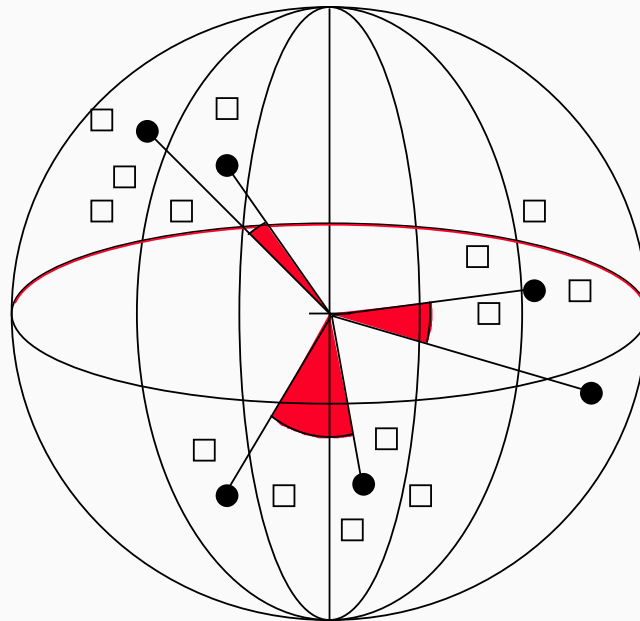


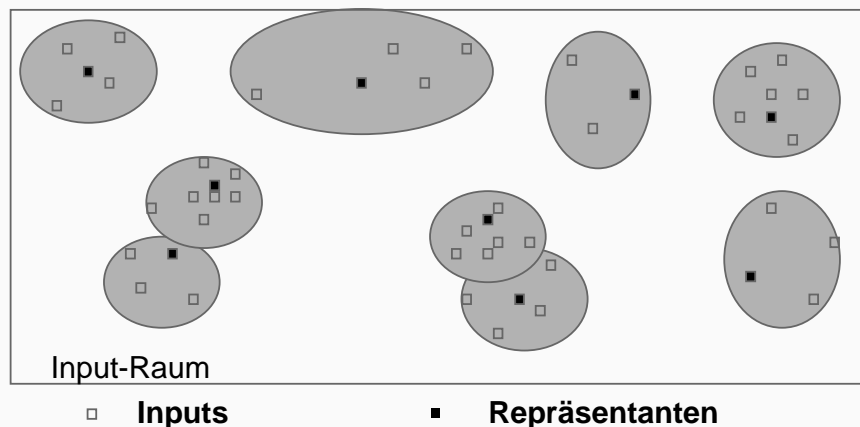
# Wettbewerbslernen



# Probleme für selbstorg. Lernen

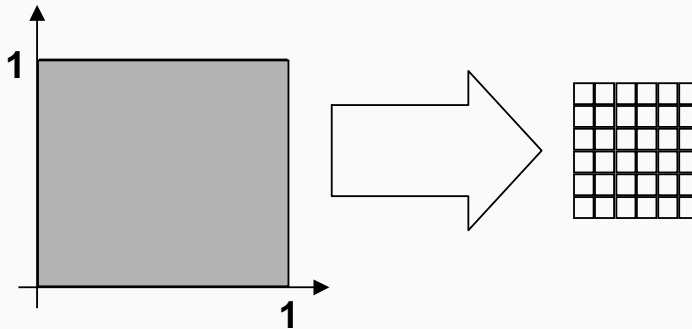
## clustering:

Die Input-Daten sollen in Clustereingeteilt werden, die nicht vom Benutzer vorgegeben sind. Ausgabe ist dann jeweils ein Label (oder Repräsentant) des Clusters, in dem der Input fällt.



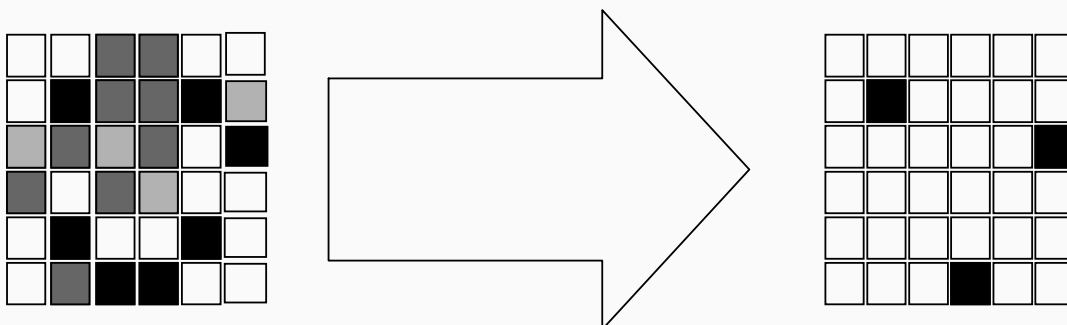
## vector quantization :

Ein kontinuierlicher Raum muß diskretisiert werden, d.h. unendlich viele Vektoren müssen auf endlich viele Vektoren klassifiziert werden. Ausgabe ist dann jeweils ein Label (oder Repräsentant) des Clusters, in den der Input fällt.



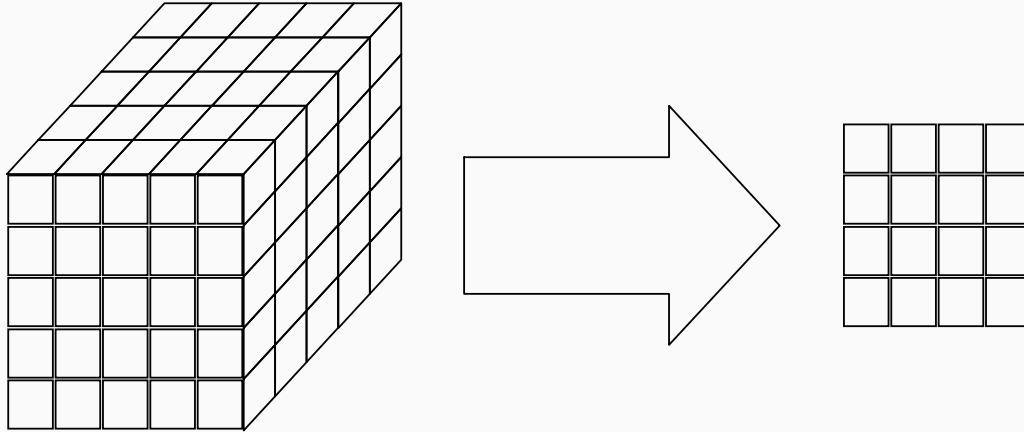
## sparse coding

sehr komplexe Eingaben müssen in eine digitale Form gebracht werden, die nur sehr wenige 1'en enthält. Ausgabe ist dann jeweils die dünn besetzte Form des Inputs.



