Multi-Layer Neural Networks and Learning Algorithms

Alexander Perzylo

22. Dezember 2003

Ausarbeitung für das Hauptseminar Machine Learning (2003)
mit LaTeX gesetzt

Diese Ausarbeitung ist eine Weiterführung zum Thema „Neural Networks: Introduction and Single-Layer Networks“ und setzt somit Wissen über den allgemeinen Aufbau von künstlichen neuronalen Netzen und deren Funktionalität voraus.

Inhaltsverzeichnis

1 Einführung

1.1 Lernen in Neuronalen Netzen

Das Trainieren eines künstlichen Neuronalen Netzes kann verschiedene theoretische Ansätze haben. So ist es möglich, durch die Einführung neuer oder der Entfernung bereits bestehender Neuronen und Verbindungen ein Netz weiter an seine Aufgabenstellung anzupassen. Denkbar wäre auch die Aktivierungs-, Propagierungs-und die Ausgabefunktion der Neuronen zu verändern. In erster Linie hat sich aufgrund der praktischen Umsetzbarkeit und beachtlicher Erfolge in der Forschung sowie des großen Anwendungsgebietes besonders die Modifizierung der Verbindungsgewichtungen zwischen den Neuronen und ihrer Schwellenwerte als Lernverfahren durchgesetzt. Doch auch die Veränderung der Topologie durch sogenannte Genetische Algorithmen kann zusätzlich genutzt werden, ein neuronales Netz weiter zu verbessern.

1.2 Arten von Lernverfahren

Man unterscheidet im Allgemeinen drei Arten des Lernens in Neuronalen Netzen. Das überwachte (engl. supervised learning), unüberwachte (engl. unsupervised learning) und bestärkende (engl. reinforcment learning) Lernen. Beim überwachten Lernen wird zu jedem Eingabemuster das gewünschte Ausgabemuster vorgegeben. Dieser externe „Lehrer“ zwingt das Neuronale Netz, die Einstellung aller seiner Verbindungsgewichte so zu verändern, daß es möglichst genau die vorgegebenen Ausgangsvektoren für die entsprechenden Eingangsvektoren berechnen kann. Durch wiederholtes Repräsentieren vieler Paare von Eingabe- und Ausgabemustern kann eine gewisse Generalisierung erreicht werden, d.h. das Neuronale Netz assoziiert selbständig auf ähnliche aber nicht gelernte Eingabemuster mit einem entsprechenden (richtigen) Ausgabemuster. Diese Art, ein Neuronales Netzwerk zu trainieren, ist üblicherweise die schnellste Methode. Allerdings ist hierbei die Vorgabe der gewünschten Aktivierungen aller Ausgangsneuronen notwendig. Das bestärkende Lernen benötigt im Prinzip ebenfalls alle Paare der Eingabe- und Ausgabemuster. Hier teilt der „Lehrer“ diese dem Neuronalen Netz aber nicht mit. Nach jedem Trainingsdurchlauf wird nur angegeben, ob das Eingabemuster richtig oder falsch klassifiziert wurde. Das Netz hat nun die Aufgabe, selbst die optimale bzw. richtige Lösung zu finden. Diese Art des Lernens ist deutlich langsamer als überwachtes Lernen. Unüberwachtes Lernen benötigt überhaupt keinen externen Lehrer. Dem Netz werden nur Eingabemuster angeboten. Dieses versucht dann die Muster nach Gemeinsamkeiten oder Ähnlichkeiten zu klassifizieren, indem es Aktivierung gleicher oder räumlich benachbarter Neuronen identifiziert. Auf diese Weise extrahiert das Neuronale Netzwerk statistische Parameter aus den Eingabemustern, um mit Hilfe derer die Muster zu klassifizieren.

2 Multi-Layer Perceptron

2.1 Aufbau

Ein Multi-Layer Perceptron ist ein mehrschichtiges Feedforward Netz. Das bedeutet, dass alle Neuronen des Netzwerks in Schichten eingeteilt sind, wobei ein Neuron einer Schicht immer mit allen Neuronen der nächsten Schicht verbunden ist. Es gibt keine Verbindungen zur vorherigen Schicht und keine Verbindungen, die eine Schicht überspringen. Zudem sind die Verbindungen zwischen den Neuronen unidirektional und nicht zyklisch. Ein Multi-Layer Perceptron berechnet somit eine Funktion auf Eingabewerten, die von den Gewichtungen der Verbindungen abhängt. Es hat keinen internen Zustand, außer den Gewichtungen selbst.

