

Beleg Neuronale Netze

Anwendung von SOM im 3D-Raum

René Liebscher
R.Liebscher@gmx.de

30. September 2000

1 Ziel

Ziel dieser Untersuchung ist die mögliche Anwendung von ein- oder zweidimensionalen SOM(self-organizing map) in Bezug auf dreidimensionale Oberflächen. Als Software kam der SNNS(Stuttgarter Neuronale Netz Simulator) zur Anwendung.

2 Ausgangsdaten

Als Ausgangsdaten wurden zufällige Koordinaten einer Kugel- bzw. Würfeloberfläche benutzt. Diese Daten beschreiben also eine geschlossene Oberfläche. Als drittes wurden die Werte einer mathematischen Funktion benutzt, die eine offene Oberfläche darstellt.

Alle Daten wurden mit Java-Programmen, ähnlich dem folgenden erstellt.

(Beispielfunktion: $f(x,y) = \cos(10 \cdot \sqrt{x^2 + y^2}) \cdot e^{(-2 \cdot \sqrt{x^2 + y^2})}$)

```
1  /* Datei: fa.java */
2  import java.lang.*;
3  import java.io.*;
4
5  public class fa{
6
7      public static void main(String argv[]){
8
9          System.out.println(
10             "SNNS pattern definition file V3.2\n"+
11             "generated at Sat Oct 30 16:50:10 MEST 1999\n"+
12             "\n"+
13             "No. of patterns : 100000\n"+
14             "No. of input units : 3\n"+
15             "\n\n");
16
17         for(int i=0;i<100000;++i){
18             double x,y,z,l;
19             x=Math.random()*2-1;
20             y=Math.random()*2-1;
21             l=Math.sqrt(x*x+y*y);
22             z=Math.cos(10*l)*Math.exp(-2*l);
23
24             // auf 5 Stellen runden damit
25             // SNNS die Daten akzeptiert
```

```

26         x=Math.round(x*100000.0)/100000.0;
27         y=Math.round(y*100000.0)/100000.0;
28         z=Math.round(z*100000.0)/100000.0;
29
30         System.out.println(x+ " "+y+ " "+z);
31     };
32 }
33 };

```

Die Daten können mittels folgender Befehlsfolge erzeugt werden.

```

{1}
# javac fa.java
# java fa >pattern_file

```

3 Lernvorgang

Als Ausgangspunkt für alle weiteren Schritte wurden zwei verschiedene Netze benutzt.

Einmal ein eindimensionales Netz aus 10000 Neuronen und ein zweidimensionales Netz aus 100·100 Neuronen. Beide Netze wurden zufällig initialisiert, wobei alle Gewichtsvektoren normalisiert wurden (interne Einstellung des SNNS.)

Als Lernparameter wurden, wenn nicht anders angegeben, folgende Werte benutzt: eindimensionaler Fall {0.8, 10000, 0.99995, 0.9999, 10000} und im zweidimensionalen Fall {0.8, 100, 0.99995, 0.99995, 100}.

Da der SNNS nur normierte Gewichtsvektoren unterstützt, wurde die Lernfunktion im SNNS geändert um beliebige Vektoren zuzulassen. Die Abstandsfunktion wurde notwendigermaßen vom Skalarprodukt auf den euklidischen Abstand geändert.

Der Lernvorgang dauert auf einem K6-2 400 ungefähr eine halbe Stunde.

4 Visualisierung der Lernergebnisse

Nach dem Lernvorgang wurden die Ergebnisse mit Hilfe von Gnuplot visualisiert. Dazu benötigt man passende Datendateien, welche sich mit folgenden awk-Skripten erzeugen lassen.

```

1 #/usr/bin/sh
2 # Datei: pat2dat
3
4 awk "/^ *-?[0-9]+\.[0-9]+.*/ { print }"

```

```

1 #/usr/bin/bash
2 # Datei: net2dat
3 # ein optionaler Parameter, der die
4 # x-Ausdehnung des Netzes angibt
5
6 teilung=${1:-0}
7
8 tr ":", " " |awk "
9 BEGIN { start_filter=0; }
10 /connection definition section.*/ { start_filter=1;i=0; }
11 /^ *[0-9]+ \| +\| +.*/{
12     if(start_filter>0){
13         a[\$4]=\$5;a[\$6]=\$7;a[\$8]=\$9;
14         print a[1] \" \" a[2] \" \" a[3];
15         i++;

```

```

16     if ( $teilung>0 && i % $teilung ==0) print "\"";
17     };
18     }
19     "

```

Anwendung auf die Muster-Dateien:

```
# bash pat2dat <pattern.pat >pattern.dat
```

Anwendung auf eindimensionale SOM:

```
# bash net2dat <net1d.net >net1d.pat
```

Anwendung auf zweidimensionale SOM:

```
# bash net2dat 100 <net2d.net >net2d.pat
```

In Gnuplot kann man die Daten folgendermaßen anzeigen lassen.

Muster-Dateien:

```
# gnuplot
gnuplot> set data style dots
gnuplot> splot "pattern.dat"

```

Eindimensionale SOM:

```
# gnuplot
gnuplot> set data style lines
gnuplot> splot "net1d.dat"

```

Zweidimensionale SOM:

```
# gnuplot
gnuplot> set data style lines
gnuplot> set hidden3d
gnuplot> splot "net2d.dat"

```

Weitere mögliche Gnuploteinstellungen entnehme man dessen Manual.