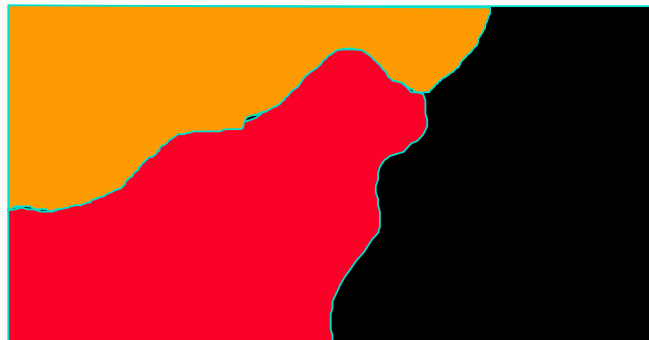
## dimensionreduction :

Eingabedaten aus einem hochdimensionalen Raum sollen in einem niedrigdimensionalen Raum repräsentiert werden. Ausgabe ist dann jeweils ein Label (oder Repräsentant) des Teilraums, in dem der Input fällt.



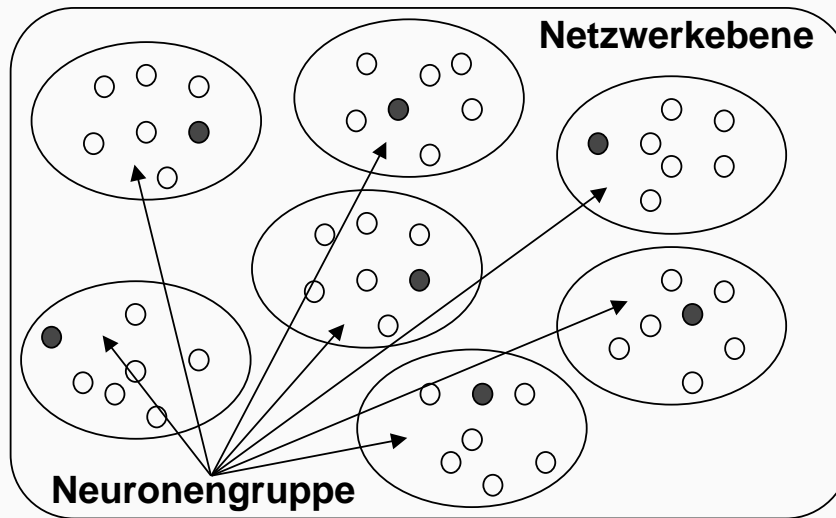
## feature extraction :

Unstrukturierte Eingabedaten sollen nach Eigenschaften (features) klassifiziert werden, die vom Benutzer nicht vorgegeben sind. Ausgabe ist dann jeweils ein Label (oder Repräsentant) für das Feature, das der Input hat.



# WinnerTakeAll-Strategie1

Nur die Gewinner jeder Gruppe nehmen am  
Propagierungs-bzw. Lernvorgang teil



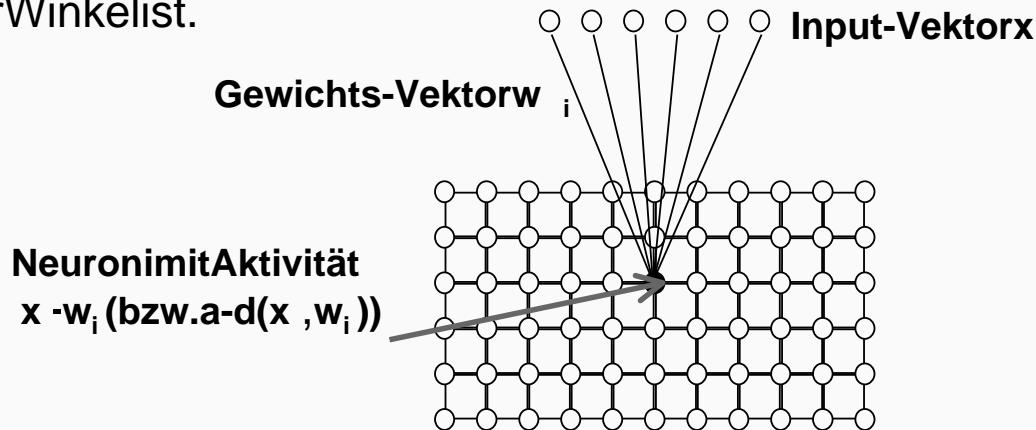
# WinnerTakeAll-Strategie2

**Um die Winner-take-all Strategie zu realisieren, kann man**

- entweder das Neuron mit minimaler Distanz von der Eingabe (maximaler Aktivität) direkt berechnen, dazu benötigt man eine **Distanzfunktion** im Eingaberaum.
- oder alle Neuronen des Clusters untereinander mit **inhibitorischen** Synapsen verbinden, so daß im Ruhezustand nur noch der Gewinner aktiv sein kann.
- oder die Neuronen mit nach "**Abstand**" im Ausgaberaum variierender Stärke inhibitorisch/excitorisch untereinander verbinden. Dabei bilden sich Nachbarschaften von Neuronen heraus.

# Sieger-Neuron

- Sieger ist das Neuron mit Gewichtsvektor (= **Repräsentant**) von der kleinsten Distanz (bzw. maximalem Skalarprodukt, Kreuzkorrelation, ...) zum Eingabevektor.
- Das Skalarprodukt ist (normiert) der cos des Winkels zwischen den beiden Vektoren und umso größer, je kleiner der Winkel ist.



# Distanzfunktionen

Auf dem Inputraum können sehr verschiedene Distanzen verwendet werden, um die maximale Ähnlichkeit zwischen Input  $x$  und Repräsentant  $w$  zu ermitteln, z.B.:

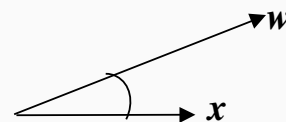
- minimale euklidische Distanz  $|x - w| = \sqrt{(x - w)^2}$
- minimale Taxi-Distanz  $|x - w| = \sum |x_i - w_i|$
- maximales Skalarprodukt  $x \cdot w$

daher die Längen der Vektoren  $x$  und  $w$  eine bedeutende Rolle spielt, kann dieses Maß nur

dann gut funktionieren, wenn alle Repräsentanten von etwa derselben Größenordnung sind, denn dann mißt das Skalarprodukt i.w. den Winkel

zwischen Input und Repräsentant, denn  $x \cdot w = |x| \cdot |w| \cdot \cos(x, w)$ .

Es wird deshalb empfohlen, bei dem Einsatz des Skalarproduktes mit jeder Gewichts Anpassung auch eine Normierung des Gewichtsvektors (z.B. auf 1) vorzunehmen.