2.2 Back-Propagation Learning

2.2.1 Beschreibung

Da Neuronale Netze aus der Natur abgeleitet wurden, stellt sich auch immer wieder die Frage nach der biologischen Plausibilität. Nach diesem Kriterium würde das unüberwachte Lernen die erste Wahl bedeuten. In technischen Problemstellungen sind aber Implementierbarkeit und Lerngeschwindigkeit ausschlaggebend, weshalb überwiegend überwachtes Lernen zur Anwendung kommt. Der bekannteste Vetreter dieser Lernmethode ist das Back-Propagation Learning. Die verschiedenen Phasen des Algorithmus lassen sich wie folgt beschreiben:

- Präsentation des Eingabemusters durch entsprechende Aktivierung der Eingabeneuronen (input units),
- Vorwärtspropagierung der angelegten Eingabe durch das Netz; Ausgabemuster für die aktuelle Eingabe erzeugen,
- Vergleich der Ausgabe mit der erwünschten Ausgabe (teaching input) liefert einen Fehlervektor (Differenz, Delta),
- Rückwärtspropagierung der Fehler von der Ausgabeschicht zur Eingabe führt zu Änderungen der Verbindungsgewichte, um den Fehlervektor zu verringern,
- Anpassung der Gewichte aller Neuronen entsprechend der berechneten Änderungen.

Dieser Vorgang wird für jedes Beispiel aus dem Trainingsset wiederholt. Das ganze Trainigsset wird dann erneut durchlaufen (Epoche) bis für die erzeugte Ausgabe des Netzes ein gewünschter Grad an Genauigkeit erreicht ist.

2.2.2 Mathematische Herleitung

Das Back-Propagation Learning basiert auf einem Gradientenabstiegsverfahren auf der Fehleroberfläche über der Menge aller Gewichtungen. Dabei werden die Gewichtungen im Neuronalen Netz so verändert, dass der Fehler der Ausgabeschicht minimiert wird. Um eine Minimierung des Fehlers zu erreichen, ermitteln wir den Gradienten der Fehlerfunktion (entspricht der Steigung) und verändern die Gewichtungen in die Richtung, für die der Ausgabefehler des Netzes in ein Tal der Fehlerfunktion absteigt.

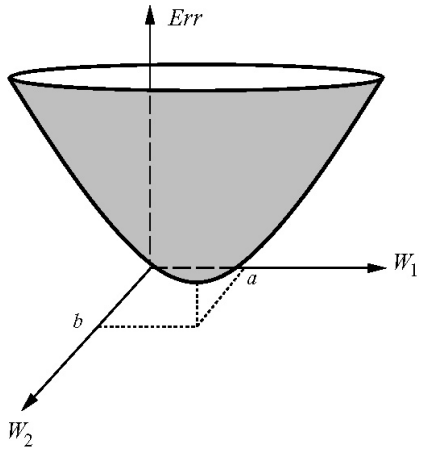


Bild1 Für Gewichte $W_1 = a$ und $W_2 = b$ ist der Fehler minimiert.

Nun wollen wir mathematisch herleiten, wie die Update-Regeln für die Gewichtungen in einem Multi-Layer Perceptron unter der Anwendung des Back-Propagation Algorithmus aussehen müssen. Wir beginnen mit der Fehlerfunktion selbst. Aufgrund der für die kommenden Berechnungen angenehmen Form, verwenden wir die Summe der quadratischen Fehler der Ausgabewerte:
Sei T die Zielausgabe (engl. target) und O die Aktivierung eines Ausgabeneurons (engl. output), dann gilt für alle Ausgabeneuronen i

$$E = \frac{1}{2}\sum_i (T_i - O_i)^2$$

Da die Aktivierung eines Neurons der Ausgabeschicht als Funktion von Gewichtungen der darüberliegenden Schichten geschrieben werden kann, gilt für die Fehlerfunktion eines allgemeinen 2 Schichten Netzwerks (mit einer verdeckten Schicht):
Sei W die Gewichtung zwischen zwei Neuronen, a die Aktivierung (output) eines Neurons und I die Eingabe (input) in ein Neuron der Eingabeschicht, dann gilt für alle Ausgabeneuronen i, verdeckten Neuronen j und Eingabeneuronen k

$$\begin{aligned} E(W) &= \frac{1}{2}\sum_i (T_i - g(\sum_j W_{j,i} a_j))^2 && (1) \\ &= \frac{1}{2}\sum_i (T_i - g(\sum_j W_{j,i} g(\sum_k W_{k,j} I_k)))^2 && (2) \end{aligned}$$

