Funktionen in den NN-Kernel implementiert:

- in `nn_net_learn.c`:
  - `nn_Net_learnQCG`
    Diese Funktion ist die modifizierte Netzlernfunktion für das qcg-Lernen. Innerhalb eines Lernschrittes wird sie zweimal durchlaufen, für den Hilfsschritt und für den Lernschritt.
- in `nn_neuron_learn.c`:
  - `nn_calc_deb`
    Diese Funktion berechnet die Skalarprodukte, die benötigt werden, um die Koeffizienten b, d und e zu berechnen.

- `nn_learn_backprop_qcg`
  Diese Funktion stellt den Hilfsschritt (1.Teilschritt) dar. Dies ist, wie schon erwähnt, ein Standard-Backpropagation-Schritt
- `nn_learn_qcg`
  Diese Funktion ist die Neuronenlernfunktion.
  Hier wird unterschieden, ob man einen Hilfsschritt oder einen „richtigen“ Lernschritt durchführt, wobei der „richtige“ Lernschritt in dieser Funktion beschrieben wird.

Kapitel 7

Vergleich mit anderen Lernverfahren

Ich habe das qcg-Verfahren mit, den in der Firma am häufigsten verwendeten Lernverfahren, Resilient Propagation, Backpropagation-online und Backpropagation-offline, verglichen. Dabei ist Resilient Propagation in der Praxis das bisher leistungsfähigste Verfahren der Klasse der offline-Verfahren und Backpropagation-online das Pendant für die online-Verfahren.

7.1 Probleme des Vergleichs

Der Vergleich von Lernverfahren für Feed-Forward-Netze bringt sehr viele Variationsmöglichkeiten mit sich. Ich möchte hier nur einige aufzählen, die zeigen werden, daß allgemeine Aussagen des Vergleichs von Lernverfahren nur sehr schwer zu tätigen sind.

- **Gestelltes Problem**
  Die Qualität eines Lernverfahrens hängt im hohen Maße von der konkreten Anwendung des KNN ab. Zum einen wäre die Klasse der Anwendung zu nennen, wie z.B. Zeichenerkennung, Handschrifterkennung, Robotersteuerung usw. Andererseits muß man in diesem Zusammenhang auch die Vorverarbeitung der Eingabedaten und die Art der Trainingspattern nennen.
  Messungen über die Güte eines bestimmten Lernverfahrens lassen sich also nicht von einem Problem auf das nächste übertragen. Diese Erkenntnis wird später auch noch ersichtlich sein, wenn wir feststellen, daß das qcg-Verfahren für das XOR-Problem sehr gut lernt, aber für größere Probleme erhebliche Schwierigkeiten hat.

- **Netztopologie**
  Die gewählte Netztopologie ist ebenfalls sehr entscheidend für die Güte eines Lernverfahrens, speziell die Netzwerktiefe, das heißt die Anzahl der Schichten. Jedes Lernverfahren reagiert verschiedenartig auf unterschiedliche Netztopologien. In der Praxis hat sich bisher gezeigt, daß Backpropagation und seine Modifikationen bei großer Netztiefe Schwierigkeiten besitzen. Auf Grund dieser Tatsache ist es praktisch, mit einem KNN anzufangen, welches nur eine verdeckte Schicht besitzt. Dann kann man gegebenenfalls verdeckte Schichten hinzufügen.
  Die Wahl der optimalen Netztopologie für eine bestimmte Anwendung ist im Moment eines der bedeutendsten Fragestellungen. Dieses Problem wird wohl aber auch in naher Zukunft nicht zufriedenstellend gelöst werden.

- **Optimale Parameterwahl**
  Ein weiterer wichtiger Punkt der Lernverfahren ist die Problematik der optimalen Wahl der Parameter. Jedes Lernverfahren hat seine eigenen speziellen Parameter. Bei Backpropagation ist es beispielsweise nur einer, die Schrittweite. Je nach Einstellung der Parameter arbeitet ein Lernverfahren gut oder schlecht. Hier kann man jedoch keine allgemeine Aussage treffen, da die optimale Einstellung wieder vom jeweiligen Problem abhängt, welches das KNN lösen soll. Das bedeutet, man muß, bevor ein Lernverfahren eingesetzt werden kann, eine Optimierung der Parameter durchführen, wo der Anwender meist auf seine Erfahrung im Umgang mit KNN angewiesen ist.
  Es gibt aber auch Lernverfahren, wie beispielsweise das qcg-Verfahren, die ihre Parameter im Laufe des Lernens automatisch bestimmen (λ^* und μ^*). Das bedeutet aber nicht grundsätzlich, daß diese Verfahren besser arbeiten als die anderen.

