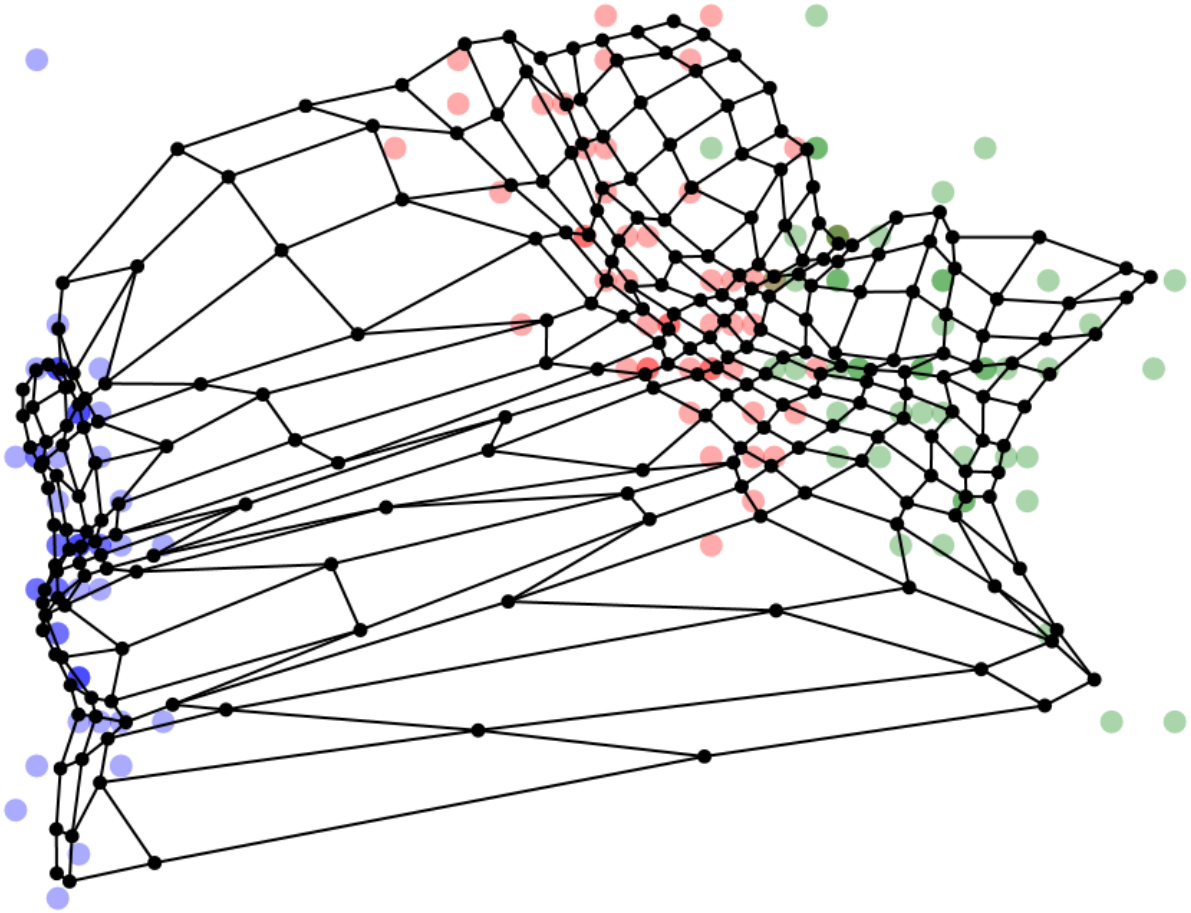

# Kohonen Netze

Selbstorganisierende Karten

Seminararbeit

Marko Bublic
bublm1@bfh.ch

# Zusammenfassung

Teuvo Kohonens selbstorganisierende Karten gehören zum Paradigma des unüberwachten Lernens. Das Ziel des Netzes ist in einem Merkmalsraum, dessen Klasseneinteilung nicht bekannt ist, eine darunterliegende Struktur zu entdecken. Kohonen Netze sind topologieerhaltende Karten: Die Neuronen im Netz haben zueinander eine Nachbarschaftsbeziehung, die nach dem ‚Erlernen' des Merkmalsraumes die Beziehungen zueinander erhalten. Das Resultat des Lernverfahrens ist eine tiefdimensionale Darstellung (Karte) eines Hochdimensionalen Raumes. Das Konzept folgt der Erkenntnis aus der Biologie, dass ähnliche Reize von Nervenzellen verarbeitet werden, die räumlich nahe bei einander liegen.

Viele Parameter (Lernrate, Nachbarschaftsbetiehungen, Anzahl Trainingsschritte) müssen vor dem Lernverfahren definiert werden. Ein Vorwissen über die Topologie des Merkmalsraumes ist bei der Justierung der Parameter hilfreich.

Der Algorithmus ist einfach umgesetzt und gibt in den meisten Fällen eine brauchbare Lösung zurück. Da das Lernverfahren nicht-deterministische Schritte besitzt, ist das resultierende Netz jedes Mal unterschiedlich.

# Inhaltsverzeichnis

1 Neuronale Netze ..... 1
1.1 Biologische Grundlage ..... 1
1.2 Modellierung des künstlichen Neurons ..... 2
1.3 Lernvorgang ..... 3
1.3.1 Algorithmus für Gewichte ..... 3
1.3.2 Online und Offline-Modus ..... 3
1.4 Anwendungen ..... 3
1.4.1 Mehrschichtige Neuronale Netze ..... 4
2 Kohonen Netze ..... 5
2.1 Teuvo Kohonen ..... 5
2.2 Strukturen im Gehirn ..... 5
2.3 Lernalgorithmus im Detail ..... 6
2.3.1 Stichprobe ..... 6
2.3.2 Neuronen ..... 6
2.3.3 Lernphase ..... 7
2.3.4 Zeitkomponente $\boldsymbol{t}$ ..... 7
2.3.5 Lernrate $\boldsymbol{\epsilon}$ ..... 8
2.3.6 Nachbarschaftsradius $\boldsymbol{\mathcal{R}}$ ..... 8
2.3.7 Entfernungsgewichtungsfunktion $\boldsymbol{h}$ ..... 10
2.4 Klassifizierung, Mapping und Einsatzgebiet ..... 10
3 Implementation ..... 11
3.1 Technologie ..... 12
3.2 Lernalgorithmus ..... 12
3.3 Zufällig ein Element wählen ..... 13
3.4 Nachbarschaft abrufen ..... 13
4 Abbildungsverzeichnis ..... 14
5 Literaturverzeichnis ..... 15

# 1 Neuronale Netze

Neuronale Netze sind Datenstrukturen, welche die Nervenzellen und deren Verbindungen untereinander, wie sie in allen höheren Lebewesen vorkommen, abbilden. Die neuronalen Netze sind ein Teilgebiet der künstlichen Intelligenz. Das Ziel ist das Verhalten von Neuronen in einem Gehirn abzubilden. Man macht sich zu Nutze, dass die Neuronen und Netze von bekannten Datensätzen ‚lernen' können, um dann neue und unbekannte Inputs schnell und zuverlässig zu verarbeiten.