5 Auswertung

Im Einzelnen entstanden folgende Abbildungen(1, 2, 3, 4) der SOM auf den 3D-Raum. (Eindimensional als Linie und zweidimensional als Fläche dargestellt.)

5.1 Kugel

Wie leicht zu erkennen ist, hat sich das SOM beinahe komplett der Kugeloberfläche angepasst. Da allerdings die gegenüberliegenden Kanten nicht verbunden sind, bleibt immer eine gewisse Lücke übrig.

5.2 Würfel

Wie auch bei der Kugel kann hier das Netz sich zwar der Oberfläche anpassen, aber diese nicht komplett umschliessen.

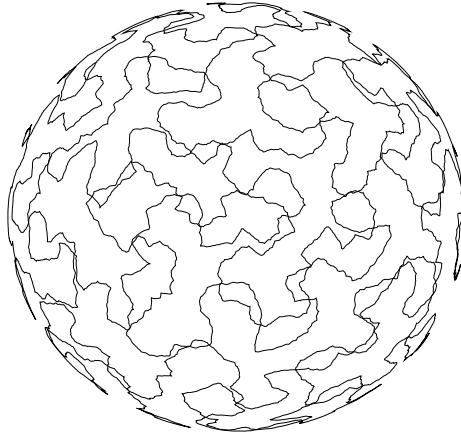


Abbildung 1: Eindimensionales SOM abgebildet auf Kugeloberfläche

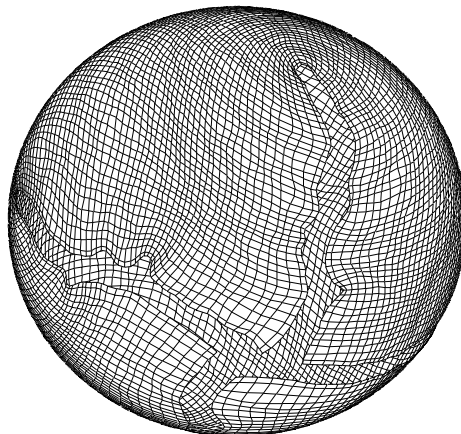


Abbildung 2: Zweidimensionales SOM abgebildet auf Kugeloberfläche

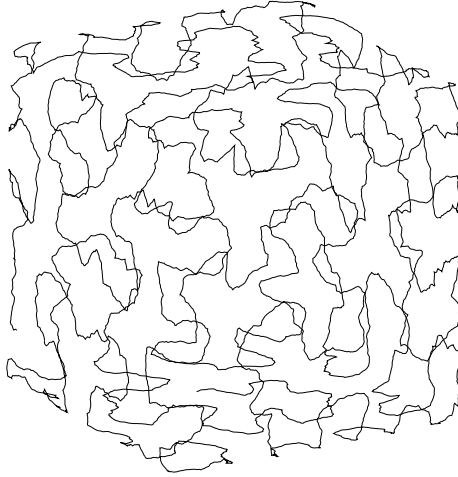


Abbildung 3: Eindimensionales SOM abgebildet auf Würfeloberfläche

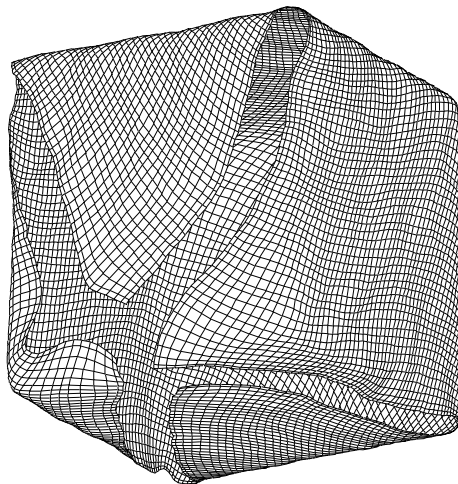


Abbildung 4: Zweidimensionales SOM abgebildet auf Würfeloberfläche

5.3 Einfluß des Radius auf die Abdeckung

Bei genauerer Betrachtung der obigen Abbildungen(1, 2, 3, 4) fällt auf, das speziell die eindimensionalen Abbildungen, nicht ihre komplette Möglichkeiten ausnutzen. Das ist offensichtlich durch die Lernparameter bedingt. Die gleichzeitige Abnahme der Adaptionstärke mit dem Radius verhindert die Ausbildung feinerer Strukturen.

Durch Festlegung eines konstanten Radius ergaben sich erhebliche Verbesserungen (Abbildungen 5, 6, 7).

