Selbstorganisierende Karten

Yacin Bessas
yb1@informatik.uni-ulm.de
Proseminar Neuronale Netze

1 Einleitung

1.1 Kurzüberblick

Die Selbstorganisierenden Karten, auch Self-Organizing (Feature) Maps, Kohonen-Karten oder Kohonen Feature Maps (benannt nach ihrem Erfinder Teuvo Kohonen) genannt, sind ein unüberwachtes Lernverfahren der Neuroinformatik. Ein unüberwachtes Lernverfahren bezeichnet maschinelles Lernen ohne vorher bekannte Ergebniswerte. Ziel dieses Lernverfahrens ist es, eine topologische Darstellung und Clusterbildung zu erreichen.

1.2 Vorbemerkung

Selbstorganisierende Karten (im Folgenden SOM genannt) gibt es auch im Gehirn. In Abb.1 ist eine Skizze der topologischen Abbildung des sensorischen Kortex zu sehen, der die sensorische Eingaben des Körpers auswertet. Dies ergibt eine Klassifizierung der Ausgaben, da Ähnliche in direkter Umgebung angesiedelt werden. Beispielsweise sind die Ausgaben der Signale an die Finger in Nachbarschaft zu einander. Hier werden Gewichtsvektoren durch Funktionen so verändert, das sich Gewichtsvektoren ähnlicher Ausgabewerte gegenseitig annähern. Solch eine Nachbarschaftsbeziehung zwischen Neuronen ist ein grundlegendes Merkmal von SOM.

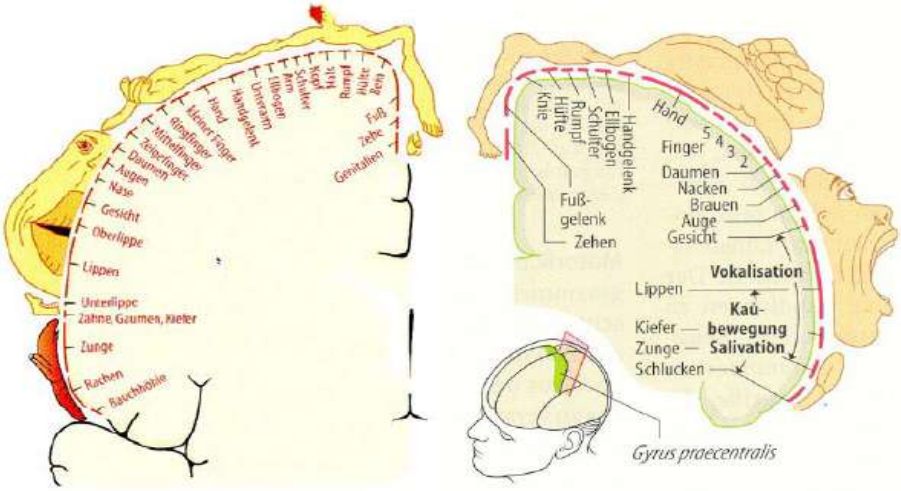

Abb.1: Der sensorische Kortex des Gehirns [4]

2 Selbstorganisierende Karten

2.1 Grundaufbau der selbstorganisierenden Karten

Selbstorganisierende Karten setzen sich zusammen aus Eingabe- und Ausgabeneuronen, die jeweils eine Schicht bilden, genannt die Eingabe- oder Ausgabeschicht. Die Ausgabeneuronen werden auch als Kohonen-Neuronen bezeichnet. Die Eingabeneuronen sind über Gewichtsvektoren komplett mit den Kohonen-Neuronen verbunden (angedeutet in Abb.2). Die Eingabeschicht besteht aus der Menge der Eingabevektoren X =(x{1}, x{2},..., x{i}), wobei i die Anzahl der Eingabevektoren ist. Die Kohonenschicht wird in Abb.2 als (quadratisches) Gitter dargestellt, wobei die Verbindungen zwischen den Ausgabeneuronen als Nachbarschaftsbeziehung zu verstehen sind. Dies kann kann man wie ein Netz sehen. Wenn man die Position eines Knotens verändert, beeinflusst das auch die Position der umliegenden Knoten. Die Kohonenschicht kann beliebig dimensional sein, am häufigsten werden hier zwei-dimensionale quadratische Gitter benutzt, da sie am einfachsten zu implementieren sind. Allerdings sind auch andere Formen möglich und manchmal auch sinnvoller je nach Problemstellung. Nachdem die Dimension und Form der Ausgabeschicht gewählt wurde, wird sie im Laufe des Lernverfahrens nicht mehr geändert. Die beiden Schichten sind durch Gewichtsvektoren $W_j = (w_{1,j}, w_{2,j}, ..., w_{i,j})$ verbunden, wobei j der Knoten aus der Kohonenschicht ist, mit dem der Gewichtsvektor verbunden ist.