Die theoretischen Grundlagen wurden in den Anfangsjahren der Informationstechnologie (40er und 50er Jahre des letzten Jahrhunderts) von verschiedenen Mathematikern, Physiologen und Psychologen erarbeitet (Prof. Dr. Gauglitz & Jürgens, 1999). Die geschaffenen Modelle gerieten in Vergessenheit, bis sie schliesslich in den 1980er Jahren wieder aufgegriffen wurden, als die Computer leistungsfähiger und Anwendungsfälle (Mustererkennung, Telekommunikation) vielfältiger wurden.

## 1.1 Biologische Grundlage

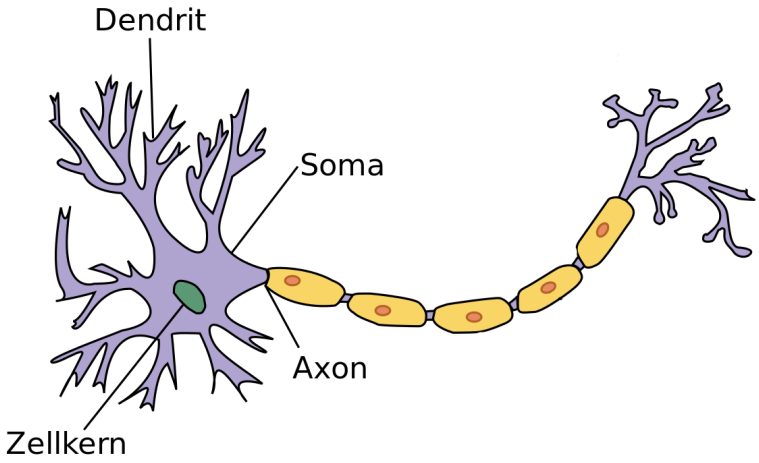


*Abbildung 1: Schematische Darstellung einer Nervenzelle. Quelle: (Wikipedia, 2016)*

Eine Nervenzelle besteht aus vier grundlegenden Bestandteilen: Zellkern, Zellkörper (Soma), mehreren Dendriten und einem Axon mit mehreren Axon-Enden. Die Dendriten empfangen elektrochemische Signale von anderen Neuronen und die Axonen leiten die Signale an andere Neuronen weiter. Die Übertragungseigenschaften der Verbindungen zwischen Dendriten und Axonen verändern sich mit der Zeit; das Neuron ‚lernt'. Jedes Neuron hat einen Schwellwert. Dieser ist abhängig davon wie viele und welche Dendriten gerade ein Signal erhalten. Wird der Schwellwert überschritten, wird ein Signal über das Axon weitergegeben an weitere Neuronen weitergegeben (Wikipedia, 2016).

## 1.2 Modellierung des künstlichen Neurons

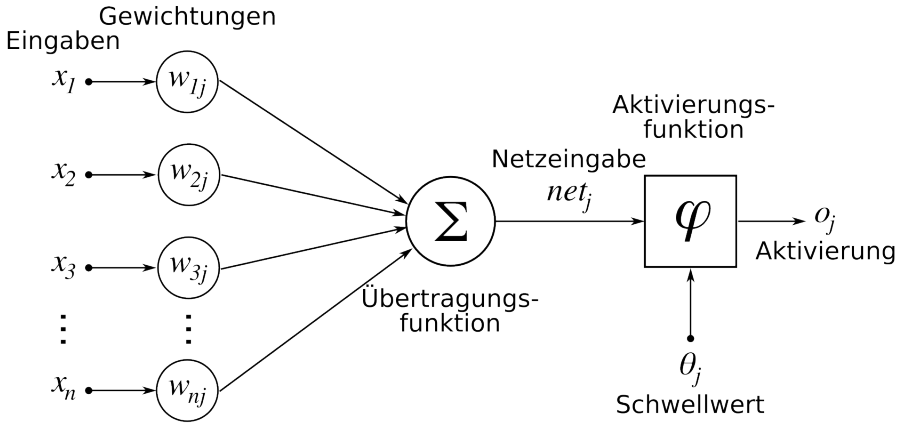


*Abbildung 2: Darstellung eines künstlichen Neurons mit seinen Elementen. Quelle: (Wikipedia, 2016)*

Ein künstliches Neuron besteht aus vier Elementen: Beliebig viele gewichtete Eingabevariabeln, eine Übertragungsfunktion, eine Aktivierungsfunktion und ein Schwellwert (Wikipedia, 2016).

Zu jedem Eingabewert $x_i$ ist ein Gewicht $w_i$ zugeordnet. Beide Werte sind üblicherweise reelle Zahlen. Je nach Anwendung können die Eingabewerte aber auch auf bestimmte Wertmengen beschränkt sein: Binär $\{1, 0\}$, diskrete Wertebereiche wie $\{-1, 0, 1\}$ oder Intervalle wie $[0, 1]$.

Die Übertragungsfunktion summiert die Werte auf:

$$f(x_1, \dots, x_n) = \sum_{i=1}^{n} w_i x_i$$

Auf das Resultat der Übertragungsfunktion wird eine Aktivierungsfunktion angewendet. Übersteigt das Resultat der Aktivierungsfunktion einen bestimmten Schwellwert, aktiviert sich das Neuron und leitet ein Signal an verbundene Neuronen oder an die Ausgabe. Es sind verschiedene Aktivierungsfunktionen gebräuchlich:

1. Hart-begrenzte Schwellwertfunktion

$$g(x) = \begin{cases} 0, & x < \theta \\ 1, & x \geq \theta \end{cases}$$

2. Stückweise lineare Schwellwertfunktionen

$$g(x) = \begin{cases} 1, & x > b \\ \frac{1}{b-a}(x-a), & a \leq x \leq b \\ 0, & x < a \end{cases}$$

3. Sigmoid-Funktionen

$$g(x) = \frac{1}{1+e^{-cx}}, mit\ c > 0\ und\ c\ konstant$$

Die letzten zwei können als unscharfe (fuzzy) Funktionen betrachtet werden. Solche Aktivierungsfunktionen leiten Resultate aus einem Intervall (üblicherweise $[0, 1]$) weiter.