Um den Gradienten der Fehlerfunktion in Bezug auf die Gewichtungen von der verdeckten Schicht zur Ausgabeschicht zu erhalten, leiten wir die Fehlerfunktion aus (1) nach $W_{j,i}$ ab:

$$\begin{aligned} \frac{\partial E}{\partial W_{j,i}} &= -a_j(T_i - O_i)g'(\sum_j W_{j,i}a_j) \\ &= -a_j(T_i - O_i)g'(in_i) \\ &= -a_j\Delta_i \end{aligned} \tag{3}$$

Die Ableitung der Fehlerfunktion (2) mit Bezug auf die Gewichtungen der Eingabeneuronen ist ein wenig komplexer, bringt aber ein ähnliches Ergebnis. Ein Differenzieren nach $W_{k,j}$ liefert:

$$\frac{\partial E}{\partial W_{k,j}} = \cdots = -I_k\Delta_j \tag{4}$$

Damit wir nun die Update-Regeln für die Gewichtungen erhalten, müssen wir uns vor Augen halten, dass das Ziel die Minimierung des Fehlers ist. Demnach müssen wir die Gewichtungen ein Stück weit in die entgegengesetzte Richtung des Gradienten verändern. Diese Änderung wird durch die Lernrate (Wert zwischen 0 und 1) skaliert:
Sei α die Lernrate des Neuronalen Netzes

Aus (3) folgt für die Gewichtungen der Neuronen zur Ausgabeschicht

$$\boxed{W_{j,i} \leftarrow W_{j,i} + \alpha \cdot a_j \cdot \Delta_i}$$

$$\Delta_i = Err_i \cdot g'(in_i)$$

Aus (4) folgt analog für die Gewichtungen der Neuronen der Eingabeschicht

$$\boxed{W_{k,j} \leftarrow W_{k,j} + \alpha \cdot I_k \cdot \Delta_j}$$

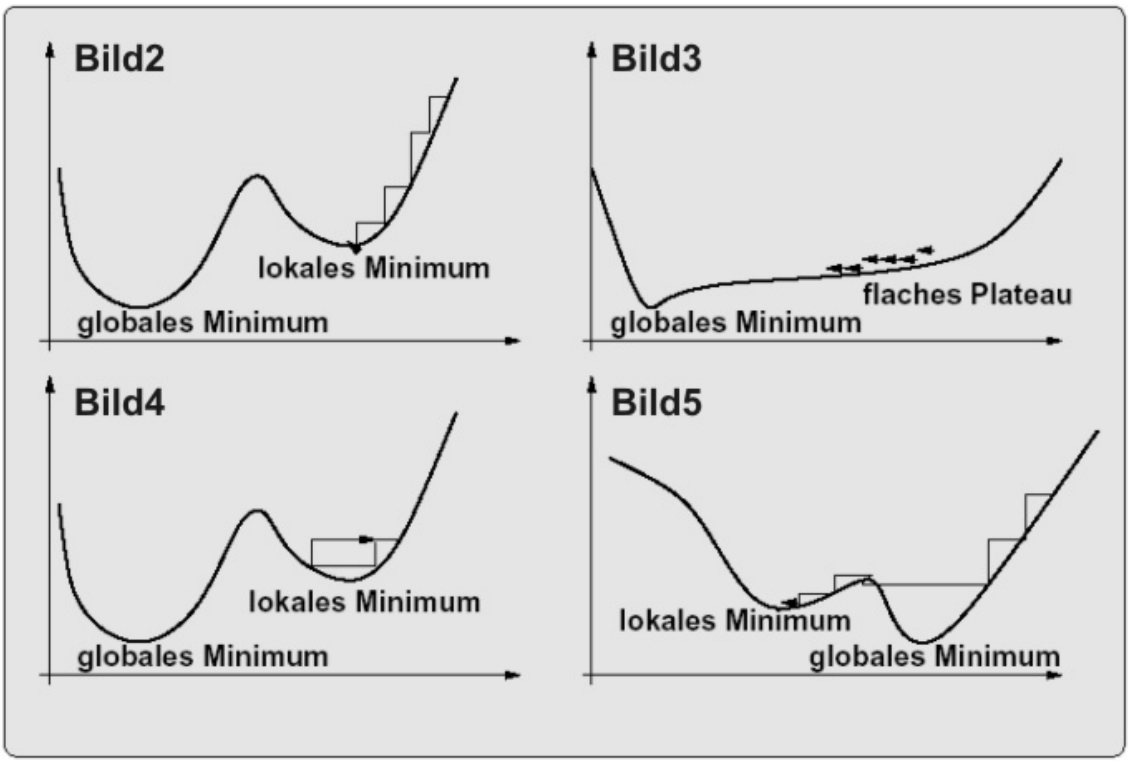
$$\Delta_j = g'(in_j)\sum_i W_{j,i}\Delta_i$$