- **Initialisierung des Netzes**
  Die zufällige Gewichts- und Schwellwertinitialisierung ist ein weiteres Problem, das beim Einsatz der Lernverfahren auftritt. Vor allem bei den Gradientenverfahren (Backpropagation und seine Modifikationen) ist dies sehr bedeutend, da diese Verfahren lokale Optimierungsverfahren sind. Bei unterschiedlicher Initialisierung eines Netzes kann man mit diesen Verfahren also auch in verschiedene lokale Minima geraten. Auch die Größe der Anfangsgewichte ist sehr entscheidend für die Qualität des Lernens. Bei der Anwendung der Lernverfahren im NN-Kernel der Planet GmbH in Schwerin wurden meist Initialgewichte im Bereich von -1 bis 1 gewählt.

- **Epochenanzahl vs. Zeitaufwand**
  Hier wird eine nächste Eigenschaft von Lernverfahren überprüft, der Aufwand, und das Problem wie man den Aufwand definiert. Beispielsweise ist die Anzahl der benötigten Epochen (Lernschritte) nur dann ein sinnvolles Maß für den Aufwand, wenn der Arbeitsaufwand pro Epoche bei den unterschiedlichen Lernverfahren gleich ist. Dies ist aber im allgemeinen nicht der Fall. Das qcg-Verfahren, zum Beispiel, benötigt für einen Lernschritt immer noch einen Schritt, den Hilfsschritt, mehr als Backpropagation. Auch der Berechnungsaufwand pro Epoche ist beim qcg-Verfahren ungemein höher als beim Backpropagation-Verfahren. (Koeffizienten- und Schrittweitenberechnung).
  Man muß letztendlich ganz genau abgrenzen, wann man ein Lernverfahren effizienter und besser einschätzt als ein anderes.

7.2 Das XOR-Problem

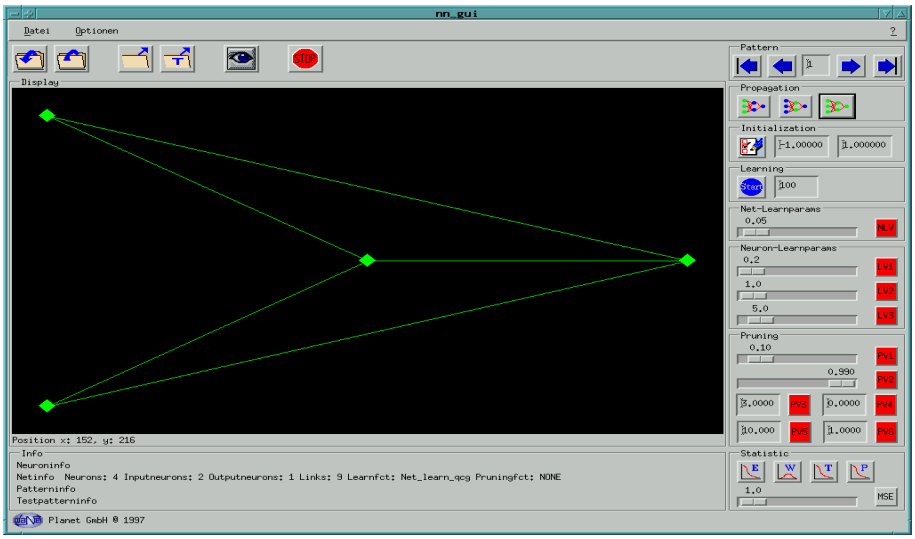

Abbildung 7.1: **XOR-Netz** `xornetz`

Das erste Testbeispiel ist das XOR-Problem. Es ist ein einfaches Problem mit 2 Eingabeneuronen, einem verdeckten Neuron und einem Ausgabeneuron. Das XOR-Netz wurde in Abbildung 7.1 in der graphischen Benutzeroberfläche des NN-Kernels, der `nn_gui`, dargestellt. Als Trainingspattern standen mir die 4 möglichen Belegungen beim XOR-Problem, $\{(0,0),(0)\}$,

$\{(1,1),(0)\},\{(0,1),(1)\}$ und $\{(1,0),(1)\}$, zur Verfügung. Die Aktivierungsfunktion war in diesem Fall die logistische Funktion (siehe Kapitel 1.2.1).
An diesem Beispiel habe ich die Konvergenzgeschwindigkeit (Anzahl der Lernschritte) von Resilient Propagation, Backpropagation-offline, Backpropagation-online und dem qcg-Verfahren getestet. Die Generalisierungsleistung spielt hier keine Rolle, da man keine adäquaten Testpattern besitzt, um diese nachzuprüfen.
Ich habe zum Verfahrensvergleich den mittleren quadratischen Fehler (mean square error, MSE) herangezogen, der sich wie folgt berechnet

$$\text{MSE} = \frac{\text{SSE}}{\text{Anzahl der Trainingspattern}} \tag{7.1}$$

wobei SSE (sum square error) den Gesamtfehler ausdrückt (siehe Formel (2.5) auf Seite 16).
fehler fehler
Als erstes habe ich das XOR-Netz mit Gewichten und Schwellwerten im Bereich von -1 bis 1 zufällig initialisiert und dann kopiert, damit ich Vergleichsfehler auf Grund von unterschiedlichen Initialisierungen ausschliessen konnte. Dann habe ich für alle 4 Verfahren eine Parameteroptimierung durchgeführt. Die Ergebnisse für das qcg-Verfahren sind in Abbildung 7.2 und in Tabelle B.3 dargestellt.
xorstepopti
xor3

