Einführung in die Theorie der
Neuronalen Netze

Vorlesung 5

Unüberwachtes Lernen

Alexander Förster
Universität Bremen

Wintersemester 2016 / 2017

Aufgabenbesprechung

1) Implementiere den Perzeptron Lernalgorithmus für n-dimensionale Vektoren und teste ihn an zufälligen Daten:
   - Wähle einen Gewichtsvektor w zufällig.
   - Generiere p Punkte zufällig im Raum und klassifizieren sie diese in P und N mit dem gegebenen Gewichtsvektor w.
   - Teste den Algorithmus in Bezug auf die Anzahl der notwendigen Iterationen mit einem neu initialisierten Gewichtsvektor w_0.
   - Zeichne einen Graphen, der n und p bezüglich der Laufzeit für bis zu 10 Dimensionen und 100 Punkte zeigt.
2) **Verständnisfrage:** Gebe ein Beispiel für eine Trainingmenge an, bei der der Perzeptron Lernalgorithmus viele Iterationen braucht. Begründe.

Zusatzaufgabe

Konvolution. Benutze ein Bildbearbeitungsprogramm, um die Kanten in einem Bild (schwarz/weiß) zu erkennen. Nehmen alle Pixel des Bildes um ein gegebenes Pixel für die Eingabemenge und den Wert des Pixels in dem Kantenbild als Klassifizierung P oder N. Lerne ein Perzeptron.

- Welche Werte haben die Gewichte?
- Teste den Algorithmus an dem ursprünglichen und einem weiteren Bild.

Zusatzaufgabe

- Gestartet wurde mit den Gewichten

  0.9003 -0.5377 0.2137

  -0.0280 0.7826 0.5242

  -0.0871 -0.9630 0.6428

  und Schwellwert -0.1105.
- Es wurden nach 280 Korrekturen 10000 Tests ohne Korrektur durchgeführt (gesamt 17090 Tests).
- Gewichtsmatrix nachher:

  -3.0997 -5.5377 -2.7863

  -5.0280 29.7826 -3.4758

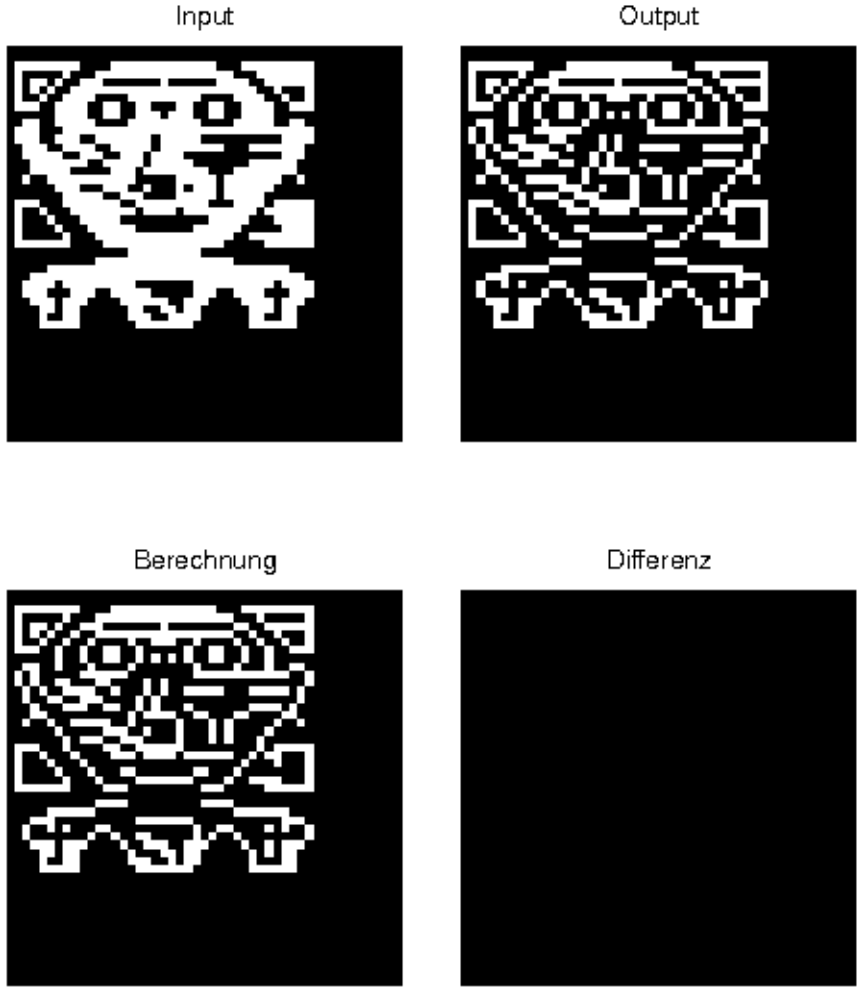
  -5.0871 -4.9630 -5.3572

  mit dem Schwellwert -0.1105.
- Normalisiert auf den Schwellwert 0.5 und das mittlere Gewicht 8 ergibt sich die Gewichtsmatrix:

