Enrico Biermann (enrico@cs.tu-berlin.de)
Timo Glaser (timog@cs.tu-berlin.de)
Marco Kunze (makunze@cs.tu-berlin.de)
Sebastian Nowozin (nowozin@cs.tu-berlin.de)

WS 2002/03
20. 1. 2003

Grundlagen neuronaler Netzwerke

Dieser Text befasst sich mit einfachen neuronalen Netzwerken im Rahmen eines Programmier Praktikums an der TU-Berlin [2]. Er ist daher in Bezug auf die neuronalen Netze sehr eingeschränkt und stellt nur eine einfache Einführung in die Thematik dar, mit dem die im Programm verwendeten Algorithmen besser verständlich sind. Für eine detaillierte Einführung in neuronale Netze ist das Werk von Andreas Zell [1] empfohlen.

1 Aufbau des neuronalen Netzwerks

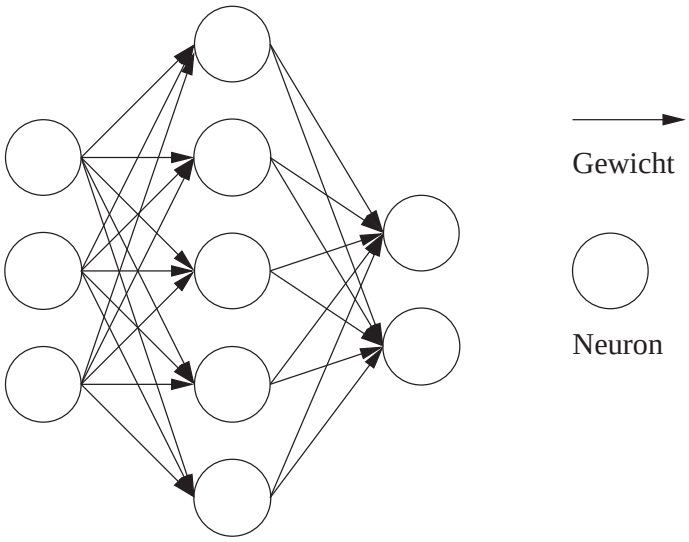

Abbildung 1: Allgemeines Schema eines Multi-Layer-Perceptron mit drei Schichten

Die Grundstruktur des verwendeten neuronalen Netzes ist in Abbildung 1 schematisiert dargestellt. Es handelt sich um ein Multi-Layer-Perceptron Netzwerk mit folgenden Eigenschaften:

- In Schichten organisiertes ebenenweise verbundenes *feed-forward* Netz

  Das neuronale Netz ist in mehrere Schichten von Neuronen eingeteilt. Es gibt nur Verbindungen von einer Schicht zur nächsten.

- Drei Arten von Neuronen: Input, Hidden, Output

  Die Art der Neuronen wird durch ihre Position innerhalb des Netzes bestimmt. Dabei sind Neuronen, die in der ersten Schicht des Netzes liegen die Input-Neuronen, die der letzten Schicht die Output-Neuronen. Alle dazwischenliegenden Neuronen sind verdeckt - gehören also zu den Hidden-Neuronen.

1.1 Aufbau eines Neurons

Der Aufbau aller Neuronen des Netzes ist in Abbildung 2 schematisiert.