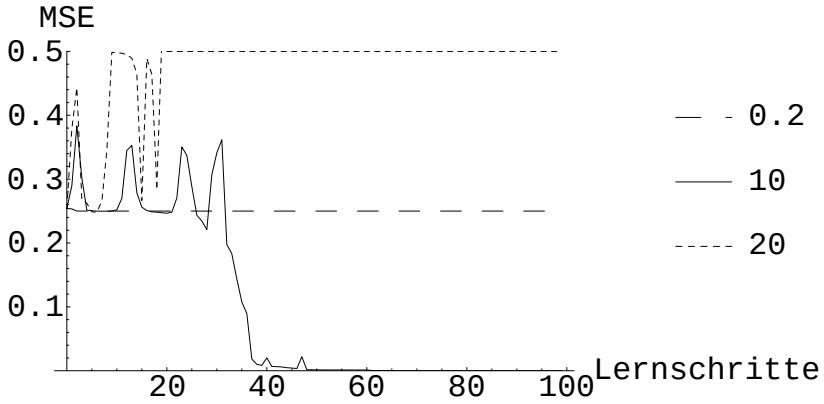

Abbildung 7.2: **XOR-qcg-Schrittweitenoptimierung** xorstepopti

Wie man der Abbildung entnehmen kann, hängt es ganz entscheidend von der Schrittweite ab, wie gut oder wie schlecht das qcg-Verfahren lernt.

Das kommt daher, daß die Gesamtfehlerfunktion mehrere große Plateaus besitzt, beispielsweise bei einem MSE von 0.25. Bei einer Schrittweite von 0.2 ist das qcg-Verfahren nicht in der Lage in einer akzeptablen Zeit von diesem Plateau herunterzukommen, da die Schrittweite für den Hilfsschritt zu gering ist. Hingegen bei einer Schrittweite von 20 vollführt das Verfahren gleich einen solch großen Schritt, daß man ein anderes Plateau bei MSE = 0.5 erreicht. Es hat sich herausgestellt, daß die Schrittweite 10 optimal für das XOR-Problem ist.
Beim Vergleich der 4 Lernverfahren (siehe Abbildungen 7.3 und 7.4) kann man ganz klar erkennen, daß das qcg-Verfahren für dieses Problem am schnellsten konvergiert. Diese Erkenntnis ist nicht verwunderlich, da die Fehlerfunktion des XOR-Problems konvex ist (mit Ausnahme der Plateaus). Wir haben in den Kapiteln 5 und 6 festgestellt, daß für diese Art von Problemen das qcg-Verfahren besonders gut geeignet ist. Auffallend ist auch, daß das Backpropagation-online-Verfahren mit Abstand das schlechteste Verfahren ist. Die anderen beiden Verfahren, Resilient Propagation und Backpropagation-offline, haben eine ungefähr gleiche Konvergenzgeschwindigkeit.
Ich möchte an dieser Stelle noch einmal betonen, daß dieses Ergebnis nicht verallgemeinert werden kann. Denn diese Vergleichsaussage ist abhängig von der Initialisierung der Gewichte und Schwellwerte und vom zu lösenden Problem. Das bedeutet, das qcg-Verfahren ist nicht grundsätzlich für jedes Problem das beste Lernverfahren, was wir im nächsten Abschnitt auch sehen werden.