  -0.6552 -1.2969 -0.5727

  -1.1627 8.0000 -0.7542

  -1.1783 -1.1456 -1.2494

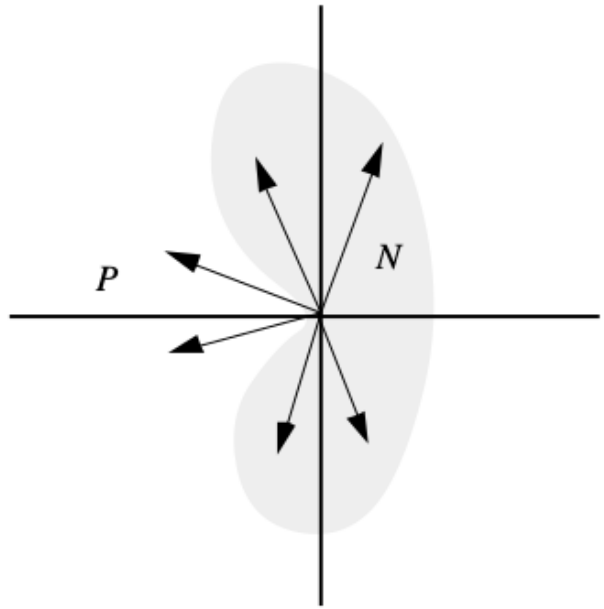


Unüberwachtes Lernen

- Der Perzeptron Lernalgoritmus ist ein Beispiel für überwachtes Lernen eines Neurons.
- Bei unüberwachtem Lernen organisiert sich das Netz selbst. Wir können zwischen Verstärkungslernen (Hebbian Learning) und Wettbewerbslernen unterscheiden.
- Beim **Verstärkungslernen** werden Gewichte (Verbindungen) verstärkt, die gleichzeitig aktiv sind.
- Beim **Wettbewerbslernen** stehen die Neuronen in Konkurrenz zueinander. Das Neuron mit der „richtigsten“ Antwort darf seine Gewichte optimieren. Alle anderen Neuronen werden unterdrückt.
- Anwendungen:
  - (verlustbehaftete) Datenkompression (Bild, Ton, hochdimensionale Daten)
  - Mustererkennung (Sprache, Zeichen)
  - Kodierung und Fehlerkorrektur in der Signalverarbeitung

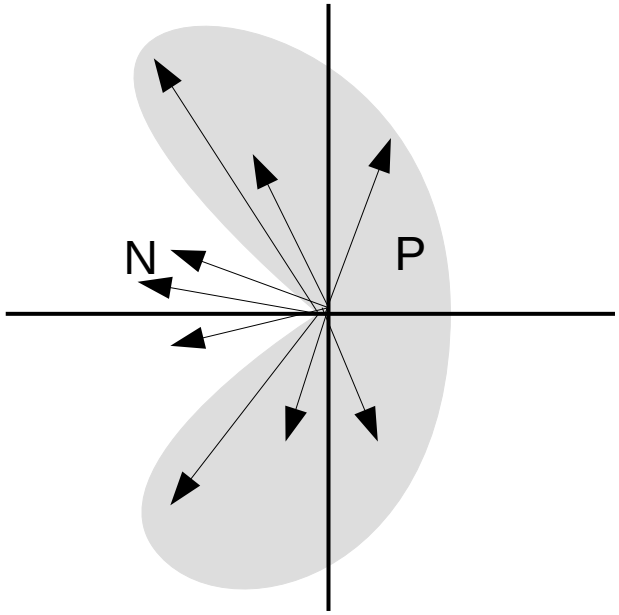
Motivation: Vom Neuron zum Netz

- Ein Perzeptron ist oft nicht genug, wenn die Daten nicht linear trennbar sind.



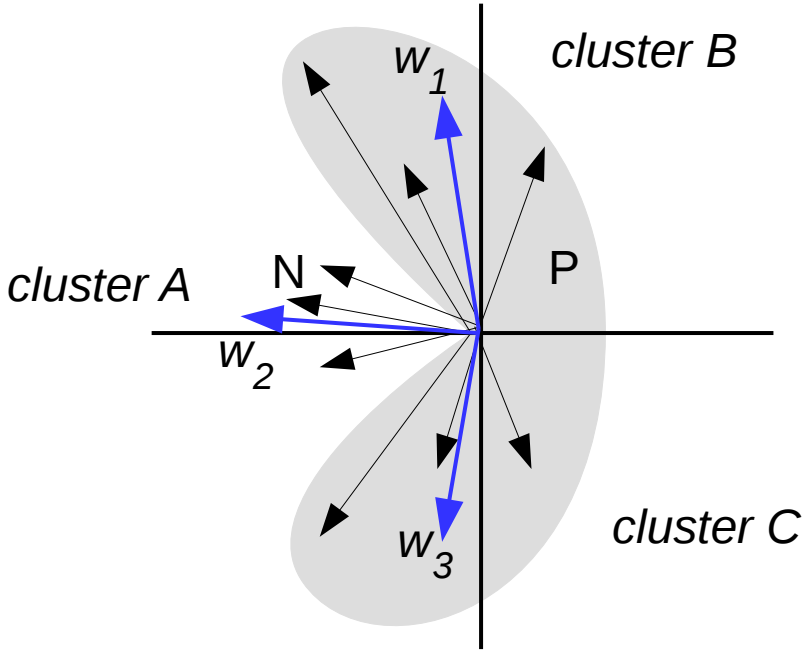
Motivation: Vom Neuron zum Netz

- Ein Perzeptron ist oft nicht genug, wenn die Daten nicht linear trennbar sind.
- Kein Perzeptron mit Gewichtsvektor w kann P und N für alle $p \in P$, $n \notin P$ mit $pw \geq 0$ und $nw < 0$ trennen.