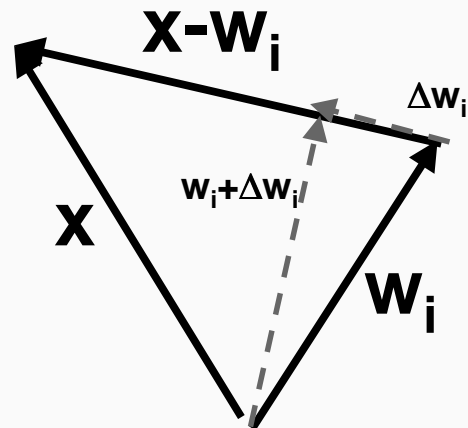


# Competitive-Lernregel1

$$\Delta w_i = \eta \cdot (x - w_i)$$

Diese Regel bewegt den Gewichtsvektor ein Stück auf den Inputvektor zu.

Die Gewichtsvektoren stellen sich deshalb auf die Mitte der Gruppe von Input-Vektoren ein, die diesen Repräsentanten als Sieger ansprechen.



# Competitive-Lernregel2

Satz:

Die Lernregel  $\Delta w_i = \eta \cdot (x - w_i)$  minimiert den quadratischen Fehler:

$$E = \frac{1}{2} \cdot (x - w_i)^2 = \frac{1}{2} \cdot \sum (x_j - w_{ij})^2$$

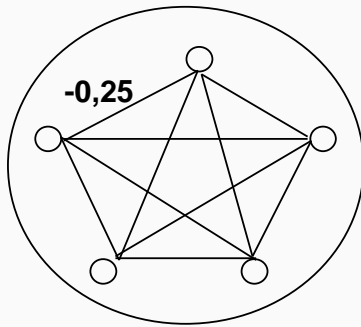
für jeden Input  $x$  mit Sieger-Neuron  $i$ .

Die Formel beschreibt nämlich gerade den Gradienten-Abstieg in der Landschaft der Distanzen zu den Repräsentanten (Fehlerlandschaft).

# Inhibitorische Cluster

Bei einem Cluster von  $n+1$  Neuronen verbindet man alle Neuronen durch Synapsen von negativen Gewicht (z.B. alle  $-1/n$ ).

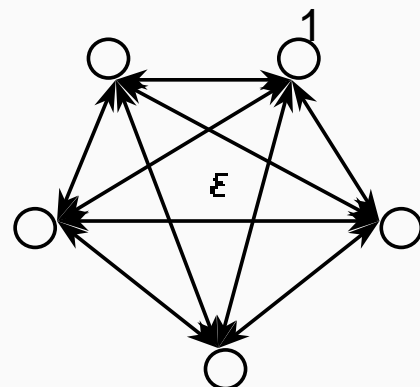
Im Ruhezustand hat dann nur noch der Gewinner positive Aktivität.



Die Synapsen zur Realisation von inhibitorischen Clustern werden problemabhängig festgelegt und nicht dem Training unterworfen.

# MAXNET (R.P. Lippmann 1982)

- Vollständig rückgekoppeltes Netz mit einheitlichen Gewichten  $w_{ik} = -\epsilon$  und Loops  $w_{ii} = 1$  ohne Lernregel.
- Verwendung zur Bestimmung des Maximums
- Auswertung durch Relaxation in einen stabilen Zustand, indem nur das Neuron mit maximalem Input aktiv ist.  
(Problem: stabiler Zustand kann nicht garantiert werden)



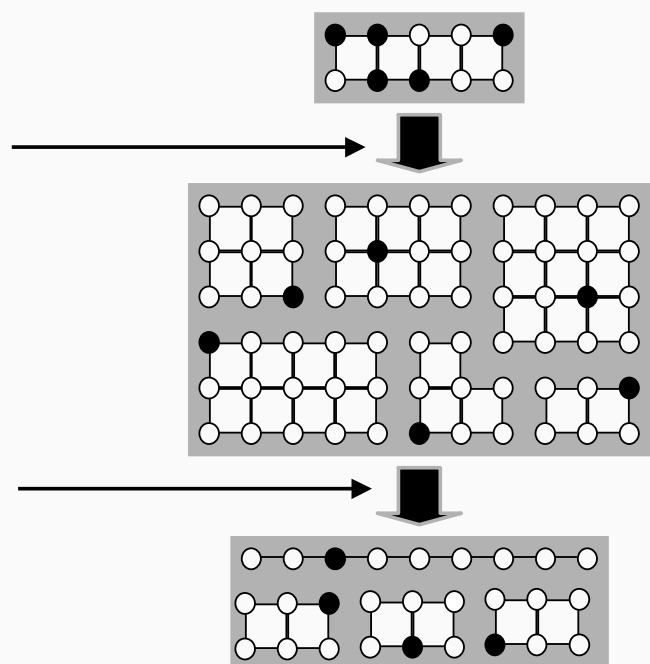
(R.P. Lippmann, B. Gold, M.L. Malpass: A Comparison of Hamming and Hopfield Neural Nets for Pattern Classification, MIT Lincoln Lab. Tech Report 769)

# competitive learning (Rumelhart/Zipser)

- Ein competitive-learning-network hat eine Input-Ebene und mehrere Cluster-Ebenen, deren Neuronen in disjunkte Clustereingeteilt sind.
- In jedem Cluster wird nun das Siegerneuron ermittelt und mit Wettbewerbsstraining  $\Delta w_i = \eta \cdot (\mathbf{x} - \mathbf{w}_i)$  die in dieses Neuron dieses Clusters führenden Verbindungen abgeändert.
- Nur das Gewinnerneuron liefert einen Output  $\neq 0$  an die nächste Schicht weiter, die nach demselben Prinzip trainiert wird.
- Die Arbeitsphase läuft ebenso ab, nur daß keine Gewichtsveränderungen mehr stattfinden.

## competitive learning struktur

- Input-Ebene
- Wettbewerbsstraining
- Cluster-Ebene(n)
- Wettbewerbsstraining
- Cluster-Ebene





# Counterpropagation1

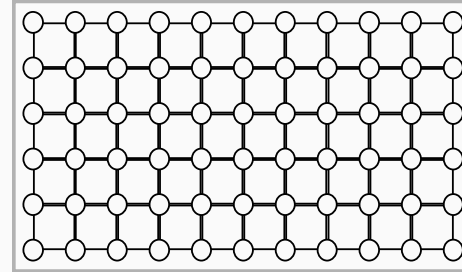
- Inputschicht



Wettbewerbslernen



- Kohonen-Schicht



Grossberg-lernen



- Outputschicht



# Counterpropagation2

- Der Input wird wie üblich in die verborgene (**Kohonen-**) Schicht propagiert (Skalarproduktregel) und daraus der Sieger (Maximum) ermittelt.
- Vereinbarungsgemäß gibt nur das Siegerneuron eine 1 an die (Grossberg-)Ausgabeschicht weiter, alle anderen 0.
- Die Gewichtsveränderung erfolgt nun wiederum nach dem Wettbewerbslernen  $\Delta \mathbf{w}_i = \eta \cdot (\mathbf{x} - \mathbf{w}_i)$
- Die Gewichtsveränderung der Gewichte  $\mathbf{v}_i$  vom Sieger zur Ausgabeschicht wird nach der Perceptron-Regel trainiert:  $\Delta \mathbf{v}_i = \alpha \cdot (\mathbf{t} - \mathbf{v}_i)$  ( $\mathbf{t}$  ist der Target-Output)
- Im **interpolativen** Modus gibt es eine Gruppe von Siegern mit Output  $y_i$  und mit  $\Delta \mathbf{v}_i = \alpha \cdot y_i \cdot (\mathbf{t} - \mathbf{v}_i)$