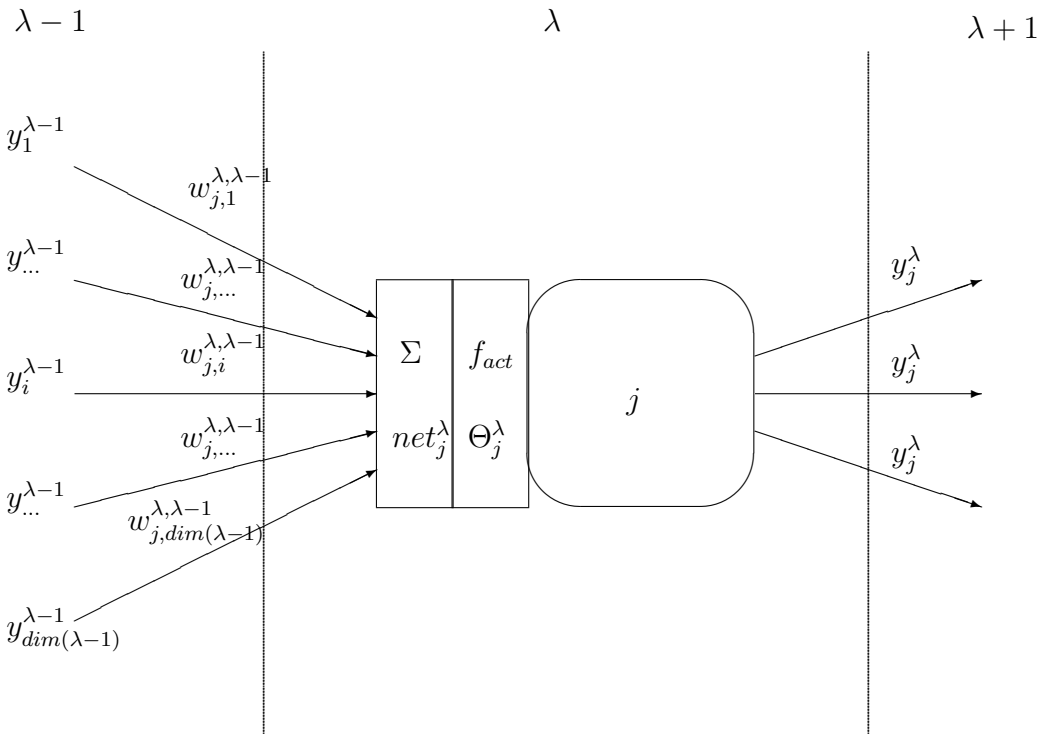

Abbildung 2: Neuron j in der Schicht λ

Dabei sind die verwendeten Terme wie folgt zu verstehen:

Term	Erklärung
λ	Eine Schicht des Netzwerkes (Ebene)
y_i^λ	Outputsignal des i. Neurons der Schicht λ
$w_{j,i}^{\lambda,\lambda-1}$	Gewicht der Verbindung zwischen Neuron i der Schicht $(\lambda - 1)$ und Neuron j der Schicht λ
$dim(\lambda)$	Anzahl der Neuronen in der Schicht λ
net_i^λ	Eingangssignal des Neurons i in der Schicht λ
Θ_i^λ	Schwellenwert des Neurons i in der Schicht λ
f_{act}	Sigmoide Aktivierungsfunktion

2 Propagierung des Eingabemusters

Wird ein Eingabevektor, genannt Muster p (Input) durch das neuronale Netz propagiert, so liegt anschliessend an den Ausgabeneuronen ein Ausgabesignal (Output) an, das das "Resultat" des neuronalen Netzes für das Muster p darstellt. Diese Ausgabesignale bilden den Ausgabevektor. Die von der Eingabe p abhängigen Parameter des Netzes erhalten als Kennzeichnung den Index p. Zum Beispiel wird aus net_i^λ jetzt $net_{p,i}^\lambda$.

Um diese Propagierung vorzunehmen, wird jede Schicht des Netzes, von links nach rechts einzeln abgearbeitet. Dabei sind mehrere Fälle zu unterscheiden:

- Die Schicht ist die Eingabeschicht

Hier wird für jedes Neuron i der Schicht λ als Ausgabesignal $y^{\lambda}_{p,i}$ der i. Eingabepegel des Eingabevektors benutzt. Da vor den Neuronen dieser Schicht keine anderen Neuronen liegen, existieren keine Gewichte, und die Eingabe wird direkt als Ausgabesignal übernommen. Würde auch hier eine Aktivierungsfunktion verwendet werden, so würde das Eingabesignal verzerrt werden, bevor eine gewichtete Summation vorgenommen werden kann, was die Lernfähigkeit des Netzes extrem einschränkt. Damit gilt für alle Neuronen i der Eingabeschicht λ:

$$y^{\lambda}_{p,i} = input_{p,i}$$

Wobei $input_{p,i}$ ein Wert des Eingabevektors des Gesamtnetzes darstellt.

- Die Schicht ist eine verdeckte (hidden) oder eine Ausgabe-Schicht

  Für die Neuronen dieser Schichten gelten die unten beschriebenen Propagierungsregeln.