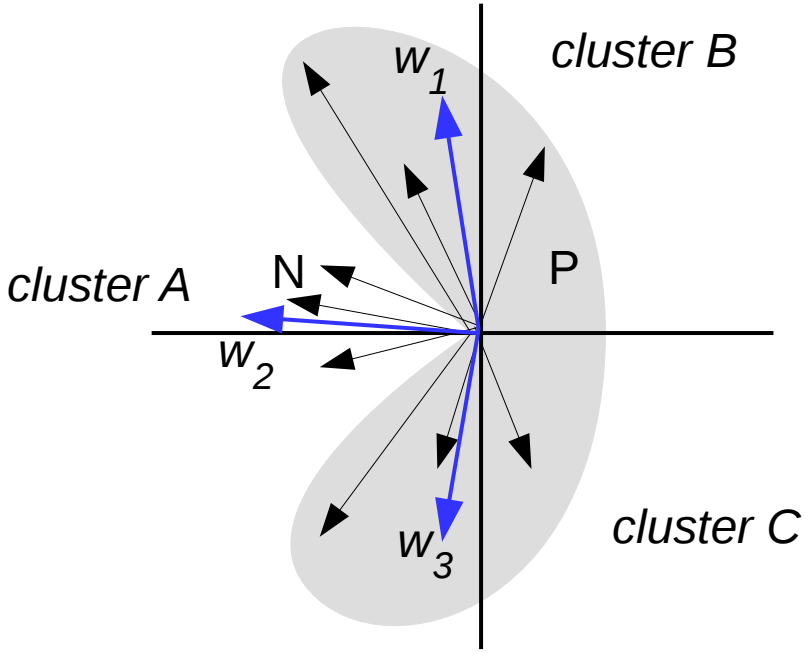
Motivation: Vom Neuron zum Netz

- Ein Perzeptron ist oft nicht genug, wenn die Daten nicht linear trennbar sind.
- Kein Perzeptron mit Gewichtsvektor w kann P und N für alle $p \in P$, $n \notin P$ mit $pw \geq 0$ und $nw < 0$ trennen.
- Wir können aber drei Vektoren w_1, w_2, w_3 finden, die als Repräsentanten für jeweils eine Region (einen Cluster) stehen.



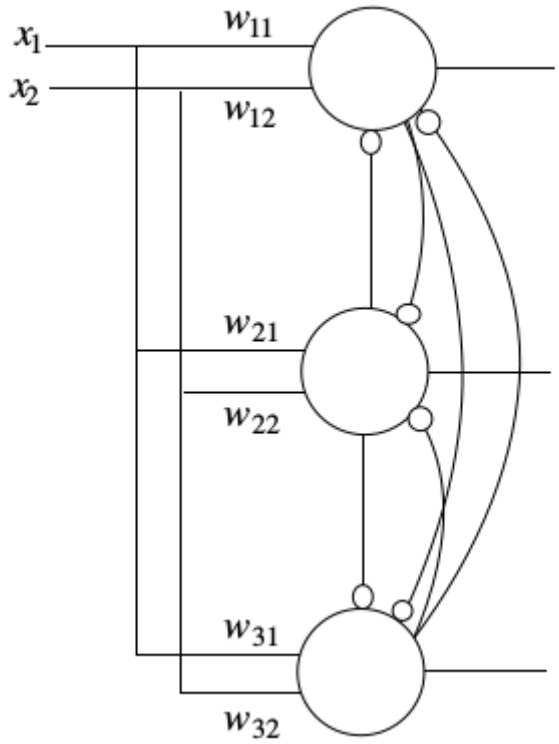
Motivation: Vom Neuron zum Netz

- Jeder Gewichtsvektor bestimmt ein Neuron, das nur feuert, wenn der Eingabevektor nahe genug am Gewichtsvektor liegt (Skalarprodukt ist positiv und groß). Halbraum geht nun nicht mehr.
- Wir können Repräsentanten per Hand auswählen.
- Im Allgemeinen ist die Anzahl der Cluster unbekannt. Das ist das **Cluster-Problem**.



Netz von konkurrierenden Neuronen

- Die Perzeptrone haben zusätzliche inhibitorische Leitungen, die nach dem Prinzip **winner-takes-all** funktioniert. Nachteil: Dafür ist **globale Information** über die das Aktivierungspotential alle Neuronen notwendig.
- Auch die Lernregel wird dementsprechend modifiziert, dass nur das am stärksten erregte Neuron seinen Gewichtsvektor ändert.
- Eine negative Menge brauchen wir in dem Sinne nicht mehr.



Konkurrenzlernalgorithmus

Eingabe: Erweiterte und *normalisierte* Eingabevektoren $X=\{x_1,...,x_l\}$ eines n-dim VR, die in k unterschiedliche Cluster klassifiziert werden sollen.

Start: Die erweiterten Gewichtsvektoren $w_1,...,w_k$ werden zufällig initialisiert und *normalisiert*.

Test:

- Wähle Vektor $x_j \in X$ zufällig.
- Berechne $x_j \cdot w_i$ für alle $i=1,...,k$.
- Wähle w_m mit $x_j \cdot w_m \geq x_j \cdot w_i$ für alle $i=1,...,k$.
- Weiter mit Update

Update:

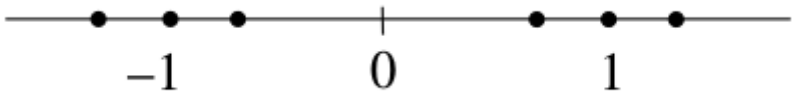
- Ersetze w_m mit w_m+x_j und normalisiere.
- Weiter mit Test.

Konkurrenzlernalgorithmus

- Knoten, die nie eine Update bekommen werden als **tote Knoten** (dead units) bezeichnet.
- Die Update-Regel $w_m \leftarrow w_m + x_j$ kann auch abgeändert werden:
  - Mit Lernrate: $\Delta w_m = \eta x_j$ mit $\eta \in (0,1]$
  - Mit Differenz: $\Delta w_m = \eta(x_j - w_m)$
  - Stapelbearbeitung: Δw_m wird für den entsprechenden Vektor m akkumuliert. Erst nach einer gewissen Zeit werden alle Gewichte korrigiert.

Konvergenzanalyse

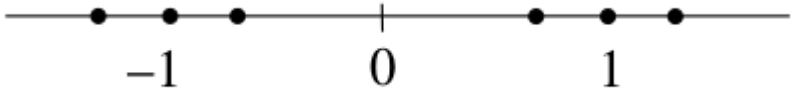
- Die Konvergenzanalyse ist bei unüberwachtem Lernen schwierig. Kann bei schlechter Initialisierung oder falscher Anzahl von Gewichtsvektoren auch nicht konvergieren.
- Beispiel im 1-dimensionalen mit Punkten {−1,3; −1,0; −0,7; 0,7; 1,0; 1,3} (nicht normiert):

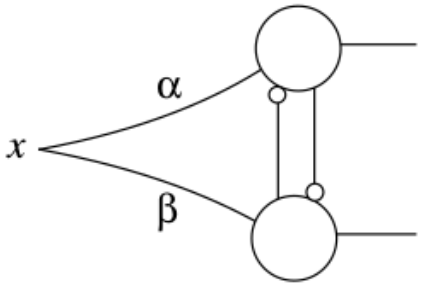


- Bei **einem** Gewicht wird es sich bei 0 einpendeln, wenn η geeignet sinkt.