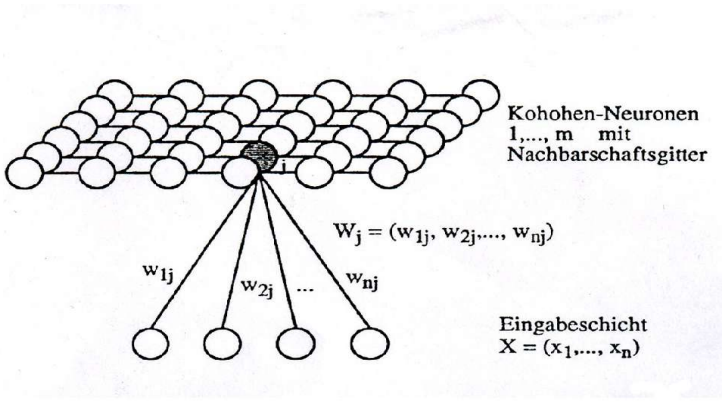

Abb.2: Visualisierung von Verbindung zwischen Eingabe- und Kohonenschicht [1]

2.2 Lernverfahren der selbsorganisierenden Karten

Die selbstorganisierenden Karten lernen dadurch, dass bei einer Eingabe, ein Siegerneuron bestimmt wird. Die Bestimmung des Siegerneurons geschieht mittels der euklidischen Norm. Hiermit wird der Abstand zwischen Eingabevektor und Gewichtsvektor bestimmt:

Euklidische Norm: $|| X - W_c || = \min_j (|| X - W_j||)$

Alternativ kann zur Bestimmung des Index c des Siegerneurons auch diese Funktion benutzt werden:

$c = \arg \min_j (|| X - W_j ||)$

wobei die Funktion arg zur Indexbestimmung des Minimums dient und hier das Siegerneuron zurück gibt.

Bei normalisierten Vektoren kann auch das Skalarprodukt benutzt werden. Bei dieser Vorgehensweise gewinnt das Neuron mit dem größten Skalarprodukt. Es kann natürlich vorkommen, dass 2 oder mehr Neuronen die gleiche euklidische Norm besitzen, hier wird per Zufall ein Siegerneuron aus der Gruppe der Neuronen bestimmt, die als Siegerneuronen in Frage kommen. Besonders an SOM ist, dass nicht nur der Gewichtsvektor des Siegerneurons verändert wird, sondern auch die Gewichtsvektoren der Neuronen in der Umgebung des Siegerneurons. Es kommt nicht darauf an, dass die Neuronen in der Nachbarschaft der Eingabe auch besonders ähnlich sind, sondern nur auf die direkte topologische Umgebung bzw. Entfernung.
Die Stärke der Veränderung der Verbindungsgewichte wird mittels dieser Funktion bestimmt:

Trainingsfunktion: $W_j(t+1) = W_j(t) + \eta(t) * h_{cj}(t) * [X(t) – W_j(t)]$

Die Operation * gilt in dieser Funktion als Multiplikation.
Die Trainingsfunktion richtet sich zum einen nach einer Nachbarschaftsfunktion (neighborhood kernel), und zum anderen nach einer zeitlichen Funktion $\eta(t)$. Diese wird dazu benutzt, um mit Fortschreiten des Lernens den Radius der Einflussnahme auf Nachbarneuronen zu reduzieren und so den Lernalgorithmus zu stabilisieren. Anfangs wird meist das komplette Netz der Neuronen verändert. Mit Fortschreiten des Lernens entsteht eine gewisse Spezialisierung. Deswegen ist es nicht mehr nötig den Radius des Lernalgorithmus so groß zu wählen, dass dieser Neuronen auf der Kompletten Karte beeinflusst. Aus diesem Grund wird $\eta(t)$ mit $0 < \eta(t) < 1$ mit der Nachbarschaftsfunktion $h_{cj}(t)$ multipliziert um die Distanz mit der Zeit zu verringern. Dies wird getan um zu vermeiden, dass die Karte aus unbekannten Gründen irgendwann nicht mehr konvergiert. Die Nachbarschaftsfunktion