# Counterpropagation3

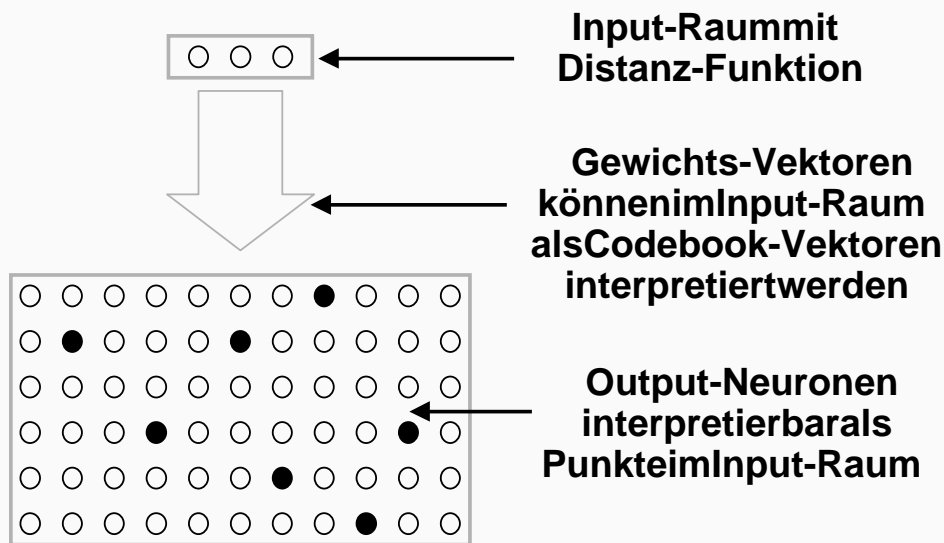
## Initialisierung der Gewichtsvektoren.

- **zufällig (normiert!)**: dabei kann es zu Problemen kommen, weil ein Großteil der Repräsentanten nie Sieger wird.
- **konvexe Kombination** : Alle Gewichtsvektoren normiert und gleich  $w$ . Inputs setzen auf  $x^{neu} = a x^{alt} + (1-a)w$  mit langsam von 0 auf 1 wachsendema.
- **verrauschte Eingaben** : Die Inputs werden zunächst stark und später immer geringer verrauscht eingegeben.
- **Veränderung aller Gewichte** : Es werden alle Gewichte von Neuronen trainiert, die eine Aktivität nahe dem Sieger haben (mit immer kleiner werdendem Abstand).
- **Conciencfactor** : Jedes Neuron zählt die Anzahl der ihn ansprechenden Inputs und mit steigender Anzahl wird (vorübergehend) die Länge (Siegesmöglichkeit) etwas reduziert.

# Input/Output-Trennung

Counterpropagation ist ein Beispiel für die Trennung des Trainings nach Input und Output. Die Kohonen-Schicht erzeugt eine Klassifikation nach Ähnlichkeit und die Grossberg-Schicht erzeugt zu jeder Klasse den Output.

- **PROBLEM:**  
Diese Methode versagt, wenn gleich klassifizierte Inputs sehr verschiedene Outputs verlangen, weil die Perceptron-Regel zwischen den verschiedenen Outputs mittelt.
- In Bereichen starker Varianz des Soll-Output solltendie Repräsentanten sehr viel dichter liegen, als bei geringer Varianz. Dies kann erreicht werden, indem man die Repräsentanten mit Input-Vektoren initialisiert, deren Output gleichmäßig verteilt sind.

**LernendeVektor-Quantisierung****LernendeVektor-Quantisierung**

- Es handelt sich um ein 2-Schichten-Netzwerk, indem jedes Input-Neuron mit jedem Output-Neuron verbunden ist.
- Die Gewichte der Neuronen zum Output-Neuron bilden den Repräsentanten  $w_i$ , genannt Codebook-Vektor.
- Das Netz soll überwachen eine vorgegebene Klasseneinteilung lernen, deshalb gibt es zu jeder Klasse einen oder mehrere Codebook-Vektoren dieser Klasse.
- Das Netz realisiert eine **nächste-Nachbar-Klassifikation**, d.h. zum Input wird der nächste Codebook-Vektor (Distanz im Input-Raum) gesucht und dessen Klasse zugeordnet.

**Lernalgorithmus der Vektor-Quantisierung**

- Zum Input  $\mathbf{x}$  wird der nächste Codebook-Vektor  $\mathbf{w}_i$  gesucht (bzgl. einer Distanz im Input-Raum).
- Der Codebook-Vektor  $\mathbf{w}_i$  wird nach der Wettbewerbsregel  $\Delta \mathbf{w}_i = \delta \cdot \eta \cdot (\mathbf{x} - \mathbf{w}_i)$  geändert, wobei  $\delta = +1$  gewählt wird, wenn die Soll-Klasse von  $\mathbf{x}$  mit der Klasse von  $\mathbf{w}_i$  übereinstimmt, und sonst  $\delta = -1$ .
  - Bei korrekter Klassifikation wird der Codebook-Vektor auf den Input zu verändert und andernfalls vom Input weg.
- Der Lernfaktor  $\eta$  ist nicht konstant, sondern wird mit der Zeit langsam auf 0 gesenkt.

**Initialisierung**

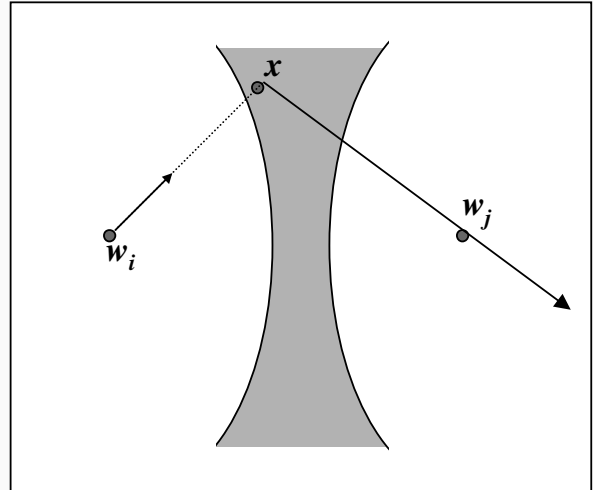
Ähnlich wie bei der Counterpropagation kann man zwischen mehreren Initialisierungsarten (mit denselben Problemen) wählen.