## 1.3 Lernvorgang

Bevor das Neuron eingesetzt wird, müssen die Gewichte $w_i$ und der Schwellwert $\theta$ zunächst erlernt werden. Diese Werte können initial zufällig gewählt werden. Weiter braucht es eine Stichprobe von Eingabeelementen, von denen das erwartete Resultat bekannt ist. In einem iterativen Training werden nun die passenden Gewichte und der passende Schwellwert gefunden:

### 1.3.1 Algorithmus für Gewichte

Für alle Stichprobenelemente $x_s$

Berechne den Output: $y = g(wx)$

Gewichte anpassen: $w = w + d * (y_s - y) * x_s$

Bis für alle Stichprobenelemente gilt: $(y_s - y) = 0$

Dabei ist $y_s$das erwartete Resultat und $d$ eine konstante Lernrate. Dies ist eine Zahl im Intervall $]0, 1]$. Die Stichprobenelemente werden in mehreren Durchläufen (Epochen) trainiert. Die Reihenfolge der Stichproben wird pro Epoche zufällig gewählt.

### 1.3.2 Online und Offline-Modus

Die oben beschriebene Variante ist der Online-Modus des Trainings. Das Anpassung der Gewichte geschieht unmittelbar nach der Berechnung des Outputs für das aktuelle Stichprobenelement. Im Offline-Modus erfolgt zunächst die Berechnung für alle Stichproben. Anschliessend wird der Unterschied zwischen Output und erwartetem Output berechnet. Beim Anpassen der Gewichte wird am Schluss der Mittelwert der Unterschiede berücksichtigt. Das Lernen der Gewichte erfolgt im Offline-Modus ebenfalls in mehreren Epochen.

## 1.4 Anwendungen

Neuronale Netze finden Anwendung bei der Klassifikation von Eingabedaten. Nach dem Training kann ein neuer Eingabewert einer von mehreren vorher definierten Kassen zugeordnet werden. In vielen Fällen lassen sich die Klassen aber nicht klar trennen. Dann kommen unscharfe Funktionen (wie oben beschrieben) zum Zug. Diese weisen einem Eingabewert Zugehörigkeitswerte im Intervall $[0, 1]$ zu jeder Klasse zu. Falls eine Gewinnerklasse gesucht ist, fällt die Entscheidung meist zu Gunsten der Klasse mit dem höchsten Zugehörigkeitswert.

### 1.4.1 Mehrschichtige Neuronale Netze

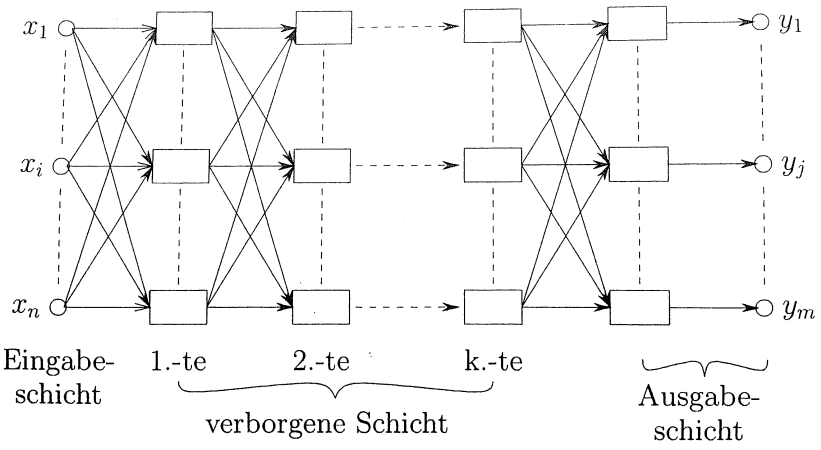


*Abbildung 3: Darstellung eines mehrschichtigen neuronalen Netzes. Quelle: (Bunke, 2006)*

Um die Struktur im Gehirn passender abzubilden, werden die Neuronen in mehreren (verborgenen) Schichten verknüpft. Neuronen auf einer Schicht sind mit allen Neuronen aus benachbarten Schichten verbunden. Die Eingabe wird so von einer Schicht auf die nächste weitergegeben bis Werte bei der Ausgabeschicht ankommen. Diese Struktur ist in der Anwendbarkeit mächtiger. Das Training ist dafür auch komplexer: Nach der Verarbeitung einer Stichprobe müssen auf jeder der Schichten und Neuronen die Gewichte angepasst werden. Bei Netzen mit vielen Schichten und umfangreichen Stichproben kann der Lernvorgang rechenaufwendig werden.

Dazu kommen weitere Schwierigkeiten:

- Das Netz kann übertrainiert werden. Wenn sich das Netz auf eine Stichprobe zu stark adaptiert, kann es vorkommen, dass unbekannte Eingaben falsch klassifiziert werden. Die richtige Anzahl Elemente pro Stichprobe lässt sich nicht bezeichnen.
- Es gibt keine allgemeingültige Empfehlung für die richtige Anzahl Schichten und die richtige Anzahl Neuronen pro Schicht. Abhängig vom Problem sind andere Konfigurationen von Vorteil. Hier muss man ebenfalls mit ‚trial-and-error' suchen.

Neuronale Netze sind in den Fällen praktisch, bei denen eine algorithmische Lösung sehr komplex oder unmöglich wäre und hauptsächlich ein Resultat und nicht der Weg zum Resultat interessiert. Das Netz und insbesondere Gewichte und Schwellwerte der einzelnen Neuronen sind für Menschen nicht interpretierbar. Das trainierte Netz ist wie eine schwarze Box, in welcher Eingabewerte zu Ausgabewerten überführt werden und wir können sie nicht aufmachen, um die Mechanismen zu analysieren.