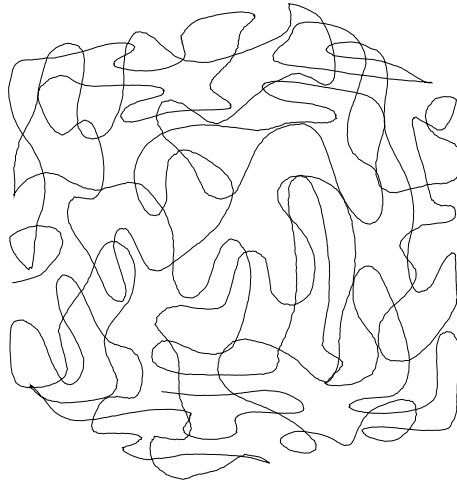


Abbildung 5: Eindimensionales SOM abgebildet auf Würfeloberfläche mit Radius 30

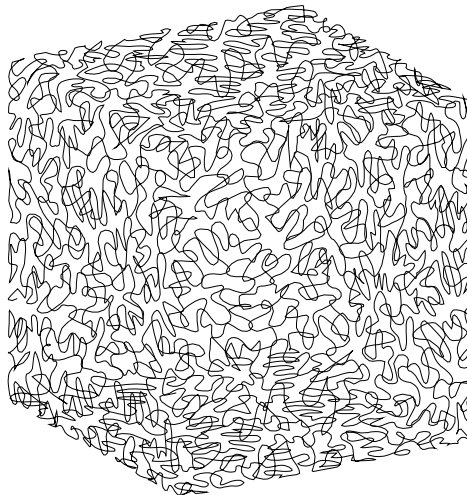


Abbildung 6: Eindimensionales SOM abgebildet auf Würfeloberfläche mit Radius 3

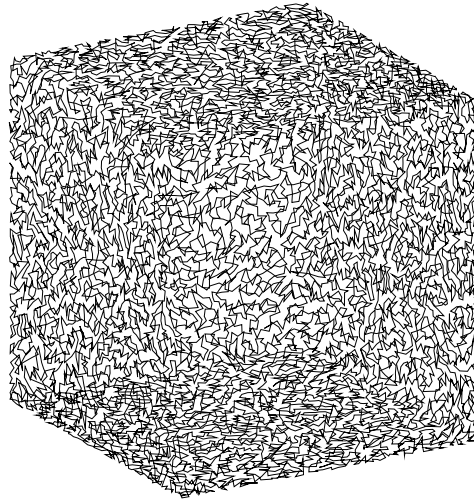


Abbildung 7: Eindimensionales SOM abgebildet auf Würfeloberfläche mit Radius 1

5.4 Abbildung der mathematischen Funktion

Als Ergebnis der mathematischen Funktion ergab sich Abbildung 8.

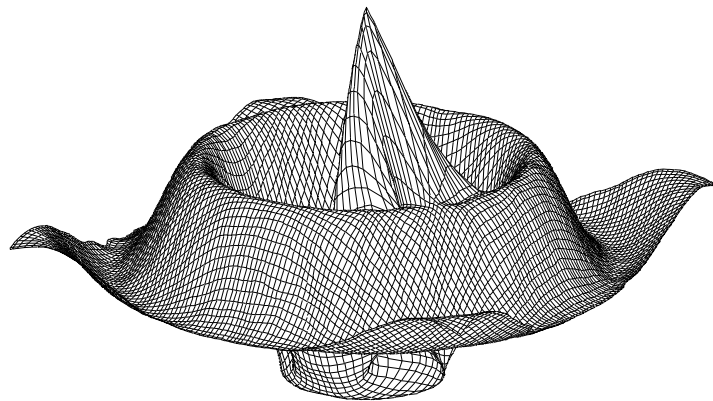


Abbildung 8: Abbildung der Funktion $f(x, y) = \cos(10 \cdot \sqrt{x^2 + y^2}) \cdot e^{(-2 \cdot \sqrt{x^2 + y^2})}$ mittels zweidimensionalen SOM

Im Grossen und Ganzen wurde die Funktion korrekt nachgebildet. Die festzustellende Abweichung (Mitte links, die Funktion ist eigentlich rotationssymmetrisch) kann möglicherweise durch weiteres Trainieren beseitigt werden. Der Grund dafür dürfte darin liegen, dass die Ausgangswerte gleichverteilt in der X-Y-Ebene erzeugt wurden, und damit für relativ steile Stellen relativ wenige Trainingsmuster zur Verfügung stehen.

6 Verbesserungen

Generell müssten wahrscheinlich die Lernparameter besser abgestimmt werden.

Für die Kugel und den Würfel wären geschlossene Netze besser geeignet.

6.1 Darstellung bei verbundenen Ecken

Wenn man die bisher sichtbaren Ecken und Kanten miteinander verbindet, erreicht man ein Ergebnis entsprechend Abbildung 9 bzw. 10.

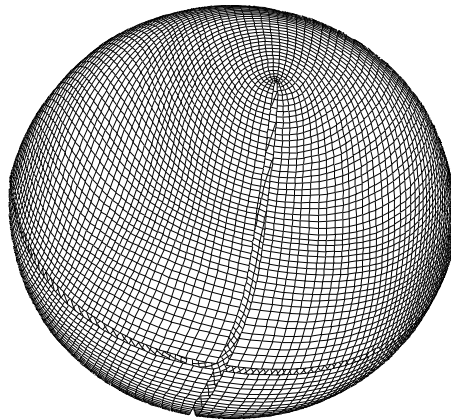


Abbildung 9: Abbildung der Kugel bei verbundenen Kanten

Die noch zu sehenden Lücken entstehen, weil Gnuplot für die Verbindung zwischen den Kanten keine Daten zur Verfügung stehen um dort die Fläche zu zeichnen. Dazu müsste man die zu zeichnenden Daten an jeder Seite um eine Reihe ergänzen, die jeweils die gegenüberliegenden Elemente darstellt.