xorvergl xorbackon

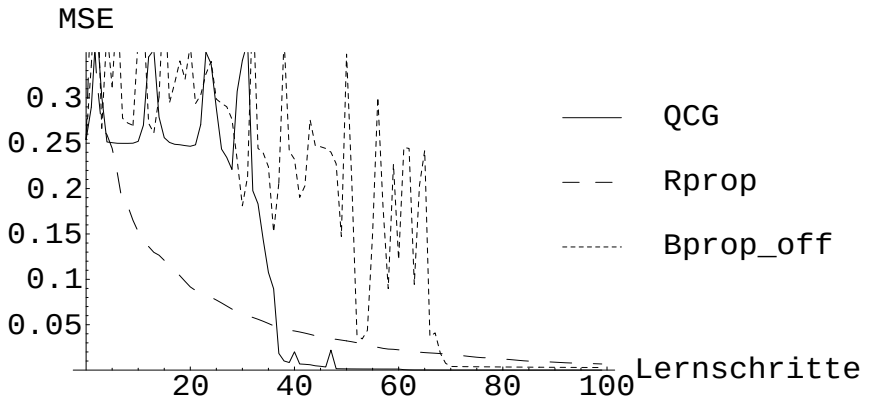

Abbildung 7.3: **XOR-Vergleich** xorvergl

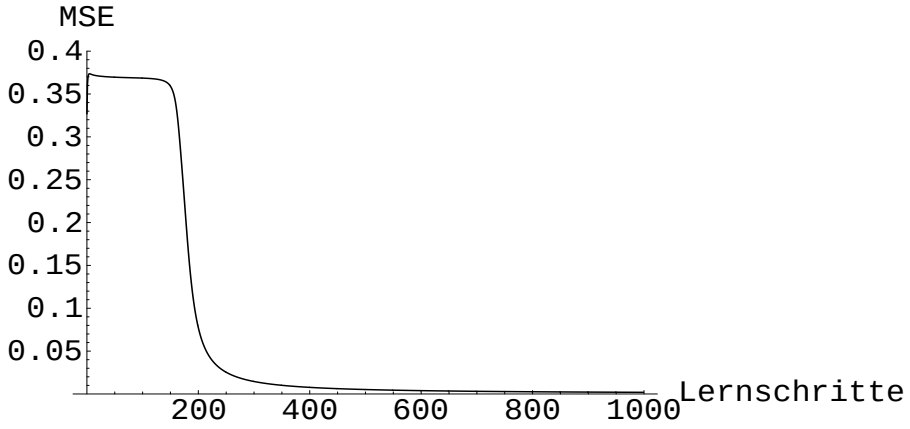

Abbildung 7.4: **XOR-Backpropagation-online** `xorbackon`

Es wurden auch noch einige andere einfache akademische Probleme für den Vergleich der 4 Verfahren herangezogen, wie zum Beispiel das OR-Problem. Auch hier arbeitete das qcg-Verfahren sehr zufriedenstellend.

7.3 Das ZAHL-Problem

Das andere Netz, welches ich zum Vergleich der Lernverfahren benutzt habe, hieß *zahl*. Dieses Netz habe ich mit dem tool `create_nn` erzeugt und dann kopiert, so daß ich für den Vergleich der Lernverfahren Fehler durch unterschiedliche Initialisierungen ausschließen konnte.
Wie der Name schon erahnen läßt, wurden mit diesem Netz Zahlen gelernt. Es war ein *1-aus-N Decoder*, d.h. die Ausgabeneuronen konnten nur die Werte 0 oder 1 annehmen, und jeweils 1 Neuron in der Ausgabeschicht stand für eine Klasse von Zahlen.
Das Netz bestand aus insgesamt 83 Neuronen, davon 60 input-Neuronen, 10 hidden-Neuronen und 13 output-Neuronen. Es hatte 2 Schichten trainierbarer Verbindungen.
zahl war vollständig ebenenweise verbunden, ohne shortcut-connections. Das Netz besaß also insgesamt 813 Verbindungen, wovon aber 83 Verbindungen vom on-Neuron ausgingen. Hierzu möchte ich noch sagen, daß zwar in diesem Netz die Neuronen der Eingabeschicht auch mit Schwellwerten versehen wurden, diese aber bei den Berechnungen nicht eingingen, was typisch ist für

Feed-Forward-Netze. Auch dieses Netz ist in Abbildung 7.5 in der `nn_gui` dargestellt.