2.1 Propagierung innerhalb eines Neurons

Das Eingangssignal eines Neurons i in der Schicht λ wird gewonnen mittels Summation der gewichteten Ausgaben der Neuronen der Vorgängerschicht. Damit gilt:

$$net^{\lambda}_{p,i} = \sum_{j=1}^{dim(\lambda-1)} w^{\lambda,\lambda-1}_{i,j} \cdot y^{\lambda-1}_{p,j}$$

Das Ausgangssignal eines Neurons i in der Schicht λ wird aus der Aktivierungsfunktion f_{act} und dem Eingangssignal $net^{\lambda}_{p,i}$ des Neurons bestimmt:

$$y^{\lambda}_{p,i} = f_{act}(net^{\lambda}_{p,i})$$

2.2 Die Aktivierungsfunktion f_{act}

Die Aktivierungsfunktion hat mehrere Aufgaben bei der Berechnung des Ausgabesignals eines Neurons. Damit ist sie entscheident für die Leistungsfähigkeit des gesamten Netzes. Man kann zeigen, dass bei Verwendung einer linearen Aktivierungsfunktion eine Schicht von veränderbaren Gewichten ausreicht, um alle mehrschichtigen Netzwerke darzustellen. Dies, und die biologische Motivation Neuronen besonders sensibel zu machen, wenn ein gewisser Schwellenwert überschritten wird, führt dazu, dass man andere, nicht lineare Aktivierungsfunktionen benutzt.

Zwei Funktionen eignen sich besonders für die Verwendung als Aktivierungsfunktion: Die Tangens-Hyperbolicus Funktion $tanh(x)$, und die logistische Funktion $\frac{1}{1+e^{-net^{\lambda}_{p,i}}}$. Beide haben einen ähnlichen Graphenverlauf, der in Abbildung 3 (Tangens-Hyperbolicus) und 4 (logistische Funktion) aufgetragen ist.

Der Schwellenwert Θ jedes Neurons - der manchmal auch als *bias* bezeichnet wird - parametrisiert die Aktivierungsfunktion. Er wird als einfacher Summand zum Eingabesignal addiert, um die Kurven längst der x-Achse zu verschieben:

$$f_{tanh}(net^{\lambda}_{p,i}, \Theta^{\lambda}_{i}) = tanh(net^{\lambda}_{p,i} - \Theta^{\lambda}_{i})$$

$$f_{log}(net^{\lambda}_{p,i}, \Theta^{\lambda}_{i}) = \frac{1}{1 + e^{-(net^{\lambda}_{p,i} - \Theta^{\lambda}_{i})}}$$

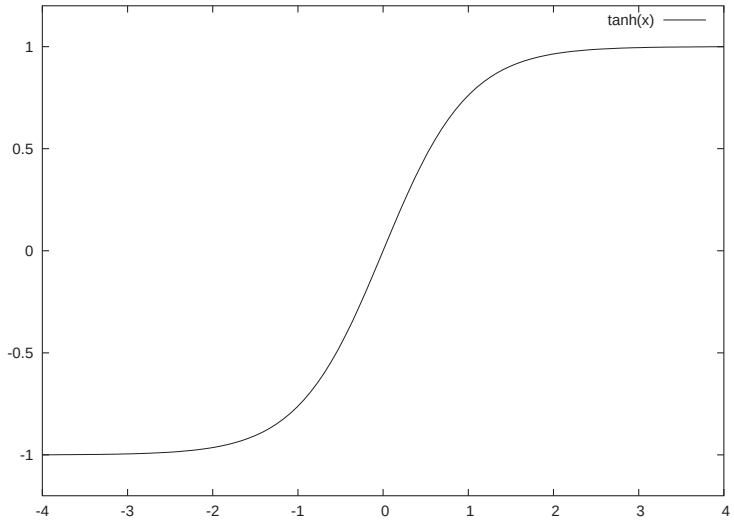

Abbildung 3: Aktivierungsfunktion $f_{tanh}(x) = tanh(x)$

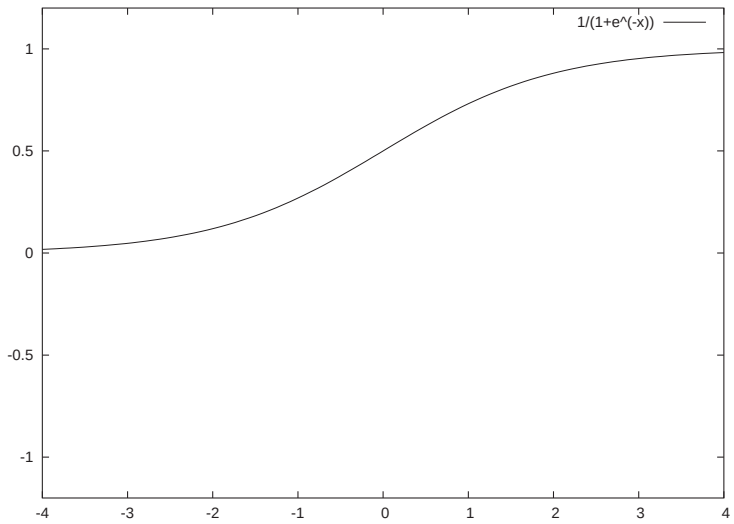

Abbildung 4: Aktivierungsfunktion $f_{log}(x) = \frac{1}{1+e^{-net_{p,i}^{\lambda}}}$