6.2 Darstellung bei Kugelsystem

Bei Verbindung einer Längskante und Verbinden gegenüberliegender Punkte an der anderen Kante, ergibt sich eine Art Kugelsystem, ähnlich dem Längen- und Breitengradssystem der Erde. (Abbildung 11, 12)

Wie man sieht ist der Würfel besser geschlossen als die Kugel, was sich wahrscheinlich dadurch erklärt das bei der Kugel an den Polpunkten sich zu viele Neuronen zu nahe kommen.

7 Anwendungsmöglichkeiten

7.1 Eindimensionale SOM

Da die Fläche gleichmäßig abgedeckt wird, kommt es für entsprechende Anwendungen in Frage. Zum Beispiel das Auftragen von Klebstoffen auf Flächen oder das Lackieren derselben, dabei ist es von Vorteil wenn die Fläche gleichmässig und Kreuzungen abgearbeitet wird. Der Radius und die Anzahl der Neuronen müssten dem

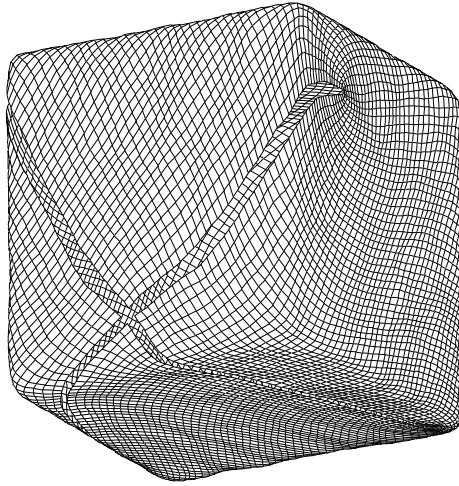


Abbildung 10: Abbildung des Würfels bei verbundenen Kanten

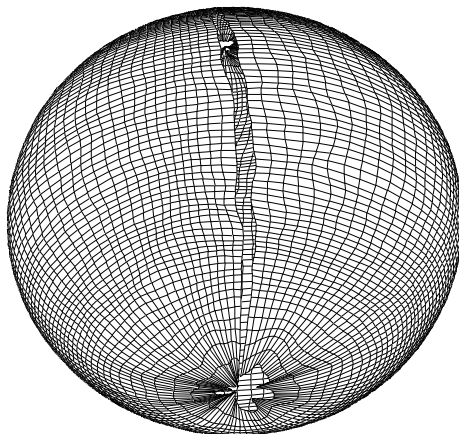


Abbildung 11: Abbildung der Kugel im Kugelsystem

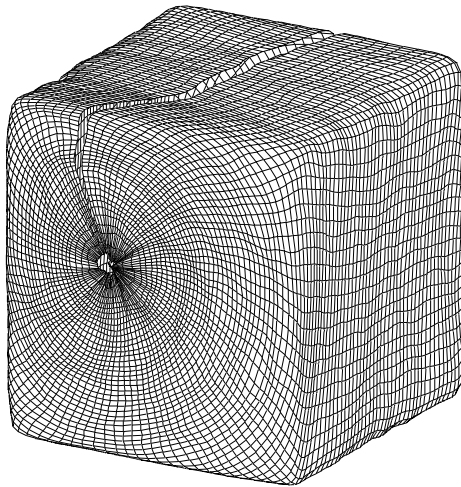


Abbildung 12: Abbildung des Würfels im Kugelsystem

Durchmesser des Farbstrahls angepasst werden. Gegenüber den normalerweise verwendeten Methoden, die z.B. Roboterarme auf langen geraden Wegen führen, wäre diese Art Wegplanung für zweistufige Systeme (Mensch: Schulter- und Handgelenk) passender, dabei kann eine grosse stabile Einheit ein kleine beweglichere führen. Der Weg für die grössere Einheit würde mit entsprechend grossem Radius berechnet, und dann weiter mit kleinerem Radius der Weg für die mobile Einheit.

7.2 Zweidimensionale SOM

Diese könnte man z.B. benutzen um Flächen in Mengen von 3D-Punkten (Messwerte) zu finden. Auch ist es denkbar damit Polygonnetze zur Visualisierung zu erzeugen, insbesondere von Daten die nicht berechenbar sind z.B. Messwerten.

Ein spezieller Fall ist hierbei die Anwendung auf Messwerte von 3D-Scannern, wobei allerdings als Vorgabe ein bereits entsprechend strukturiertes Netz zur Anwendung kommen sollte. (Zum Beispiel ein Ellipsoid, wenn man die Daten eines Kopfes einlernen will.)

Weiterhin kann man durch Aufsummieren der Flächenstücke eine Näherung für den Flächeninhalt der abgebildeten Oberfläche erhalten.