zahlnetz

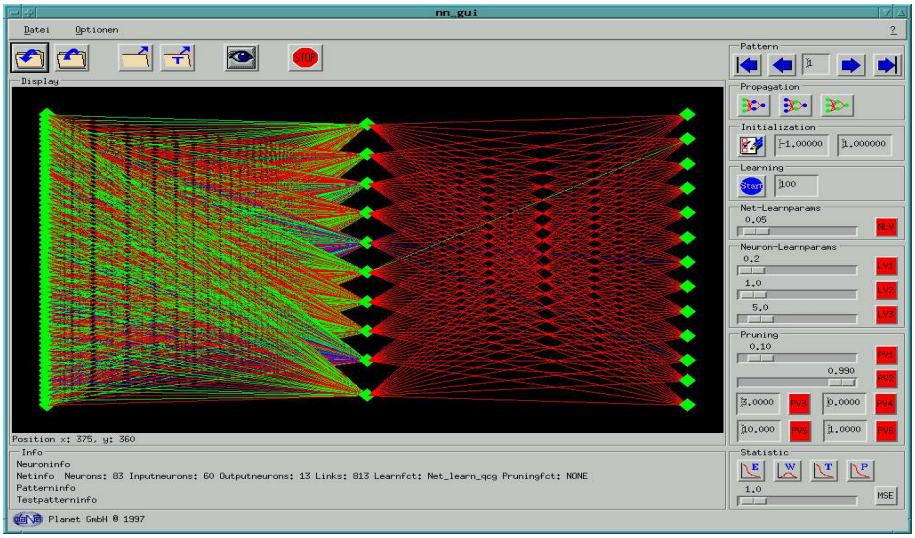

Abbildung 7.5: **ZAHL-Netz** `zahlnetz`

Zum Trainieren des Netzes standen mir 2750 Trainingspattern zur Verfügung. Zur Kontrolle der Generalisierungsfähigkeit benutzte ich 2564 Testpattern.

Als erstes habe ich eine Patternpaketoptimierung für das qcg-Verfahren durchgeführt (vergleiche Kapitel 3.2.1). Dabei hat sich herausgestellt, daß für dieses Problem die Patternpaketgröße unerheblich für die Lernqualität des qcg-Verfahrens ist. Je kleiner aber die Anzahl der Trainingspattern in einem Paket ist, desto mehr Lernschritte werden durchgeführt, und das bedeutet, das qcg-Verfahren ist dann zeitaufwendiger. Deshalb habe ich von nun an mit dem vollen Patternpaket (2750 Trainingspattern in einem Paket) weiter gearbeitet.
Nach der Patternpaketoptimierung wurde eine Parameteroptimierung des qcg-Verfahrens durchgeführt. Bei den anderen 3 Lernverfahren habe ich die Parameter, welche in der Vergangenheit, beim Lernen mit diesen Verfahren, in der Firma herausgefunden wurden, als optimale Parameter gesetzt.

Aussagen zum Vergleich der Lernverfahren mit Begründungen und Erklärungen (siehe Hefter)