- **zufällig:**  
Die Gefahr besteht, daß einige Codebook-Vektoren nie Sieger werden und daß die zufällige Klassenzuordnung nur unzureichend mit den Inputs übereinstimmt.
- **Input-Auswahl:**  
Die Codebookvektoren samt zugehörigen Klassen werden aus den Inputs ausgewählt, dabei ist darauf zu achten, daß genügend Repräsentanten aus jeder Klasse gewählt werden.

**erweiterteLernenregelnLVQ2,LVQ3**

- StattnureinenSiegerzuermittelnwerdendiebeidenbesten Codebook-Vektoren  $\mathbf{w}_i, \mathbf{w}_j$  ermittelt.
- LVQ2:Gelerntwird,wenn  $\mathbf{w}_i, \mathbf{w}_j$  ausverschiedenenKlassensind,  $\mathbf{x}$ derKlassevon  $\mathbf{w}_i$ angehört und  $\mathbf{x}$ ineinemFensterzwischen denbeidenKlassenliegt:  

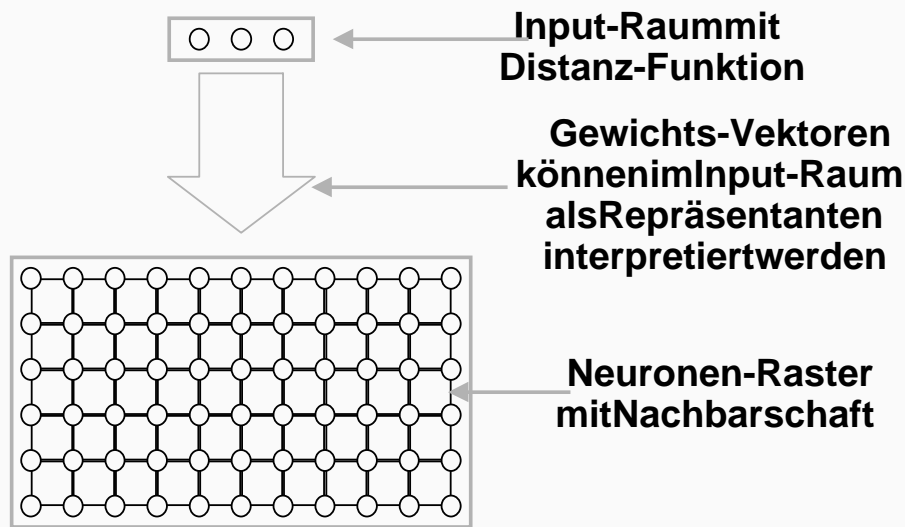
$$\frac{1+v}{1-v} d(\mathbf{x}, \mathbf{w}_i) > \frac{1-v}{1+v} d(\mathbf{x}, \mathbf{w}_j) \quad v \in [0.2, 0.3]$$
- Lernregel:  $\Delta \mathbf{w}_i = \delta \eta \cdot (\mathbf{x} - \mathbf{w}_i)$
- LVQ3:zusätzlichlernen,wenn  $\mathbf{x}, \mathbf{w}_i, \mathbf{w}_j$ inderselbenKlassesind.

**LVQAnmerkungen** (Kohonen)**DieLeistungsfähigkeitderLVQhängtabvon:**

- derZahlderCodebook-VektorenproKlasse
- derInitialisierungderCodebook-Vektoren
- derverwendetenLernregel
- derrichtigenLernratefürdieeinzelnenSchritte
  - EineOptimierungdesLVQistzuerreichen,wennmandenNeuronen individuelleLernraten  $\eta$ gibtunddiesenachderRegel:  
 $\eta = \eta / (1 + \delta \cdot \eta)$ mitdenGewichtenupdated(OptimizedLVQ)
- demAbbruchkriteriumzurTerminierungdes Lernens

# Merkmals-Karten(SOM) (Kohonen)

## selbstorganisierteKarten



# SOMAufbau

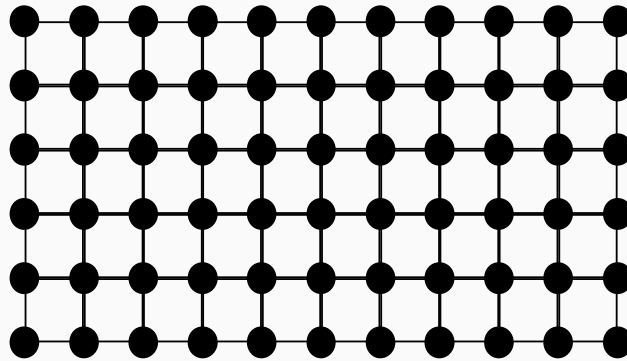
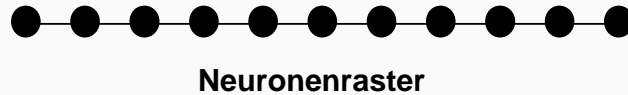
## selbstorganisierteKarten

- Die Netzstruktur ist ein 2-Schichten-Netz, dessen Ausgangsschicht aber in Rasterform angeordnet ist. (Rechteck- oder Waben-Form)
- Die Gewichtsvektoren werden zufällig initialisiert und als Repräsentanten im Inputraum interpretiert.
- Das Lernverfahren ist das nun zu beschreibende Nachbarschaftslernen  $\Delta w_k = \eta \cdot (x - w_i) \cdot h_r(d(i,k))$  mit den Parametern:
  - Lernfaktor  $\eta$
  - Distanzfunktion  $h_r$  im Input-Raum
  - Abstandsmaß im Neuronengitter
  - Reichweite im Neuronengitter
  - Reichweitenfunktion  $h_r$ : Abstand  $\rightarrow$  Gewicht

# Nachbarschaften1

## 1)Output-NeuronenineinemRasteranordnen

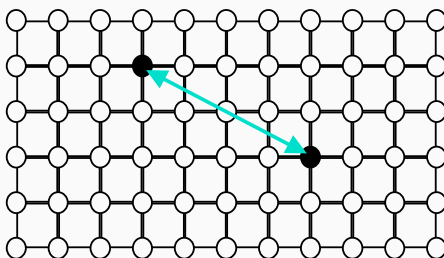
(meist1-oder2-dimensional,möglichistaberauchhöher-dimensional)