Anmerkung: Da für die Berechnung des Gradienten die Ableitung der Aktivierungsfunktion benötigt wird, und daher die sign- und step-Funktionen dafür nicht verwendet werden können, wird meist die sigmoid Funktion benutzt. Ihr Vorteil ist die einfache Berechnung der Ableitung:

$$\frac{\partial sigmoid}{\partial x} = sigmoid(1 - sigmoid(x))$$

2.2.3 Konvergenzprobleme

Wie jedes andere Gradientenabstiegsverfahren hat auch der Back-Propagation Algorithmus Probleme in Hinblick auf Effizienz und Konvergenz. Da beim Gradientenabstieg nur lokale Informationen berücksichtigt werden, kann er lediglich lokale Minima aufspüren (*siehe Bild2*). Außerdem kommt es auf flachen Plateaus der Fehlerfunktion zu einer verlangsamten Konvergenz. Da in solchen Fällen die Steigung nahe 0 ist, werden auch die Gewichtungen nur in sehr kleinen Schritten angepasst, was die Lerndauer erheblich verlängern kann (*siehe Bild3*). In engen Schluchten der Fehlerfunktion, kann es zu Oszillationen kommen, bei denen die Gewichtungen so verändert werden, dass der Fehler von der einen auf die andere Seitenwand der Schlucht springt und nicht zum Minimum absteigt (*siehe Bild4*). Desweiteren können globale Minima durch eine zu groß gewählte Lernrate wieder verlassen werden (*siehe Bild5*).



2.3 Anwendungsbeispiele

2.3.1 Text vorlesen (NETtalk,1987)

Das NETtalk Programm (Sejnowski und Rosenberg) ist ein Neuronales Netz, das lernt geschriebenen englischen Text vorzulesen. Die Eingabe besteht aus dem auszusprechenden Buchstaben, sowie dessen Vorgänger und drei Nachfolgern. Das Netz, für das Resultate vorliegen, besteht aus 80 verdeckten Neuronen und einer Ausgabeschicht, die die Merkmale des akustischen Signals erzeugt (hoher Ton, tiefer Ton, betont, unbetont, ...). Abhängig vom untersuchten Wort, benötigt man manchmal zwei oder mehr Buchstaben um eine Ausgabe zu generieren. In diesem Fall ist die korrekte Ausgabe für den zweiten Buchstaben nichts. Nach 50 Trainingsdurchläufen mit einer Trainingsmenge von 1024 Wörtern, arbeitete das Programm auf dieser zu 95 Prozent korrekt. Eine Testmenge wurde zu 78 Prozent richtig erkannt. Nun mag dieses Ergebnis nicht besonders gut erscheinen, dies liegt daran, dass nur ein kleiner Kontext für das zu erzeugende Phonem berücksichtigt wurde.

2.3.2 Erkennung handschriftlich verfasster Zahlen (1989)

Le Cun hat ein Neuronales Netzwerk implementiert, das darauf ausgelegt ist Postleitzahlen von handbeschriebenen Briefumschlägen zu erkennen, um eine automatische Sortierung zu ermöglichen. Als Eingabe dient ein 16x16 Pixel-Array. Das Netz besteht aus 3 verdeckten Schichten mit 768, 192 bzw. 30 Neuronen. Ein vollständig verbundenes Netz dieser Größe hätte 200000 Verbindungsgewichtungen und wäre nur schwer zu trainieren. Deshalb wurden die Neuronen der ersten verdeckten Schicht in 12 Gruppen zu jeweils 64 Neuronen eingeteilt, die jeweils einen 5x5 Pixel großen Teilausschnitt der Eingabe untersuchten. Damit konnte die Gesamtzahl der Verbindungen auf 9760 reduziert werden. Nach einem Training auf 7300 Beispielen, wurde die Testmenge zu 99 Prozent korrekt verarbeitet.