Anhang A

Programm zum Beispiel aus Kapitel 5.3

Das MatLab-Programm beispiel.m:

```
% ========================================================
% Beispiel zum Vergleich von cg-Verfahren und qcg-Verfahren
% ========================================================

% Erklaerung der Variablen
%      f_...   -> jeweiliger Fehler
%      g_...   -> jeweiliger Gradient des Fehlers
%      z_...   -> jeweilige Suchrichtung
%      step    -> Schrittweite des Gradientenschrittes
%      C       -> Matrix
%      p       -> Vektor
%      x_...   -> jeweiliger Loesungsvektor
%      tol     -> Fehlerschranke
%      v_error -> Hilfsvariable zur Fehlerberechnung
%      error   -> Fehler der Iteration
%      it      -> Anzahl der Iterationsschritte
%      n       -> Dimension des Systems
%      control -> flag zur Kontrolle der spd von C
%      cnt_pd  -> Variable zur Kontrolle der pos. Def. von C
%      cnt_sym -> Variable zur Kontrolle der Symmetrie von C


% Initialisierung der Anfangsgroessen
% -----------------------------------
```

```
tol = 0.00001;
it = 0;
control = 1;

% Genauigkeit der Variablen erhoehen
% ----------------------------------
format long;


% Anfangsmenue
% ------------
disp('           ---------------------------------------')
disp('           Loesen des Beispiels aus Kapitel 5.3')
disp('           ---------------------------------------')
disp(' ')
disp('Es soll das System Cx + p = 0 geloest werden')
disp('wobei C eine spd-Matrix ist')
disp(' ')

% Eingabe von Matrix C, Vektor p und Startvektor x
% ------------------------------------------------
disp('Waehle die Dimension')
n = input('Eingabe: ');
disp(' ')
while control == 1
  disp('Gebe die Matrix- und die Vektorelemente ein')
  disp('(zeilenweise von links nach rechts)')
  disp(' ')
  for i = 1:n
     for j = 1:n
        C(i,j) = input('c = ');
     end
     p(i,1) = input('p = ');
     x_ite(i,1) = input('x_anfang = ');
  end

  % Kontrolle der spd-Eigenschaft von C
  % -----------------------------------
  cnt_sym = 0;
  cnt_pd = 0;
  for i = 1:n
```

```
      if det(C(1:i,1:i)) > 0
        cnt_pd = cnt_pd + 1;
      else
        disp(' ')
        disp('Die Matrix C muss positiv definit sein')
        disp(' ')
        break
      end
  end
  for i = 1:n
     for j = i+1:n
        if C(i,j) == C(j,i)
          cnt_sym = cnt_sym + 1;
        else
          disp(' ')
          disp('Die Matrix C muss symmetrisch sein')
          disp(' ')
          break
        end
     end
  end
  if (cnt_sym == (n^2-n)/2) & (cnt_pd == n)
    control = 0;
  end
end

% Ausgabe der Matrix und der Vektoren zur Kontrolle
% -------------------------------------------------
disp(' ')
disp('Es wurde folgendes System gewaehlt')
disp('Die Matrix C:')
disp(C)
disp('Der Vektor p:')
disp(p)
disp('Der Startvektor x:')
disp(x_ite)

% Menue zur Auswahl der Verfahren
% -------------------------------
disp(' ')
disp('Waehle eines der beiden Verfahren')
```

```
disp(' ')
disp('                  cg-Verfahren ..... 1')
disp('                  qcg-Verfahren ..... 2')
disp(' ')
answer = input('Eingabe: ');

if answer == 2
   disp('                                  Das qcg-Verfahren')
   disp('                                  -----------------')

   disp('Der Iterationsfehler:')
   disp('--------------------')

   % 1. Schritt
   % ----------
   g_old = C * x_ite + p;
   f_old = 1/2 * x_ite' * C * x_ite + p' * x_ite;
   step = (g_old' * g_old) / (g_old' * C * g_old);
   z_new = -step * g_old;
   x_ite = x_ite + z_new;

   % Berechnung des Fehlers
   % ----------------------
   v_error = C * x_ite + p;
   error = 0;
   for i = 1:n
      error = v_error(i,1)^2 + error;
   end

   disp(error)
   it = it + 1;

   % Iteration
   % ---------
   while error >= tol
      z_old = z_new;
      f_urold = f_old;
      g_urold = g_old;
      f_old = 1/2 * x_ite' * C * x_ite + p' * x_ite;
      g_old = C * x_ite + p;
      z_new = -g_old;
```

```
      % Hilfsschritt
      % ------------
      x_hilf = x_ite + z_new;
      f_hilf = 1/2 * x_hilf' * C * x_hilf + p' * x_hilf;

      % Koeffizientenberechnung
      % -----------------------
      e = -z_new' * z_new;
      f = f_old;
      a = f_urold - f;
      b = e - (z_new' * g_urold);
      c = f_hilf - e - f;

      % Berechnung von lambda und mu
      % ----------------------------
      quot = 4 * a * c - b^2;
      lambda = b * e  / quot;
      mu = - 2 * a * e / quot;

      % Berechnung der neuen Suchrichtung
      % ---------------------------------
      z_new = mu * z_new + lambda * z_old;

      % qcg-Schritt
      % -----------
      x_ite = x_ite + z_new;

      % Berechnung des Fehlers
      % ----------------------
      v_error = C * x_ite + p;
      error = 0;
      for i = 1:n
         error = v_error(i,1)^2 + error;
      end

      disp(error)
      it = it + 1;
   end

else
```

```
disp('                                    Das cg-Verfahren')
disp('                                    ----------------')
disp(' ')

disp('Der Iterationsfehler:')
disp('--------------------')

% 1. Schritt
% ----------
g = C * x_ite + p;
z_new = -g;
step = (-z_new' * g) / (z_new' * C * z_new);
x_ite = x_ite + step * z_new;

% Berechnung des Fehlers
% ----------------------
v_error = C * x_ite + p;
error = 0;
for i = 1:n
   error = v_error(i,1)^2 + error;
end

disp(error)
it = it + 1;

% Iteration
% ---------
while error>=tol
   z_old = z_new;
   g = C * x_ite + p;
   z_new = -g_old;

   % Berechnung von beta
   % -------------------
   beta = (z_old' * C * g) / (z_old' * C * z_old);

   % Berechnung der neuen Suchrichtung
   % ---------------------------------
   z_new = -g + beta * z_old;

   % Berechnung der neuen Schrittweite step
```

```
      % ------------------------------------
      step = (-z_new' * g) / (z_new' * C * z_new);

      % cg-Schritt
      % ----------
      x_ite = x_ite + step * z_new;

      % Berechnung des Fehlers
      % ----------------------
      v_error = C * x_ite + p;
      error = 0;
      for i = 1:n
         error = v_error(i,1)^2 + error;
      end

      disp(error)
      it = it + 1;
   end
end

disp('Anzahl der Iterationsschritte:')
disp(it)
disp(' ')
% Ausgabe der iterierten Loesung
% ------------------------------
disp('Die iterierte Loesung:')
disp(x_ite)
% Berechnung der exakten Loesung
% ------------------------------
x_exakt = C\(-p);
disp('Die exakte Loesung:')
disp(x_exakt)
disp('Ende der Iteration')
clear;
```