# 2 Kohonen Netze

Im Bereich der künstlichen Intelligenz gibt es zwei grundlegend unterschiedliche Lern-Paradigmen:

- Überwachtes Lernen. Bei diesem Prozess ist eine Stichprobe gegeben, dessen Soll-Resultate (Klassenzugehörigkeit) bekannt sind. Die Stichprobe wird verwendet, um die Parameter (Gewichte bei neuronalen Netzen) zu trainieren. Dieses Verfahren ist in Kapitel 1.3 grob aufgezeigt.
- Unüberwachtes Lernen. Bei dieser Variante ist nur eine Stichprobe gegeben. Die Soll-Resultate sind nicht bekannt. Das bedeutet: Die Klassenzugehörigkeit der Stichprobenelemente sowie die Anzahl der verschiedenen Klassen ist unbekannt.

Unüberwachtes Lernen ist dann attraktiv, wenn die manuelle Klassifizierung der Stichprobe sehr aufwändig ist. Desweitern kann das Verfahren Muster in grossen, unstrukturierten Datensätzen aufspüren (Bunke, 2006). Diese sogenannte Clustering-Analyse ist von Grosser Bedeutung im Data Mining.

Die Kohonen Netze gehören zu den unüberwachten Lernverfahren. Das Ziel von Kohonens Ansatz ist eine tiefdimensionale Darstellung (dreidimensional oder kleiner) von hochdimensionalen Daten. Dabei soll die Topologie der Stichprobe erhalten bleiben. Das heisst: Reize die nahe bei einander liegen im Stichprobenraum, sollen im Kohonen Netz auch benachbart sein. Ein solches Netz ist wie eine Kartographierung des Stichprobenraumes. Die tiefdimensionale Abbildung der Daten ist nachbarschaftserhaltend. So können Klassen erkannt und definiert werden.

## 2.1 Teuvo Kohonen

Kohonen, geboren 1934, ist ein finnischer Ingenieur und emeritierter Professor an der staatlichen Akademie von Finnland. Für seine Forschung in den Bereichen der Selbstorganisation, der neuronalen Netze und der Mustererkennung erhielt er diverse Auszeichnungen und Ehrendoktorwürden. Während seiner Arbeit veröffentlichte er über 300 Forschungsarbeiten; darunter befindet sich die 1982 erschienene Abhandlung über selbstorganisierende Karten (Kohonen D. E., 2001).

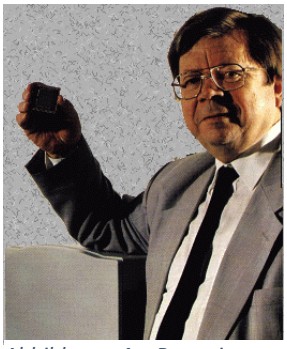

*Abbildung 4: Portrait von Teuvo Kohonen, 2000. Quelle: (Kohonen D. E., 2001)*

## 2.2 Strukturen im Gehirn

Der Ansatz von Kohonen orientiert sich an Erkenntnissen aus der biologischen Signalverarbeitung. Reize, die unser Nervensystem erreichen, haben eine multidimensionale Struktur. Im menschlichen Auge beispielsweise befinden sich mehrere Millionen lichtempfindlicher Sinneszellen, welche kontinuierlich Informationen an das Gehirn weiterleiten. Die Nervenzellen in der Grosshirnrinde, welche diese Reize weiterverarbeiten, sind topologisch häufig linear oder planar untereinander verbunden und liegen nahe bei einander (Kohonen & Honkela, 2007).

Aktive Gruppen von Nervenzellen regen verbundene Zellen in der nahen Nachbarschaft an und unterdrücken die Aktivität von Gruppen, die weiter entfernt liegen. Dieses Verhalten nennt sich laterale Hemmung. Ähnliche Reize werden so von Neuronen verarbeitet, die sich nahe beieinander befinden (Wikipedia, 2016).

## 2.3 Lernalgorithmus im Detail

### 2.3.1 Stichprobe

Der Ausgangspunkt des Lernalgorithmus ist eine Stichprobe $M = \{m_1, \dots, m_n\}$, deren Elemente n-dimensionale Vektoren sind:

$$m_i = \begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_p \end{pmatrix} \mid x_i \in \mathbb{R}$$

### 2.3.2 Neuronen

Dazu ist eine Menge von Neuronen gegeben. Jedes Neuron besteht aus einem Gewichtsvektor $w_i$ und einer Position auf dem Gitter $k_i$. Der Gewichtsvektor hat dieselbe Dimension, wie die Elemente der Stichprobe. Die Werte dieses Vektors werden initialisiert mit Zufallswerten.

$$N = \{n_1, \dots, n_q\}$$

$$n_i = (w_i, k_i) \mid w_i = \begin{pmatrix} y_1 \\ y_2 \\ \dots \\ y_p \end{pmatrix}, y_i \in \mathbb{R}\ und\ k_i \in K$$

Wobei $K$ abhängig von der Topologie des Gitters ist. Bei einem planaren Gitter wäre $k_i$ ein Vektor im zweidimensionalen Raum ($\mathbb{R}^2$). Der Raum des Gitters kann von den beiden Vektoren aufgespannt werden, die sich aus der Hauptkomponentenanlyse (PCA) der Stichproben ergeben. Die Neuronen sind untereinander verbunden, so dass ein Gitter entsteht. Jedes Neuron kann 4, 6 oder 8 direkte Nachbaren haben.

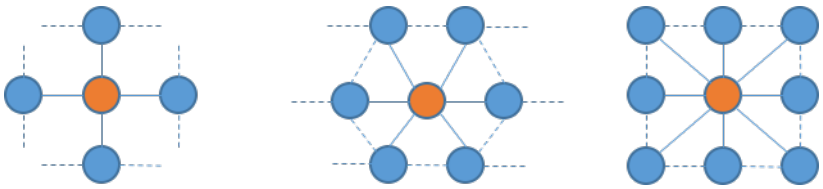

*Abbildung 5: Drei verschiedene Nachbarschafts-Konfigurationen*

Aber auch andere Topologien, wie zum Beispiel lineare und kugelförmige sind je nach Stichprobe sinnvoll. Damit der Algorithmus ein nützliches Resultat produziert, müssen Informationen über die Topologie der Stichprobe bekannt sein.