$$h\{cj\}(t) = h(\| r\{c\} – r\{j\} \|, t) = h(z , t) = h'(z, d)$$

hängt von der Entfernung z der Neuronen c sowie j ab(in der Nachbarschaftsfunktion $z = \| r_c – r_j \|$) und von der zeitlichen Komponente t. Statt der t wird auch oft die Distanz d benutzt. Hierzu wird eine andere Funktion h'(z,d) benutzt um auf das gleiche Ergebnis von h(z,t) zu kommen. Wichtig dabei ist, je größer t der Funktion h(z,t) wird, desto mehr nähert sich d der Funktion h'(z,d) 0 an um den gleichen Effekt zu erzeugen. Je größer der Zeitraum des Lernens ist, desto geringer wird die maximale Distanz zu Nachbarneuronen die mit verändert werden. Zur Verdeutlichung der Auswirkung einer solchen Nachbarschaftsbeziehung dient Abb.3.

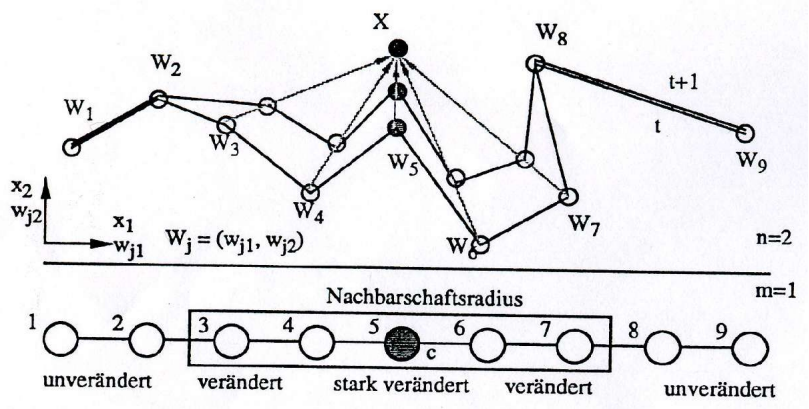

Abb.3: Verdeutlichung der Veränderungen der Neuronen durch Nachbarschaftsbeziehung
m = Dimension der Eingabeschicht; n = Dimension der Ausgabeschicht, t = Zeitliche Komponente [1]

Meistens werden schon bewährte Distanzfunktionen h(z,d) benutzt. Diese haben oft gemeinsam, dass die Werte dieser Funktionen im Intervall [0,1] verlaufen.
Zwei Beispiele für bewährte Funktionen sind z.B.:

$h(gauss)_{z,d} = e\ exp(-(z/d)^2)$
$h(cylinder)_{z,d} = 1$ falls $z<d$ oder 0 sonst.

Es gibt aber auch Distanzfunktionen die ungeeignet für SOM sind, als Beispiel hierzu ist die Mexican-Hat-Funktion zu nennen. Diese ist definiert als die 2. Ableitung der normalisierten Gaussfunktion. Da ein Teil dieser Nachbarschaftsfunktion negativ ist, werden Neuronen in einem gewissen Radius nicht mehr angezogen sondern abgestoßen. Abb.4 zeigt eine Zusammenschau einiger Nachbarschaftsfunktionen.

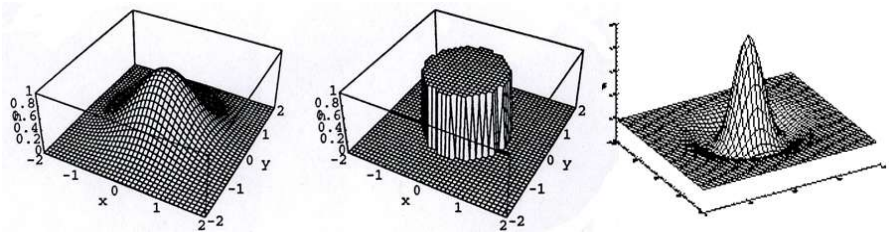

Abb.4: Nachbarschaftsfunktionen : Gausssche Glockenfunktion [1], Zylinderfunktion [1] und Mexican-Hat [7]