2.3.3 Auto fahren (ALVINN,1993)

Das Neuronale Netz ALVINN (Autonomous Land Vehicle In a Neural Network) hat gelernt entlang einer Spur auf Autobahnen zu fahren, indem es vorher einen Menschen bei gleicher Tätigkeit beobachtet hat. Eine Fülle von Sensoren, wie z.B. eine Videokamera und ein Radar, werden ausgewertet und in ein 30x32 Pixel-Array umgewandelt, welches dem Netz als Eingabe dient. Desweiteren besteht das Netz noch aus genau einer verdeckten Schicht mit 5 Neuronen und einer Ausgabeschicht von 30 Neuronen. Dabei steht jedes Ausgabeneuron für einen bestimmten Lenkwinkel, wobei das Neuron mit höchster Aktivierung die Richtung in die gelenkt werden soll angibt. Nach einer Trainingsdauer von 5 min und einem Lernvorgang mittels Back-Propagation von 10 min, ist ALVINN bereit selbstständig zu fahren. Dies hat das Programm schon eindrucksvoll unter Beweis gestellt, indem es Strecken von bis zu 150km mit Geschwindigkeiten bis zu 120km/h zurückgelegt hat.

3 Hopfield-Netz

3.1 Aufbau

In einem Hopfield Netz ist jedes Neuron mit allen anderen Neuronen, außer mit sich selbst, verbunden. Das ganze Netz besteht aus einer einzelnen Schicht, die sowohl als Eingabe- wie auch als Ausgabeschicht fungiert. Die Ein- bzw. Ausgaben sind bipolar kodiert und die Verbindungsgewichte zwischen den Neuronen sind symmetrisch ($W_{j,i} = W_{i,j}$). In Hopfield Netzen findet die ursprüngliche Hebb'sche Lernregel Anwendung.

3.2 Hebb'sche Lernregel

Die von dem Psychologen Donald O. Hebb 1949 aufgestellte Lernregel, mit welcher er versuchte, die Lernfähigeit des Gehirns zu erklären, besagt: Das Gewicht von einem Neuron i zu einem anderen Neuron j wird genau dann erhöht , wenn j eine Eingabe von i erhält und beide Neuronen gleichzeitig stark aktiv sind. Mathematisch lautet diese Regel:
Sei α die Lernrate, W das Gewicht zwischen zwei Neuronen und a die Aktivierung eines Neurons, dann gilt für zwei Neuronen i und j

$$\Delta W_{j,i} = \alpha \cdot a_i \cdot a_j$$

Die Lernrate skaliert die Stärke der Gewichtsänderung in Abhängigkeit der Ausgaben der zwei verbundenen Neuronen.
Die Hebb'sche Lernregel findet hauptsächlich bei binären Aktivierungswerten Anwendung. Hierbei muss beachtet werden, dass im Falle der Kodierung mit {0,1} nur positive Gewichtsänderungen möglich sind. Dies ist natürlich nicht wünschenswert. Daher benutzt man die Kodierung {-1,1}, da in diesem Fall die Hebb'sche Lernregel eine Verringerung der Gewichte liefert, falls die Ausgabe des Vorgängerneurons nicht mit der Ausgabe des Nachfolgerneurons übereinstimmt.

3.3 Lernvorgang

Das Hopfield Netz dient der Mustervervollständigung bzw. Mustererkennung. Um jedoch in der Lage zu sein, bestimmte Muster zu erkennen, muss man sie dem Netz zunächst präsentieren. Das Netz „merkt“ sich diese und ist dann in der Lage neue Eingaben anhand der abgespeicherten Informationen zu klassifizieren.