3 Fehlerterm

Um den Begriff "Fehler eines neuronalen Netzes" zu definieren, benutzt man eine Fehlerfunktion. Diese berechnet den aktuellen Fehler des Netzes für ein Trainingsmuster. Sie ist definiert als:

$$E = \frac{1}{2} \left\| t_{p,i} - y_{p,i}^{\lambda,L} \right\| = \frac{1}{2} \sum_i \left(t_{p,i} - y_{p,i}^{\lambda,L} \right)^2$$

Wobei λ der letzte Layer - der Output-Layer - des Netzes ist und $t_{p,i}$ die gewünschte Netzausgabe des i. Ausgabeneurons bei Eingabemuster p ist.

4 Lernverfahren: Backpropagation

Als Lernverfahren benutzen wir die populäre "Backpropagation" Methode. Sie ist eine Spezialisierung der allgemeinen Hebbschen Lernregel:

Wenn ein Neuron j eine Eingabe von Neuron i erhält und beide gleichzeitig stark aktiviert sind, dann erhöhe das Gewicht $w_{j,i}$.

Mathematisch gilt für ein Neuron i in der Schicht λ, und ein Neuron j in der Schicht $\lambda + 1$:

$$\Delta w_{j,i}^{\lambda+1,\lambda} = \eta y_i^{\lambda} net_j^{\lambda+1}$$

wobei $\Delta w_{j,i}^{\lambda+1,\lambda}$ die Änderung des Gewichtes $w_{j,i}^{\lambda+1,\lambda}$ ist. Weiter ist η eine gewählte Konstante, die der Lernrate des Gewichtes entspricht. y_i^{λ} und $net_j^{\lambda+1}$ sind wie oben definiert.

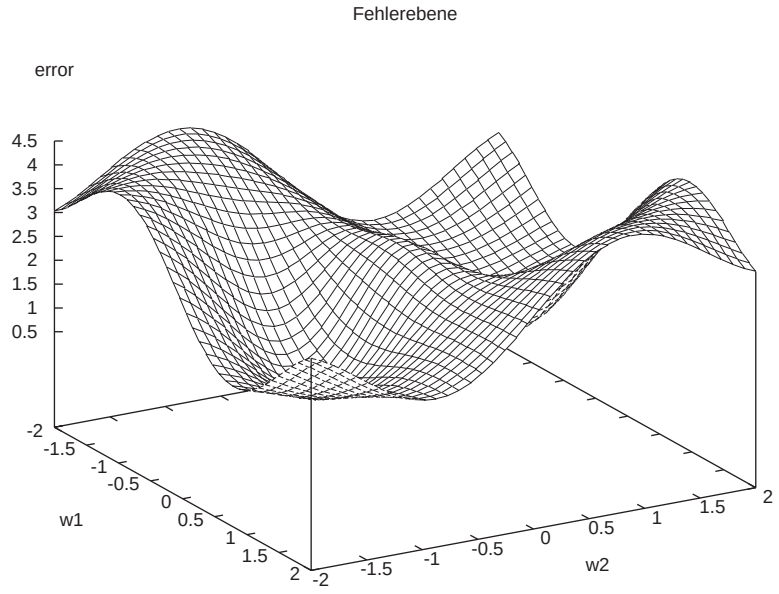

Abbildung 5: Beispiel: Fehlerebene für zwei Gewichte w_1 und w_2

Stellt man sich die Fehlerwerte auf eine Ebene der Dimension der Anzahl der Gewichte projiziert vor, so bildet eine Konfiguration des Netzes für mehrere Muster die Gesamtfehlerrate auf einen Punkt dieser Fehlerebene ab. Ein Beispiel für zwei Gewichte ist in Abbildung 5 gezeigt. Die Höhe eines Punktes auf dieser Fläche gibt Auskunft über den summierten Fehler des Netzes bei Eingabe aller Trainingsmuster. Im Spezialfall dass nur ein Muster propagiert wird, bildet es einen einzelnen Fehlerwert ab. Da die Ebene nicht global bekannt ist, können nur Einzelmessungen für spezielle Gewichtskonfigurationen vorgenommen werden.