### 2.3.3 Lernphase

Meistens wird das Kohonen Netz mit der gleichen Stichprobe in mehreren Epochen trainiert. Das heisst jedes Element aus der Stichprobe wird mehrmals eingegeben. Eine Epoche der Lernphase ist folgendermassen definiert (Wikipedia, 2016):

1. Wähle gleichverteilt zufällig ein Element $m^*$ aus der Stichprobe
2. Suche das Neuron $n_j$, dessen Gewichtsvektor den kleinsten Abstand zum gewählten Element besitzt. Die Metrik für das Distanzmass kann beliebig gewählt werden; häufig wird der euklidische Abstand der Vektoren berechnet.
$$w_j = \min\{d_{euklid}(m^*, w_j) \mid j = 1, \dots, q\}$$
3. Nun verschiebt sich das Neuron des gefundenen Gewichtsvektors und die benachbarten Neuronen in Richtung des Stichprobenelements. Die Verschiebung der Neuronen ist der Lernschritt und kann mit folgender Adaptionsregel ausgedrückt werden:
$$w_l^{t+1} = w_l^t + \epsilon^t * h^t * (m^* - w_l^t)$$
4. Reduktion der Lernrate und des Nachbarschaftsradius
5. Zurück zu Schritt 1. Solange es noch Elemente hat, die noch nicht durchlaufen wurden.

Im Lernschritt wird das gefundene Neuron (gelb) an das gewählte Element (weisser Punkt) aus der Stichprobe (blau) ‚herangezogen'. Gleichzeitig zieht das Neuron seine Nachbarschaft ebenfalls zum Element mit. Nach endlich vielen Iterationen hat das Gitter die Verteilung der Stichprobe approximiert (Mcld, 2010)

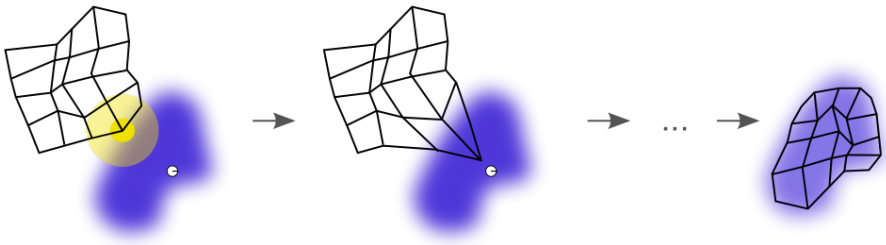

*Abbildung 6: Illustration des Trainings eines Kohonen Netzes. Quelle: (Mcld, 2010)*

### 2.3.4 Zeitkomponente $t$

Die hochgestellten Indices in der Adaptionsregel in Schritt 3. des Algorithmus weisen darauf hin, dass die Lernrate $\epsilon$ und Entfernungsgewichtungsfunktion $h$ abhängig sind von der bisher verstrichenen Zeit. Nach jedem Adaptionsschritt wird ein Zähler inkrementiert. Dieser Schrittzähler wird nach einer Epoche nicht zurückgesetzt. Es werden die absolut durchgeführten Schritte gezählt. Nach der Durchführung von einer maximalen Anzahl Adaptionsschritten wird das Training terminiert. Typischerweise wird jedes Stichprobenelement einige hundert bis tausend Mal eingegeben.

### 2.3.5 Lernrate $\epsilon$

Die Lernrate ist abhängig von den bisher durchgeführten Adaptionsschritten. Nach jedem Schritt wird diese geringfügig reduziert. Am Anfang des Trainings ist es nützlich eine hohe Lernrate zu haben, damit sich das Gitter möglichst schnell an die grobe Verteilung des Stichprobenraumes anpassen kann. Später sollen sich die Neuronen feiner anschmiegen und währenddessen das Gitter gleichmässig über den Stichprobenraum legen, um eine topologierhaltende Kartierung des Stichprobenraumes zu realisieren

Eine Regel für die Reduktion der Lernrate könnte so aussehen (Wikipedia, 2016):

$$\epsilon^t = \epsilon_{start} * \left(\frac{\epsilon_{end}}{\epsilon_{start}}\right)^{\frac{t}{t_{max}}}$$

Für eine Start-Lernrate $\epsilon_{start} = 0.9$,, End-Lernrate $\epsilon_{end} = 0.1$ und wenn insgesamt 100'000 Adaptionsschritte durchgeführt werden sollen, sähe der Verlauf so aus:

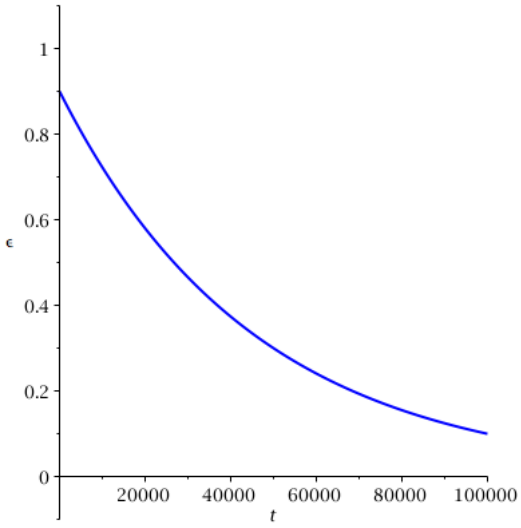


*Abbildung 7: Reduktion der Lernrate während der gesamten Lernphase*

Denkbar wären auch lineare Funktionen, halbe Gaussglocken oder Treppenfunktionen.

### 2.3.6 Nachbarschaftsradius $\mathcal{R}$

Im Schritt 3. des Algorithmus zieht das gefundene Neuron seine Nachbarschaft in Richtung des Stichproben-Elements mit. Nachbarn sind alle Neuronen, die in $\mathcal{R}$ Schritten erreichbar sind. Zwei direkt verbundene Neuronen haben die Distanz $d_{\mathcal{R}}(n_i, n_k) = 1$. Mit diesen Annahmen können wir die Nachbarschaftsverhältnisse eines Neurons so visualisieren:

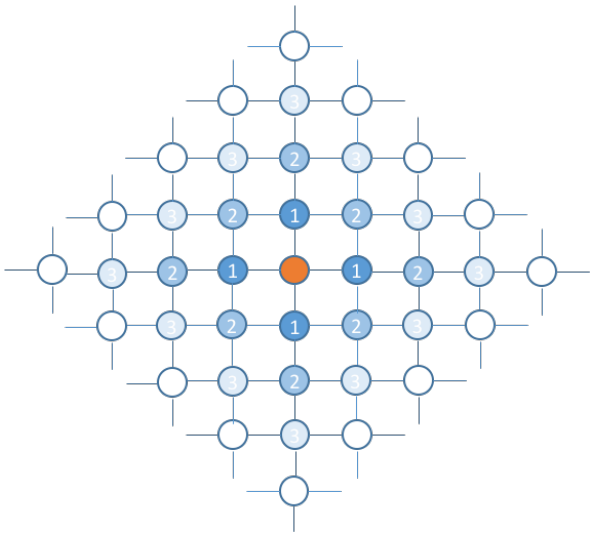


*Abbildung 8: Illustration der Nachbar-Distanz von Neuronen im Kohonen Netz*

Analog zur Lernrate ist es sinnvoll den Nachbarschaftsradius zu Beginn des Trainings gross zu bestimmen, damit grosse Teile des Gitters zum Stichprobenraum hingezogen werden. Eine Regel zur Reduktion des Radius könnte so aussehen (Wikipedia, 2016):

$$\mathcal{R}^t = \left\lfloor \mathcal{R}_{start} * \left(\frac{\mathcal{R}_{end}}{\mathcal{R}_{start}}\right)^{\frac{t}{t_{max}}} \right\rfloor$$

Für einen Start-Radius $\mathcal{R}_{start} = 16$, einen Minimal-Radius von $\mathcal{R}_{end} = 1$ und wenn wieder 100'000 Schritte ausgeführt werden, ergibt sich folgende Treppenkurve:

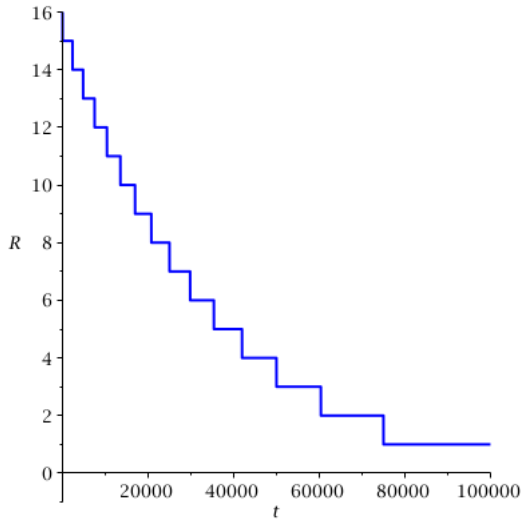


*Abbildung 9: Reduktion des Nachbarschaftsradius während der gesamten Lernphase*

### 2.3.7 Entfernungsgewichtungsfunktion $h$

Neuronen, die topologisch näher benachbart sind zum gefundenen Neuron $n_j$, sollen im Adaptionsschritt stärker angepasst werden. Mit der Entfernungsgewichtungsfunktion $h$. Mit Hilfe einer Gaussfunktion, passen sich näher benachbarte Neuronen stark ans gewählte Stichproben-Element an, während die Anziehung entferntere gehemmt wird (Lippe, 2007):

$$h^t = e^{-\left(\frac{d_{\mathcal{R}}(n_i,n_k)}{\mathcal{R}^t}\right)^2}$$

Diese Gewichtungsfunktion ist abhängig vom Nachbarschaftsradius und verändert sich deshalb während der Lernphase. Veranschaulichung: Die Kurve bei verschiedenen Nachbarschaftsradien

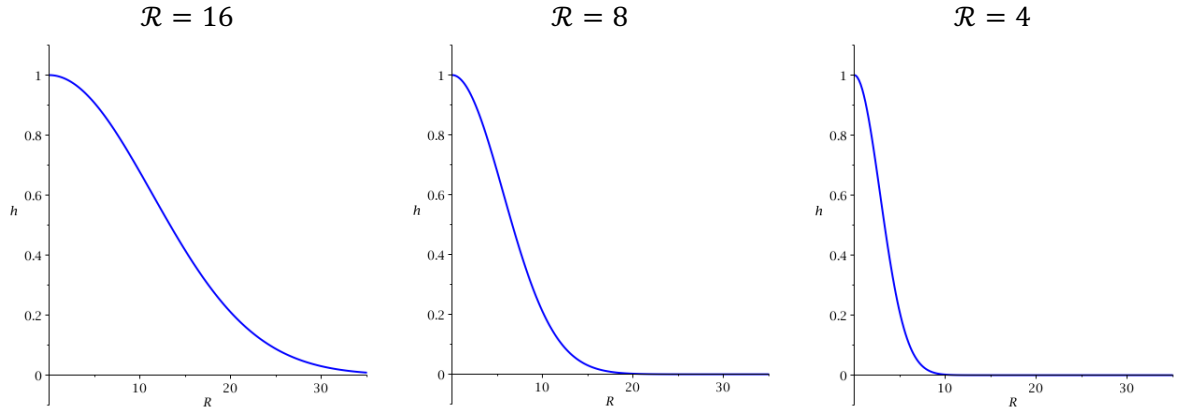


*Abbildung 10: Entfernungsgewichtsfunktion in Abhängigkeit des Nachbarschaftsradius*

## 2.4 Klassifizierung, Mapping und Einsatzgebiet

Nach der Durchführung des Lernalgorithmus hat sich das Kohonen Netz an den Stichprobenraum angenähert. Das tiefdimensionale Gitter lässt sich für Menschen interpretierbar als zwei- oder dreidimensionale Karte darstellen.

Das trainierte Kohonen Netz präsentiert gefundene Cluster aber deren Bedeutung und die Unterteilung in disjunkte Klassen wird dem Nutzer überlassen. Nachdem Klassen definiert sind, kann ein neuer Datensatz an durch das Netz Verarbeitet werden: Der neue Vektor wird mit allen Gewichtsvektoren des Netzes verglichen. Das ähnlichste Neuron gewinnt. Der neue Datensatz wird derjenigen Klasse zugeordnet, der das Gewinnerneuron angehört.

Kohonens selbstorganisierende Karten finden in Verwendung in der Custer-Analyse und bei Quantisierungsalgorithmen zur Farbreduktion (Wikipedia, 2016).