Konvergenzanalyse

- Die Konvergenzanalyse ist bei unüberwachtem Lernen schwierig. Kann bei schlechter Initialisierung oder falscher Anzahl von Gewichtsvektoren auch nicht konvergieren.
- Beispiel im 1-dimensionalen mit Punkten {−1,3; −1,0; −0,7; 0,7; 1,0; 1,3} (nicht normiert):

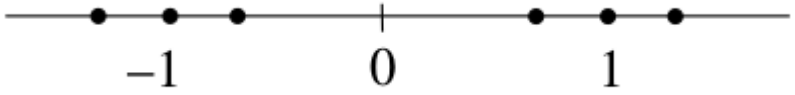


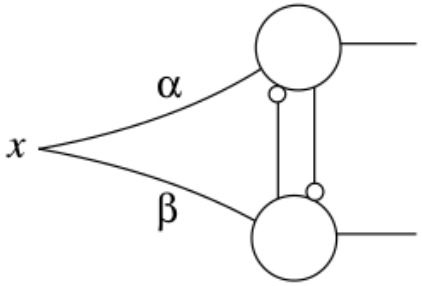


- Bei **zwei** Gewichten α,β ist eine stabile Lösung bei z.B. α=-1 und β=1. Die Gewichte verlassen wegen des großen Abstands nicht „ihr Gebiet“.

Konvergenzanalyse

- Die Konvergenzanalyse ist bei unüberwachtem Lernen schwierig. Kann bei schlechter Initialisierung oder falscher Anzahl von Gewichtsvektoren auch nicht konvergieren.
- Beispiel im 1-dimensionalen mit Punkten {−1,3; −1,0; −0,7; 0,7; 1,0; 1,3} (nicht normiert):

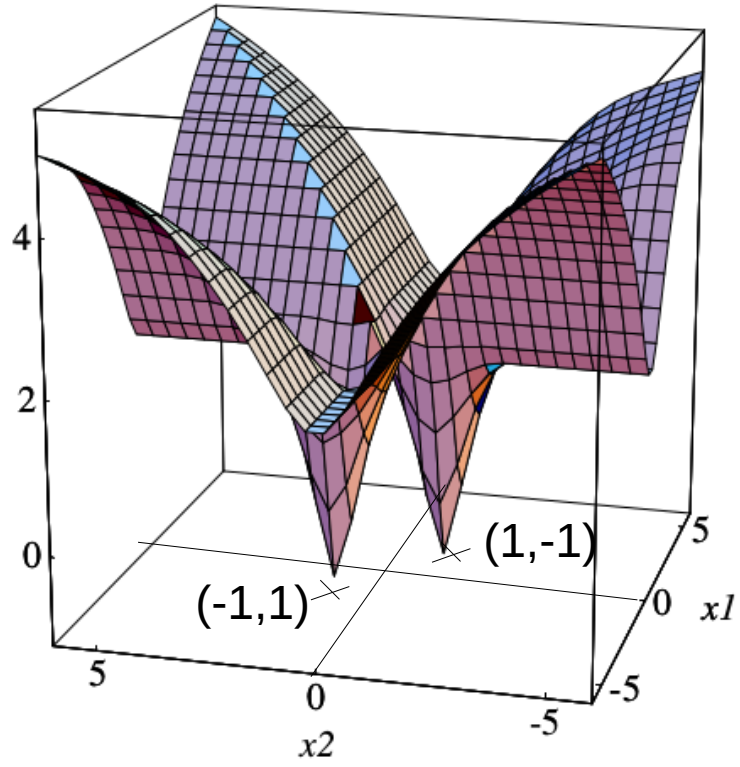




- Bei **zwei** Gewichten α,β und der Initialisierung α=0 und β=10 wird β nie gewählt und α deckt beide Cluster ab.

Konvergenzanalyse

- Je nach Anfangsbedingungen fallen die Gewichte (beide!) in (ein!) globales Minimum oder sie bleiben auf einem Plateau (lokales Minimum).

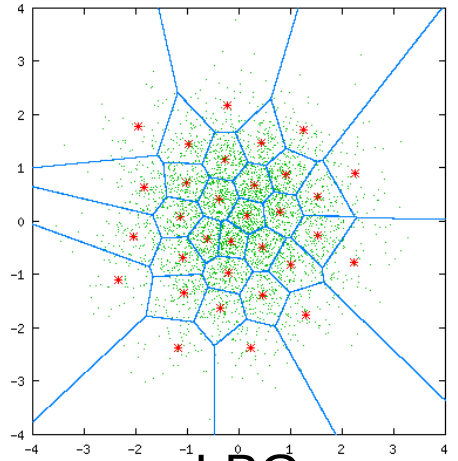


Vektorquantisierung

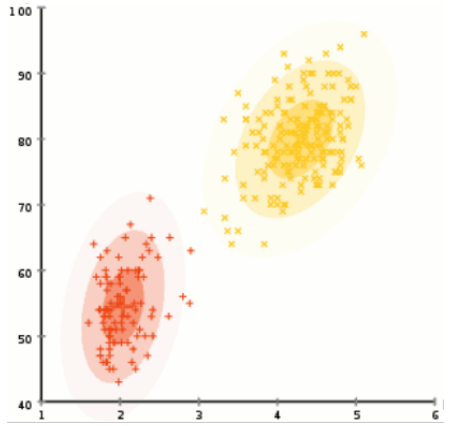
- Der Konkurrenzlernalgorithmus ist ein Vektorquantisierungsalgorithmus (VQ).
- Bei der VQ werden die Datensätze in Merkmalsvektoren zusammengefasst.
- Andere bekannte Verfahren zum Clustering sind
  - k-means (nicht zu verwechseln mit k-nächsten Nachbarn)
  - LBG (Linde-Buzo-Gray)
  - EM (Expectation-Maximization)
  - ...

