Institut für Parallele und Verteilte Systeme
Universitätsstraße 38
D-70569 Stuttgart

Studienarbeit Nr. 2355

Verwendung von hierarchischen selbstorganisierenden Karten zur Erkennung von komplexen Aktionen

Louis Bergmann

Studiengang:	Informatik
Prüfer:	Prof. Dr. rer. nat. habil. Paul Levi
Betreuer:	Dipl.-Inf. Kai Häussermann
begonnen am:	25.10.2011
beendet am:	23.04.2012
CR-Klassifikation:	I.2.6, I.2.9, I.5.3

Zusammenfassung

Abstract

Selbstorganisierende Karten sind ein noch relativ junges und sehr aktives Forschungsgebiet. Bisher wurden einige Methoden vorgestellt, um mit Hilfe von SOMs Zeitreihen zu verarbeiten. In dieser Arbeit soll ein Ansatz vorgestellt werden, der zwei selbstorganisierende Karten verwendet. Die erste verarbeitet Posen, die zweite jeweils eine geordnete Menge dieser Posen, nämlich Gesten. Interessant an SOMs ist, dass sie völlig unüberwacht lernen können. Im Vergleich zu klassischen Verfahren zur Dimensionreduktion oder zum Clustering arbeitet eine SOM auch wesentlich effizienter. Im Rahmen des RoboEarth Projektes, wird deshalb in dieser Arbeit untersucht, wie Gesten mit der beschriebenen Hierarchie von SOMs gelernt, erkannt und verglichen werden können. Zum Sammeln von realistischen Bewegungsdaten wird die Microsoft Kinect verwendet. Schließlich wird eine Software entwickelt, um diese Daten auf die besagte Weise zu verarbeiten und den Ansatz zu testen.

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Der Mensch hat ein nahezu unerschöpfliches Repertoire von verschiedensten Bewegungen. Darunter befinden sich einige Standardbewegungen. Diese nur unscharf zu definierende Gruppe beinhaltet z.B. das Gehen, das Greifen nach einem Objekt, Nahrungsaufnahme, Begrüßungsrituale, Positionswechsel zwischen liegender, sitzender und stehender Haltung oder das Öffnen von Türen.
Auch wenn der Mensch nur zwischen Bewegungen aus dieser kleinen Menge von Bewegungen unterscheiden müsste, ist dies ein komplexes Problem. Eine Bewegung wird nie zweimal hintereinander exakt gleich ausgeführt, weder in ihrer Form noch in ihrer Geschwindigkeit. Durch die hohe Anzahl an Freiheitsgraden und unterschiedlicher Körperformen, hat der Mensch hier mit großen Ungenauigkeiten zu kämpfen. Aus diesem Grund lässt sich nicht jede Bewegung ein mal lernen und abspeichern, um zukünftig ausgeführte - oder bei anderen Personen betrachtete - mit diesen direkt vergleichen und dadurch zu klassifizieren.
Sowohl das Lern- als auch das Erkennungsverfahren muss eine große Unschärfe in der zeitlichen sowie in der räumlichen Dimension erlauben. Von theoretischer Seite betrachtet handelt es sich hier um das Problem des Lernens und Vergleichens von allgemeinen Zeitreihen höherer Dimension.
Eine Zeitreihe besteht aus einer zeitlich geordneten Folge von Momentaufnahmen einer oder mehreren statistischen Variablen. Sie bildet in elektronischen Systemen immer einen zeitdiskreten Verlauf ab. In dieser Studienarbeit sollen darin Gesten abgebildet werden.
Diese Arbeit widmet sich neben dem Lernen und Erkennen vor allem dem Vergleich von Gesten, genauer gesagt mit der Bildung einer Distanzmetrik für Gesten unterschiedlicher

Form und Dauer. Mit Geste ist in dieser Arbeit eine allgemeine Bewegung mit bestimmter Dauer gemeint. Zudem unterliegt die Erfassung der Geste den Beschränkungen des Kinect-Sensors. Dazu später mehr.

1.2 Das RoboEarth Projekt

Diese Arbeit geht aus dem *RoboEarth* genannten Projekt hervor. Das Projekt beschäftigt sich mit der Frage der Machbarkeit von einem globalen Wissensspeicher für Roboter. Verschiedenartige Roboter speichern selbst erlerntes Wissen in den globalen Speicher und können Wissen anderer Roboter wieder davon abrufen.
Die Datenbank enthält folgende Informationen:

- Bilder und Objektmodelle zur Objekterkennung
- Navigationsdaten wie Karten oder Weltmodelle
- Prozesse wie Aktionspläne, Informationen zur Manipulierbarkeit von Objekten oder einzelne Bewegungen.

Außerdem kann diese Wissensbasis weitere intelligente Dienste anbieten. Denkbar wäre z.B. offline learning oder Bilder mit Metadaten zu versehen.

1.3 Aufbau der Arbeit

Im zweiten Kapitel werden selbstorganisierende Karten (auch self-organizing maps bzw. SOMs) beschrieben. Dies umfasst neben geschichtlichen Informationen die mathematische Definition, das Lernverfahren sowie die Fähigkeiten von SOMs.

Verschiedene Verfahren zur Gestenerkennung und das Konzept des in der Arbeit vorgestellten Verfahrens wird im dritten Kapitel beschrieben. Dabei wird auch auf einige Probleme eingegangen.

Im vierten Kapitel wird das Softwaredesign und die gemachten Experimente skizziert.

Zudem werden die Experimente ausgewertet.

Am Ende gibt es eine Zusammenfassung und einen Ausblick.

2 Self Organizing Map (SOM)

2.1 Geschichte

Die SOM [KSH01] wurde ursprünglich von dem finnischen Ingenieur *Teuvo Kohonen* entwickelt, der das erste Paper dazu 1982 veröffentlichte. Die selbstorganisierende Karte oder Kohnen-Karte sollte ein Modell für die räumliche Organisationsstruktur sein, wie sie vor allem im zerebralen Cortex zu finden ist. Obwohl es bereits ähnliche Modelle gab, waren die SOMs ein wichtiger Schritt, da die Fähigkeit zur selbstorganisation in älteren Modellen wie das von Malsburg (1973, [vdM73]) und Amari (1980, [Ama80] schlechter ausgeprägt war.
Die grundlegende Entwicklung, die den SOMs ihre Mächtigkeit brachte, war zwei Teilsysteme zu einem zu verknüpfen. Das erste System ist ein künstliches neuronales Netzwerk nach dem *Winner-Takes-All* Prinzip, bei dem das am stärksten aktivierte Neuron das einzig relevante Neuron ist. Das zweite Teilsystem ist der Lernalgorithmus, der das Gewinnerneuron und innerhalb des Gitters die davon benachbarten Neuronen in Richtung des Eingabesignals bewegt.

2.2 Mathematische Definition

Eine SOM besteht aus einem Netz $\mathcal{W}$ von Neuronen, die jeweils ein gewisses Gewicht $\vec{w}_{\vec{r}} \in \mathbb{R}^n$ besitzen. Im Folgenden wird der Begriff Neuron mit mit dem zugehörigen Gewicht gleichgesetzt. $\vec{r} \in \mathbb{R}^d$ gibt dabei den Index bzw. die Position innerhalb des Netzes an. Das Netz ist ein d-dimensionales regelmäßiges Gitter. In Abbildung 2.3 ist eine zweidimensionale SOM abgebildet; $\vec{r}$ ist in diesem Fall also aus $\mathbb{R}^2$.

Als **Eingabe** erhält das Netz einen Vektor aus dem selben Raum wie der der Gewichtsvektoren $\vec{w}_{\vec{r}}$, also auch aus $\mathbb{R}^n$.

Als **Ausgabe** der SOM erhält man den Index $\vec{r}$ des Gewinnerneurons.

Eine allgemeine SOM W bildet also eine Funktion folgender Form:

$$W : \mathbb{R}^n \rightarrow \mathbb{R}^d ; d <= n \tag{2.1}$$

Es sollte gelten $d <= n$. Wäre d, also die Dimension des Gitters größer, würde dies nur die Komplexität steigern, jedoch keine weiteren Informationen liefern. Ist $d = n$ wird von der SOM nur die Eigenschaft der Vektorquantisierung verwendet, jedoch nicht die Fähigkeit zur Dimensionsreduktion. Erst bei $d < n$ oder besser $d << n$ entfaltet sich das volle Potenzial der SOM.

Die Neuronen stehen für Prototypen im Merkmalsraum $\mathbb{R}^n$. Die Menge der Prototypen definiert ein Voronoi-Diagramm ([OBSC00]), d.h. eine Zerlegung des Raums in die Menge $\mathtt{V} = \{V_{\vec{r}_i} | i = 1, \ldots, n\}$ der Voronoizellen der Gewichtsvektoren $\vec{w}_{\vec{r}}$, so dass

$$\mathbb{R}^n = V_{\vec{r}_1} \cup V_{\vec{r}_2} \cup \cdots \cup V_{\vec{r}_n}.$$

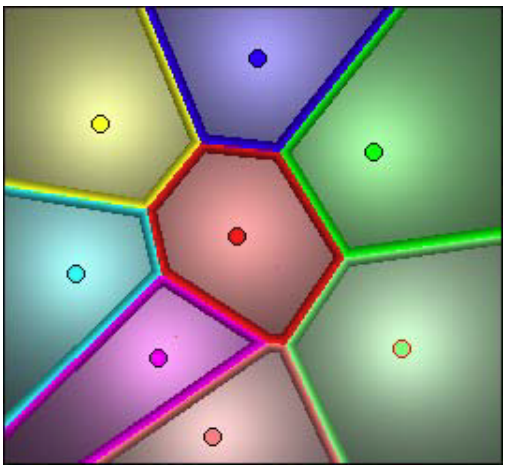


Abbildung 2.1: Koloriertes Voronoi Diagramm. [TW01] Jede Farbfläche repräsentiert eine Voronoizelle. Die Punkte sind jeweils z.B. die Gewichte von verschiedenen Neuronen.

Die Voronoizelle $V_{\vec{r}}$ des Vektors $\vec{w}_{\vec{r}}$ ist definiert als das Gebiet aller $\vec{x}$, die zu $\vec{w}_{\vec{r}}$ maximal den gleichen Abstand haben wie zu allen anderen Neuronen:

$$V_{\vec{r}_i} = \left\{ \vec{x} \,|\, d(\vec{x}, \vec{w}_{\vec{r}_i}) \leq d(\vec{x}, \vec{w}_{\vec{r}_j}), \forall i \neq j \right\} \tag{2.2}$$

In Abbildung 2.1 ist ein beispielhaftes Voronoi Diagramm zu sehen. Betrachtet man die Übereinstimmung des Eingabevektors mit dem Maß für die Erregung des Neurons, so ist die Voronoizelle der Bereich im Merkmalsraum, in dem das jeweilige Neuron das am stärksten erregte Neuron im gesamten Netz ist. Jenes Neuron wird Gewinnerneuron oder *Best-Matching-Unit* genannt. Die Voronoizelle ist für Neuronen in einer selbstorganisierenden Karte somit gleichbedeutend mit deren rezeptiven Feld.

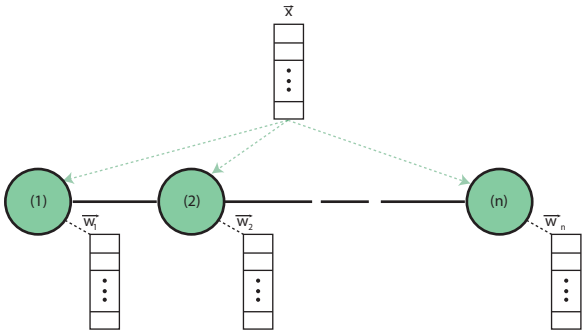


Abbildung 2.2: Eindimensionale selbstorganisierende Karte. Alle Neuronen in der Karte sind mit der Eingabe $\vec{x}$ verbunden. Außerdem ist jedem Neuron i ein Gewicht $\vec{w}_i$ zugeordnet. Dieses Gewicht repräsentiert wiederum einen Punkt im Eingaberaum.

2.3 Lernalgorithmus

Eine SOM besteht wie bereits erwähnt aus den zwei Teilsystemen

- *Winner-Takes-All* Netz und
- dem Lernalgorithmus, der die Gewichte der Neuronen anpasst.