# 3 Implementation

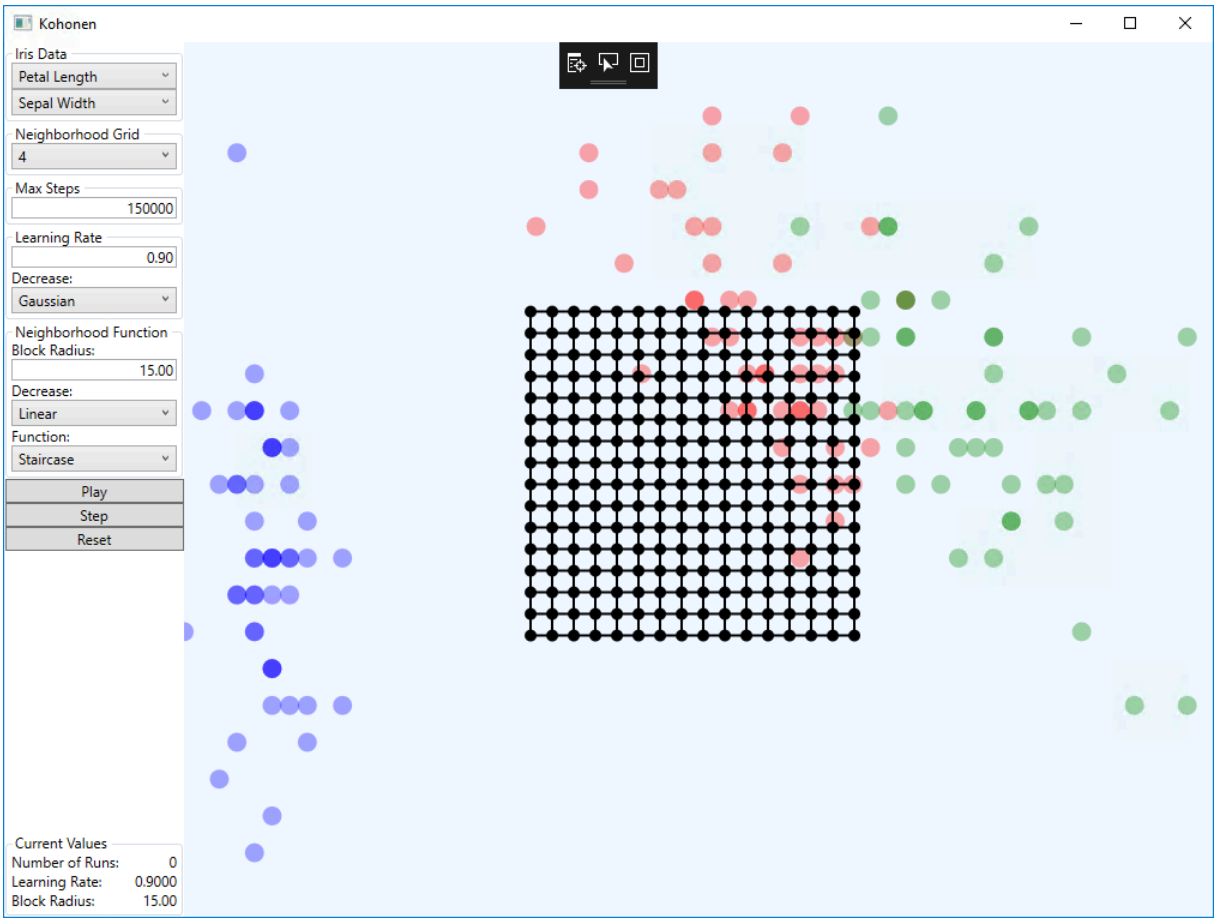


*Abbildung 11: Bildschirmfoto der Benutzeroberfläche für die Simulation des Lernalgorithmus*

Um den Lernalgorithmus genau zu verstehen und um in das laufende Geschehen hineinschauen zu können, bedurfte es das Erstellen einer einfach bedienbaren Test-Applikation. Die vielen wählbaren Startparameter der Kohonen Netze sind auf der Benutzeroberfläche leicht konfigurierbar:

- Stichproben: Wählbar sind verschiedene Darstellungen des bekannten Iris-Datensatzes (Wikipedia, 2016).
- Terminierungsbedingung: Maximal durchzuführende Adaptionsschritte
- Start-Lernrate: Die Lernrate verringert sich nach dem Verfahren aus Kapitel 2.3.5. Die End-Lernrate ist festgelegt auf eine sehr kleine positive Double Zahl
- Start-Blockradius: Der Blockradius verringert sich nach dem Verfahren aus Kaptel 2.3.6. Der End-Blockradius ist festgelegt auf 1 (gefundenes Neuron und 0 Nachbaren).

Die Applikation hat keine Abhängigkeiten zu externen Programmbibliotheken und führt, für ein einfaches Debugging, alle Operationen auf einem einzigen Thread aus. Der Programmcode ist auf Github veröffentlicht:

https://github.com/uzapy/ch.bfh.bti7311.f2016.kohonen

## 3.1 Technologie

Die Applikation baut auf dem .NET-Technologie von Microsoft auf. Der Programmcode ist in C# geschrieben; die Benutzeroberfläche ist als WPF beschrieben und eine Datei-basierte Datenbank persistiert die Stichproben zwischen den Durchführungen.

## 3.2 Lernalgorithmus

Dieser Programm-Ausschnitt wird einmal pro Epoche ausgeführt.

```
public void Algorithm()
{
    // Nachbarschaft des ausgewählten Neurons: N+t
    Dictionary<Neuron, double> Neighborhood;

    // Zufällig durch die Input-Daten gehen
    IrisData = Shuffle(IrisData);

    foreach (IrisLib iris in IrisData)
    {
        // Rt: Adaptionsradius um das Gewinner-Neuron auf der Karte
        BlockRadius = BlockRadiusStart *
            Math.Pow((BlockRadiusEnd / BlockRadiusStart), (Steps / StepsMax));

        // Die zeitabhängige Lernrate εt
        LearningRate = LearningRateStart *
            Math.Pow((LearningRateEnd / LearningRateStart), Steps / (StepsMax));

        // Das Neuron mit der maximalen Erregung wird ermittelt.
        // Die NeuronMap-Liste wird neu sortiert. Dann wird der erste in der Liste gewählt
        // Die Metrik ist der euklidische Abstand zum Input-Vektor.
        // Das Position Atribut der Neuronen und der Iris sind mehrdimensionale Vektoren.
        Neuron closest = NeuronMap.OrderBy(n => (n.Position - iris.Position).Length).First();

        // Alle Nachbaren innerhalb des Blockradius des gewählten Neurons (inklusive selbst) auslesen.
        Neighborhood = closest.GetNeighborhood(BlockRadius, new Dictionary<Neuron, double>());

        // Neuron und dessen Nachbarschaft ein Stück in die Richtung des Input-Vektors bewegen
        foreach (KeyValuePair<Neuron, double> n in Neighborhood)
        {
            // Die zeitabhängige Entfernungsgewichtungsfunktion ht
            double weight = Math.Exp(-Math.Pow(n.Value / BlockRadius, 2));

            n.Key.Position = n.Key.Position + LearningRate * weight * (iris.Position - n.Key.Position);
        }

        // Schrittzähler inkrementieren
        Steps++;
    }
}
```