$$\arg\min_{\mathbf{S}} \sum_{i=1}^{k} \sum_{\mathbf{x} \in S_i} \|\mathbf{x} - \boldsymbol{\mu}_i\|^2$$

k-means



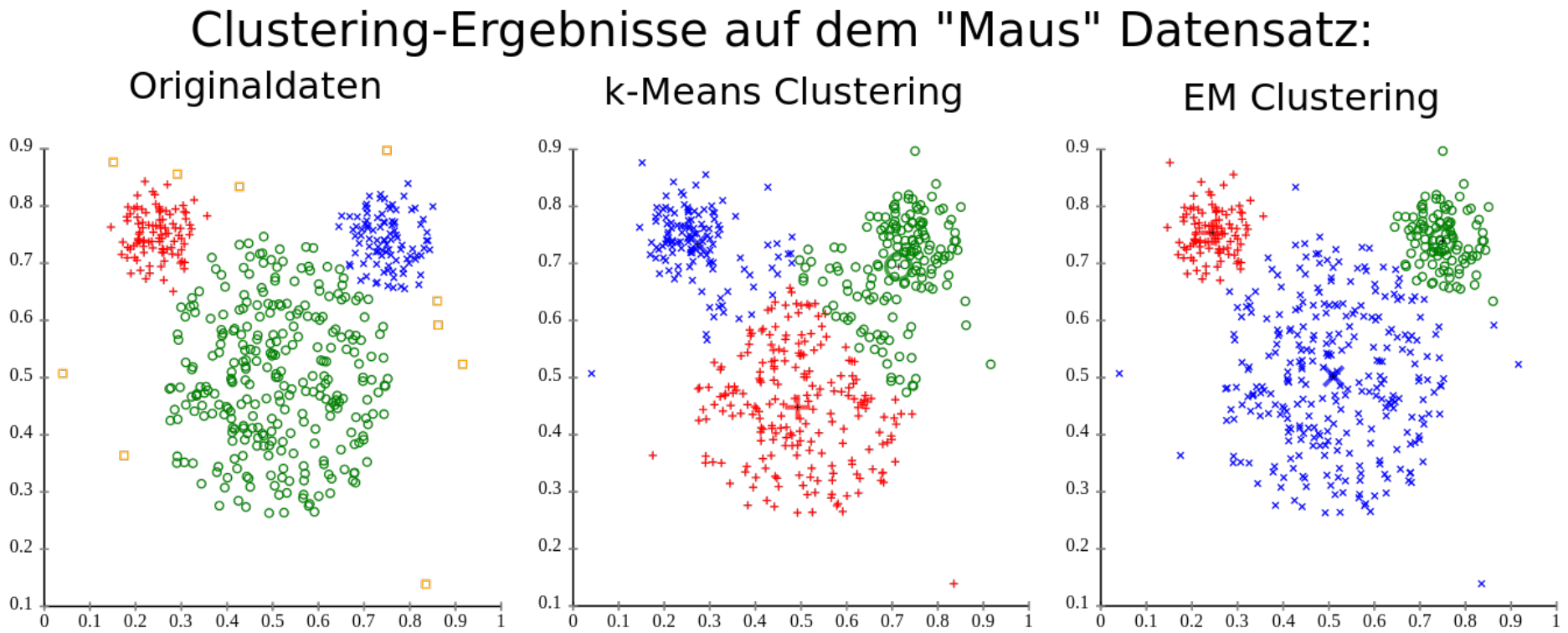
LBG



EM

Quelle: en.wikipedia.org/wiki/Expectation-maximization_algorithm
Quelle: www.data-compression.com/vq.shtml
Quelle: en.wikipedia.org/wiki/K-means_clustering

Vergleich k-means - EM



Quelle: de.wikipedia.org/wiki/EM-Algorithmus

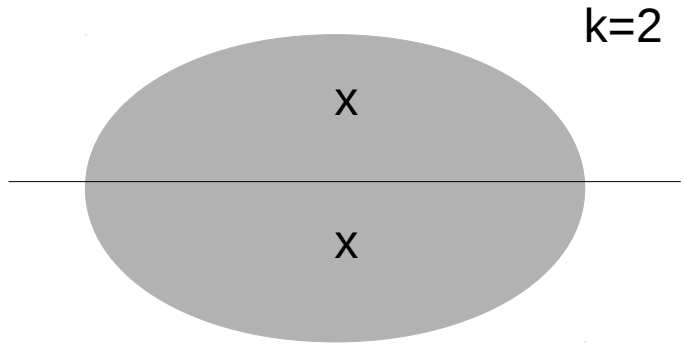
Cluster-Problem

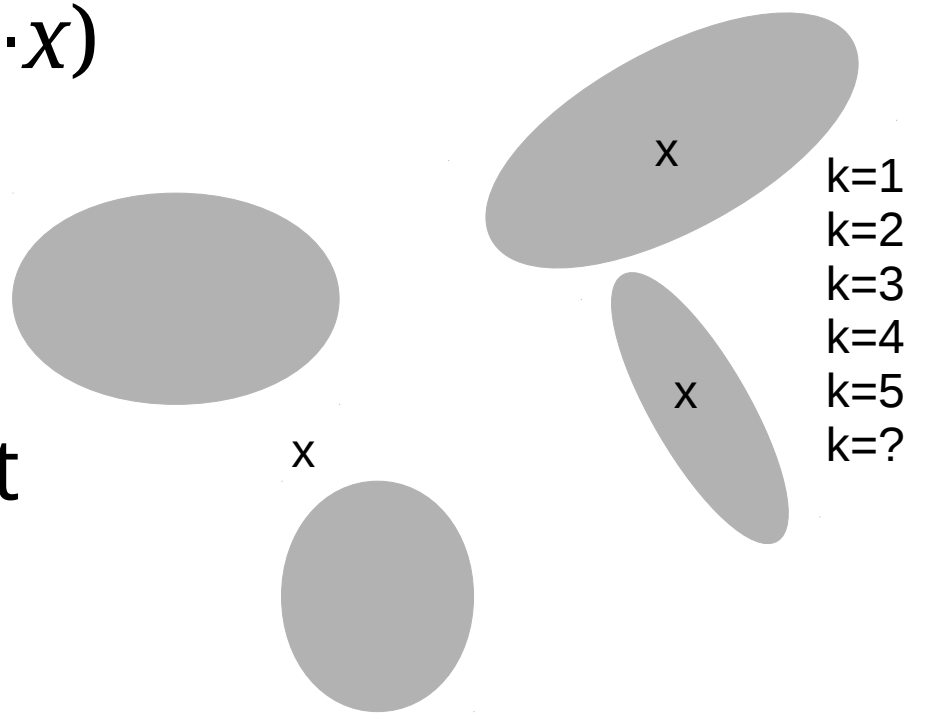
- Zu viele Cluster oder zu wenig Cluster?
- Kostenfunktion kann helfen:

  Kosten = $k+\alpha\sum_{j=1\ldots k}(1-w_j\cdot x)$

  mit $x\in$Cluster j

  α geeignet setzen
- Kosten müssen minimiert werden!





Hauptkomponentenanalyse

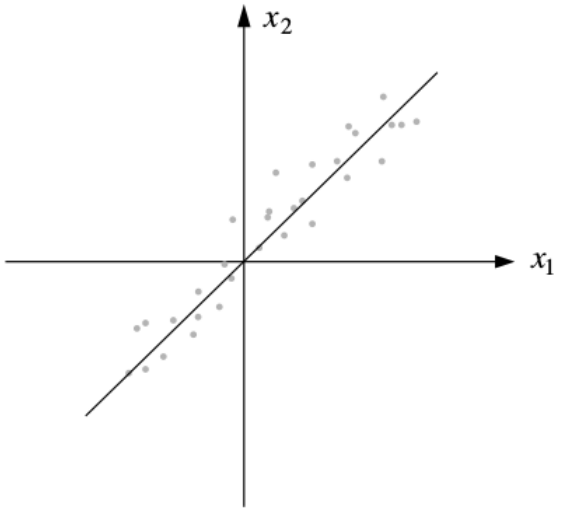
- Die Hauptkomponentenanalyse (Principal Component Analysis, PCA) findet die Richtung in den Daten, an denen die Varianz am größten ist, d.h. die Richtung, an der der Informationsgehalt am größten ist.
- Dies kann zur Dimensionsreduktion der Daten genutzt werden, sodass möglichst wenig Information verloren geht und trotzdem der Datensatz kleiner ist.
- Das hilft evtl. auch bei späterem Lernen.

Hauptkomponentenanalyse

- **Definition:** Gegeben ist eine Menge $X = \{x_1, x_2, \ldots, x_m\}$ von n-Dim Vektoren ist gegeben. Die erste Hauptkomponente von X ist ein Vektor w, der

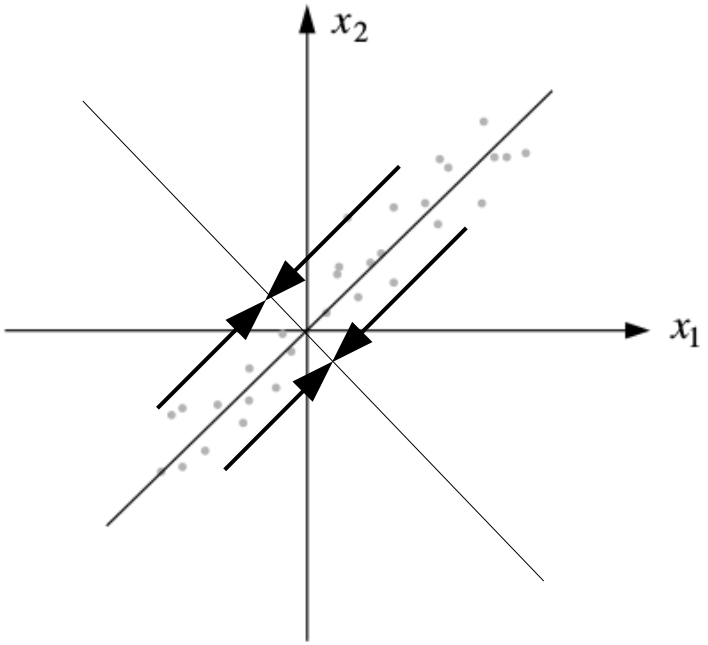
$$\frac{1}{m}\sum_{i=1}^{m} \|\mathbf{w} \cdot \mathbf{x}_i\|^2,$$

maximiert.



Hauptkomponenten

- Die 2. Hauptkomponente wird berechnet, indem von jedem Vektor x_i dessen Projektion auf die 1. Hauptkomponente abgezogen wird.



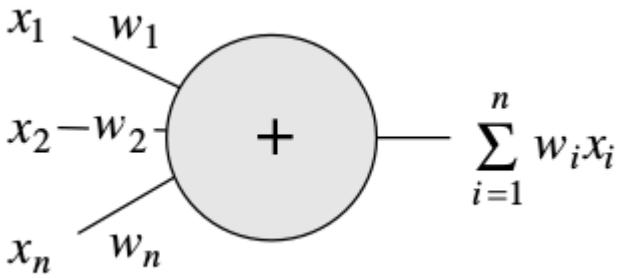
- Die 2. Hauptkomponente ist orthogonal zur ersten.
- Die 3. Hauptkomponente usw. wird rekursiv genauso berechnet.

Klassische PCA

- Das übliche Vorgehen zur Berechnung der Hauptkomponenten ist es die Eigenvektoren der Kovarianzmatrix der Daten zu berechnen.
- Die erste Hauptkomponente ist Richtung des längsten Eigenvektors.
- Diese Methode ist zuverlässig und nicht besonders schwierig, aber mit wachsender Anzahl von Vektoren steigt auch der Aufwand.
- **Nachteil:** Wenn sich die Eingabevektoren ändern muss alles neu berechnet werden.

Hauptkomponentenanalyse

- Die zugehörige Berechnungseinheit ist ein linearer Assoziator. Ein Perzptron ohne Schwellwertberechnung.