Um das Gewinnerneuron zu ermitteln wird eine Distanzmetrik benötigt, die den Abstand zweier Vektoren aus dem Raum $\mathbb{R}^n$ definiert. Meist wird hierfür der euklidische Abstand

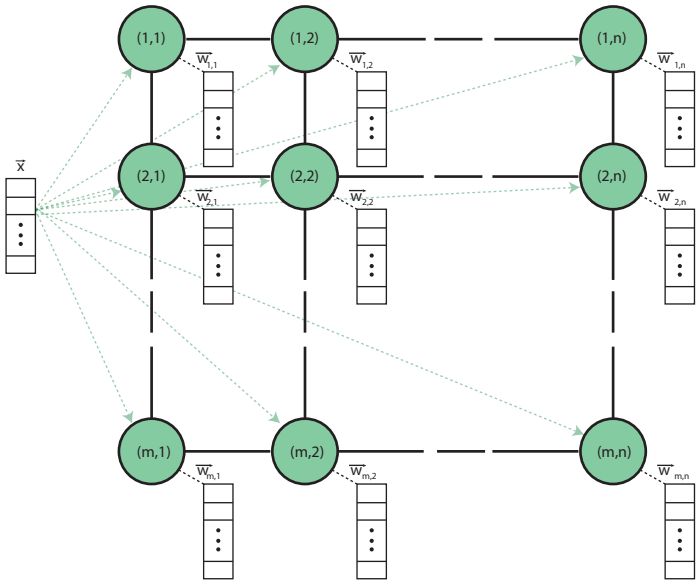


Abbildung 2.3: Zweidimensionale selbstorganisierende Karte. Alle Neuronen in der Karte sind mit der Eingabe $\vec{x}$ verbunden. Außerdem ist jedem Neuron (i, j) ein Gewicht $\vec{w}_{(i,j)}$ zugeordnet. Dieses Gewicht repräsentiert wiederum einen Punkt im Eingaberaum.

$$d(\vec{x}, \vec{y}) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots + (x_n - y_n)^2} = ||\vec{x} - \vec{y}|| \tag{2.3}$$

verwendet. Das Neuron dessen Gewicht den kleinsten Abstand zum Eingabevektor hat, ist dann das Best-Matching-Unit bzw. das Gewinnerneuron $\vec{w}_{\vec{r}_{win}}$ mit

$$\vec{r}_{win}(\vec{x}) = \left\{ \vec{r} \,|\, \min_{\vec{r}_i} d(\vec{w}_{\vec{r}_i}, \vec{x}) \right\} \tag{2.4}$$

2.3.1 Weitere Metriken

Es sind auch andere Abstandsmetriken verwendbar. Allgemein kann die p-Norm verwendet werden. Sie hat die Form:

$$||\vec{x}||_p = \left(\sum_{i=1}^{n} x_i^p \right)^{1/p} \quad |1 < p < \infty \tag{2.5}$$

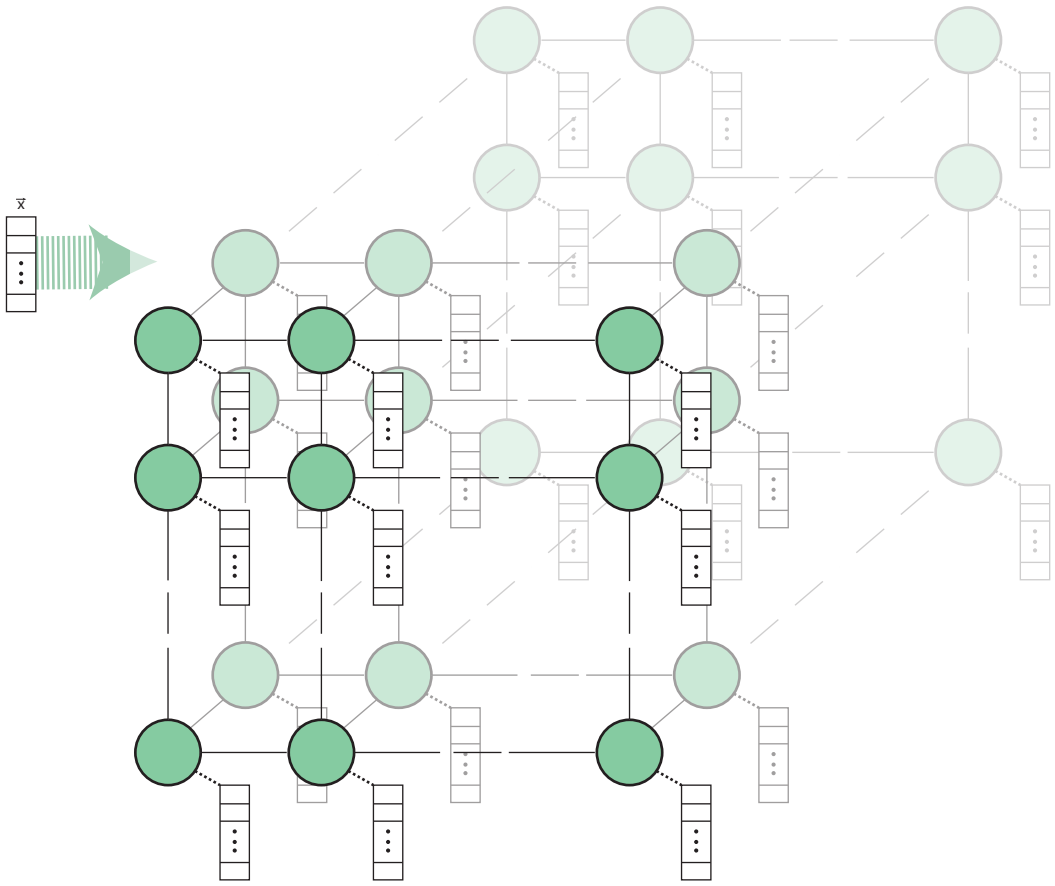


Abbildung 2.4: Dreidimensionale selbstorganisierende Karte. Alle Neuronen in der Karte sind mit der Eingabe $\vec{x}$ verbunden. Außerdem ist jedem Neuron (k, i, j) ein Gewicht $\vec{w}_{(k,i,j)}$ zugeordnet. Dieses Gewicht repräsentiert wiederum einen Punkt im Eingaberaum.

Die euklidsche Abstandsnorm ist ein Sonderfall dieser, nämlich hat für diesen Fall p den Wert 2. Für $p = 1$ erhält man die Manhattan Distanz. Der Name Manhattan ist angelehnt an den Stadtteil von New York, da die Straßen dort, abgesehen vom Broadway, rechtwinklig zueinander stehen. Um nun von einem Punkt zu einem anderen zu kommen, kann man nicht dem direkten Weg, also die Luftlinie , sondern nur auf Parallelen zu den zwei Achsen x und y gehen. Dies ist in Abbildung 2.5 skizziert. Es ist leicht sichtbar, dass die Manhattan Distanz stets größere Werte liefert als die euklidsche Distanz, da sie

lediglich alle Elemente eines Vektors aufsummiert:

$$||\vec{x}||_1 = \left(\sum_{i=1}^{n} x_i^1 \right)^{1/1} = \sum_{i=1}^{n} x_i \tag{2.6}$$

Der andere Extremfall zu $p = 1$ ist $p = \infty$. Hiermit folgt nämlich

$$||\vec{x}||_\infty = \left(\sum_{i=1}^{n} x_i^\infty \right)^{1/\infty} = \max_i x_i \tag{2.7}$$

. Es wird mit $||\vec{x}||_\infty$ also immer nur das größte Element des Vektors $\vec{x}$ berücksichtigt. Diese Norm ist in Abbildung 2.5 als blaue Linie skizziert. Sie liefert damit gleichzeitig auch die kleinsten Abstandswerte dieser drei p-Normen.

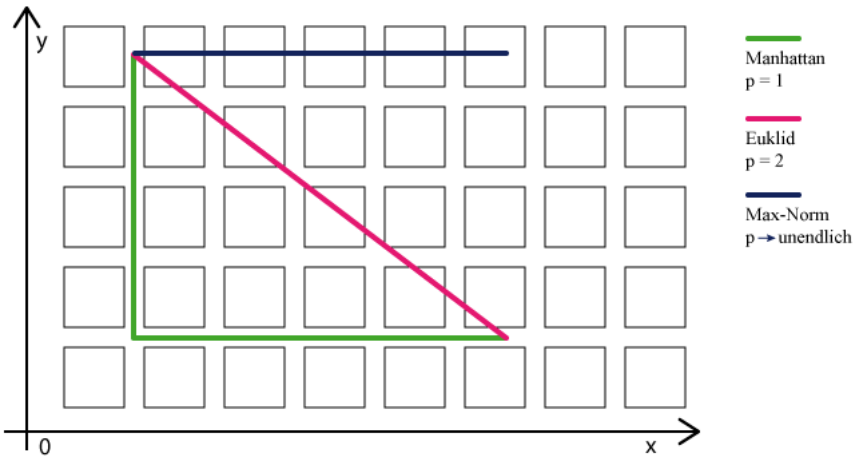


Abbildung 2.5: Skizze der Manhattan Distanz $||\vec{x}||_1$ (grün) im Vergleich mit der euklidschen Distanz $||\vec{x}||_2$ (rot) und der Maximums-Norm $||\vec{x}||_\infty$ (blau).

Häufig wird im Zusammenhang mit SOMs auch die Levensthein Metrik [Lev66] verwendet, welche in [Nav01] nochmals ausführlich beschrieben wird. Hierfür werden drei Operationen der Form $\delta(x, y)$ definiert:

1. Einfügen: $\delta(\epsilon, a)$ (Buchstabe a einfügen)
2. Löschen: $\delta(a, \epsilon)$ (Buchstabe a löschen) und
3. Ersetzen: $\delta(a, b)$ (Buchstabe a mit Buchstabe b ersetzen)

Jede Operation hat die Kosten von 1, sie können also gleichwertig verwendet werden. Die Levensthein Distanz erfüllt auch alle Bedingungen, die eine Metrik erfüllen muss, nämlich:

$$d(x,y) \geq \quad 0 \tag{2.8}$$
$$d(x,y) = \quad d(y,x) \tag{2.9}$$
$$d(x,z) \leq \quad d(x,y) + d(y,z) \tag{2.10}$$

Des weiteren gilt für die Levensthein Metrik

$$0 \leq d(x,y) \leq max(|x|,|y|) \tag{2.11}$$

Der Abstand ist also höchstens so groß wie der längste String lang ist. In der Literatur wird dieses Suchproblem auch *string matching with k differences* genannt.

2.3.2 Anpassung der Gewichte

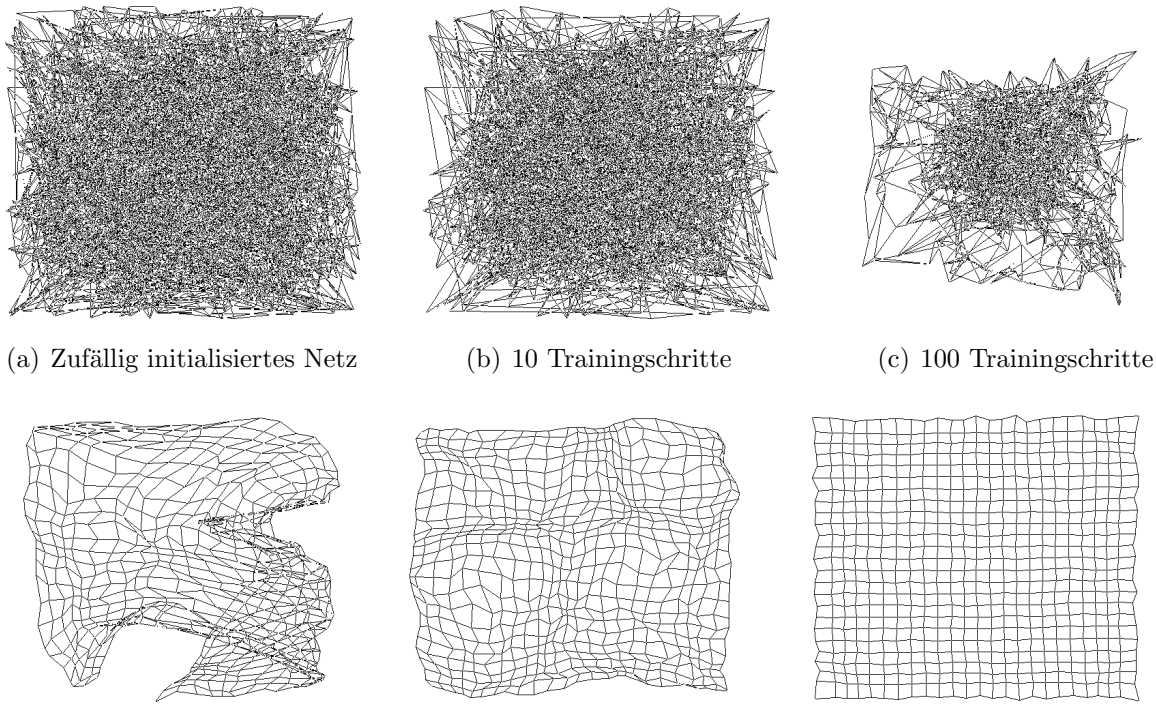
Die Lernfunktion berechnet das Gewicht jeweils eines Neurons und benötigt als Eingabeparameter

- die Koordinaten des Gewinnerneurons $\vec{r}_{win}$,
- den Eingabevektor $\vec{x}$,
- die Lerngeschwindigkeit α,
- der Lernradius σ sowie
- das Gewicht $\vec{w}_{\vec{r}_i}$ das neu berechnet werden soll

und hat folgende Form:

$$\vec{w}_{\vec{r}_i}(t+1) = \vec{w}_{\vec{r}_i}(t) + \alpha \cdot N(\vec{r}_i, \vec{r}_{win}, \sigma) \cdot (\vec{w}_{\vec{r}_{win}}(t) - \vec{w}_{\vec{r}_i}(t)) \tag{2.12}$$

Die Parameter α und σ werden mit jedem Lernschritt kleiner, sind also monoton fallend, wenn eine Konvergenz erreicht werden soll. α liegt im Intervall $(0,1]$. σ ist abhängig von



(a) Zufällig initialisiertes Netz (b) 10 Trainingschritte (c) 100 Trainingschritte

(d) 1000 Trainingschritte (e) 10000 Trainingschritte (f) 100000 Trainingschritte

Abbildung 2.6: Entfaltung einer zweidimensionalen selbstorganisierenden Karte in einem zweidimensionalem Eingaberaum.

der Netzgröße. Je größer σ ist, desto höher ist der Einflussradius des Best-Matching-Unit. Die Funktion $N(\vec{x}, \vec{y})$ kann z.B. die klassische Normalverteilung

$$N(\vec{x}, \vec{y}, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{||\vec{x}-\vec{y}||}{\sigma}\right)^2} \tag{2.13}$$

sein.