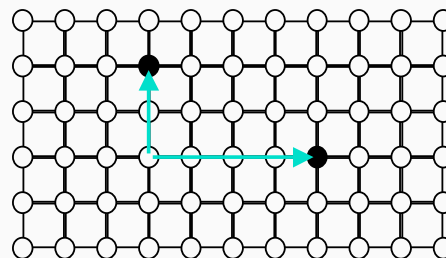
# Nachbarschaften2

## 2)Abstandd(i,j)aufdemRasterfestlegen.

euklidischerAbstand



Taxi-Abstand



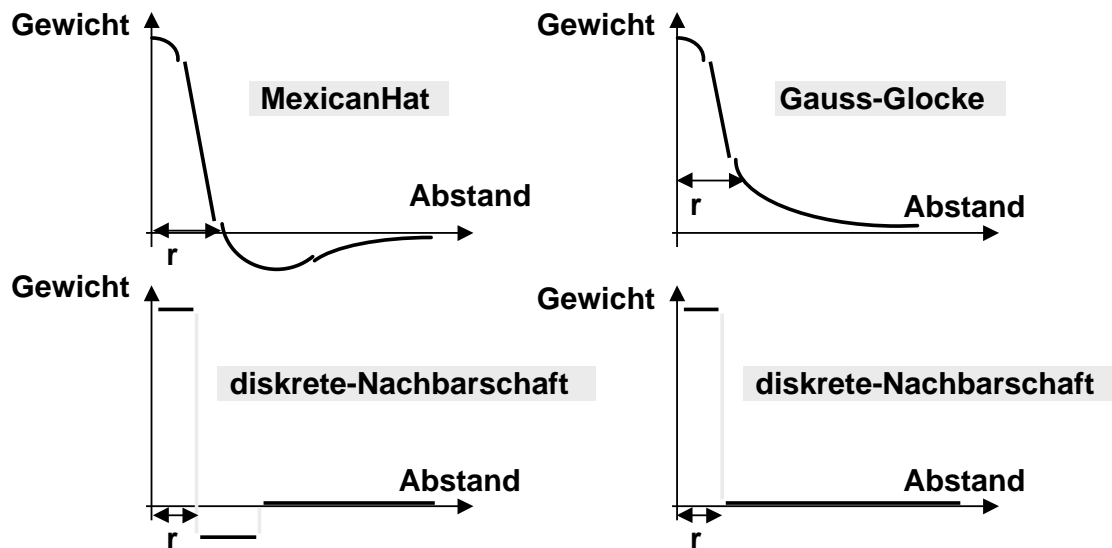
Hamming-Abstand

Anzahl der sich  
unterscheidenden  
Koordinaten

Maximum-Abstand

Maximum der  
Beträge der einzelnen  
Koordinaten

## 3) Funktion $h_r$ : Abstand Gewicht wählen.



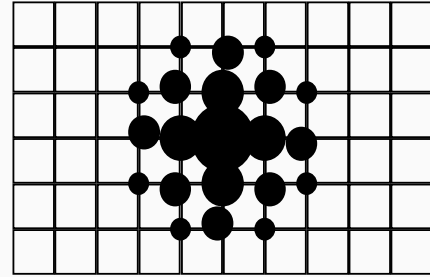
## 4) Nachbarschafts-Training.

- Zuerst wird aus dem Input das Sieger-Neuron ermittelt auf der Basis einer Distanzfunktion  $|...|$  auf dem Input-Raum.
- Nach der competitive-learning-Regel wird die Gewichtsänderung  $\Delta w$  für das Siegerneuron berechnet.
- Für die Neuronen  $j$  mit Abstand  $d(i,j)$  (im Raster) wird der Anteil dieses Neurons an der Gewichtsänderung vom Gewinner berechnet, d.h. der zugehörige Gewichtsvektor wird um  $\Delta w \cdot h_r(d(i,j))$  geändert.
- Damit das Training zu einem Ende kommt, muß die Lernkonstante  $\eta$  bei der Berechnung von  $\Delta w$  schrittweise herabgesetzt werden bis schließlich 0 erreicht wird.



## Erregungsfeld

Einderartiger Ansatz führt dazu, daß das Neuronenraster in einen Ausgabezustand hat, indem eine ganze Nachbarschaft des Gewinners aktiv ist.



Hier wird also nicht nur der Gewinner in das Training eingebezogen, sondern eine ganze Umgebung des Gewinners, die Reichweite entscheidet über die Größe der Umgebung.

## Lernen im Erregungsfeld

$$\Delta w_k = \eta \cdot (x - w_i) \cdot h(d(i, k))$$

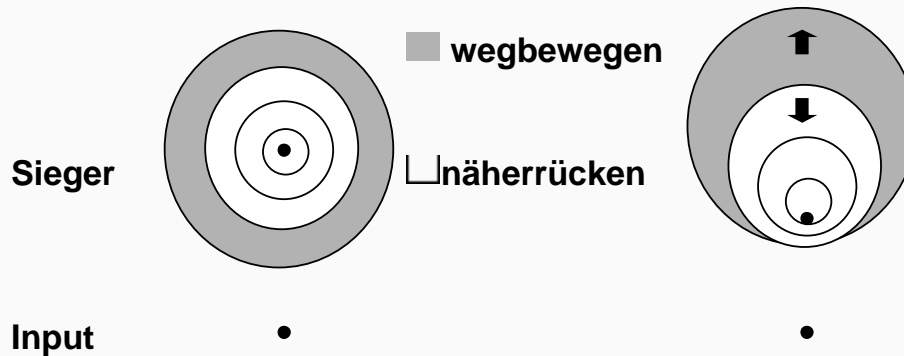
Diese Variante des Wettbewerbs-Lernens hat den Effekt, daß auch die Gewichtsvektoren in der Nähe des Gewinners an den Input-Vektor herangezogen werden, wogegen die etwas weiter entfernten Neuronen ihren Gewichtsvektor genau entgegengesetzt oder gar nicht bewegen.

# LernschrittSkizze

- Die Veränderung der Umgebung des Siegers bei einem Lernschritt kann man sich etwa wie folgt vorstellen:

Umgebung des Siegers  
vordem Lernen

Umgebung des Siegers  
nachdem Lernen

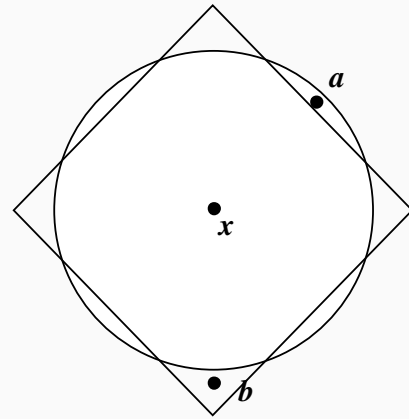


# Ziele des Nachbarschaftslernens

- Durch das Nachbarschaftslernen wird angestrebt, daß Inputs, die im Input-Raum nahe beieinander liegen, Neuronen aktivieren, die auch im Raster nahe beieinander liegen (topologische Abbildung).
- In Bereichen hoher Input-Dichte liegen auch viele Repräsentanten (hohe Repräsentantendichte).
- Diese eignen sich besonders zur Visualisierung, da zusammenhängende Bereiche ähnlicher Eingaben auch auf zusammenhängende Gebiete im Raster abgebildet werden.
- Durch die Lage im Raster können besonders wichtige Repräsentanten markiert und Übergänge zwischen diesen sichtbar gemacht werden.