Es könnte zwar angenommen werden, dass dies eine bessere Auftrennung geben würde, aber es wäre nun möglich, dass ein Siegerneuron bei der nächsten Eingabe wieder abgestoßen wird und sich damit die vorherige Eingabe größtenteils egalisiert. Außerdem steigert sich so die Wahrscheinlichkeit, dass die Ausgabeschicht nicht konvergiert. Darauf wird später näher eingegangen. Nicht nur ungeeignete Nachbarschaftsfunktionen können für einen topologischen Defekt verantwortlich bei SOM sein. Hierbei ist zu beachten, dass die Distanzkomponente d richtig gewählt werden muss. Falls d zu Beginn des Lernverfahrens zu niedrig gewählt wurde oder zu schnell verringert wird, kann es zu Defekten führen.
In Abb.5 ist eine SOM dargestellt, die sich topoligisch korrekt entfaltet und ausserdem eine topologisch nicht korrekt entwickelte Karte, die unter Umständen durch weitere Eingaben nicht wieder korrigiert werden kann. An dieser Karte mit Defekt sieht man, dass das Ergebnis zumindest Teilweise korrekt ist, aber dadurch, dass nicht genug Veränderungen an Nachbarneuronen vorgenommen wurde, wurde hier der Mittelteil falsch angeordnet. Ein topologischer Defekt kann sich an beliebiger Stelle einer SOM befinden.Am Ende, wenn das Lernen ohne Fehlfunktionen vollendet wurde, sollte das Ergebnis ein vollständiges aufgespanntes Netz sein, ähnlich Abb.5 (3. Bild von links). Um so ein zufriedenstellendes Ergebnis zu erreichen benötigt es oft mehrere tausend Eingaben.

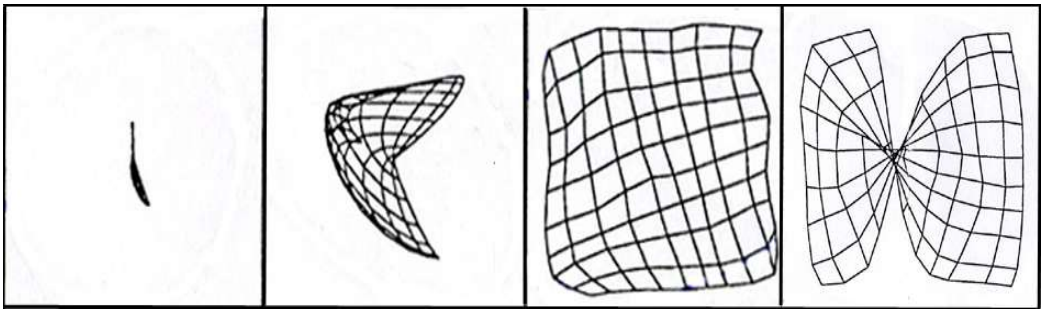

Abb.5: von links nach rechts: Entfaltung einer SOM: nach 10, 100 und 3000 Eingaben [1] . Eine Kohonenkarte mit topologischem Defekt [1]

2.3 Anwendungsbeispiel

Gegeben sei eine SOM mit 4 Eingabevektoren, mit Dimension 3:

[0.8 0.7 0.4] , [0.6 0.9 0.9], [0.3 0.4 0.1], [0.1 0.1 0.3]

Gewichtsvektoren seien:

$$\begin{bmatrix} 0.5 & 0.4 \\ 0.6 & 0.2 \\ 0.8 & 0.5 \end{bmatrix}$$

Die Lernrate η(t) sei 0.5 und der Radius der Radius der Nachbarschaftsfunktion sei 0, d.h. nur das Siegerneuron selbst wird verändert.
Nun vergleicht man den ersten Eingabevektor mit den beiden Gewichtsvektoren.
Für den Ersten ergibt sich der Abstand d1:

$d_1 = (0.5 – 0.8)^2 + (0.6 – 0.7)^2 + (0.8 – 0.4)^2 = 0.26$

Für den Zweiten ergibt sich der Abstand d2:

$d_2 = (0.4 – 0.8)^2 + (0.2 – 0.7)^2 + (0.5 – 0.4)^2 = 0.42$

Die Distanz zu Gewichtsvektor 1 ist geringer und damit wird dieser verändert.
Dies geschieht mittels dieser Formel:

$$w_{ij}(n+1) = w_{ij}(n) + 0.5 [x_i – w_{ij}(n)] \qquad (1)$$

Dadurch ergeben sich neue Gewichte:

$$\begin{bmatrix} 0.65 & 0.4 \\ 0.65 & 0.2 \\ 0.6 & 0.5 \end{bmatrix}$$

Nun wird der nächste Eingabevektor mit den Gewichtsvektoren verglichen:

$d_1 = (0.65 – 0.6)^2 + (0.65 – 0.9)^2 + (0.60 – 0.9)^2 = 0.155$

$d_2 = (0.4 – 0.6)^2 + (0.2 – 0.9)^2 + (0.5 – 0.9)^2 = 0.69$

Gewichtsvektor 1 hat den geringeren Abstand zu Eingabevektor 2 und wird wieder mit (1) verändert. Dies ergibt Folgende neue Gewichtsvektoren:

$$\begin{bmatrix} 0.625 & 0.400 \\ 0.775 & 0.200 \\ 0.750 & 0.500 \end{bmatrix}$$

Diese werden nun mit Eingabevektor 3 verglichen:

$d_1 = (0.625 – 0.3)^2 + (0.775 – 0.4)^2 + (0.750 – 0.1)^2 = 0.67$

$d_2 = (0.4 – 0.3)^2 + (0.2 – 0.4)^2 + (0.5 – 0.1)^2 = 0.21$

Da nun die Distanz zwischen Eingabevektor 3 und Gewichtsvektor 2 kleiner ist, wird dieser mit (1) wie folgt verändert:

$$\begin{bmatrix} 0.625 & 0.350 \\ 0.775 & 0.300 \\ 0.750 & 0.300 \end{bmatrix}$$

Diese Gewichtsvektoren werden nun mit Eingabevektor 4 verglichen:

$d_1 = (0.625 - 0.1)^2 + (0.775 - 0.1)^2 + (0.750 - 0.3)^2 = 0.93$

$d_2 = (0.350 - 0.1)^2 + (0.300 - 0.1)^2 + (0.300 - 0.3)^2 = 0.10$

Da die Distanz zu Gewichtsvektor 2 kleiner ist, wird dieser wieder verändert. Dadurch entstehen diese Gewichtsvektoren nach Eingabe aller 4 Vektoren:

$$\begin{bmatrix} 0.625 & 0.225 \\ 0.775 & 0.200 \\ 0.750 & 0.300 \end{bmatrix}$$

3.0 Zusammenfassung

Die wichtigsten Merkmale der SOM sind rückblickend, ihr unüberwachtes Lernverfahren und ihr Clustering, welches durch Nachbarschaftsbeziehungen der Neuronen zustande kommt. Wichtig hierbei ist die Vorgehensweise indem ein Siegerneuron bestimmt wird und dann durch eine Bestimmung der Nachbarneuronen, die durch dieses Siegerneuron mitbeeinflusst werden, und der Stärke der Veränderung. Je nach Distanz zum Siegerneuron werden dann die Nachbarneuronen mitverändert.

Allerdings müssen Vorkehrungen getroffen werden, wie das Festlegen einer Gitterstruktur, um ein solches unüberwachtes Lernen zu ermöglichen. Auch ist dieses Verfahren nicht frei von Fehlfunktionen, aber dies kann durch sorgfältige Wahl von Anfangsbedingungen und der Nachbarschaftsbeziehung vermieden werden.

4. Quellen

[1] A. Zell. *Simulation Neuronaler Netze*. Addison – Wesley 1996.

[2] R. Calan. *The Essence of Neural Networks*. Prentice Hall, 1999.

[3] http://www.cl.uni-heidelberg.de/kurs/ws04/ml/ML-2004-Teil%2007.pdf (Download 20.2.2005

[4] http://www-lehre.inf.uos.de/~sblohm/documents/Bachelor-Kolloquium.pdf (Download 20.2.2005)

[5] http://www.fmi.uni-passau.de/lehrstuehle/demeer/seminars/ws_04_05/self_org_04_05/documents/lovasz_ausarbeitung.pdf (Download 20.2.2005)

[6] http://www.informatik.htw-dresden.de/~iwe/Belege/Hoehne/SOM_Anwenderdoku.pdf (Download 13.4.2005)

[7] http://iraf.noao.edu/iraf/web/ADASS/adass_proc/adass_95/freemanp/img17.gif (Download 20.2.2005)