Der gesamte Lernalgorithmus sieht nun folgendermaßen aus:

1. Gewinnerneuron ermitteln mit Gleichung 2.4
2. Iterieren über alle Neuronen $\vec{w}_{\vec{r}_i}$ mit Gleichung 2.12

Als Ergebnis werden die Gewichte des Gewinnerneurons und der umliegenden Neuronen in Richtung des Eingabevektors $\vec{x}$ verschoben. Dadurch, dass die in der Gitterstruktur benachbarten Neuronen auch mitgezogen werden, spricht man von einer Entfaltung des Netzes. Dies ist ein Abbildung 2.6 veranschaulicht.

2.4 Vektorquantisierung

Ein Grund für die Anwendung von SOMs ist ihre Generalisierungseigenschaft. Ähnlich zu k-means ordnen SOMs ein Eingabedatum, welches in einem bestimmten Gebiet liegt, einer bereits bekannten Klasse zu. Der Begriff Klasse kann in diesem Zusammenhang zwei Bedeutungen haben. Die erste ist die vermutlich intuitivere, nämlich einer vom Anwender bereitgestellten Bezeichnung für eine Gruppe von sich ähnlichen Daten. Die zweite abstraktere Bedeutung ist die Zuordnung zu einem Gewichtsvektor. Das zugehörige Neuron repräsentiert bereits für sich einen Bereich im Eingaberaum, bezeichnet also eine aus menschlicher Sicht abstrakte Klasse; sie ist im Normalfall wenn überhaupt eine Teilmenge einer bezeichenbaren Klasse. Um eine intuitive Klasse in einer SOM zu kennzeichnen, wird in der Regel einem ganzen Gebiet in der Karte, also einer Gruppe von Neuronen, ein sinnvoller Bezeichner zugeordnet. Dieser Vorgang wird *Labeln* genannt.

Wie bereits erwähnt, wird ein Wertebereich im Eingaberaum einem Gewichtsvektor zugeordnet, also einem Vektor, der ein Repräsentant für eine Menge von Werten aus dessen Raum ist. Diese Eigenschaft nennt man *Vektorquantisierung*([GG91]). Der Ursprungsraum wird nur mit einer Gewissen Anzahl von Vektoren repräsentiert. Vektorquantisierung ist also eine Art von *verlustbehafteter Komprimierung*. Welcher Wertebereich jeweils von einem Vektor repräsentiert wird, hängt, abgesehen von der Anzahl der Vektoren, maßgeblich von dem Verfahren und der verwendeten Distanzmetrik ab. Man verwendet normalerweise den euklidischen Abstand

$$d(\vec{x}, \vec{y}) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots + (x_n - y_n)^2} = ||\vec{x} - \vec{y}||. \tag{2.14}$$

In der Sprachverarbeitung werden die einzelnen Dimensionen zusätzlich noch unterschiedlich gewichtet, um die Eigenschaften des menschlichen Gehörs berücksichtigen zu können. Der Mensch nimmt Töne unterschiedlicher Frequenzen bei selbem Schalldruckpegel nicht gleich laut wahr([Kin27]).

Eine SOM ist ein Beispiel für lernende Vektorquantisierung, wobei Vektorquantisierung natürlich nicht zwingend durch ein Lernverfahren erreicht werden muss. Die naive Methode wäre, in einen Eingaberaum eine gleichmäßige Struktur von Vektoren zu legen. Diese Vektoren würden den umspannenden Teil des Raumes gleichmäßig ausfüllen. Die Vektoren auf der Oberfläche des entstandenen Gitters teilen sich den gesamten Wertebereich des nicht abgedeckten Raumes auf. Sie können also durchaus Werte repräsentieren,

die unendlich groß bzw. klein sind. Dies ist z.B. in $\mathbb{R}$ oder auch $\mathbb{Z}$ der Fall.

Formal betrachtet ordnet die Vektorquantisierung eine Menge von Eingabevektoren jeweils dem nächstgelegenen Codevektor zu. Dieser Codevektor besitzt wiederum einen Index, über den er identifiziert wird. Im Codebuch sind wiederum die Werte der Codevektoren sowie die Zuordnung von selbigen zu Indizes zu finden. Das Problem bei der Komprimierung ist, eine möglichst optimale Verteilung der Codevektoren im Eingaberaum zu finden. Ansätze dazu gibt es viele: Gradientenabstiegsverfahren, Genetische Algorithmen, Selbstorganisierende Karten und weitere.

Die Fähigkeiten der SOM gehen jedoch über die Vektorquantisierung hinaus. Sie erzeugt nämlich zusätzlich eine Ähnlichkeitsmetrik für die Codevektoren. Liegen die Codevektoren innerhalb der Netzstruktur der Karte nebeneinander, sind sie - ausgenommen weniger Ausnahmen - gleichzeitig auch zueinander ähnlich. Sie erhalten die Topologie des Merkmalsraumes.

2.5 Topologieerhaltung

Die Neuronen einer noch ungelernten SOM liegen verteilt im Eingaberaum. Die Lage der Neuronen hängt von der Art der Initialisierung ab, wird aber in der Regel zufällig gewählt. Das Netz liegt also - wie in Abbildung 2.6 a) zu sehen - völlig ungeordnet im Ursprungsraum. Neuronen, die in der Gitterstruktur eigentlich nebeneinander liegen, sind zu Beginn wenn überhaupt nur zufällig benachbart. Nehmen wir den Fall an, dass zwei im Gitter benachbarte Neuronen im Ursprungsraum weit auseinanderliegen: Wir wissen, dass sich auch die zum Gewinnerneuron benachbarte Neuronen zum Eingabedatum bewegen. Dies findet jedoch durch die Funktion $N(\vec{x}, \vec{y}, \sigma)$ bei größer werdendem Abstand eigentlich immer langsamer statt. Da jedoch zusätzlich der Abstand zwischen Eingabedatum und zu berechnendem Neuron einen linearen Einfluss hat, werden weiter entfernt liegende Neuronen trotzdem annähernd so weit zum Eingabedatum bewegt, wie das Gewinnerneuron. Durch diesen Mechanismus entwickeln Gitternachbarn auch im Merkmalsraum eine räumliche Nähe zueinander, was zu der erwähnten *Entfaltung* des Netzes führt. Ist dieser Effekt in einer frühen Lernphase noch sehr erwünscht, verhindert er später, dass das Netz detaillierter werden, also feinere Strukturen abbilden, kann. Aus

diesem Grund wird sowohl die Lerngeschwindigkeit als auch der in $N(\vec{x}, \vec{y}, \sigma)$ verwendete Einzugsradius σ mit jedem Lernschritt verringert. Welchen Teil des Eingaberaums die Karte am Ende abdeckt, hängt maßgeblich von der Werteverteilung der Menge der Eingabedaten ab. Leider kann nie davon ausgegangen werden, dass das gesamte Netz gleichmäßig im Ursprungsraum liegt. So ist es möglich, dass sich *Falten* in der Karte entwickeln. Zwischen den Gewichten zweier benachbarten Neuronen entstehen Sprünge im Merkmalsraum, die Kartentopologie bildet also nicht zwingend die ursprüngliche Topologie über das gesamte Netz ab; sie ist häufig nur stückweise vorhanden.

2.6 Dimensionsreduktion und Distanzmetrik

Zusätzlich zu der Fähigkeit der topologieerhaltenden Vektorquantisierung lassen sich mit selbstorganisierenden Karten multidimensional skalieren. Dadurch, dass der Eingaberaum auch eine höhere Dimension aufweisen darf als die Gitterstruktur der SOM, können Eingabedaten in niedrigdimensionale Räume übertragen werden. Wenn vorhanden, können mit dieser Methode Unterräume im Merkmalsraum gefunden werden; z.B. Daten auf einem zweidimensionalen Band, welches in einem dreidimensionalen Raum liegt. Die Koordinaten der Neuronen können dann für Koordinaten auf dem Band stehen, wenn dieses Flach auf einer Ebene liegen würde.

Leider bietet diese Methode der Dimensionsreduktion keine Garantie für eine optimale Einhaltung von Abständen zwischen je zwei Datenpunkten im Merkmals- sowie im Ausgaberaum; die klassische multidimensionale Skalierung (MDS) ist jedoch in der Lage, Abstände weitgehend maßstabsgetreu zu skalieren.

Die MDS wurde ursprünglich entwickelt, um Zusammenhänge hochdimensionaler psychologischer Daten in einem zwei- oder dreidimensionalem Raum und die Ähnlichkeit bzw. den Abstand verschiedener Objekte aus einem Hochdimensionalem Raum sichtbar zu machen. Zu der Entwicklung haben z.B. [Tor52], [Kru64], [Coo64] und [Sam69] beigetragen. Ein Maß für topographische Störungen in einer SOM wurde von [Duc94] entwickelt.

Um eine gültige Skalierung zu berechnen, müssen klassische Verfahren bei x Datenpunkten eine Funktion mit $x(x-1)/2$ Variablen (Abständen) minimieren, um eine in

ihrem Sinne optimale Verteilung der Position der Repräsentanten im Merkmalsraum zu berechnen. Der SOM-Algorithmus benötigt $x * n$ (Merkmalsraum hat Dimension n) Rechenschritte, um alle Datenpunkte zu berücksichtigen. Wann ist es also effizienter eine SOM für das Problem zu verwenden? Setzt man die beiden Aufwandsabschätzungen in eine Ungleichung

$$\begin{aligned} \frac{x(x-1)}{2} &> x * n \\ (x-1) &> 2n \\ x &> 2n+1 \end{aligned}$$

erhält man eine Rechenvorschrift ([DN96]). Sobald also die Ungleichung

$$x > 2n + 1 \tag{2.15}$$

zutrifft, benötigt die Berechnung der Skalierung mit einer SOM einen geringeren Aufwand als die klassische MDS. Angenommen man möchte Daten aus einem Raum mit einer Dimensionalität von 100 multidimensional skalieren, träfe diese Ungleichung bereits bei 202 Datenpunkten zu. Möchte man jedoch mit heutzutage relevanten Datenmengen umgehen, liegt die Anzahl der Datenpunkte in einem Datensatz leicht bei 10^6 und höher. Dies macht die Anwendung der Sammonschen multidimensionalen Skalierung zumindest schwierig.