Ziel des Lernverfahrens ist es, durch mehrere Messungen die lokalen und möglichst auch die globalen Täler der Ebene zu erreichen. Die Fehlerebene bezüglich des Eingabemusters p wird mit E_p bezeichnet.

4.1 Herleitung Backpropagation

Sei $\frac{\partial E_p}{\partial w_{i,j}^{\lambda,\lambda-1}}$ der Gradient bezüglich dem Gewicht $w_{i,j}^{\lambda,\lambda-1}$. Da wir die Täler der Fehlerebene erreichen wollen, muss die Gewichtsänderung in Richtung des negativen Gradienten vollzogen werden. Es ergibt sich allgemein für alle Muster p:

$$\Delta w_{i,j}^{\lambda,\lambda-1} = \sum_p -\eta \frac{\partial E_p}{\partial w_{i,j}^{\lambda,\lambda-1}}$$

Für den Faktor $\frac{\partial E_p}{\partial w_{i,j}^{\lambda,\lambda-1}}$ gilt nach der Kettenregel für Gradienten:

$$\frac{\partial E_p}{\partial w_{i,j}^{\lambda,\lambda-1}} = \underbrace{\frac{\partial E_p}{\partial net_{p,i}^{\lambda}}}_{1} \cdot \underbrace{\frac{\partial net_{p,i}^{\lambda}}{\partial w_{i,j}^{\lambda,\lambda-1}}}_{2}$$

Für den zweiten Faktor gilt:

$$\frac{\partial net_{p,i}^{\lambda}}{\partial w_{i,j}^{\lambda,\lambda-1}} = \frac{\partial}{\partial w_{i,j}^{\lambda,\lambda-1}} \sum_{j=1}^{dim(\lambda-1)} w_{i,j}^{\lambda,\lambda-1} \cdot y_{p,j}^{\lambda-1} = y_{p,j}^{\lambda-1}$$

Wir definieren das Fehlersignal δ wie folgt:

$$\delta_{p,i}^{\lambda} = -\frac{\partial E_p}{\partial net_{p,i}^{\lambda}}$$

Dann lässt sich die Gewichtsänderung zusammenfassen zu:

$$\Delta w_{i,j}^{\lambda,\lambda-1} = \eta \sum_p y_j^{\lambda-1} \cdot \delta_{p,i}^{\lambda}$$

mit

$$\delta_{p,i}^{\lambda} = -\frac{\partial E_p}{\partial net_{p,i}^{\lambda}} = \underbrace{-\frac{\partial E_p}{\partial y_{p,i}^{\lambda}}}_{1} \cdot \underbrace{\frac{\partial y_{p,i}^{\lambda}}{\partial net_{p,i}^{\lambda}}}_{2}$$

Für (2) ist:

$$\frac{\partial y_{p,i}^{\lambda}}{\partial net_{p,i}^{\lambda}} = \frac{\partial}{\partial net_{p,i}^{\lambda}} \cdot f_{act}(net_{p,i}^{\lambda}) = f'_{act}(net_{p,i}^{\lambda})$$

Für (1) mit $\delta_{p,i}^{\lambda}$ unterscheiden wir zwei Fälle:

- i ist Index eines Ausgabeneurons (Fall 1)
  Mit dem Erwartungswert der Ausgabe, $t_{p,i}$ lässt sich der Faktor ersetzen durch:

$$-\frac{\partial E_p}{\partial y_{p,i}^{\lambda}} = (t_{p,i} - y_{p,i}^{\lambda})$$

- i ist Index bezüglich einer verdeckten Ebene (Fall 2)

$$-\frac{\partial E_p}{\partial y_{p,i}^{\lambda}} = -\sum_{k=1}^{dim(\lambda+1)} \frac{\partial E_p}{\partial net_{p,k}^{\lambda+1}} \cdot \frac{\partial net_{p,k}^{\lambda+1}}{\partial y_{p,i}^{\lambda}} = \sum_{k=1}^{dim(\lambda+1)} \left(\delta_{p,k}^{\lambda+1} \cdot \frac{\partial}{\partial y_{p,i}^{\lambda}} \sum_{j=1}^{dim(\lambda)} w_{k,j}^{\lambda+1,\lambda} \cdot y_{p,j}^{\lambda} \right)$$

$$= \sum_{k=1}^{dim(\lambda+1)} \delta_{p,k}^{\lambda+1} \cdot w_{k,i}^{\lambda+1,\lambda}$$

  k ist der Laufindex für alle Neuronen der dahinterliegenden Schicht.