- „Merkvorgang“: Dem Netz wird ein Muster eingegeben. Nun erfolgt für alle vorhandenen Gewichte des Netzes eine Modifikation. Dabei geht man wie folgt vor:
  Die Gewichte zwischen zwei Neuronen gleicher Aktivität (beide -1 bzw. beide 1) werden um die Lernrate erhöht, da:

$$\Delta W_{j,i} = \alpha \cdot (-1) \cdot (-1) = \alpha \cdot 1 \cdot 1 = \alpha$$

Bei unterschiedlicher Aktivität (-1 und 1), wird das Gewicht zwischen den betroffenen Neuronen um die Lernrate verringert, da:

$$\Delta W_{j,i} = \alpha \cdot (-1) \cdot 1 = \alpha \cdot 1 \cdot (-1) = -\alpha$$

Dieser Ablauf wird für jedes zu erlernende Muster wiederholt, dabei ist zu beachten, dass die Anzahl erlernbarer Muster nach oben beschränkt ist und von der Musterbeschaffenheit und der Größe des Hopfield Netzes abhängt.

- Vorgang der Musterkennung: Dem Netz wird eine unbekannte Eingabe präsentiert. Der Nachfolgezustand des Netzes berechnet sich nach folgendem Schema:
  Für jedes Neuron im Hopfield Netz summieren wir alle eingehenden Gewichte auf, wobei wir bei einem gegenüberliegenden aktiven Neuron (Ausgabe 1) addieren und bei einem gegenüberliegenden passiven Neuron (Ausgabe -1) subtrahieren. Ergibt sich für diese Summe ein Wert größer gleich 0, so ist das untersuchte Neuron aktiv, ansonsten passiv. Der geschilderte Ablauf wird solange wiederholt, bis sich keine Änderung mehr ergibt. Das Ergebnis des Netzes entspricht nun einem vorher gespeicherten Muster, welches der unbekannten Eingabe am ähnlichsten ist, womit eine Klassifizierung stattgefunden hat.

4 Kohonen Self-Organizing-Map (SOM)

4.1 Aufbau

Eine Kohonen Self-Organizing-Map besteht aus einer Eingabeschicht und einer sogenannten Kohonenschicht (Kartenraum). Jedes Neuron der Kohonenschicht ist mit allen Eingabeneuronen, sowie den anderen Neuronen der Kohonenschicht verbunden. Die Gewichte der Verbindungen sind, wie bei nahezu allen Neuronalen Netzen, zu Beginn zufällig ausgewählt.

4.2 Lernvorgang

SOM können ausgenutzt werden, um eine topologieerhaltende Verteilung von Objekten durchzuführen. Ausgangspunkt ist ein Eingangsraum mit Reizen, sowie eine neuronale Schicht(Kohonen-Schicht). Die Neuronen besitzen durch ihre Gewichte eine Position im Eingaberaum, welche sich iterativ den Reizen anpasst. Wichtiges Merkmal dieses Ansatzes ist die Vernetzung: ein Neuron spricht auf einen Reiz an und ändert darauf seine Gewichte, d.h. seine Position, in Richtung auf den Reiz. Allerdings bewirkt dies nun auch eine Änderung der Gewichte seiner Nachbarn, die sich ebenfalls auf den Reiz zubewegen. Dies hat letztendlich den Effekt, dass sich an räumlichen Positionen mit vielen Reizen auch entsprechend viele Neuronen anlagern. Da für diesen Vorgang keine externe Steuerung notwendig ist, spricht man hierbei von einem unüberwachten Lernverfahren. Der Lernalgorithmus läßt sich wie folgt beschreiben:

- Stimuluswahl: Aus der Menge der Reize im Eingaberaum wird ein Reiz zufällig ausgewählt und dem SOM präsentiert
- Gewinnerneuron: Es wird das Neuron bestimmt, dessen Gewichtsvektor W dem Eingabevektor I (Reiz) am ähnlichsten ist. Ähnlichkeit wird hierbei als räumliche Nähe definiert. Diese kann durch die Berechnung der euklidischen Distanz ermittelt werden:
$$\sqrt{(I - W_i)^2}$$
- Anschließend werden die Gewichte des Neurons, sowie der Neuronen in seiner Nachbarschaft angepasst, so dass die Gewichtsvektoren dem Eingabevektor ähnlicher werden. Der Grad der Anpassung wird durch die Lernrate α skaliert:
$$\Delta W_i = \alpha(I - W_i)$$
- Die Lernschritte werden iterativ wiederholt, bis ein Abbruchkriterium erfüllt ist:
$$|\Delta W < \varepsilon|$$

Die Größe der Nachbarschaft und die Lernrate nehmen im Laufe des Trainings ab, um nach einer anfänglich groben Umordnung eine Konvergenz des Netzes zu ermöglichen.