Eine wichtige Tatsache wurde bisher noch nicht betrachtet. Dazu ein Beispiel: Es sei eine zweidimensionale Karte mit einer Kantenlänge von drei Neuronen gegeben. Diese Karte wurde außerdem mit einem Datensatz fertig trainiert. Nun kann der Fall eintreten, dass der Abstand des Gewichtsvektors des Neurons oben in der Mitte zum Gewichtsvektor des mittleren Neurons 10 mal so groß ist, wie die Abstände zwischen dem mittleren und dem unteren Gewichtsvektor. Eine SOM gibt somit als keinerlei Aufschlüsse über quantitative Distanzen, sondern nur qualitative. Während die MDS von Sammon ([Sam69] versucht, maßstabsgetreu zu skalieren, skaliert eine SOM nur topologiererhaltend. Eine selbstorganisierende Karte versucht also die Vorschrift

$$\begin{aligned} &d(\vec{w}_{\vec{r}_i}, \vec{w}_{\vec{r}_j}) + d(\vec{w}_{\vec{r}_j}, \vec{w}_{\vec{r}_k}) > d(\vec{w}_{\vec{r}_i}, \vec{w}_{\vec{r}_k}) \\ &\quad \Leftrightarrow d(\vec{r}_i, \vec{r}_j) + d(\vec{r}_j, \vec{r}_k) > d(\vec{r}_i, \vec{r}_k) \\ &\qquad\qquad \forall \vec{r} \in \mathcal{W} \end{aligned}$$

zu erhalten. Sammons Kriterium geht jedoch weiter:

$$d(\vec{w}_{\vec{r}_i}, \vec{w}_{\vec{r}_j}) = t \cdot d(\vec{r}_i, \vec{r}_j) \quad \forall \vec{r} \in \mathcal{W},\, t \in \mathbb{R} \tag{2.16}$$

Es lassen sich also auch quantitative Rückschlüsse auf Distanzen im Merkmalsraum ziehen. [Tro08] schlägt vor, sich die Effizienz von SOMs bei der Dimensionsreduktion bzw. dem Clustering bei einer hohen Anzahl von Datensätzen zu nutze zu machen und das Ergebnis mit MDS weiter zu verarbeiten. Damit sollen die einzelnen Neuronen möglichst maßstabsgerecht im Ausgaberaum positioniert werden, ohne, dass dazu ein Rechenaufwand wie bei MDS nötig wäre. Die Methode dazu ist folgende:

1. Trainiere die SOM mit dem gewünschten Datensatz
2. Wende Sammons MDS auf die Gewichtsvektoren der trainierten SOM an und platziere die Neuronen entsprechend im Ausgaberaum

2.7 Visualisierung

Eine essentielle Aufgabe von selbstorganisierenden Karten ist die Visualisierung von hochdimensionalen Daten. Im Raum $\mathbb{R}^t$ mit $t >> 3$ liegende Daten, können nicht mehr ohne spezielle Techniken visualisiert werden. Eine häufig verwendete Methode ist der sogenannte Scatterplot (Siehe Abbildung 2.7). Hier werden jeweils zwei Dimensionen des Datensatzes in einem Plot dargestellt. Wird dies mit allen Dimensionspaaren gemacht, erhält man den Scatterplot. Die Diagonale ist eine Symmetrieachse. Diese Visualisierungstechnik stößt bei noch höheren Dimensionalitäten schnell an die Grenzen der Einsetzbarkeit, da die Übersicht verloren geht und Zusammenhänge zwischen mehreren Dimensionen nicht mehr erfassbar sind. Andere Verfahren, die versuchen, alle Dimensionen getrennt voneinander bzw. paarweise darzustellen, stoßen auf die selben Probleme wie der Scatterplot.

Aus diesem Grund wird auf verschiedene Arten der Dimensionsreduktion zurückgegriffen ([Yin07]). Eine sehr einfache Variante ist die lineare PCA, die *Principal Component Analysis*. Sie wurde bereits 1901 von Pearson [Pea01] vorgeschlagen. Hierbei werden anhand der zwei mithilfe der Autokorrelationsmatrix der Daten berechnete Geraden in den Merkmalsraum gelegt. Diese Geraden stehen bei der Visualisierung im zweidimen-

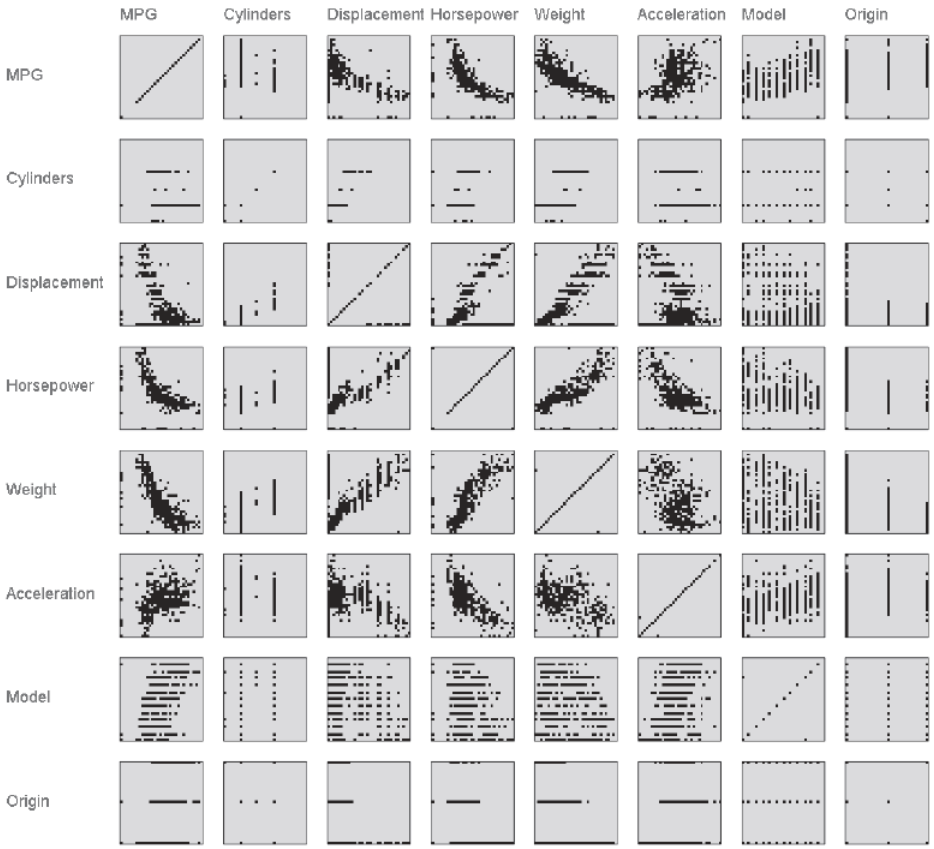


Abbildung 2.7: Scatterplot eines 7-dimensionalen Datensatzes mit Eigenschaften von Autos. Quelle: [EDF08]

sionalen für die beiden Achsen x und y, auf welchen die Varianz der Daten am höchsten ist. Auch hier treten jedoch Schwierigkeiten auf, wenn die Daten nichtlinear im Raum verteilt sind, wenn sie also beispielsweise in einem kurvenförmigen Unterraum liegen. Dieser Raum ist mit zwei Geraden nicht repräsentierbar. Deshalb wurde die nicht lineare PCA entwickelt, die versucht, über beispielsweise ein Gradientenabstiegsverfahren Punkte einer Kurve so in den Merkmalsraum zu legen, dass die Summe der Abstände der Datenpunkte zu dem jeweils nächsten Repräsentanten möglichst klein wird.

Eine weitere Entwicklung ist die bereits erwähnte sehr rechenintensive multildimensionale Skalierung, wie sie z.B. Sammon ([Sam69]) vorgeschlagen hat. Mit MDS können

hochdimensionale Daten in einem zweidimensionalem Raum sogar unter Einhaltung der Abstände im Merkmalsraum visualisiert werden.

Um nun das Fehlen der maßstabsgetreuen multidimensionsalen Skalierung mit SOMs teilweise auszugleichen, wurde von ([Ult03b]) eine Erweiterung namens U-Matrix vorgeschlagen. Diese Erweiterung modifiziert nicht den Lernalgorithmus der SOM. Lediglich die Visualisierung wird so verändert, dass Entfernungen im Merkmalsraum zweier innerhalb der SOM benachbarter Gewichte erfassbar werden. Hierzu erhält jedes Neuron zusätzlich zu seiner Position $\vec{r}$ innerhalb des Gitters sowie zu seinem Gewichtsvektor $\vec{w}_{\vec{r}}$ einen Wert $u_{\vec{r}}$, der für den mittleren Abstand eines Neurons $\vec{r}$ zu seinen Nachbarn steht. Als Distanznorm wird die selbe Norm wie bei der Berechnung des Best-Matching-Unit verwendet. Nachdem der Wert $u_{\vec{r}}$ für alle Neuronen berechnet wurde, wird jedem dieser Werte beispielsweise ein Grauwert zugeordnet. Dieser Wert wird in der Visualisierung der SOM als Farbe der Neuronen verwendet. Wie im Beispiel in Abbildung 2.8 zu sehen. Die Zuordnung des mittleren Abstandes u zu einem 8 Bit Grauwert g erfolgt über die Gleichung

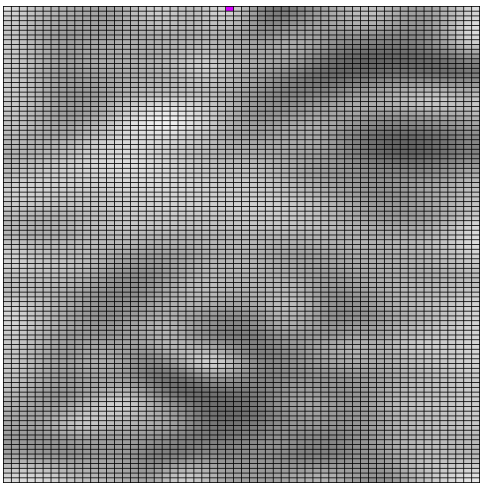
$$g = u^a \cdot 255 \tag{2.17}$$

mit $a \in \mathbb{R}$ und $u \in [0; 1]$. Der Größte Abstand wird mit $u = 0$ und der kleinste Abstand mit $u = 1$ repräsentiert. Die Zuordnung zwischen dem mittleren Abstand und dem u-Wert ist linear. Um die berechneten mittleren Abstände sinnvoll auf das Intervall $[0; 1]$ skalieren zu können, wird sowohl das Maximum d_{max} als auch das Minimum d_{min} aller mittleren Abstände $d_{\vec{r}_i}$ gesucht. $u_{\vec{r}_i}$ berechnet sich dann über

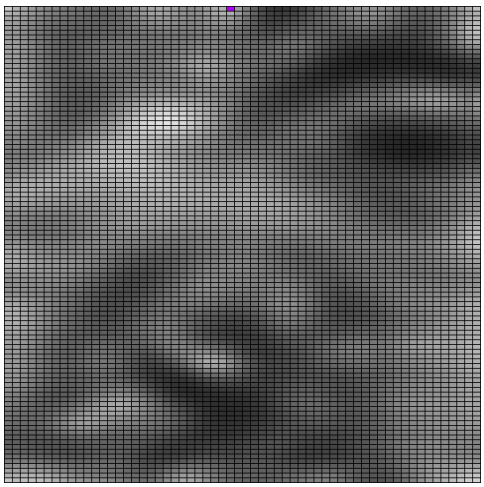
$$u_{\vec{r}_i} = 1 - \frac{d_{\vec{r}_i} - d_{min}}{d_{max} - d_{min}}. \tag{2.18}$$

Anstatt eine lineare Zuordnung zwischen u und g zu wählen, kann auch eine nichtlineare gewählt werden. Je höher dieser Wert ist, desto höher ist die Auflösung für kleine Abstände. Sinnvoll ist ein hoher Wert vor allem dann, wenn wenige sehr große Werte für u berechnet wurden. Dies würde ohne Nichtlinearität nämlich dazu führen, dass die gesamte Karte bis auf wenige Stellen sehr hell wird und somit Informationen über kleine und mittlere Abstände verloren gehen.

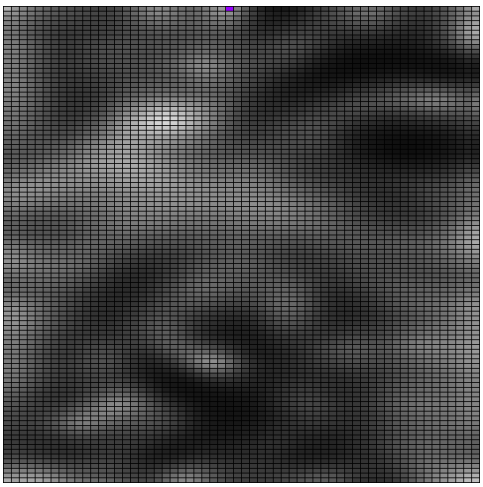
Die Hauptaufgabe der Visualisierungsmethode mit der U-Matrix ist das Finden von Clustern in einer SOM. Ansammlungen von Neuronen mit ähnlichen Gewichten werden hell eingefärbt, hohe Unterschiede zwischen Gewichten wiederum dunkel. So können



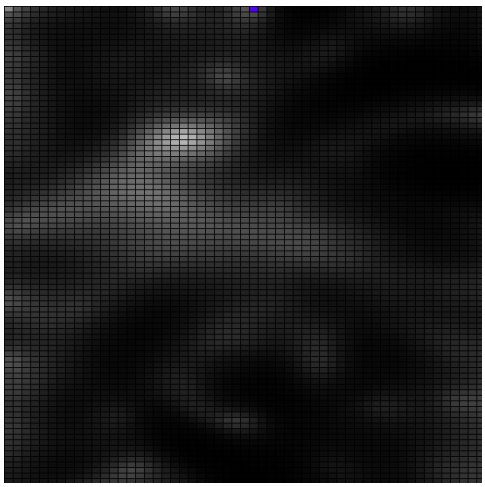
(a) 8 Bit Grauwert g mit $g = u^1 \cdot 255$ berechnet.



(b) 8 Bit Grauwert g mit $g = u^2 \cdot 255$ berechnet.