8 Codeänderungen im SNNS

Die Änderungen betreffen nur das Lernverhalten beim Kohonen-Lernverfahren. Die zu ändernde Funktion findet sich in der Kernel-Datei `learn_f.c` (bei Version 4.1 ab Zeile 6221.)

Die letzten zwei Änderungen funktionieren nur korrekt, wenn man ein zweidimensionales Netz hat und der Lernradius die Grösse des Netzes nicht überschreitet. Um auch den eindimensionalen Fall zu bearbeiten, müssten einige zusätzliche Checks eingebaut werden.

8.1 Nichtnormierte Koordinaten

Diese Änderung schaltet die Normierung der Eingabemuster und Gewichtsvektoren aus. Es sind nur geringe Veränderungen notwendig, die hier mit ##### markiert worden sind.

```

/*****
FUNCTION : propagateNet_kohonen

PURPOSE  : Propagate and train a pattern
NOTES    :
UPDATE   : 07.02 1994 by Sven Doering

                Copyright (c) 1992-1995 Neuro Group, Univ. of Tuebingen, FRG
*****/

static float propagateNet_kohonen(int pattern_no, int sub_pat_no, float height,
                                float radius, int sizehor)
{
    register struct Link *link_ptr;
    register struct Site *site_ptr;
    register struct Unit *unit_ptr;
    register struct Unit *winner_ptr;
    register Patterns in_pat;
    register int      NoOfCompounds, sizever, verwin, horwin, hor, ver, helpver,
    helphor, range;
    float            maximum, sum_error, deviat, learn_error, sum;
    float            unit_ptr_net;
    register TopoPtrArray topo_ptr;
    float            adapt;
    int               winner, current_no;

    /* calculate the activation and the output values          */
    /* of the input units (Input Layer)                       */

    NoOfCompounds = NoOfInputUnits;
    sizever = NoOfHiddenUnits / sizehor;

    sum = 0.0;

    /* calculate startaddress for input pattern array */
    in_pat = kr_getSubPatData(pattern_no, sub_pat_no, INPUT, NULL);

    topo_ptr = topo_ptr_array;

    /* copy pattern into input unit's activation and calculate output of the
       input units */
    while ((unit_ptr = *++topo_ptr) != NULL) { /* topo_ptr points to the
                                                unit stuctures (sorted by:
                                                input-, hidden- and
                                                output-units, separated
                                                with NULL pointers) */

        /*****
        /*sum += *in_pat * *in_pat;*/

        if (unit_ptr->out_func == OUT_IDENTITY)
            /* identity output function: there is no need to call the output
               function */
            unit_ptr->Out.output = unit_ptr->act = *in_pat++;
    }
}

```

```

else
    /* no identity output function: calculate unit's output also */
    unit_ptr->Out.output =
        (*unit_ptr->out_func) (unit_ptr->act = *in_pat++);
}

/*#####if (sum != 0.0) */
/* normalize the inputvector */
/*normalize_inputvector(sum);*/

/* propagate Kohonen Layer */

/* calculate the activation and the output values */
/* of the competitive units (hidden layer) */

/* #####*/
/* winner is determined using the euclidian distance(dot product) */

winner_ptr = NULL;
maximum = -1.0e30;          /* contains the maximum of the activations */
current_no = 0;

/* propagate hidden units */
while ((unit_ptr = *++topo_ptr) != NULL) { /* topo_ptr points to a
                                           (topological sorted) unit
                                           structure */

    unit_ptr_net = 0.0;
    if (UNIT_HAS_DIRECT_INPUTS(unit_ptr)) { /* the unit has direct links */
        FOR_ALL_LINKS(unit_ptr, link_ptr)
            /*#####*/
            unit_ptr_net -= (link_ptr->weight - link_ptr->to-
>Out.output)*(link_ptr->weight - link_ptr->to->Out.output);
    } else { /* the unit has sites */
        FOR_ALL_SITES_AND_LINKS(unit_ptr, site_ptr, link_ptr)
            /*#####*/
            unit_ptr_net -= (link_ptr->weight - link_ptr->to-
>Out.output)*(link_ptr->weight - link_ptr->to->Out.output);
    }

    if (maximum < unit_ptr_net) { /* determine winner unit */
        winner_ptr = unit_ptr;
        maximum = unit_ptr_net;
        winner = current_no;
    }
    current_no++;
    /* reset output and activation of hidden units */
    unit_ptr->Out.output = unit_ptr->act = (FlintType) 0;
}

/* the competitive winner is chosen */

winner_ptr->Out.output = winner_ptr->act = (FlintType) 1;
winner_ptr->bias++;
winner_ptr->value_a = (FlintType) (pattern_no + 1);

/* store number of according pattern in winner unit */

horwin = winner % sizehor;
verwin = winner / sizehor;