- Es wird die Projektion des Vektors auf die jeweilige Hauptkomponente berechnet.

Oja Algorithmus

Eingabe: Eingabevektoren $X=\{x_1,...,x_l\}$, um den Ursprung zentriert, eines n-dim VR, deren 1. Hauptkomponente berechnet werden soll.

Start:

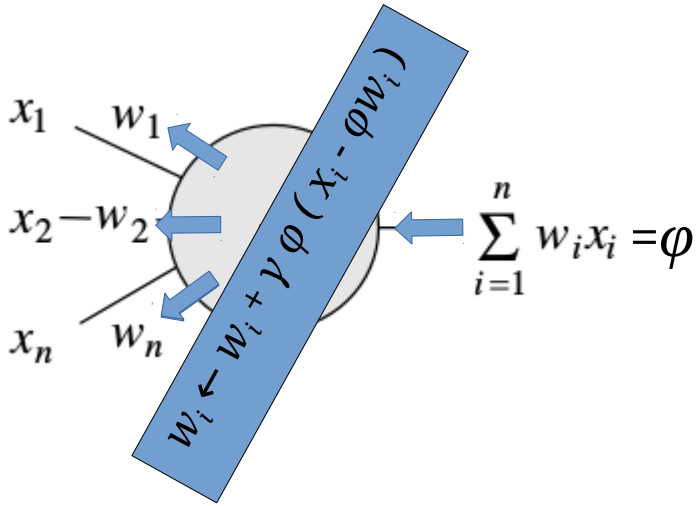
- Der Gewichtsvektoren w wird zufällig initialisiert ($w \neq 0$).
- Die Lernrate γ wird mit $0 < \gamma \leq 1$ gewählt.

Update:

- Wähle Vektor $x \in X$ zufällig.
- Berechne das Skalarprodukt $\varphi = x \cdot w$.
- Setze $w \leftarrow w + \gamma \varphi (x - \varphi w)$.
- Verringere γ.
- Weiter mit Update.

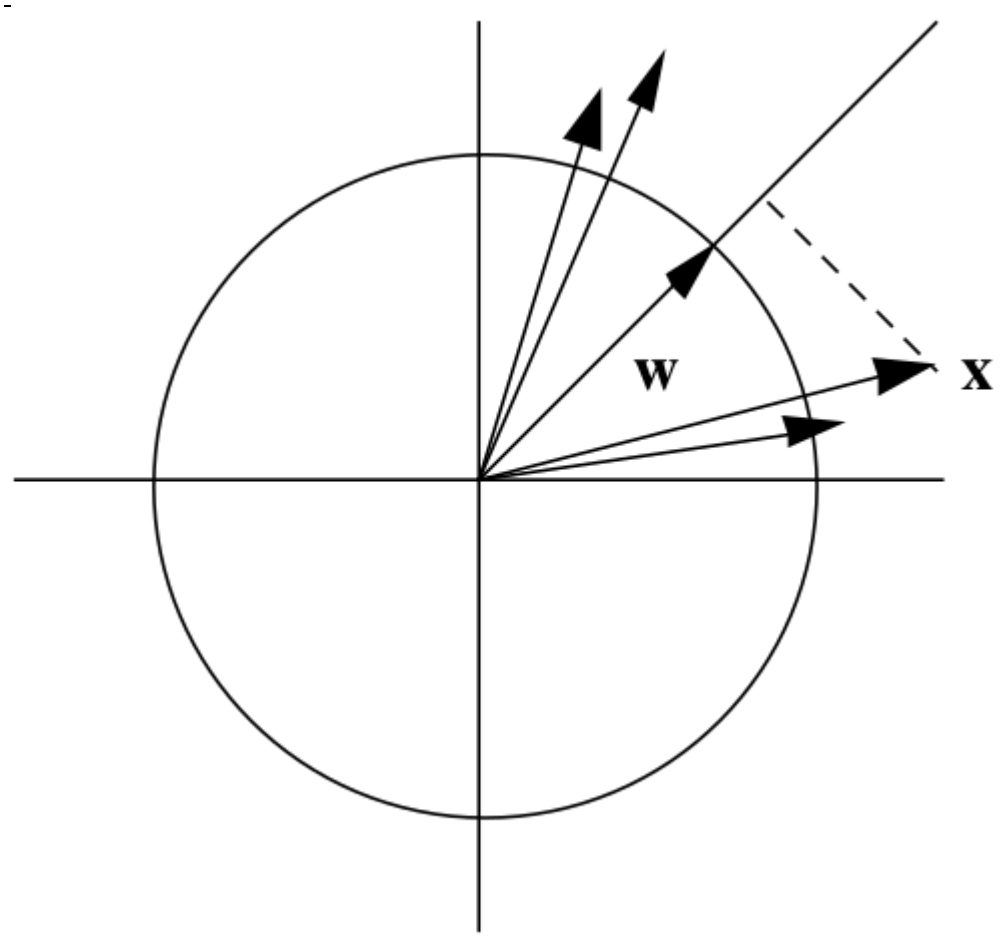
Eigenschaften vom Oja Algorithmus

- Der Gewichtsvektor wird automatisch normalisiert.
- Außer dem Skalarprodukt wird alles lokal berechnet. Das Skalarprodukt wird vom linearen Assoziator berechnet.