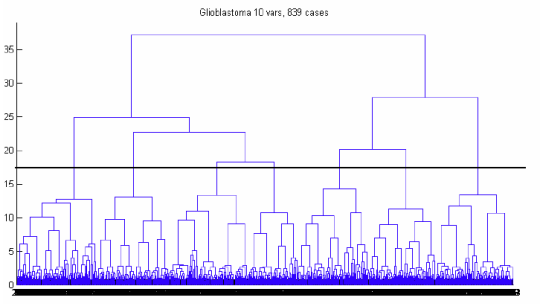


(c) 8 Bit Grauwert g mit $g = u^3 \cdot 255$ berechnet.

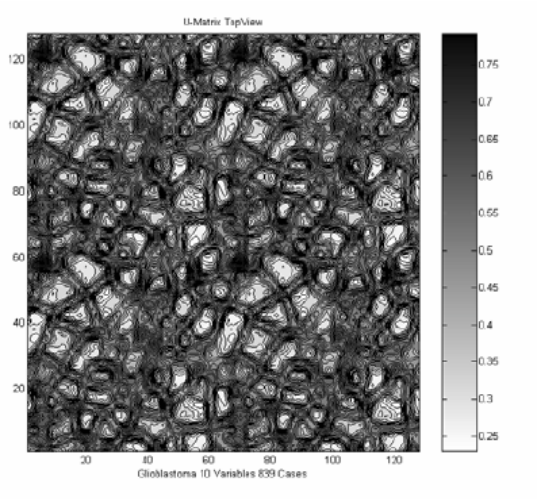


(d) 8 Bit Grauwert g mit $g = u^6 \cdot 255$ berechnet.

Abbildung 2.8: Selbstorganisierende Karte mit visualisierter U-Matrix. Kartengröße: 60 x 60 Neuronen. Helle Grauwerte stehen für geringe Abstände und dunkle Grauwerte für große Abstände zu Nachbarneuronen. Der Merkmalsraum ist in diesem Fall $\mathbb{R}^{18}$. Die verarbeiteten Daten sind mit Hilfe der Kinect aufgezeichnete 3D-Koordinaten der Hände, Füße, des Kopfes und des Torsos. Die Abbildungen unterscheiden sich jeweils in der Zuordnung des mittleren Abstandes der Nachbarn u zum Grauwert g, mit welchem der Wert repräsentiert wird. Die Werte u werden normalisiert auf das Intervall $[0; 1]$. 0 steht dabei für den höchsten, 1 für den kleinsten berechneten mittleren Abstand.



(a) Dendogramm berechnet mit der WARD Methode. Für Clusterradien größer 15 erhält man damit sieben Cluster.



(b) Visualisierung der Nachbarschaftsbeziehungen mit der U-Matrix. Es sind keine klaren Strukturen zu erkennen.

Abbildung 2.9: Ein Datensatz bestehend aus Daten über Hirntumore wird mit zwei Methoden auf Cluster untersucht. Die U-Matrix liefert keine sinnvollen Clusterstrukturen.

Cluster schnell und treffsicher erkannt werden. Doch dies funktioniert nicht immer, was ein Fall von Ultsch [Ult03b] belegt. Er untersuchte einen Datensatz von Gehirntumoren und stellte dabei fest, das die Fähigkeiten der U-Matrix nicht ausreichten. Die Clusteranalyse mit der agglomerativen WARD [EHW86] Methode lieferte, wie in Abbildung 2.9 a) zu sehen, 7 große Cluster, die einen Radius größer 15 besitzen. Die U-Matrix Visualisierung jedoch lieferte, wie in Abbildung 2.9 b) zu sehen, nur viele kleine Cluster, mit ihr ließen sich also keine Rückschlüsse auf die reale Verteilung der Daten ziehen.

In Verbindung mit der P-Matrix jedoch lieferte die Clusteranalyse mit Hilfe einer SOM brauchbare Ergebnisse. Die P-Matrix enthält Informationen über die Punktdichte im Merkmalsraum in der Umgebung eines Gewichtes. Zur Berechnung der Dichte wird die Pareto Density Estimation (PDE) [Ult03a] verwendet. PDE misst die Dichte als Anzahl der Punkte in einer Hypersphäre. Es kann natürlich auch jede andere Methode zur Berechnung der Punktdichte verwendet werden. Hohe Werte in der P-Matrix stehen also für hohe Punktdichten, kleine Werte für lose im Raum verteilte Punkte. Fasst man die U- und P-Matrix zusammen, erhält man die sogenannte U*-Matrix. Für die U*-Matrix

gilt:

- Ist die Punktdichte um ein Neuron so hoch wie die mittlere Punktdichte, ist der Wert der U*-Matrix für dieses Neuron so hoch wie der Wert der U-Matrix.
- Ist die Punktdichte höher als die mittlere Punktdichte, sind die Abstände zu den Nachbargewichten vermutlich Entfernungen innerhalb eines Clusters, der U*-Wert ist also größer(heller) als der U-Wert. Entfernungen werden also kleiner gewertet, als sie eigentlich sind.
- Ist die Punktdichte geringer als die mittlere Punktdichte, sind die Abstände zu den Nachbarn hauptsächlich Abstände am Rande eines Clusters. U*-Matrix-Werte sind also kleiner (dunkler) als der U-Wert.

Das Ergebnis der Anwendung der U*-Matrix ist in Abbildung 2.10 zu sehen. Blaue Stellen bzw. Täler in der Grafik stehen für Cluster. Diese Cluster sind weitgehend identisch zu den mit der WARD Methode gefundenen Clustern.

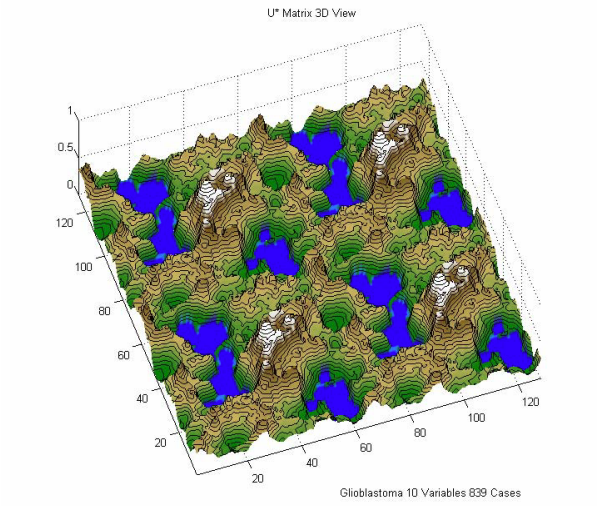


Abbildung 2.10: Anwendung der U*-Matrix Methode auf einen Datensatz bestehend aus Daten über Hirntumore. Die U*-Matrix Methode liefert deutliche Cluster (blau) im Gegensatz zur U-Matrix wie in Abbildung 2.9 b) zu sehen.

2.8 Anwendungsgebiete

Selbstorganisierende Karten finden in vielen verschiedenen Bereichen Anwendung; nämlich immer dann, wenn es darum geht, eine große Menge hochdimensionaler Daten zu analysieren. SOMs helfen dabei, die Dimensionalität der Daten zu reduzieren, eine Distanzmetrik für die Daten zu finden, diese zu Clustern, zu Labeln und neu hinzugekommen Daten zu klassifizieren. All diese Prozesse können online ablaufen. Die Karte muss nicht mit jedem weiteren Datum von von Grund auf neu berechnet werden. Sehr häufig werden Gebiete in der Karte, die sich einander ähnelnde Daten repräsentieren, manuell gekennzeichnet, also *gelabelt*, um daraufhin vom Menschen lesbare Informationen über zu klassifizierende Daten zu erhalten.

3 Lernen und Vergleichen von Gesten

3.1 Zeitreihen

Eine für folgende Abschnitte grundlegende Komponente ist die Zeitreihe [Bri81]. Zeitreihen werden wie die SOMs in sehr vielen Fachbereichen verwendet. So gibt es in der Ökonomie Zeitreihen von Aktienwerten oder von der Höhe des Leitzins, in der Biologie werden beispielsweise Zeitreihen des Wachstums von Zellkulturen angefertigt, in der Medizin sind es Verläufe von Blutwerten oder Herzfrequenzen und in der Signalverarbeitung werden Sensorwerte über einen gewissen Zeitraum aufgezeichnet. Formal werden Zeitreihen in allen Bereichen gleich definiert. Eine Zeitreihe ist immer eine geordnete Menge von Werten. Die Werte haben sinnvollerweise innerhalb einer Zeitreihe die selbe Form. Ein Wert ist stets ein Vektor einer gewissen Dimensionalität. Der Raum, aus dem die einzelnen Elemente des Vektors stammen, kann jede erdenkliche Menge von Symbolen sein. Denkbar wären Mengen von Wörtern, von Bilder, Buchstaben, Zahlen oder sonstigen Symbolen. Diese Arbeit beschränkt sich auf Werte aus dem Raum $\mathbb{R}^n$.

Eine Zeitreihe wird also wie folgt definiert:

$$S = \{\vec{s}_t | t \in 1, .., T\} \quad , T \in \mathbb{Z}, \vec{s}_t \in \mathbb{R}^n, n \in \mathbb{N} \tag{3.1}$$

T ist die Anzahl der Elemente einer Zeitreihe, n ist die Dimensionalität eines Elementes der Zeitreihe S.

Auch eine Geste kann als Zeitreihe interpretiert werden. Ein Element s einer solchen Zeitreihe beschreibt dann eine einzelne Pose und enthält Koordinaten von definierten Punkten am Körper. Eine geordnete Menge solcher Posen können dann als Geste interpretiert werden. In 3.1 ist ein Beispiel abgebildet.

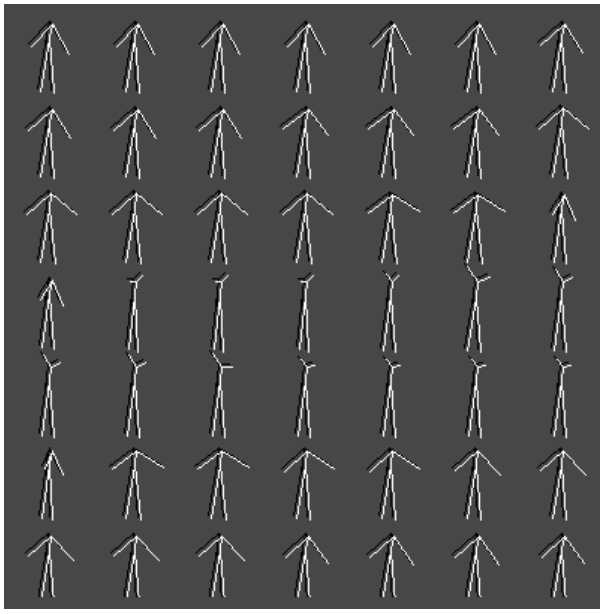


Abbildung 3.1: Visualisierung einer Geste, die an ein Flügelschlagen erinnert. Die abgebildete Zeitreihe umfasst 49 Zeitpunkte. Die Posenfolge ist Zeile für Zeile und von links nach rechts zu lesen. Zugrundeliegende Daten stammen aus einer Aufzeichnung mit einer Kinect. Die verwendete Methode wird in Kapitel 3.3 beschrieben.

3.2 Vorhandene Konzepte zum Lernen und Erkennen von Gesten

Es existieren viele Methoden, um Gesten zu erkennen, wie in [MA07] zu lesen ist. Eine sehr weit verbreitete Methode zur Gestenerkennung ist die Verwendung von Hidden-Markov-Modellen [RJ86] (HMM). Verwendet werden auch *recurrent fuzzy networks* [JK05], *time delay neural networks* [YAT02] oder *endliche Automaten* [HTH00]. Im Folgenden werden zwei Konzepte im Detail vorgestellt.

3.2.1 Konzept 1: Gestenerkennung mit Hilfe einer SOM und Hidden-Markov-Modellen

Das Team um Caridakis [CKP+08] schlägt vor, Gesten mit Hilfe einer Kombination aus

1. einer selbstorganisierenden Karte (SOM, [KSH01]) und
2. Hidden-Markov-Modellen [RJ86]

zu lernen und zu erkennen.