```

```

/*****
/* Train the SOM */

/* Only the weights of links that go to the winner and its */
/* neighbourhood are adjusted, the others remain the same. */
/* The incoming weights to the competitive units are adapted */
/* as follows: */

/* weight(new) = weight(old) + adapt * (output - weight(old)) */

/* where adapt is the learning rate (0 < adapt <= 1.0) */
/* and output is the value of the input unit vector */

*****/

for (ver = 0; ver < sizever; ver++)
    for (hor = 0; hor < sizehor; hor++)
        if ((hor < radius + horwin) &&
            (hor > horwin - radius) &&
            (ver < radius + verwin) &&
            (ver > verwin - radius)) {
            helpver = (float) ((ver - verwin) * (ver - verwin));
            helphor = (float) ((hor - horwin) * (hor - horwin));
            adapt = height * exp(-(helpver + helphor) /
                                   (float) (radius * radius));

            sum = 0.0;
            range = ver * sizehor + hor + 1 + NoOfCompounds;

            /* get unit pointer of unit in adaptation range */
            unit_ptr = kr_getUnitPtr(range);

            if(!IS_SPECIAL_UNIT(unit_ptr)){
                if (unit_ptr->flags & UFLAG_DLINKS) { /* the unit has */
                                                         /* direct links */
                    FOR_ALL_LINKS(unit_ptr, link_ptr) {
                        deviat=link_ptr->to->Out.output - link_ptr->weight;
                        learn_error = adapt * deviat;
                        link_ptr->weight += learn_error;
                        /* this is needed for the normalization of the
                           weight_vector */
                        /******
                        /*sum += link_ptr->weight * link_ptr->weight;*/
                    }
                } else { /* the winner unit has sites */
                    FOR_ALL_SITES_AND_LINKS(winner_ptr,site_ptr,link_ptr) {
                        deviat=link_ptr->to->Out.output - link_ptr->weight;
                        learn_error = adapt * deviat;
                        link_ptr->weight += learn_error;
                        /* this is needed for the normalization of the
                           weight_vector */
                        /******
                        /*sum += link_ptr->weight * link_ptr->weight;*/
                    }
                }
            }
            /* if (sum != 0.0) *****/
            /*normalize_weight(unit_ptr, sum);*/

```

```

    }
}
sum_error = 0.0;          /* 0.0 is chosen arbitrarily and serves no
                           purpose */
return (sum_error);
}

```

8.2 Ecken und Kanten verbinden

Diese Änderung verbindet die Ecken miteinander, dazu wurde der zweite Teil der Funktion (Train the SOM) etwas weitgehender geändert. Der veränderte Abschnitt ist wieder durch ##### markiert.

```

/*****
FUNCTION : propagateNet_kohonen

PURPOSE  : Propagate and train a pattern
NOTES    :
UPDATE   : 07.02 1994 by Sven Doering

                Copyright (c) 1992-1995  Neuro Group, Univ. of Tuebingen, FRG
*****/

static float propagateNet_kohonen(int pattern_no, int sub_pat_no, float height,
                                  float radius, int sizehor)
{
    register struct Link *link_ptr;
    register struct Site *site_ptr;
    register struct Unit *unit_ptr;
    register struct Unit *winner_ptr;
    register Patterns in_pat;
    /* neu */
    register int coord_overflow_direction, my_ver, my_hor;
    /* ende neu */
    register int    NoOfCompounds, sizever, verwin, horwin, hor, ver, helpver,
    helphor, range;
    float          maximum, sum_error, deviat, learn_error;
    float          unit_ptr_net;
    register TopoPtrArray topo_ptr;
    float          adapt;
    int            winner, current_no;

    /* calculate the activation and the output values          */
    /* of the input units (Input Layer)                       */

    NoOfCompounds = NoOfInputUnits;
    sizever = NoOfHiddenUnits / sizehor;

    /* calculate startaddress for input pattern array */
    in_pat = kr_getSubPatData(pattern_no, sub_pat_no, INPUT, NULL);

    topo_ptr = topo_ptr_array;

    /* copy pattern into input unit's activation and calculate output of the
       input units */
    while ((unit_ptr = *++topo_ptr) != NULL) { /* topo_ptr points to the
                                                unit stuctures (sorted by:

```

```

input-, hidden- and
output-units, separated
with NULL pointers) */

if (unit_ptr->out_func == OUT_IDENTITY)
    /* identity output function: there is no need to call the output
    function */
    unit_ptr->Out.output = unit_ptr->act = *in_pat++;
else
    /* no identity output function: calculate unit's output also */
    unit_ptr->Out.output =
        (*unit_ptr->out_func) (unit_ptr->act = *in_pat++);
}

/* propagate Kohonen Layer */

/* calculate the activation and the output values */
/* of the competitive units (hidden layer) */

/* winner is determined using the euclidian distance */

winner_ptr = NULL;
maximum = -1.0e30;          /* contains the maximum of the activations */
current_no = 0;

/* propagate hidden units */
while ((unit_ptr = *++topo_ptr) != NULL) { /* topo_ptr points to a
                                           (topological sorted) unit
                                           structure */
    unit_ptr_net = 0.0;
    if (UNIT_HAS_DIRECT_INPUTS(unit_ptr)) { /* the unit has direct links */
        FOR_ALL_LINKS(unit_ptr, link_ptr)
            unit_ptr_net -= (link_ptr->weight - link_ptr->to-
>Out.output)*(link_ptr->weight - link_ptr->to->Out.output);
    } else { /* the unit has sites */
        FOR_ALL_SITES_AND_LINKS(unit_ptr, site_ptr, link_ptr)
            unit_ptr_net -= (link_ptr->weight - link_ptr->to-
>Out.output)*(link_ptr->weight - link_ptr->to->Out.output);
    }

    if (maximum < unit_ptr_net) { /* determine winner unit */
        winner_ptr = unit_ptr;
        maximum = unit_ptr_net;
        winner = current_no;
    }
    current_no++;
    /* reset output and activation of hidden units */
    unit_ptr->Out.output = unit_ptr->act = (FlintType) 0;
}

/* the competitive winner is chosen */

winner_ptr->Out.output = winner_ptr->act = (FlintType) 1;
winner_ptr->bias++;
winner_ptr->value_a = (FlintType) (pattern_no + 1);

/* store number of according pattern in winner unit */

horwin = winner % sizehor;