# Distanz||imInput-Raum

- Die Distanzfunktion  $\|\cdot\|$  im Inputraum wird beim Training und beim Propagieren zur Ermittlung des Siegers eingesetzt.
- Daß euklidische Distanz  $\sum x_i^2$  und Taxi-Distanz  $\sum |x_i|$  zu verschiedenen nächsten Nachbarn führen kann, kann man sich an folgendem Bild klarmachen, indem  $a$  der euklidische Norm nächster Nachbar von  $x$  ist, nach Taxi-Distanz aber  $b$ .



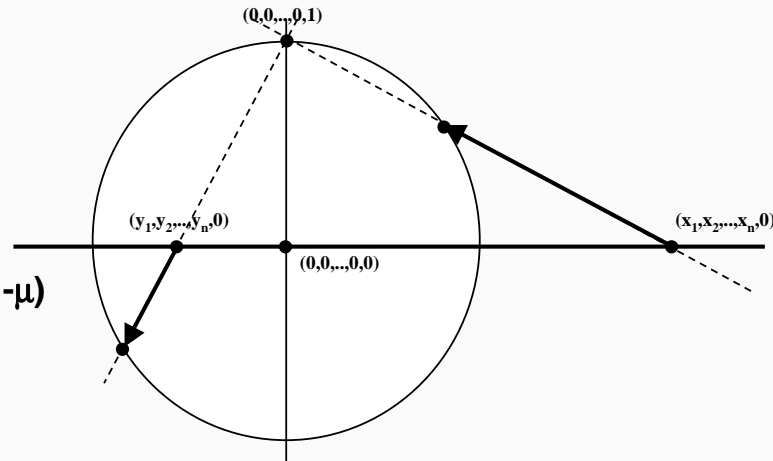
# Skalarprodukt

- Wählt man das Skalarprodukt  $|x||a|\cos(x,a)$  im Input-Raum als Distanzmaß zur Siegerbestimmung, dann werden Repräsentanten  $a$  großer Länge als Sieger bevorzugt.
- Es ist also entscheidend, alle Repräsentanten während des Trainings auf etwa dieselbe Länge zu halten.
- Dann hängt die Maximierung des Distanzmaßes nur von der Richtung von  $x$  (d.h. nur von den Größenverhältnissen zwischen den Komponenten von  $x$ ) und nicht von der Länge von  $x$  ab, deswegen empfehlen einige Autoren auch die Input-Vektoren auf dieselbe Länge zu normieren.

# Berücksichtigung der Länge

- Will man trotz Verwendung des Skalarproduktes neben der Richtung auch die Länge der Inputs berücksichtigen, so empfiehlt es sich, den Input-Raum auf die Einheitskugel des  $n+1$  Dimension größeren Raums zu projizieren:

- $f(x_1, x_2, \dots, x_n, 0) = (\lambda x_1, \lambda x_2, \dots, \lambda x_n, 1 - \lambda)$
- $\lambda^2 \sum x_i^2 + (1 - \lambda)^2 = 1$   
 $\lambda = 2 / (1 + \sum x_i^2)$
- Umkehrung:  
 $f^{-1}(u_1, u_2, \dots, u_n, 1 - \lambda) = (\mu u_1, \mu u_2, \dots, \mu u_n, \mu(1 - \lambda) + 1 - \mu)$
- $\mu(1 - \lambda) + 1 - \mu = 0$   
 $\mu = 1 / \lambda$



# Variante des Nachbarschaftslernens

Verringerung der Reichweite während des Trainings.

Nach einer groben topologischen Anordnung der Repräsentanten wird die Feineinstellung noch individuell gemacht.

Initialisierung der Gewichtsvektoren mit zufällig ausgewählten Inputs

Dadurch wird jedes Neuron wenigstens einmal der Sieger.

Normierung aller Input-Vektoren und Gewichte auf Länge 1.

Dann entspricht minimaler Winkel der minimalen Distanz und dem maximalen Skalarprodukt

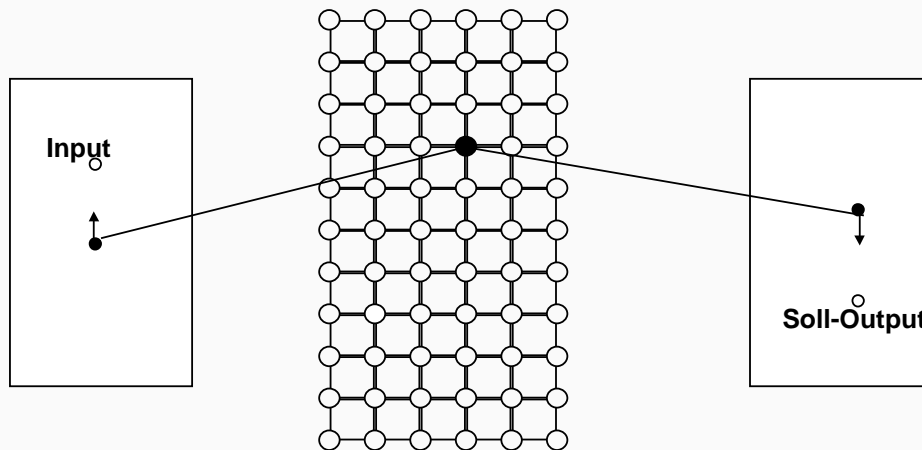
$$|x - a|^2 = |x|^2 - 2 \cdot x \cdot a + |a|^2 = 2 - 2 \cdot x \cdot a = 2 \cdot (1 - \cos(x, a)).$$

Zählender Gewinn-Häufigkeit der Neuronen mit sinkender Gewinnchance (conscience factor) bei hoher Trefferrate.

Dann werden auch nicht oder selten gewinnende Neuronen in den Lernprozess stärker einbezogen.

# Motorische Karten (Ritter, Schulten)

- Um Kohonens Algorithmus auch auf überwachtes Lernen anzuwenden, kann man die SOM (auch **sensorische Karte** genannt) zur **motorischen Karte** verallgemeinern.
- Zu jedem Trainingsinput hat man auch einen Soll-Output.



# Prinzip motorischer Karten

- Zu jedem Rasterneuron gehören zwei Gewichtsvektoren  $w_i$  (zum Input) und  $v_i$  (zum Output).
- Bei der Eingabe von  $x$  wird der zu  $x$  nächste Repräsentant  $w_i$  im Inputraum gesucht (ist der Gewinner). Der zu  $x$  gehörige Output  $u$  spielt bei der Minimumsuche keine Rolle.
- Wird nicht trainiert, ist  $v_i$  der Output des Netzes.
- Training:  
In der Nachbarschaft des Gewinners wird nach der Kohonen-Regel  $\Delta w_k = \eta \cdot (x - w_i) \cdot h_r(d(i, k))$  gelernt. Auch der Ausgabe repräsentant  $v_i$  und dessen Umgebung wird trainiert:  $\Delta v_k = \mu \cdot (u - v_i) \cdot h_r(d(i, k))$ .