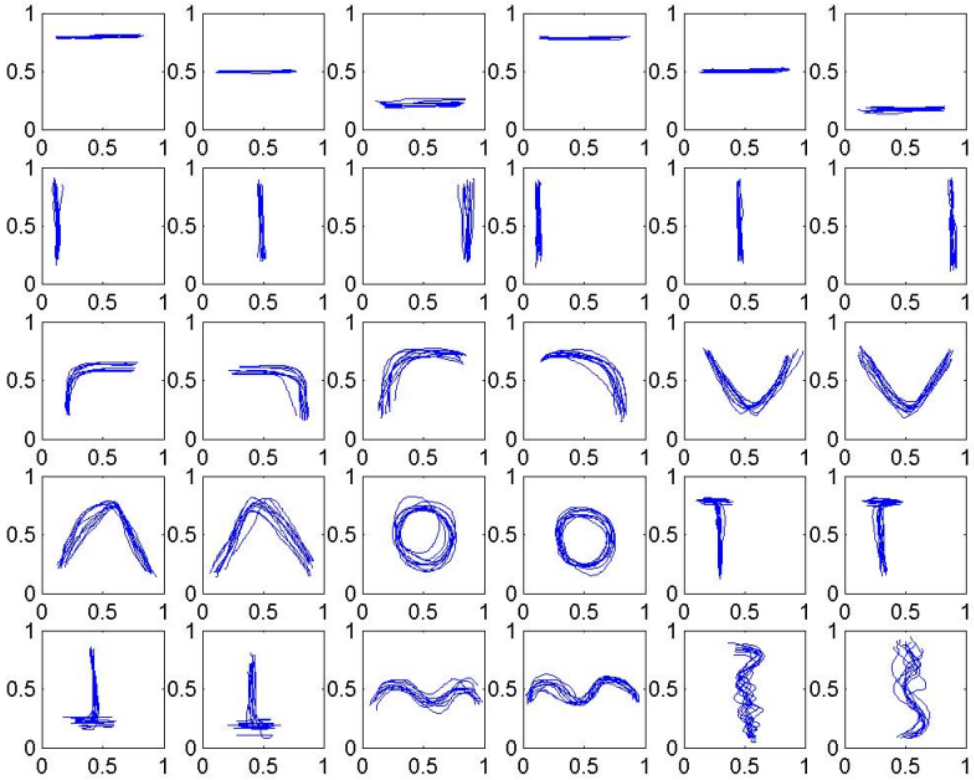


Abbildung 3.2: Datensatz mit 30 Gesten. Diese Gesten wurden in der beschriebenen Arbeit aus Kapitel 3.2.1 verwendet.(Quelle: [CKP+08])

Die SOM verarbeitet Positionsdaten und dient der Vektorquantisierung, also der Generalisierung. Sie hat also den selben Zweck wie bei der in dieser Arbeit vorgeschlagenen Methode.

Die Koordinanten der BMUs werden als Symbole betrachtet. Eine Folge dieser Symbole repräsentiert eine Geste. Diese Folgen dienen als Basis für das Lernen von Hidden-Markov-Modellen.

Jedes HMM repräsentiert eine Geste. Nun wird für jedes dieser HMMs aus allen Trainingsdaten der generalisierte Median mit Hilfe der Levensthein Metrik [Lev66] berechnet und dieser mit abgespeichert.

Anstatt HMMs aus Positionsdaten zu erzeugen, erzeugt ein zweiter Prozess HMMs aus einer Menge von Trajektorienänderungen. Hierfür werden Winkel mit einer begrenzten Menge von Werten beschrieben, die als Symbole behandelt werden. Diese zweite Klasse von HMMs soll die Erkennung verbessern und wird verwendet, wenn die Erkennungsprozedur mit der ersten Klasse keine eindeutigen Ergebnisse liefert.

Eine Geste gilt dann als erkannt, wenn drei Bedingungen erfüllt sind:

1. Die höchste berechnete Wahrscheinlichkeit aller HMMs aus der Menge der ersten Klasse ist größer als ein Wert α.
2. Der Abstand zwischen den beiden höchsten berechneten Wahrscheinlichkeiten ist größer als β.
3. Die die Geste repräsentierende Symbolfolge wird mit dem generalisierten Median des HMMs mit der größten Wahrscheinlichkeit verglichen. Der Abstand muss kleiner als ein Wert γ sein.

Die durchschnittliche Erkennungsrate dieser Methode lag bei 93 %. Ein Standardverfahren aus der Literatur, welches auch HMMs verwendet, hatte mit den selben Daten eine Erkennungsrate von 85 %. Die verwendeten Gestendaten sind in Abbildung 3.2 zu sehen.

3.2.2 Konzept 2: Gestenerkennung mit Hilfe einer SOM, eines endliches Automaten und SVMs

Oshita und Matsunaga verwenden zur Gestenerkennung [OM10] eine Kombination aus

1. einer selbstorganisierenden Karte (SOM, [KSH01]),
2. eines endlichen Automaten und
3. einer Support-Vector-Machine (kurz SVM, [CST00]).

Auch hier verarbeitet die SOM Positionsdaten und dient der Vektorquantisierung, also der Generalisierung. Einander ähnliche Posen werden einem Neuron oder nahen Nachbarn diesem zugeordnet. Im Allgemeinen muss die Neuronenzahl in einer SOM manuell festgelegt werden. Das Team um Oshita verwendete eine Automatisierung. Ein Optimierungskriterium

$$e = e_1 + k \cdot e_2 \tag{3.2}$$

wurde festgelegt. e soll minimiert werden. e_1 ist die Fehlerrate, mit der korrekte Eingaben nicht erkannt werden, e_2 ist die Rate, mit der falsche Eingaben als korrekt eingestuft werden. $k \in \mathbb{R}$ ist ein Parameter, der die beiden Fehlerraten gewichtet. k soll je nach Anwendungsfall gewählt werden; zwei Extremfälle werden angegeben:

1. Soll die Gestenerkennung für ein Computerspiel verwendet werden, ist ein kleines e_1 wichtig, k sollte als klein gewählt werden. Die Gestenerkennung arbeitet dann sehr tolerant. Wenn Gesten von der Sollgeste abweichen, werden sie eher akzeptiert.
2. Wird die vorgeschlagene Gestenerkennung jedoch in einem Sicherheitskritischen Bereich eingesetzt, sollten False-Positves unbedingt vermieden werden und die Fehlerrate e_2 wird durch ein hohes k entsprechend höher als e_1 gewichtet.

Wenn eine im Sinne der Gleichung 3.2 optimale selbstorganisierende Karte gefunden wurde, werden deren Neuronen als Zustände des Automaten interpretiert und Kanten zwischen diesen Zuständen erstellt. Eine Kante wird immer dann erstellt, wenn während der Ausführung einer zu lernenden Geste ein unmittelbarer Übergang zwischen den beiden entsprechenden Neuronen stattfindet. Außerdem hat jeder Zustand eine reflexive Kante sowie eine Kante zum Startzustand.

Eine SVM je Zustand lernt nun die Bedingungen für einen Zustandsübergang. Ob der aktuelle Zustand sich ändert bzw. zu welchem Zustand gewechselt wird, hängt nur vom aktuell gemessenen Merkmalsvektor und dem davon aktivierten Neuron ab.

Eine Geste wird nur dann als gültig erkannt, wenn durch die Folge ihrer Posenvektoren Neuronen einer bereits bekannten Folge entspricht. Die bekannten Folgen stammen aus den Trainingsdaten. Das macht diese vorgestellte Gestenerkennung untolerant gegenüber Rauschen.

Diese Methode kann also weder ähnliche Gesten als solche erkennen noch kompensieren, wenn ein einzelner Zustand der Geste auf das Nachbarneuron fällt, was offensichtlich ein schwerwiegender Nachteil ist. Eine SOM ermöglicht zwar durch ihre Fähigkeit der Vektorquantisierung gerade diese Eigenschaften, sie wird jedoch nur auf Ebene der Positionen im Raum verwendet. Die Erkennung der Zeitreihe an sich verfügt, durch die Verwendung einer begrenzten Menge von unterschiedlich geformten Tupeln von Zuständen für die eine Geste gültig ist, über begrenzte Generalisierungseigenschaften.
Hierzu ein Beispiel: Man nehme an, die Neuronenfolge (3, 51, 140) wird als Geste A abgespeichert; nehme man weiter den wohlgemerkt sehr wahrscheinlichen Fall an, dass eine geringe Abweichung der Bewegung zu der Neuronenfolge (3, 52, 140) führe. Obwohl absolut gesehen die Abweichung verschwindend gering sein kann, erhält man durch die Beschränkung auf den Abgleich von Zustandsfolgen eine andere Geste oder die Geste wird als unbekannt eingestuft. Die Trainingsdaten müssten also vermutlich alle gewünschten Zustandsfolgen für jede zu erkennende Geste umfassen, um resistent gegenüber Abweichungen in der Bewegung zu sein.

Das Optimierungskriterium für die Neuronenanzahl der SOM aus Gleichung 3.2 führt für kleine k dazu, dass die Auflösung der Karte geringer wird. Die Abgrenzung einer Geste wird also unschärfer, da die Voronoizelle um ein Neuron im Merkmalsraum größer wird. Die SOM kann deshalb kleinere Strukturen schlechter abbilden.

3.3 Das vorgeschlagene Konzept

Um nun Gesten unterschiedlicher Länge vergleichen zu können, wird ein Ansatz mit hierarchisch angeordneten SOMs gewählt. Die erste zweidimensionale SOM erhält Roh-

daten der Kinect. Dies sind die Positionen der Hände, des Kopfes, des Torsos und der Füße. Mit drei Dimensionen je Punkt gilt für den Eingabevektor $\vec{x}$:

$$\vec{x} \in \mathbb{R}^{18}$$

Die Hauptaufgabe dieser Karte besteht in der Dimensionsreduktion der Daten die Posen beschreiben, was den Aufwand der Weiterverarbeitung auf nächster Ebene verringern soll.

Die Eingabe für die zweite SOM ist immer eine Konkatenation von genau 50 Posen $\vec{p}_j$ mit $0 \leq j < 50$, die in der Karte der ersten Ebene in zweidimensionale Vektoren kodiert wurden. Für den zweiten Eingabevektor $\vec{y}$ gilt also mit zwei Dimensionen je Pose und 50 Zeitpunkten:

$$\vec{y} \in \mathbb{R}^{100}$$

Koordinaten der Neuronen der ersten SOM werden in Y^2 mit $Y = [0, 1]$ übertragen, da der verwendete Lernalgorithmus einen stetigen Zahlenraum benötigt. Die Elemente r_j der Koordinate des Gewinnerneurons werden dazu jeweils durch die Seitenlänge L der SOM geteilt, sind dann also in $[0, 1]$.

Um für die Visualisierung der in der zweiten SOM kodierten Gesten auf die Gewichte der Neuronen der ersten SOM zugreifen zu können, werden die Koordinaten jeder Pose wieder mit L multipliziert.

3.4 Das Problem der konstanten Eingabedimensionalität bei SOMs

Ein Problem, dass sich für eine SOM stellt, die Gesten verarbeiten soll ist, dass Gesten naturgemäß unterschiedlicher Dauer sind. Angenommen es sind im Folgenden zwei Gesten zu vergleichen. Die erste Geste g_1 ist ein schnelles Auf- und Ab-Bewegen beider Arme, was an ein schnelles Flügelschlagen (siehe Abbildung 3.1) erinnert. Die zweite Geste g_2 ist eine Beschreibung eines Kreises mit dem rechten Arm. Die erste Geste sei 30 Zeitschritte lang, die zweite umfasse 70 Zeiteinheiten, sie haben also unterschiedliche Längen. Des Weiteren betrage die Dimensionalität der SOM in diesem Fall 50.

Die Dimensionalität der Eingabe soll also immer gleich der Dimensionalität der Gewichte sein. In dieser Arbeit werden die Eingabezeitreihen deshalb stets normiert; und zwar so, dass sie strikt 50 Zeitpunkte lang sind. Der folgende Algorithmus wird zur Streckung der zu verarbeitenden Gesten verwendet:

Gegeben:

- Anzahl z der benötigten Zeitpunkte
- Eingabe $\vec{x}$
- Anzahl n der in der Eingabe vorhandenen Zeitpunkte

Der Ablauf (für Streckungsfaktor > 1):

```
1  Vektor streckung(Vektor x){
2    // Länge der Eingabe speichern
3    int n = x.laenge();
4    // Fehlerintegrator initialisieren
5    double error = 0;
6    // Streckung berechnen
7    double streckung = n/z;
8
9    // Schleife über Sollanzahl Zeitpunkte
10   for (int i = 0; i < z; i++)
11   {
12     // Fehler integrieren
13     error += streckung - 1;
14     // Solange Fehler >= 1
15     while (error >= 1){
16       // hinter Stelle i Wert der Stelle i einfügen
17       x.insert(i, x[i]);
18       // Fehler soll kleiner als 1 werden
19       error--;
20     }
21   }
22   return x;
23 }
```

Leider führt dieser Ansatz dazu, dass Start- und Endzeitpunkt einer Geste bekannt sein müssen, um sie als Eingabedatum für die Gesten verarbeitende SOM zur Verfügung stellen zu können. Dies wirkt sich auch auf den Prozess der Erkennung von Gesten aus. Es ist damit nicht direkt möglich, eine Geste ohne das Wissen über Start- und Endzeitpunkt selbiger klassifizieren zu können.