```

```

verwin = winner / sizehor;

/*****
/* Train the SOM */

/* Only the weights of links that go to the winner and its */
/* neighbourhood are adjusted, the others remain the same. */
/* The incoming weights to the competitive units are adapted */
/* as follows: */

/* weight(new) = weight(old) + adapt * (output - weight(old)) */

/* where adapt is the learning rate (0 < adapt <= 1.0) */
/* and output is the value of the input unit vector */

*****/

/* # Anfang ##### */
for (ver = verwin-radius; ver <= verwin+radius; ver++)
    for (hor = horwin-radius; hor <= horwin+radius; hor++)
        {
            coord_overflow_direction=0;

            helpver = (float) ((ver-verwin) * (ver-verwin));
            helphor = (float) ((hor-horwin) * (hor-horwin));
            adapt = height * exp(-(helpver + helphor) /
                (float) (radius * radius));

            /* Seiten "vernaehen" */
            my_ver=ver;
            my_hor=hor;

            if(my_ver<0)coord_overflow_direction|=1<<0;
            if(my_ver>=sizever)coord_overflow_direction|=1<<1;
            if(my_hor<0)coord_overflow_direction|=1<<2;
            if(my_hor>=sizehor)coord_overflow_direction|=1<<3;

            switch(coord_overflow_direction){

            case 0x1:
                my_ver=-my_ver-1;
                my_hor=sizehor-my_hor-1;
                break;

            case 0x2:
                my_ver=2*sizever-my_ver-1;
                my_hor=sizehor-my_hor-1;
                break;

            case 0x4:
                my_ver=sizever-my_ver-1;
                my_hor=-my_hor-1;
                break;

            case 0x8:
                my_ver=sizever-my_ver-1;
                my_hor=2*sizehor-my_hor-1;

```

```

        break;

    case 0x5:
        my_ver=sizever+my_ver;
        my_hor=sizehor+my_hor;
        break;

    case 0xa:
        my_ver=my_ver-sizever;
        my_hor=my_hor-sizehor;
        break;

    case 0x9:
        my_ver=sizever+my_ver;
        my_hor=my_hor-sizehor;
        break;

    case 0x6:
        my_ver=my_ver-sizehor;
        my_hor=sizehor+my_hor;
        break;

    default:
        break;

};

range = my_ver * sizehor + my_hor + 1 + NoOfCompounds;

/* # Ende ##### */

/* get unit pointer of unit in adaptation range */
unit_ptr = kr_getUnitPtr(range);

if(!IS_SPECIAL_UNIT(unit_ptr)){
    if (unit_ptr->flags & UFLAG_DLINKS) { /* the unit has */
                                        /* direct links */
        FOR_ALL_LINKS(unit_ptr, link_ptr) {
            deviat=link_ptr->to->Out.output - link_ptr->weight;
            learn_error = adapt * deviat;
            link_ptr->weight += learn_error;
        }
    } else { /* the winner unit has sites */
        FOR_ALL_SITES_AND_LINKS(winner_ptr,site_ptr,link_ptr) {
            deviat=link_ptr->to->Out.output - link_ptr->weight;
            learn_error = adapt * deviat;
            link_ptr->weight += learn_error;
        }
    }
}

sum_error = 0.0; /* 0.0 is chosen arbitrarily and serves no
                 purpose */
return (sum_error);
}

```

8.3 Kugelförmig verbinden

Hier wurde das Netz kugelförmig verbunden. Der geänderte Bereich entspricht dem obigen.

```

/*****
FUNCTION : propagateNet_kohonen

PURPOSE : Propagate and train a pattern
NOTES   :
UPDATE  : 07.02 1994 by Sven Doering

                Copyright (c) 1992-1995 Neuro Group, Univ. of Tuebingen, FRG
*****/

static float propagateNet_kohonen(int pattern_no, int sub_pat_no, float height,
                                float radius, int sizehor)
{
    register struct Link *link_ptr;
    register struct Site *site_ptr;
    register struct Unit *unit_ptr;
    register struct Unit *winner_ptr;
    register Patterns in_pat;
    /* neu */
    register int my_ver, my_hor;
    /* ende neu */
    register int    NoOfCompounds, sizever, verwin, horwin, hor, ver, helpver,
    helphor, range;
    float          maximum, sum_error, deviat, learn_error;
    float          unit_ptr_net;
    register TopoPtrArray topo_ptr;
    float          adapt;
    int            winner, current_no;

    /* calculate the activation and the output values          */
    /* of the input units (Input Layer)                       */

    NoOfCompounds = NoOfInputUnits;
    sizever = NoOfHiddenUnits / sizehor;

    /* calculate startaddress for input pattern array */
    in_pat = kr_getSubPatData(pattern_no, sub_pat_no, INPUT, NULL);

    topo_ptr = topo_ptr_array;

    /* copy pattern into input unit's activation and calculate output of the
       input units */
    while ((unit_ptr = *++topo_ptr) != NULL) { /* topo_ptr points to the
                                                unit structures (sorted by:
                                                input-, hidden- and
                                                output-units, separated
                                                with NULL pointers) */

        if (unit_ptr->out_func == OUT_IDENTITY)
            /* identity output function: there is no need to call the output
               function */
            unit_ptr->Out.output = unit_ptr->act = *in_pat++;
        else
            /* no identity output function: calculate unit's output also */
            unit_ptr->Out.output =
                (*unit_ptr->out_func) (unit_ptr->act = *in_pat++);
    }
}