5 Fazit - Lernen in Neuronalen Netzen

5.1 Vorzüge

Gründe, die für die Verwendung Neuronaler Netze, also der Simulation biologischer Neuronaler Netze sprechen, gibt es genug:

- Lernen statt programmieren: Da bei vielen Problemstellungen keine mathematischen Verfahren oder Funktionen bekannt sind, ist es bisher nicht möglich solche Problemstellungen einem Computer auf „herkömmliche“ Weise beizubringen. Um dies zu können, müsste man das Problem in eine mathematische Aussage fassen können.
- Adaptierfähigkeit: Mit einer neuen Klasse von Problemstellungen kann ein Neuronales Netz sein Können erweitern.
- Fehlertoleranz: Bei geeigneter Netztopologie kann ein Neuronales Netz eine gewisse Fehlertoleranz gegenüber dem Ausfall von Neuronen erreichen.
- Massiver Parallelismus: Neuronale Netze sind massiv parallel, d.h. die Verarbeitungsschritte können verteilt und gleichzeitig ausgeführt werden.

- Generalisierungsfähigkeit: Nachdem ein Neuronales Netz mit einer Anzahl von Trainingsmustern trainiert worden ist, kann es auch völlig neue Trainingsmuster klassifizieren.

- Robustheit bzgl. sogenannter „verrauschter Daten“: Neuronale Netze sind in der Lage, nachdem sie trainiert worden sind, auch „verrauschte Daten“, also z.B. Daten mit Messfehlern, richtig zu klassifizieren.

5.2 Probleme

Es gibt natürlich auch einige Punkte, die gegen die Verwendung von Künstlichen Neuronalen Netzen sprechen:

- Keine Introspektion möglich: Neuronale Netze haben nicht die Fähigkeit sich selbst zu analysieren, d.h. herauszufinden, ob sie eine Aufgabe bereits beherrschen bzw. wie viel sie von einer Aufgabe beherrschen oder nicht.

- Lernen ist relativ langsam: Bis ein Neuronales Netz einen komplexeren Sachverhalt bis zu einem annehmbaren Grad gelernt hat, sind relativ viele Trainingsdaten notwendig, die das Netz verarbeiten muss. Diese Trainingsdauer kann sehr viel (Rechen-)Zeit in Anspruch nehmen.

- Blackbox-Problem: Die Ergebnisse eines Neuronalen Netzes sind schwierig zu analysieren bzw. zu bewerten, da man nicht nachvollziehen kann, wie das Ergebnis im Einzelnen zu Stande kommt.

- Optimierung sehr aufwendig: Da man bei Neuronalen Netzen nicht nur die Gewichte optimieren kann, sondern auch die Topologie, wird schnell deutlich, dass dadurch für jede in Frage kommende Topologie alle Gewichte von neuem optimiert werden müssen. Dies ist allerdings sehr aufwendig.

- keine Garantie für (optimale) Lösung: Ein Neuronales Netz kann unter Umständen überhaupt nicht gegen eine Lösung konvergieren oder nur gegen eine lokal beste Lösung.

- Gefahr des Überlernens (engl. overfitting): Wird ein Neuronales Netz zu stark auf bestimmte Trainingsmuster getrimmt, nimmt die Generalisierungsfähigkeit zunehmend ab

Literatur

[1] Bishop: *Neural Networks for Pattern Classification*
Oxford University Press (1996), ISBN 0-1985-3864-2 .

[2] Stuart J. Russell, Peter Norvig: *Artificial Intelligence: A Modern Approach*
Prentice Hall (2002), ISBN 0-1379-0395-2 .

[3] Kristof Van Laerhoven: *Real-time Analysis of Data from Many Sensors with Neural Networks*
(2001) <http://citeseer.nj.nec.com/vanlaerhoven01realtime.html>

[4] Maintainer:Warren S. Sarle: *Neural Networks FAQ*
(2002) <ftp://ftp.sas.com/pub/neural/FAQ.html>

[5] Usenet Newsgroup: *comp.ai.neural-nets*

[6] Walter Schmidt, Jörg Knappen, Hubert Partl, Irene Hyna:
LaTeX2e-Kurzbeschreibung
(2003) <ftp://dante.ctan.org/tex-archive/info/lshort/german/>