4 Experimente

4.1 Kinect

Abbildung 4.1: Microsoft Kinect. Die Kinect verfügt über einen Sensor zur Distanzmessung, eine RGB-Kamera, Mikrofone sowie einen Motor, der ihren Sensorblock drehen kann. Quelle: www.microsoft.com

Die Verfügbarkeit der Microsoft Kinect für Windows und Linux Systeme erlaubt es, Daten über Körperbewegungen leicht zu akquirieren und den in Kapitel 3.3 vorgeschlagenen Ansatz zu testen. Der für Windows, Linux und Mac erhältliche Kinect-Treiber von Primesense erkennt eine vor ihr stehende Person und legt in diese ein einfaches Skelett, welches aus Unter- und Oberarmen, dem Kopf, dem Torso sowie Unter- und Oberschenkeln besteht. Für folgende Tests wurden folgende Versionen der Treiberkomponenten verwendet:

- OpenNI-Win32-1.3.2.1-Dev
- SensorKinect-Win-OpenSource32-5.0.3.3
- NITE-Win32-1.4.0.5-Dev

4.2 Die verwendete Software

Um das Konzept testen zu können, wurde eine Software entwickelt. Sie bietet dem Benutzer eine Schnittstelle, mit der zu Beginn die Posen verarbeitende SOM trainiert werden kann. Eingabedaten sowie Gewichte der SOM können abgespeichert und wieder abgespielt bzw. geladen werden. Über eine in der Hand gehaltene Funkmaus wird der Start- und der Endzeitpunkt der aufzuzeichnenden Bewegung signalisiert. Der Lernfortschritt kann bei beiden SOMs verfolgt werden.

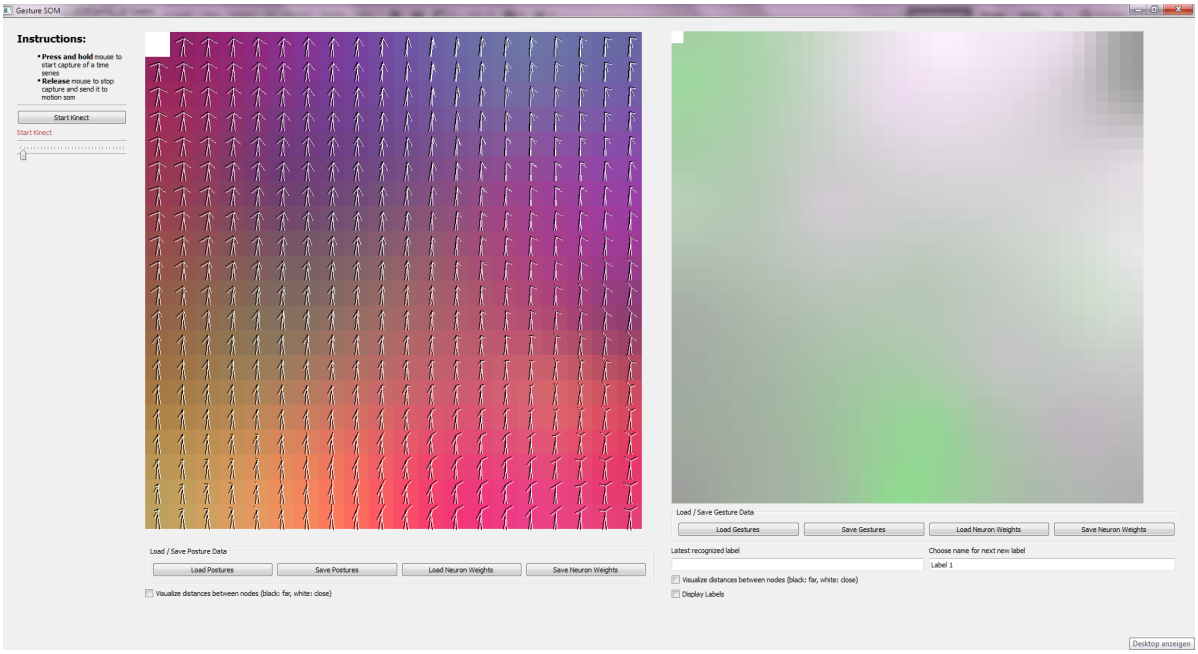


Abbildung 4.2: Screenshot der verwendeten Software. Sowohl die Neuronen der Posen-SOM (links) als auch die der Gesten-SOM (rechts) werden mit Farben abhängig der ersten 3 Komponenten ihres Gewichtes eingefärbt. In der linken SOM ist außerdem die Visualisierung der durch die Gewichte repräsentierten Posen sichtbar.

Außerdem werden zwei Visualisierungen angeboten. Die erste (Abbildung 4.2) färbt die Neuronen entsprechend der ersten drei Werte der Gewichte ein. Die Werte stehen dabei für die drei Komponenten des RGB Farbraumes. Für die zweite Visualisierung (Abbildung 4.3) wird die U-Matrix verwendet. Die Zuordnung von Nachbarschaftsabständen zu

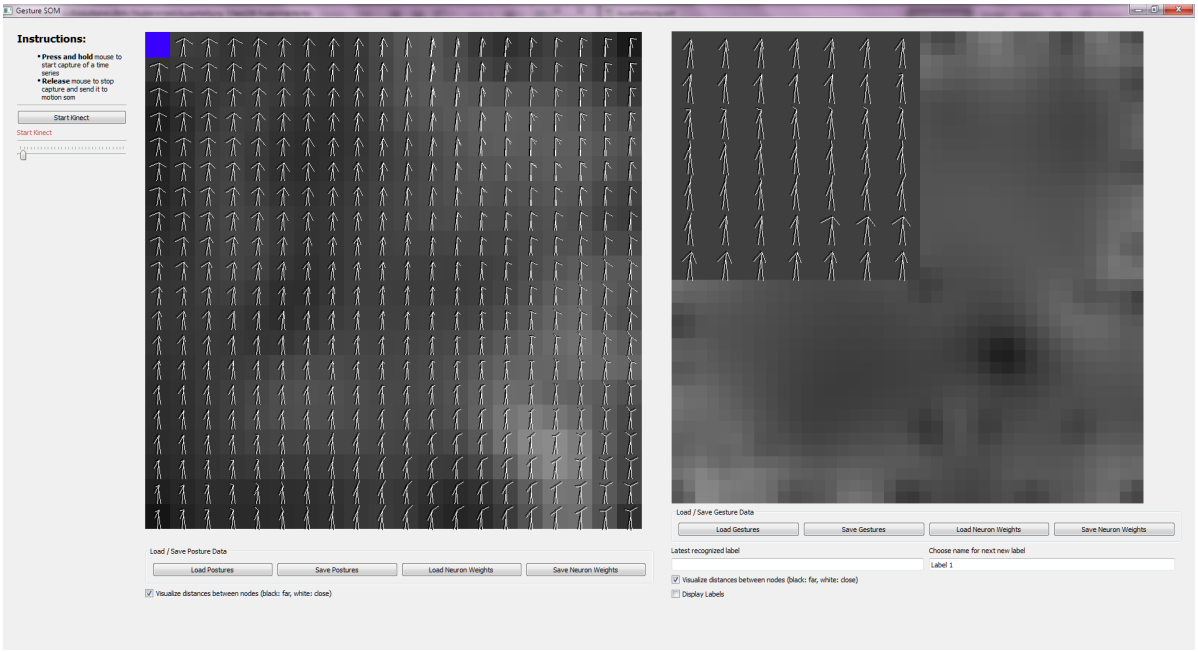


Abbildung 4.3: Screenshot der verwendeten Software. Sowohl die Neuronen der Posen-SOM (links) als auch die der Gesten-SOM (rechts) werden entsprechend des Wertes der U-Matrix gefärbt. Dunkle Farben stehen für kurze, helle Farben für große Abstände. In der rechten Karte wird die Geste eines Neurons visualisiert, dessen Gewicht ein Flattern mit beiden Armen repräsentiert.

Grauwerten kann über einen Schieberegler angepasst werden, um Cluster stets optimal sichtbar machen zu können. Zudem wird in der ersten SOM auf Neuronen die jeweils entsprechende Pose abgebildet. In der zweiten SOM kann jeweils die Folge der in den Gewichten kodierten Posen grafisch dargestellt (Abbildung 4.3) werden.

4.3 Experimente

Um das vorgeschlagene Konzept zu testen, wurde für vier verschiedene Gesten geprüft, wie gut sie erkannt werden. Folgende Gesten werden verwendet:

- Beide Arme auf und ab bewegen (Fliegen)
- Mit der rechten Hand im Uhrzeigersinn einen großen Kreis beschreiben
- Mit der rechten Hand gegen den Uhrzeigersinn einen großen Kreis beschreiben
- Winkbewegung; Die rechte Hand wird auf Kopfhöhe von rechts nach links nach rechts nach links bewegt

Jede Geste wird zehn mal aufgezeichnet.

Zuerst wurde die Posen-SOM vollständig (4000 Lernschritte lang) trainiert. Die Parameter hierfür sind wie folgt gewählt:

- Lerngeschwindigkeit zu Beginn: $\alpha_0 = 0.01$
- Einflussradius zu Beginn: $\sigma_0 = 10$
- Breite des Netzes: 20
- Höhe des Netzes: 20
- Lernschritte: 4000

Danach wurde die Gesten-SOM trainiert. Abgesehen von jeweils 5 Beispielen für die vier oben genannten Testgesten, dienen verschiedenste Bewegungen als Eingabe für das Netz. Die Parameter für die Gesten-SOM folgen:

- Lerngeschwindigkeit zu Beginn: $\alpha_0 = 0.05$
- Einflussradius zu Beginn: $\sigma_0 = 40$
- Breite des Netzes: 40
- Höhe des Netzes: 40
- Lernschritte: 2000

Das Ergebnis des erste Versuchs ist in Abbildung und 4.4 zu 4.5 sehen. In der ersten Abbildung ist die Gesten-SOM mit U-Matrix Visualisierung zu sehen. In der zweiten sind die Neuronen markiert, die beim Testen der vier Gesten Best-Matching-Unit waren. Sind weniger als 10 Neuronen markiert, gab es Dopplungen. Wo diese auftraten, ist nicht markiert. Es ist zu erkennen, dass die Kreisgeste im Uhrzeigersinn auf zwei Cluster aufgeteilt ist. Einer der Cluster liegt sehr nah an der Wink-Geste. Die andere Kreisgeste

Abbildung 4.4: Versuch 1. Posen-SOM in U-Matrix Visualisierung.

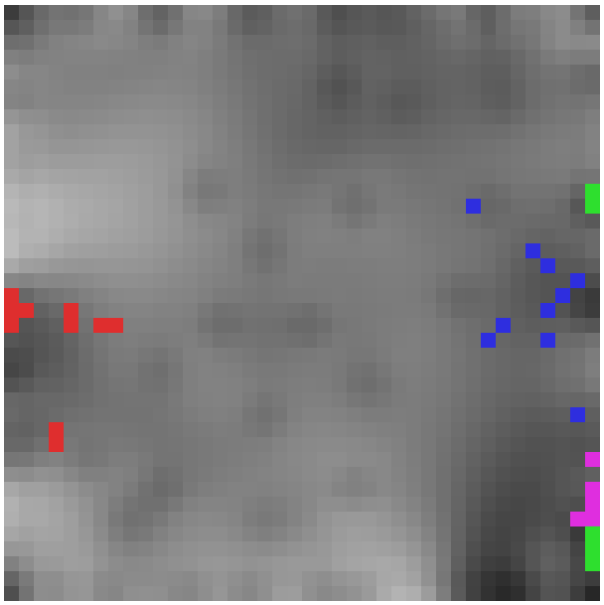


Abbildung 4.5: Versuch 1. Posen-SOM in U-Matrix Visualisierung. Best-Matching-Units wurden markiert. Eine Farbe entspricht einer Geste. Zehn Wiederholungen je Geste wurden getestet. Rot: Fluggeste; Grün: Kreis im Uhrzeigersinn; Blau: Kreis gegen den Uhrzeigersinn; Pink: Winken.

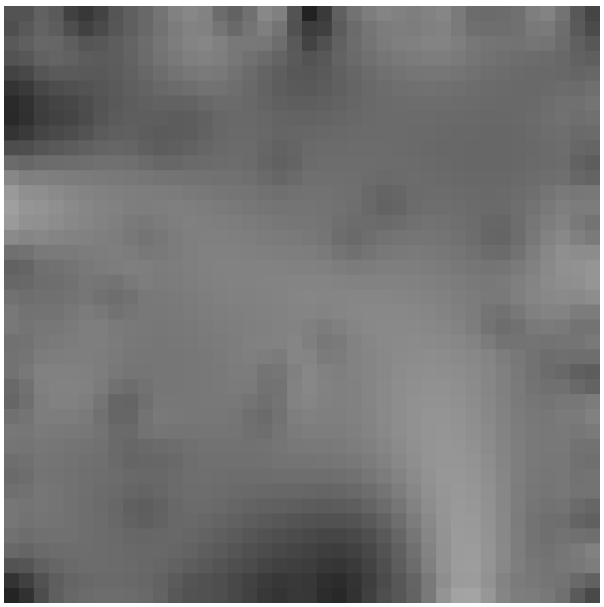


Abbildung 4.6: Versuch 2.Posen-SOM in U-Matrix Visualisierung.