Damit ergeben sich die allgemeinen Lernregeln für das Backpropagation Verfahren:

Allgemeine Formeln der Backpropagation Lernmethode

- Für den Fall 1 (Ausgabeschicht):

$$\delta_{p,i}^{\lambda} = (-1) \cdot f'_{act}(net_{p,i}^{\lambda}, \Theta_i^{\lambda}) \cdot (t_{p,i} - y_{p,i}^{\lambda})$$

- Für den Fall 2 (verdeckte Schichten):

$$\delta_{p,i}^{\lambda} = f'_{act}(net_{p,i}^{\lambda}, \Theta_i^{\lambda}) \cdot \left(\sum_{k=1}^{dim(\lambda+1)} \delta_{p,k}^{\lambda+1} \cdot w_{k,i}^{\lambda+1,\lambda} \right)$$

Die Neuronen der Eingabeschicht werden von der Backpropagation Regel überhaupt nicht beeinflusst.

Dabei gilt für die Anwendung von δ auf das Gewicht w und den Schwellenwert Θ:

$$\Delta w_{i,j}^{\lambda,\lambda-1} = -\varepsilon \cdot \delta_{p,i}^{\lambda} \cdot y_{p,j}^{\lambda-1}$$

$$\Delta \Theta_i^{\lambda} = \varepsilon \cdot \delta_{p,i}^{\lambda}$$

Wobei ε ein gewählter Lernfaktor ist, der sich normalerweise auf den Bereich zwischen 10^{-3} und $0,5$ beschränkt.

4.2 Backpropagation mit f_{tanh}

Mit

$$f_{act}(net_{p,i}^{\lambda}, \Theta_i^{\lambda}) = tanh(net_{p,i}^{\lambda} - \Theta_i^{\lambda})$$

und

$$f'_{act}(net_{p,i}^{\lambda}, \Theta_i^{\lambda}) = tanh'(net_{p,i}^{\lambda} - \Theta_i^{\lambda}) = 1 - tanh^2(net_{p,i}^{\lambda} - \Theta_i^{\lambda})$$

gilt

- für den Fall 1 (Ausgabeschicht):

$$\delta_{p,i}^{\lambda} = (-1) \left(1 - (y_{p,i}^{\lambda})^2\right) (t_{p,i} - y_{p,i}^{\lambda})$$

- für den Fall 2 (verdeckte Schichten):

$$\delta_{p,i}^{\lambda} = \left(1 - (y_{p,i}^{\lambda})^2\right) \cdot \sum_{k=1}^{dim(\lambda+1)} \delta_{p,k}^{\lambda+1} \cdot w_{k,i}^{\lambda+1,\lambda}$$

4.3 Backpropagation mit f_{log}

Mit

$$f_{act}(net^{\lambda}_{p,i}, \Theta^{\lambda}_i) = \frac{1}{1 + e^{-(net^{\lambda}_{p,i} - \Theta^{\lambda}_i)}}$$

und

$$f'_{act}(net^{\lambda}_{p,i}, \Theta^{\lambda}_i) = \left(\frac{1}{1 + e^{-(net^{\lambda}_{p,i} - \Theta^{\lambda}_i)}}\right)' = \frac{e^{-(net^{\lambda}_{p,i} - \Theta^{\lambda}_i)}}{\left(e^{-(net^{\lambda}_{p,i} - \Theta^{\lambda}_i)} + 1\right)^2} = f_{act}(net^{\lambda}_{p,i}, \Theta^{\lambda}_i) \cdot \left(1 - f_{act}(net^{\lambda}_{p,i}, \Theta^{\lambda}_i)\right)$$

gilt

- für den Fall 1 (Ausgabeschicht):

$$\delta^{\lambda}_{p,i} = (-1) \cdot \frac{e^{-(net^{\lambda}_{p,i} - \Theta^{\lambda}_i)}}{\left(e^{-(net^{\lambda}_{p,i} - \Theta^{\lambda}_i)} + 1\right)^2} \cdot (t_{p,i} - y^{\lambda}_{p,i})$$