## 3.3 Zufällig ein Element wählen

Damit die Reihenfolge, mit welcher die Input-Vektoren verarbeitet werden, zufällig ist, wird die Liste vor der Schleife durcheinandergebracht. Zum Einsatz kommt das Fischer-Yates-Shuffle (Wikipedia, 2016)

```
private Random random = new Random();

private List<T> Shuffle<T>(List<T> list)
{
    int current = list.Count;
    while (current > 1)
    {
        current--;
        int other = random.Next(current + 1);
        T otherObject = list[other];
        list[other] = list[current];
        list[current] = otherObject;
    }
    return list;
}
```

## 3.4 Nachbarschaft abrufen

Mit der rekursiven Methode GetNeighborhood kann die Nachbarschaft eines Neurons abgerufen werden:

```
// blockRadius: Umkreis-Radius der Nachbarschaft
// neighborhood: Kollektion der Neuronen in der Nachbarschaft
internal Dictionary<Neuron, double> GetNeighborhood(double blockRadius,
     Dictionary<Neuron, double> neighborhood)
{
    // Methdenblock nur ausführen, wenn das Neuron noch nicht in der Kollektion vorhanden ist.
    if (!neighborhood.ContainsKey(this))
    {
        // Aktuelles Neuron zur Nachbarschaft hinzufügen, mit der Entfernung zum aufrufenden Neuron
        neighborhood.Add(this, BlockRadiusStart - blockRadius);
        // Blockradius verringern
        blockRadius--;
        // Falls der Aktuelle Blockradius grosser als 1 ist,
        // rekursiv die Nachbarschaft des aktuellen Neurons hinzufügen
        if (blockRadius > 1)
        {
            // Für alle Nachbaren des aktuellen Neurins ausführen.
            foreach (Neuron n in Neighbours)
            {
                n.GetNeighborhood(blockRadius, neighborhood);
            }
        }
    }
    // Kollektion zurückgeben
    return neighborhood;
}
```

# 4 Abbildungsverzeichnis

Abbildung 1: Schematische Darstellung einer Nervenzelle. Quelle: (Wikipedia, 2016) ..... 1
Abbildung 2: Darstellung eines künstlichen Neurons mit seinen Elementen. Quelle: (Wikipedia, 2016) ..... 2
Abbildung 3: Darstellung eines mehrschichtigen neuronalen Netzes. Quelle: (Bunke, 2006) ..... 4
Abbildung 4: Portrait von Teuvo Kohonen, 2000. Quelle: (Kohonen D. E., 2001) ..... 5
Abbildung 5: Drei verschiedene Nachbarschafts-Konfigurationen ..... 6
Abbildung 6: Illustration des Trainings eines Kohonen Netzes. Quelle: (Mcld, 2010) ..... 7
Abbildung 7: Reduktion der Lernrate während der gesamten Lernphase ..... 8
Abbildung 8: Illustration der Nachbar-Distanz von Neuronen im Kohonen Netz ..... 9
Abbildung 9: Reduktion des Nachbarschaftsradius während der gesamten Lernphase ..... 9
Abbildung 10: Entfernungsgewichtsfunktion in Abhängigkeit des Nachbarschaftsradius ..... 10
Abbildung 11: Bildschirmfoto der Benutzeroberfläche für die Simulation des Lernalgorithmus ..... 11

# 5 Literaturverzeichnis

Bunke, P. H. (2006). *Künstliche Intelligenz.* Bern, Bern, Switzerland: Universität Bern.

Kohonen, D. E. (2001). *Laboratory of Computer Science*. Abgerufen am 22. 5 2016 von http://www.cis.hut.fi/research/som-research/teuvo.html

Kohonen, D., & Honkela, T. (2007). *Scholarpedia*. Abgerufen am 22. 5 2016 von http://www.scholarpedia.org/article/Kohonen_network

Lippe, P. D. (2007). Abgerufen am 22. 5 2016 von Westfälische Wilhelms-Universität, Münster: http://wwwmath.uni-muenster.de:8010/Professoren/Lippe/lehre/skripte/wwwnnscript/strfx/Kohonen.html

Mcld. (2010). *An illustration of the training of a self-organizing map.* Wikipedia.

Prof. Dr. Gauglitz, G., & Jürgens, C. (1999). *Neuronale Netze - Einführung*. Abgerufen am 22. 5 2016 von http://www.chemgapedia.de/vsengine/vlu/vsc/de/ch/13/vlu/daten/neuronalenetze/einfuehrung.vlu/Page/vsc/de/ch/13/anc/daten/neuronalenetze/snn1_6.vscml.html

Wikipedia. (19. 4 2016). Abgerufen am 22. 5 2016 von https://de.wikipedia.org/wiki/K%C3%BCnstliches_Neuron#Modellierung

Wikipedia. (19. 4 2016). Abgerufen am 22. 5 2016 von https://de.wikipedia.org/wiki/K%C3%BCnstliches_Neuron#Biologische_Motivation

Wikipedia. (10. 5 2016). Abgerufen am 22. 5 2016 von https://de.wikipedia.org/wiki/Selbstorganisierende_Karte#Laterale_Umfeldhemmung

Wikipedia. (10. 5 2016). Abgerufen am 22. 5 2016 von https://de.wikipedia.org/wiki/Selbstorganisierende_Karte#Formale_Beschreibung_des_Trainings

Wikipedia. (3. 5 2016). (D. v. problems, Produzent) Abgerufen am 22. 5 2016 von https://en.wikipedia.org/wiki/Iris_flower_data_set

Wikipedia. (8. 4 2016). Abgerufen am 22. 5 2016 von https://en.wikipedia.org/wiki/Fisher%E2%80%93Yates_shuffle

Wikipedia. (10. 4 2016). Abgerufen am 22. 5 2016 von https://de.wikipedia.org/wiki/Selbstorganisierende_Karte