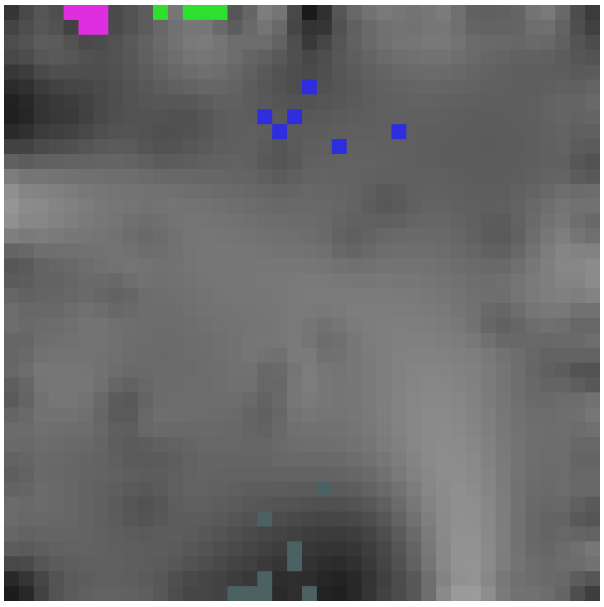


Abbildung 4.7: Versuch 2. Posen-SOM in U-Matrix Visualisierung. Best-Matching-Units wurden markiert. Eine Farbe entspricht einer Geste. Zehn Wiederholungen je Geste wurden getestet. Dunkelgrün: Fluggeste; Hellgrün: Kreis im Uhrzeigersinn; Blau: Kreis gegen den Uhrzeigersinn; Pink: Winken

liegt zwischen den bisher genannten Clustern. In der Posen-SOM gab es keine Neuronen, die Posen repräsentierten, in denen sich einer der beiden Arme oberhalb der Schulter befindet. Dass die BMUs dieser drei Gesten so verteilt sind, kann deshalb daran liegen, dass diese Gesten jeweils mit sehr ähnlichen Gewichten repräsentiert werden mussten, die auch nicht mit der originalen Bewegung übereinstimmen.

Die Fluggeste hat Neuronen auf der anderen Seite der Karte aktiviert. Hier ist auch ein deutlicher Cluster (dunkle Fläche bei roten Markierungen) zu erkennen.

Um nun einen Vergleich zu haben, wird die Posen-SOM besser trainiert und dabei mehr Wert auf Posen gelegt, die benötigt werden, um die ersten drei Gesten zu repräsentieren. Sowohl die verwendeten Daten der Trainingsgesten als auch der Vergleichsgesten sind die selben wie im ersten Versuch.

In Abbildung 4.6 und 4.7 ist das Ergebnis des zweiten Versuchs abgebildet. In der Posen-SOM finden sich jetzt einige Neuronen, die Posen repräsentieren, in denen sich die Arme oberhalb der Schultern befinden. In Abbildung 4.7 ist nun leicht erkennbar, wo die den Gesten entsprechenden Clusterzentren liegen. Außerdem sind die einzelnen Gesten besser voneinander abgegrenzt, was auch eine bessere Erkennungsrate vermuten lässt.

5 Zusammenfassung und Ausblick

Die vielfältige und unkomplizierte Einsetzbarkeit von selbstorganisierenden Karten verleitet leicht zur unbedachten Verwendung. Obwohl das Lernen unüberwacht stattfindet, hängt das Ergebnis stark von der Wahl der Parameter, sowie der Lerndaten ab. Sollen Aussagen, die sich unterstützend durch SOMs treffen lassen verlässlich sein, muss das Ergebnis vorher mit verschiedenen Parametersätzen, Lerndaten sowie Visualisierungen verifiziert werden. Auch klassische Verfahren wie die multidimensionale Skalierung von Sammon oder Clustering-Algorithmen sollten zumindest auf einen Teil der zu verarbeitenden Daten angewandt werden, um das Ergebnis besser interpretieren zu können.

Der vorgestellte Ansatz zum unüberwachten Lernen, Vergleichen und Erkennen von Gesten ist vielfältig erweiterbar und könnte in Zukunft näher studiert werden. Jedoch sollten einige Probleme gelöst werden.

Problem 1: Wenn die SOMs fertig gelernt, also alle Lernschritte absolviert hat, können keine neuen Informationen mehr in das Netz gespeichert werden. Entweder sollte hier mit Growing SOMs experimentiert werden oder das Kriterium, dass α und σ monoton fallen, muss ignoriert werden. Angenommen es existiert eine Methode, die automatisch einen Wert berechnet, der eine maximal erlaubte Distanz der Eingabe zu jedem Gewicht in der SOM definiert. Wenn dieser Wert nun überschritten wird, bedeutet dies, dass die SOM sich anpassen müsste. Die Anpassung könnte z.B. über eine Erhöhung der Lernparameter α und σ geschehen, wenn der Lernprozess bereits abgeschlossen ist.

Problem 2: Der Start- sowie der Endzeitpunkt einer Geste muss bekannt sein. Momentan müssen diese Zeitpunkte vom Menschen vorgegeben werden. Dies verhindert eine unüberwachte Erkennung von Gesten mit diesem Verfahren. Eine mögliche, wenn auch rechenintensive Lösung könnte darin liegen, dass während des Erkennungsvorgangs zu jedem Zeitpunkt parallel mehrere Zeitreihen verschiedener Längen gespeichert werden.

Die Software skaliert diese wieder auf die für die Gesten-SOM benötigte Länge. Zu jedem Zeitpunkt müssten also parallel eine hohe Anzahl von Zeitreihen, die jeweils unterschiedlich weit in die Vergangenheit zurück reichen und auf eine Einheitslänge skaliert wurden, in der SOM verarbeitet werden.

Selbstorganisierende Karten sind noch eine sehr junge Entwicklung. Auch wenn sich sehr viele Wissenschaftler mit SOMs beschäftigen, gibt es sicherlich noch viele sinnvolle Erweiterungen und Optimierung zur klassischen, von Kohonen vorgeschlagenen, SOM zu entwickeln und zu erforschen. Auch bereits vorgeschlagene Abwandlungen von SOMs, wie die Recurrent SOM [KVHK97], Growing SOM [ZF05] oder die Temporal Kohonen Map (TKM, [CT93]) können noch gründlicher erforscht werden.

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[Ama80] Shun-Ichi Amari. Topographic organization of nerve fields. *Bulletin of Mathematical Biology*, 42(3):339–364, 1980.

[Bri81] David R. Brillinger. *Time series : data analysis and theory*. Holden-Day, San Francisco, CA, 1981.

[CKP+08] George Caridakis, Kostas Karpouzis, Christos Pateritsas, Athanasios Drosopoulos, Andreas Stafylopatis, and Stefanos Kollias. Hand trajectory based gesture recognition using self-organizing feature maps and markov models. pages 1105–1108, June 2008.

[Coo64] C.H. Coombs. *A theory of data*. Wiley, 1964.

[CST00] Nello Cristianini and John Shawe-Taylor. *An introduction to support Vector Machines: and other kernel-based learning methods*. Cambridge University Press, New York, NY, USA, 2000.

[CT93] Geoffrey J. Chappell and John G. Taylor. The temporal kohonen map. *Neural Netw.*, 6(3):441–445, March 1993.

[DN96] Wlodzislaw Duch and Antoine Naud. Simplexes, multi-dimensional scaling and self-organized mapping. Technical report, In Proceedings of the 8th joint EPS-APS International Conference on Physics Computing ?96, 1996.

[Duc94] Wlodzislaw Duch. Quantitative measures for the self-organizing topographic maps, 1994.

[EDF08] N. Elmqvist, P. Dragicevic, and J.-D. Fekete. Rolling the dice: Multidimensional visual exploration using scatterplot matrix navigation. *Visualization and Computer Graphics, IEEE Transactions on*, 14(6):1539 –1148, nov.-dec. 2008.

[EHW86] A. El-Hamdouchi and P. Willett. Hierarchic document classification using ward's clustering method. In *Proceedings of the 9th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '86, pages 149–156, New York, NY, USA, 1986. ACM.

[GG91] Allen Gersho and Robert M. Gray. *Vector quantization and signal compression*. Kluwer Academic Publishers, Norwell, MA, USA, 1991.

[HTH00] Pengyu Hong, Matthew Turk, and Thomas S. Huang. Gesture modeling and recognition using finite state machines. In *In Proceedings of the Fourth IEEE International Conference and Gesture Recognition*, pages 410–415, 2000.

[JK05] C. F. Juang and Ksuan-Chun Ku. A recurrent fuzzy network for fuzzy temporal sequence processing and gesture recognition. *Systems, Man and Cybernetics, Part B, IEEE Transactions on*, 35(4):646–658, 2005.

[Kin27] B. A. Kingsbury. A direct comparison of the loudness of pure tones. *Phys. Rev.*, 29(4):588–600, Apr 1927.

[Kru64] J. Kruskal. Nonmetric multidimensional scaling: A numerical method. *Psychometrika*, 29(2):115–129, June 1964.

[KSH01] T. Kohonen, M. R. Schroeder, and T. S. Huang, editors. *Self-Organizing Maps*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 3rd edition, 2001.

[KVHK97] Timo Koskela, Markus Varsta, Jukka Heikkonen, and Kimmo Kaski. Time series prediction using recurrent som with local linear models. *INTERNATIONAL JOURNAL OF KNOWLEDGE-BASED INTELLIGENT ENGINEERING SYSTEMS*, 2(2):60–68, 1997.

[Lev66] VI Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*, 10:707, 1966.

[MA07] Sushmita Mitra and Tinku Acharya. Gesture recognition: A survey. *IEEE TRANSACTIONS ON SYSTEMS, MAN AND CYBERNETICS - PART C*, 37(3):311–324, 2007.

[Nav01] Gonzalo Navarro. A guided tour to approximate string matching. *ACM Comput. Surv.*, 33(1):31–88, March 2001.

[OBSC00] Atsuyuki Okabe, Barry Boots, Kokichi Sugihara, and Sung Nok Chiu. *Spatial tessellations: Concepts and applications of Voronoi diagrams*. Probability and Statistics. Wiley, NYC, 2nd edition, 2000. 671 pages.

[OM10] Masaki Oshita and Takefumi Matsunaga. Automatic learning of gesture recognition model using som and svm. In *Proceedings of the 6th international conference on Advances in visual computing - Volume Part I*, ISVC'10, pages 751–759, Berlin, Heidelberg, 2010. Springer-Verlag.

[Pea01] K. Pearson. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine*, 2(6):559–572, 1901.

[RJ86] L. R. Rabiner and B. H. Juang. An introduction to hidden markov models. *IEEE ASSp Magazine*, 1986.

[Sam69] J. W. Sammon. A nonlinear mapping for data structure analysis. *IEEE Trans. Comput.*, 18(5):401–409, May 1969.

[Tor52] W. S. Torgerson. Multidimensional scaling: I. theory and method. *Psychometrika*, 17:401–419, 1952.

[Tro08] M. W. Trosset. Representing clusters: K-means clustering, self-organizing maps, and multidimensional scaling. Technical report, Department of Statistics, Indiana University, 2008.

[TW01] Ru Telea and Jarke J. Van Wijk. Visualization of generalized voronoi diagrams. In *In Proceedings Data Visualization*, 2001.

[Ult03a] Alfred Ultsch. Pareto density estimation: A density estimation for knowledge discovery. In *Innovations in Classification, Data Science, and Information Systems - Proceedings 27th Annual Conference of the German Classification Siciety (GfKL'03)*, pages 91–100, 2003.

[Ult03b] Alfred Ultsch. U*-matrix: a tool to visualize clusters in high dimensional data. 2003.

[vdM73] Chistoph von der Malsburg. Self-organization of orientation sensitive cells in the striate cortex. *Kybernetik*, 14(2):85–100, 1973.

[YAT02] Ming Hsuan Yang, Narendra Ahuja, and Mark Tabb. Extraction of 2d motion trajectories and its application to hand gesture recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(8):1061–1074, August 2002.

[Yin07] Hujun Yin. Nonlinear dimensionality reduction and data visualization: A review. *International Journal of Automation and Computing*, 4(3):294, 2007.

[ZF05] Junlin Zhou and Yan Fu. Clustering high-dimensional data using growing som. In Jun Wang, Xiao-Feng Liao, and Zhang Yi, editors, *Advances in Neural Networks ? ISNN 2005*, volume 3497 of *Lecture Notes in Computer Science*, pages 822–822. Springer Berlin / Heidelberg, 2005.

Erklärung

Hiermit versichere ich, diese Arbeit
selbständig verfasst und nur die
angegebenen Quellen benutzt zu haben.

(Louis Bergmann)