```

```

/* propagate Kohonen Layer */

/* calculate the activation and the output values */
/* of the competitive units (hidden layer) */

/* #####*/
/* winner is determined using the euclidian distance */

winner_ptr = NULL;
maximum = -1.0e30;          /* contains the maximum of the activations */
current_no = 0;

/* propagate hidden units */
while ((unit_ptr = *++topo_ptr) != NULL) { /* topo_ptr points to a
                                           (topological sorted) unit
                                           structure */
    unit_ptr_net = 0.0;
    if (UNIT_HAS_DIRECT_INPUTS(unit_ptr)) { /* the unit has direct links */
        FOR_ALL_LINKS(unit_ptr, link_ptr)
            unit_ptr_net -= (link_ptr->weight - link_ptr->to-
>Out.output)*(link_ptr->weight - link_ptr->to->Out.output);
    } else { /* the unit has sites */
        FOR_ALL_SITES_AND_LINKS(unit_ptr, site_ptr, link_ptr)
            unit_ptr_net -= (link_ptr->weight - link_ptr->to-
>Out.output)*(link_ptr->weight - link_ptr->to->Out.output);
    }

    if (maximum < unit_ptr_net) { /* determine winner unit */
        winner_ptr = unit_ptr;
        maximum = unit_ptr_net;
        winner = current_no;
    }
    current_no++;
    /* reset output and activation of hidden units */
    unit_ptr->Out.output = unit_ptr->act = (FlintType) 0;
}

/* the competitive winner is chosen */

winner_ptr->Out.output = winner_ptr->act = (FlintType) 1;
winner_ptr->bias++;
winner_ptr->value_a = (FlintType) (pattern_no + 1);

/* store number of according pattern in winner unit */

horwin = winner % sizehor;
verwin = winner / sizehor;

/******/
/* Train the SOM */

/* Only the weights of links that go to the winner and its */
/* neighbourhood are adjusted, the others remain the same. */
/* The incoming weights to the competitive units are adapted */
/* as follows: */

/* weight(new) = weight(old) + adapt * (output - weight(old)) */

```

```

/* where adapt is the learning rate (0 < adapt <= 1.0)          */
/* and output is the value of the input unit vector            */

/*****/

for (ver = verwin-radius; ver <= verwin+radius; ver++)
  for (hor = horwin-radius; hor <= horwin+radius; hor++)
    {

      helpver = (float) ((ver-verwin) * (ver-verwin));
      helphor = (float) ((hor-horwin) * (hor-horwin));
      adapt = height * exp(-(helpver + helphor) /
                           (float) (radius * radius));

      /* Seiten "vernaehen" horizontal liegender "Zylinder/Globus" */
      my_ver=ver;
      my_hor=hor;

      /* Zylinder Laengsseite */
      while(my_ver<0)my_ver+=sizever;
      while(my_ver>=sizever)my_ver-=sizever;

      /* Polseiten */
      if(my_hor<0){
        my_hor=-my_hor-1; my_ver=(sizever/2+my_ver)%sizever;
      };
      if(my_hor>=sizehor){
        my_hor=2*sizehor-my_hor-1; my_ver=(sizever/2+my_ver)%sizever;
      };

      range = my_ver * sizehor + my_hor + 1 + NoOfCompounds;

      /* get unit pointer of unit in adaptation range */
      unit_ptr = kr_getUnitPtr(range);

      if(!IS_SPECIAL_UNIT(unit_ptr)){
        if (unit_ptr->flags & UFLAG_DLINKS) { /* the unit has */
                                              /* direct links */
          FOR_ALL_LINKS(unit_ptr, link_ptr) {
            deviat=link_ptr->to->Out.output - link_ptr->weight;
            learn_error = adapt * deviat;
            link_ptr->weight += learn_error;
          }
        } else { /* the winner unit has sites */
          FOR_ALL_SITES_AND_LINKS(winner_ptr,site_ptr,link_ptr) {
            deviat=link_ptr->to->Out.output - link_ptr->weight;
            learn_error = adapt * deviat;
            link_ptr->weight += learn_error;
          }
        }
      }
    }
  }
sum_error = 0.0; /* 0.0 is chosen arbitrarily and serves no
                 purpose */
return (sum_error);
}

```