- für den Fall 2 (verdeckte Schichten):

$$\delta^{\lambda}_{p,i} = \frac{e^{-(net^{\lambda}_{p,i} - \Theta^{\lambda}_i)}}{\left(e^{-(net^{\lambda}_{p,i} - \Theta^{\lambda}_i)} + 1\right)^2} \cdot \left(\sum_{k=1}^{dim(\lambda+1)} \delta^{\lambda+1}_{p,k} \cdot w^{\lambda+1,\lambda}_{k,i}\right)$$

5 Konkrete Algorithmen

5.1 Propagate

```
foreach layer in mlpnet {
   foreach neuron in layer {
      if layer.isinput == true {
         neuron.output = mlpnet.input[neuron.index]
      } else {
         neuron.input = 0.0
         foreach precessorneuron of neuron {
            neuron.input += precessorneuron.weight[neuron] * precessorneuron.output
         }
         neuron.output = activationfunction (neuron.input, neuron.theta)
      }
   }
}
```

5.2 Backpropagate

```
foreach layer in reverse (mlpnet) {
   // skip backpropagation in case of input layer
   if layer.isinput == true
      continue
   foreach neuron in layer {
      if layer.isoutput == true {
         neuron.delta = -1 * activationfunctionderivate (neuron.input) *
            (mlpnet.optimaloutput[neuron.index] - neuron.output)
```

```
      } else {
         neuron.delta = 0.0
         foreach sucessorneuron of neuron {
            neuron.delta += neuron.weight[sucessorneuron] * sucessorneuron.delta
         }
         neuron.delta *= activationfunctionderivate (neuron.input)
      }
   }
}
```

5.3 Postprocessing

```
foreach layer in mlpnet {
   // ignore the input layer for any postprocessing step
   if layer.isinput == true
      continue
   foreach neuron in layer {
      neuron.thetadiff = epsilon * neuron.delta
      foreach sucessorneuron of neuron {
         foreach weight from neuron to sucessorneuron {
            neuron.weightdiff[sucessorneuron] =
               (-1) * epsilon * sucessorneuron.delta * neuron.output
         }
      }
   }
}
```

5.4 Update

```
foreach layer in mlpnet {
   // ignore the input layer for any update step
   if layer.isinput == true
      continue
   foreach neuron in layer {
      neuron.theta += neuron.thetadiff
      foreach sucessorneuron of neuron {
            neuron.weight[successorneuron] += neuron.weightdiff[sucessorneuron]
      }
   }
}
```

6 Erweiterung des Lernverfahrens

Einfache Backpropagation, wie etwa die oben beschriebene Variante, hat Probleme, wenn die Fehlerebene bestimmte Eigenschaften zeigt. Dies ist etwa bei flachen Plateaus und bei sehr steilen Stellen der Fall. Ausserdem ist eine Übertrainierung des Netzes möglich, wenn zu viele Trainingszyklen durchlaufen werden.

6.1 Momentum-Term

Zwei der oben genannten Probleme lassen sich lindern, indem man einen zusätzlichen sogenannten "Momentum-Term" benutzt. Dieser beeinflusst die Gewichtsänderung abhängig von der vorherigen Gewichtsänderung. Dazu benutzt man einen zusätzlichen Index für die Gewichtsänderung

Δw, der den Zeitpunkt der Änderung angibt. Die aktuelle Änderung Δw_{t+1} bezieht sich auf die jeweils vorherige Gewichtsänderung Δw_t. Für die Änderung eines Gewichtes (und Schwellenwertes) gilt dann die allgemeine Form:

$$\Delta w_{t+1,i,j}^{\lambda,\lambda-1} = -\varepsilon \cdot \delta_{p,i}^{\lambda} \cdot y_{p,j}^{\lambda-1} + \alpha \cdot \Delta w_{t,i,j}^{\lambda,\lambda-1}$$

$$\left(\Delta\Theta_{t+1,i}^{\lambda} = \varepsilon \cdot \delta_{p,i}^{\lambda} + \alpha \cdot \Delta\Theta_{t,i}^{\lambda}\right)$$

Dabei ist α ein Momentumfaktor, der üblicherweise zwischen 0.5 und 0.9 gewählt wird. Für die Schwellenwertänderung haben wir in der Literatur keine Hinweise gefunden, daher ist sie nur eine Vermutung.