- Der Netzoutput  $v_i$  kann auch als eine echte Netzausgabe realisiert werden, wenn man alle Rasterneuronen außer dem Sieger eine 0 ausgeben läßt und den Sieger eine 1.
- Mit dieser Festsetzung kommt in der Ausgangsschicht genau der Vektor  $v_i$  an.
- Die gewünschte Ausgabe in der Rasterschicht läßt sich dadurch realisieren, daß man die Raster-Neuronen mit einer radialen Basisfunktional Outputfunktion versieht und deren Radius ( $\approx 1/\text{Steigung}$ ) abhängig von der Dichte der Repräsentanten klein genug wählt.
- Damit macht der Sieger einen Output nahe bei 1, wogegen die übrigen Neuronen fast 0 ausgeben.

## Problem motorischer Karten

- Da die Ermittlung des Siegerneurons nur vom Input abhängt, ist die Modellierung von Funktionen, die auf kleinen Distanzen hohe Varianz haben sehr problematisch.
- Dieses Problem kann abgemildert werden, wenn man beim Training (mit abnehmender Gewichtung) auch die Distanz der Outputs mit in die Siegerermittlung einbezieht ( $|x - w_i| + \lambda |u - v_i|$ ,  $\lambda \rightarrow 0$ )

Dies hat den Effekt, daß sehr nahe beieinander liegende Inputs mit erheblichen unterschiedlichen Outputs verschiedene Repräsentanten anziehen und damit an solchen kritischen Stellen die Repräsentanten erheblich dichter liegen als anderswo.

# AdaptiveResonanceTheory(ART)

- Das **Stabilitäts-Plastizitäts-Dilemma**  
Wie kann ein Netzwerk neue Assoziationen dazulernen, ohne alte Assoziationen zu vergessen?
- Die Adaptive Resonanztheorie (Carpenter/Grossberg) versucht eine Lösung dieses Problems im Falle der  
**Klassifikation binärer Eingabemuster**
- Das Lernverfahren verbindet ein Wettbewerbslernen mit **Aufmerksamkeit**-gesteuerter Topologieveränderung:
  - Suche zu einer Eingabe die bestmögliche Klassifikation
  - Ist diese Klassifikation zu schlecht (Abstand des Prototyps vom Input größer als die Aufmerksamkeit) erzeuge ein neues Neuron mit zugehörigen Verbindungen die eine gute Klassifikation abgibt.
  - Ist die Klassifikation gut, verbessere sie mit Wettbewerbslernen.

## Aufbau von ART

- Das Netz hat zwei Schichten, die Eingabeschicht (**comparison-layer**) mit  $n$  Neuronen und die Ausgabeschicht (**recognition-layer**)
- Zu jedem Neuron  $j$  der Ausgabeschicht gibt es einen reellen Verbindungsvektor  $w_j = (w_{j1}, \dots, w_{jn})$  und einen binären Prototypen  $b_j = (b_{j1}, \dots, b_{jn})$ .
- Es gibt eine Konstante  $\rho \in [0, 1]$ , die sog. **Aufmerksamkeit**, die ein Maß dafür ist, wie grob oder fein die Klasseneinteilung werden soll (meistens  $0.7 < \rho < 0.99$ ).
- Initialisierung:  $b_j = (1, \dots, 1)$ ,  $w_j = \alpha \cdot b_j$   
mit  $\alpha = L / (L - 1 + |b_j|)$  für ein  $L > 1$  und  $|x| = \sum x_i$ ,  
 $|x|$  ist das **Hamming Gewicht** (=Anzahl der 1'en) des Vektors  $x$ .

# Arbeitsweise von ART

ART ist konzipiert für binäre Eingabevektoren, es gibt aber auch Erweiterungen (ART2, ART3) auf reelle Eingaben. Nach der Initialisierung arbeitet ART in 4 Phasen:

- Erkennungsphase( **recognition**)  
es wird das zur Eingabe am besten passende Output-Neuron ermittelt.
- Vergleichsphase( **comparison**)  
Die Eingabe wird mit dem Prototyp verglichen.
- Suchphase( **search**)  
Fällt der Vergleich zu schlecht aus, wird das nächstbeste Output-Neuron ermittelt(--> Vergleichsphase)
- Lernphase( **training**)  
Es wird nötigenfalls ein neues Neuron in die Ausgangsschicht gesetzt und die Gewichte und Prototypen des Gewinners angepaßt.

# ART recognition

- Der Eingabevektor  $x$  wird mit der Gewichtsmatrix  $W$  multipliziert  $a = Wx$  und liefert für jedes Ausgabeneuron  $j$  einen Wert  $a_j$ .
- Unter diesen wird das Maximum (der Sieger) gesucht und die (vorläufige) Netzausgabe ist der Einheits-Vektor  $u$  mit  

$$u_j = (a_j \text{ maximal}) ? 1 : 0$$
 Interpretation: Die Eingabe  $x$  gehört zu Klasse  $j$ !
- Fällt  $u$  bei der Vergleichsphase durch, so wird  $a_j = 0$  gesetzt und  $u$  Neuberechnet.
- Ist  $u = 0$ , so muß ein neues Ausgabeneuron mit Prototyp  $b_j = (1, \dots, 1)$ , Gewicht  $w_j = L / (L - 1 + |b_j|) \cdot b_j$  eingerichtet werden, das dann Gewinner ist.



# ARTcomparison

- Der momentane Outputvektor  $u$  wird mit der Matrix  $B$  multipliziert und produziert damit den Prototyp  $b_j$ .
- Nun wird der Prototyp  $b_j$  mit dem aktuellen Input  $x$  verglichen:  
Berechne das **Hamming-Gewicht** (=Anzahl der 1'en)  $|x|$  und  $|x \text{ AND } b_j|$  der Vektoren  $x$  und  $x \text{ AND } b_j$  und berechne deren Quotient.  
(wieviel Prozent der in  $x$  besetzten Stellen sind auch in  $b_j$  besetzt?)
- Ist  $|x \text{ AND } b_j| / |x| > \rho$ , so war der Vergleich erfolgreich (z.B. wenn  $b_j = (1, \dots, 1)$  ist) und es geht weiter zum *training*.
- Ist der Vergleich nicht erfolgreich, so wird  $a_j = 0$  gesetzt und in die recognition-Phase zurückgekehrt.

# ARTtraining

- In diesem Schritt werden sowohl der Prototyp wie auch der Gewichtsvektor des Gewinnerneurons angepaßt, dabei ist der Gewichtsvektor immer die "normalisierte" Form des Prototyps.
- $b_j^{\text{neu}} = x \text{ AND } b_j^{\text{alt}}$ .
- $w_j^{\text{neu}} = L / (L - 1 + |b_j|) \cdot b_j$ .
- Folgende Eigenschaft des Trainings sind bewiesen:
  - Jeder Trainingsvektor findet seinen Gewinner ohne Suche.
  - Das Training ist stabil: kein Inputwechsel seine Klasse.
  - Das Training terminiert