Anhang B

Tabellen zum Vergleich der Verfahren

Schritte	mittlerer quadratischer Fehler (MSE)		
	qcg-Verfahren	Rprop	Bprop-offline
0	0.254064	0.254064	0.254064
10	0.252036	0.152424	0.365914
20	0.246624	0.097758	0.358099
30	0.341185	0.061437	0.180997
40	0.020155	0.043125	0.232320
50	0.001350	0.032208	0.348161
60	0.001137	0.022337	0.121741
70	0.000005	0.017228	0.004055
80	0.000005	0.012264	0.003437
90	0.000005	0.008707	0.003079
100	0.000005	0.006739	0.002822

Tabelle B.1: **XOR-Vergleich1** `xor1`

Schritte	mittlerer quadratischer Fehler (MSE)
	Bprop-online
0	0.326796
100	0.368556
200	0.077343
300	0.014552
400	0.007691
500	0.005156
600	0.003880
700	0.003096
800	0.002571
900	0.002197
1000	0.001918

Tabelle B.2: **XOR-Vergleich2** `xor2`

Schritte	mittlerer quadratischer Fehler (MSE)				
	0.2	5	10	15	20
0	0.254064	0.254064	0.254064	0.254064	0.254064
10	0.250004	0.249999	0.252036	0.263512	0.498083
20	0.249999	0.249998	0.246624	0.247158	0.499992
30	0.249999	0.249997	0.341185	0.499031	0.499992
40	0.249999	0.250008	0.020155	0.498319	0.499992
50	0.249999	0.248700	0.001350	0.494253	0.499992
60	0.249999	0.242127	0.001137	0.482048	0.499992
70	0.249999	0.205148	0.000005	0.185142	0.499991
80	0.249999	0.118939	0.000005	0.243220	0.499991
90	0.249999	0.014291	0.000005	0.295643	0.499991
100	0.249999	0.004603	0.000005	0.117214	0.499991

Tabelle B.3: **XOR-Schrittweitenoptimierung beim qcg-Verfahren** `xor3`

Anhang C

Zusammenfassung und Ausblick

C.1 Zusammenfassung

Es gibt zwei grundsätzliche Arten von Lernverfahren, die *deterministischen*, welche ich in der vorliegenden Arbeit behandelt habe, und die *stochastischen* Lernverfahren. Die erste Klasse benutzt den Gradienten der Fehlerfunktion F, wo hingegen die andere den Gradienten nicht benötigt.
Zu der ersten Klasse gehört unter anderem Backpropagation mit seinen Modifikationen, welches alle lokale Optimierungsverfahren sind.
Im Gegensatz dazu sind die stochastischen Lernverfahren globale Optimierungsverfahren, was auf den ersten Blick schließen läßt, daß diese Verfahren den deterministischen Verfahren deutlich überlegen sind. Dieser Vorteil jedoch zieht eine Menge Nachteile nach sich. Dadurch, daß das Auffinden globaler Minima sehr stark zufallsabhängig ist, sind diese Lernalgorithmen sehr zeit- und rechenaufwendig.
Da man bei den KNN sehr viele Freiheitsgrade(Gewichte und Schwellwerte) besitzt, haben sich die stochastischen Verfahren in der Praxis noch nicht bedeutend gegenüber den deterministischen Verfahren durchgesetzt. Es ist immer noch gebräuchlich, das deterministische Lernen einzusetzen und von mehreren Startpunkten aus zu beginnen.
Die populärsten Vertreter der stochastischen Lernverfahren sind *simulated anealing, threshold-accepting, Sintflut-Algorithmus* und *genetische Algorithmen*. Detaillierte Auskunft zu diesen Verfahren kann man in [1] und [8] bekommen.

Ein anderes großes Problem im praktischen Umgang mit KNN ist die optimale Bestimmung der Netztopologie. Die Netztopologie ist im höchsten Maße dafür verantwortlich, wie gut das Netz eine gestellte Aufgabe löst. Eine Her-

angehensweise zur Lösung dieses Problems, die *Netzwerkspezifikation*, wird genauer in [anders] [8] Kapitel 5 behandelt.
In [zell] [1] werden einige Lernverfahren vorgestellt, die die optimale Netzwerkarchitektur automatisch bestimmen sollen, wie z.B. *Cascade-Correlation*.

Das Ziel dieser Diplomarbeit bestand darin, eine Variante des Standard-Backpropagation-Lernverfahrens herzuleiten und in der Praxis zu testen. Dabei hat sich herausgestellt, daß das qcg-Verfahren die gleichen Probleme aufzeigt, wie die meisten anderen Modifikationen von Backpropagation. An kleinen akademischen Beispielen zeigt es sehr gute Lernleistungen, aber bei praktischen großen Problemen, wie dem ZAHL-Netz, hat es sehr große Schwierigkeiten. Es läßt sich feststellen, daß sich bis heute noch keine der vielen Varianten entscheidend in der Praxis durchgesetzt hat, vorallem nicht gegenüber dem Backpropagation-online-Verfahren.