Die Verwendung des Momentum-Terms bewirkt eine Beschleunigung in weiten Plateaus der Fehlerebene, also eine Erhöhung der Gewichtsänderung Δw. In sehr unebenen Teilen der Fehlerfläche bewirkt der Momentum-Term ein Abbremsen. Der zusätzliche Aufwand beschränkt sich auf die Speicherung jeweils einer zusätzlichen Gewichtsänderung.

6.2 Weight-Decay

Um einer Übertrainierung des Netzes vorzubeugen und eine bessere Generalisierung zu erreichen, kann man das Weight-Decay Verfahren anwenden. Dabei werden zu grosse Gewichte vermieden, da sie eine unebenere Fehlerfläche zur folge hätten, die das generische Lernen erschwert. Dazu ändert man die Fehlerfunktion ab, so dass grosse Gewichte zu einem grösseren Fehler führen:

$$E_{wd} = E + \frac{d}{2} \cdot \sum_{\lambda,i,j} \left(w_{j,i}^{\lambda+1,\lambda}\right)^2$$

Der erste Summand ist dabei die alte Fehlerfunktion E, der zweite und neue Summand summiert alle Quadrate der Gewichte des Netzes auf. Damit führen betragsmässig geringere Gewichte zu einem kleineren Gesamtfehler des Netzes.

Da die Fehlerfunktion zur Herleitung des Backpropagation Verfahrens verwendet wurde, muss dieses neu hergeleitet werden. Für die Änderung eines Gewichtes ergibt sich dann unter Beachtung der Zeitindexe t und $t+1$:

$$\Delta w_{t+1,i,j}^{\lambda,\lambda-1} = -\varepsilon \cdot \delta_{p,i}^{\lambda} \cdot y_{p,j}^{\lambda-1} - d \cdot w_{t,i,j}^{\lambda,\lambda-1}$$

Für die Schwellenwerte konnten wir keine analoge Funktion in der Literatur finden, da dort meistens mit "on"-Neuronen gearbeitet wird. Wir denken, die Änderungsfunktion muss folgendermassen lauten:

$$\left(\Delta\Theta_{t+1,i}^{\lambda} = \varepsilon \cdot \delta_{p,i}^{\lambda} - d \cdot \Theta_{t,i}^{\lambda}\right)$$

Der variable Faktor d ist klein zu wählen, da grosse Werte dazu führen, dass das Netz gar nicht trainiert werden, weil alle Gewichte zu klein bleiben. Über sinnvolle Werte des Faktors d sind in der Literatur unterschiedliche Angaben zu finden. Bei unseren Experimenten stellte sich der in [1] vorgeschlagene Wertebereich von 0.005 bis 0.03 als viel zu hoch heraus. Der Wertebereich von 0.00005 bis 0.0001 der in [4] vorgeschlagen wird erwies sich als deutlich besser und führte zu betragsmässig kleineren Schwellenwerten und Gewichten.

6.3 Toleranz der Optimaldifferenz

Eine recht einfache Methode um dem Übertrainieren des Netzes entgegen zu wirken ist das tolerieren kleiner Differenzen zwischen der Netzausgabe und der optimal erwarteten Ausgabe. Bleibt der Betrag der Differenz

$$(t_{p,i} - y_{p,i}^{\lambda})$$

unter einem bestimmten Grenzwert, so wird die Differenz auf Null gesetzt, und die Backpropagation bleibt wirkungslos. Damit bleiben alle Summanden, die aus diesem speziellen Ausgabeneuron folgen bei einem Wert von Null, und der δ Wert der davor liegenden Neuronen ist unabhängig von diesem Teil des Ausgabevektors. Häufig benutzte Werte liegen im Bereich von $0,0$ bis $0,2$. Durch eine Verwendung dieser Toleranzbedingung kann ein Übertrainieren teilweise verhindert werden [3].

Literatur

[1] Dr. Andreas Zell, "Simulation neuronaler Netze", ISBN 3-486-24350-0

[2] C++-Programmier-Praktikum, "Projektaufgabe Pj-NN"
`http://ni.cs.tu-berlin.de/lehre/C-Praktikum/data/Projekte/projekt_nn.pdf`

[3] Stuttgarter Neuronale Netze Simulator, Handbuch
`http://www-ra.informatik.uni-tuebingen.de/SNNS/UserManual/node52.html`

[4] Anders Krogh, John A. Hertz, "A Simple Weight Decay Can Improve Generalization"
`http://citeseer.nj.nec.com/krogh92simple.html`