C.2 Ausblick

Da die KNN immer mehr Einsatzgebiete in der Praxis finden, ist es unerläßlich auf diesem Gebiet intensiv weiter zu forschen. Man muß hier zugleich anmerken, daß die KNN in vorher nicht für möglich gehaltenen Anwendungsbereichen erfolgreich zum Einsatz kommen. Einige interessante Einsatzmöglichkeiten werden in [zell] [1] im Teil 4 vorgestellt und beschrieben.
In den nächsten Jahren werden sicherlich solche Lernverfahren verstärkt zum Einsatz kommen, die ihre optimale Netztopologie selber bestimmen und festlegen.
Die Verbindung zwischen KNN und Evolutionsalgorithmen (genetische Algorithmen, genetische Programmierung, Evolutionsstrategien) wird in naher Zukunft verstärkt werden. Bisher haben sich diese Verfahren beispielsweise gegenüber Backpropagation noch nicht durchgesetzt.
In theoretischer Hinsicht wird man sich auch in den nächsten Jahren an den biologischen Vorbildern orientieren, um die Informationsverarbeitung im menschlichen Gehirn noch besser und detaillierter zu verstehen. Auf praktischem Gebiet scheinen diese Modelle aber ungeeignet, gegenüber den einfacheren mathematischen Modellen.

Es wäre sicherlich interessant zu testen, wie das qcg-Verfahren als online-Verfahren arbeitet. Ich habe im Kapitel 3 schon einmal bemerkt, daß dieses Verfahren in der jetzigen Form ein offline-Verfahren ist, da es bei der quadratischen Approximation auf vergangenheitsbezogene Werte zurückgreift. Eine Idee ist, für jedes Trainingspattern mehrere Lernschritte hintereinander

durchzuführen (bei Backpropagation-online ist es genau 1 Lernschritt). Damit erreicht man den Effekt, daß sich die Vergangenheitswerte auf das gleiche Pattern beziehen. Dann könnte man vielleicht sukzesssive die Patternpaketgröße erhöhen, wie im Kapitel 3.2.1 vorgeschlagen wurde. Der Vergleich dieses qcg-Verfahrens zum Backpropagation-online-Verfahren wäre sehr interessant.

Abschließend läßt sich noch anmerken, daß trotz aller Fortschritte im Bereich der KNN, die Leistungsfähigkeit der Neuronalen Netze in absehbarer Zeit nicht die des leistungsfähigsten Biologischen Neuronalen Netz, dem menschlichen Gehirn, erreichen wird.

Literaturverzeichnis

zell [1] Andreas Zell. *Simulation neuronaler Netze*. Addison Wesley Longman Verlag GmbH, 1994

rojas [2] Raúl Rojas. *Theorie der neuronalen Netze - eine systematische Einführung*. Springer Verlag, 1993

hagen1 [3] Burkhard Lenze. *Mathematische Aspekte neuronaler Netze - Kurseinheit 1*. Fernuniversität Hagen, 1996

hagen2 [4] Burkhard Lenze. *Mathematische Aspekte neuronaler Netze - Kurseinheit 5*. Fernuniversität Hagen, 1996

stoer1 [5] Josef Stoer. *Numerische Mathematik 1*. 7. Auflage, Springer Verlag, 1994

stoer2 [6] Josef Stoer, Roland Bulirsch. *Numerische Mathematik 2*. 3. Auflage, Springer Verlag, 1990

deufl [7] Peter Deuflhard, Andreas Hohmann. *Numerische Mathematik - Eine algorithmisch orientierte Einführung*. Walter de Gruyter Verlag, 1991

anders [8] Ulrich Anders. *Statistische neuronale Netze*. Verlag Franz Vahlen GmbH München, 1997

barner [9] Martin Barner, Friedrich Flohr. *Analysis 2*. 2. Auflage, Walter de Gruyter Verlag, 1989

bronst [10] Bronstein, Semendjajew, Musiol, Mühlig. *Taschenbuch der Mathematik*. Verlag Harri Deutsch, 1993

gantma [11] F.R. Gantmacher. *Matrizentheorie*. VEB Deutscher Verlag der Wissenschaften, 1986

hesten [12] Magnus Hestenes. *Conjugate Direction Methods in Optimization*. Springer Verlag, 1980

schawe [13] Robert Schaback, Helmut Werner. *Numerische Mathematik*. Springer Verlag, 1992

schwa [14] Hans Rudolf Schwarz. *Numerische Mathematik*. B.G. Teubner Stuttgart, 1986

Erklärung:

Hiermit erkläre ich, daß ich diese Diplomarbeit eigenständig angefertigt, sowie alle benutzten Quellen und Zitate ausreichend referenziert habe. Es wurden keine anderen als die angegebenen Quellen benutzt.
Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Rostock, 11. Juli 1999

Conny Dethloff