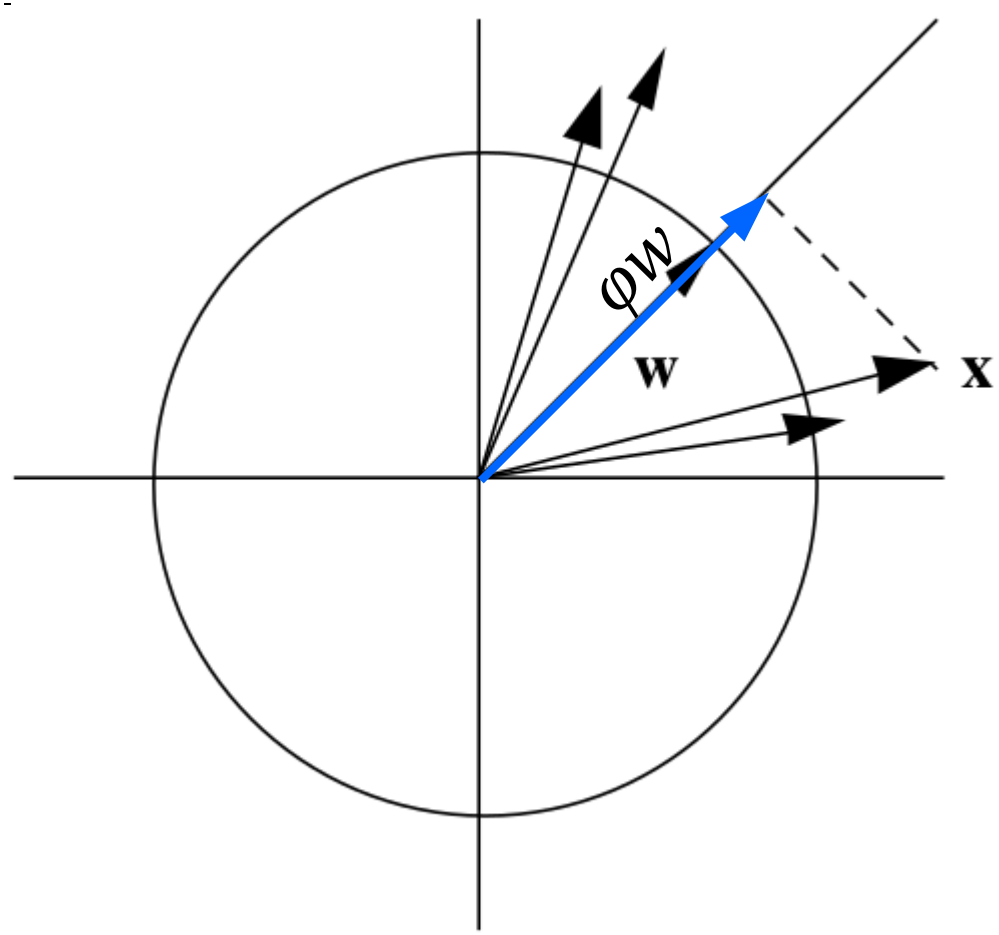
Konvergenz des Lernalgorithmus

- Wir beginnen mit $|w|=1$.
- Der Vektor x wird ausgewäh
- Das Skalarprodukt $\varphi = x \cdot w$ korrespondiert zur Projektion von x auf w.
- Der Vektor $x - \varphi w$ ist orthogonal zu w.
- Somit bewegt sich w beim Update $w \leftarrow w + \gamma\, \varphi\, (x - \varphi w)$ auf x zu (wenn γ geeignet gewählt ist).



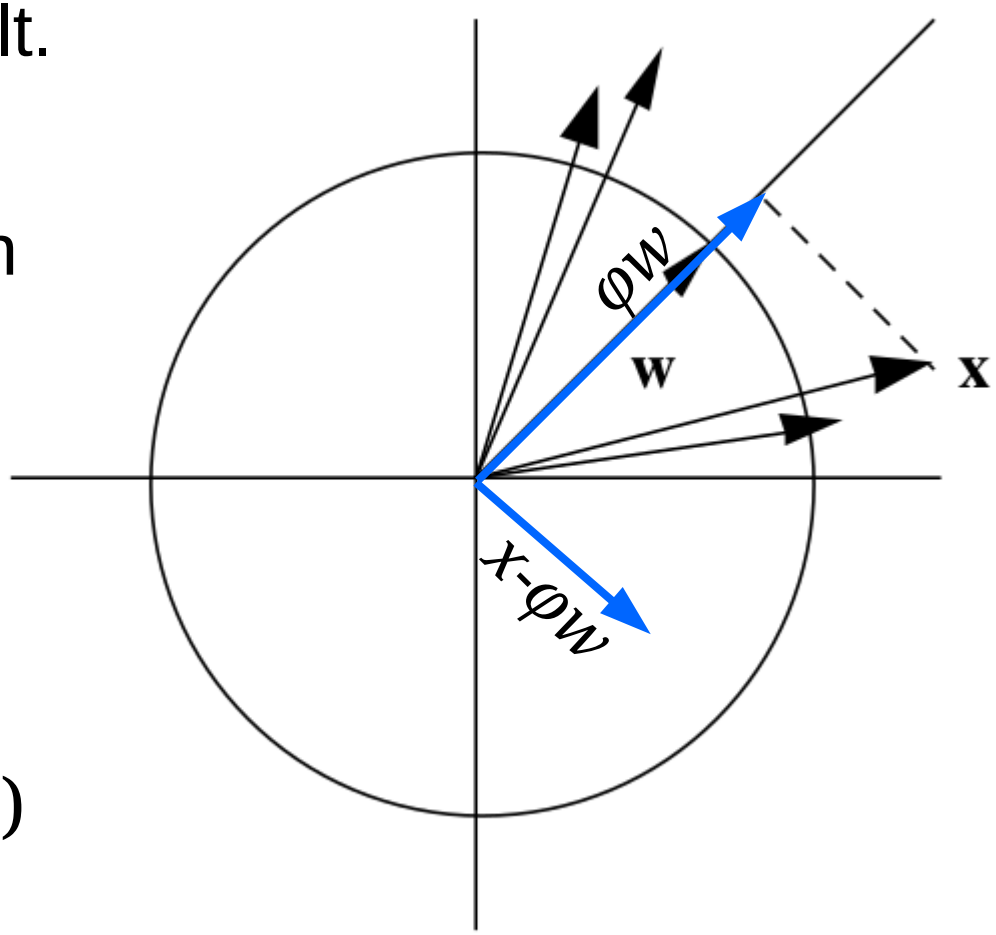
Konvergenz des Lernalgorithmus

- Wir beginnen mit $|w|=1$.
- Der Vektor x wird ausgewäh
- Das Skalarprodukt $\varphi = x \cdot w$ korrespondiert zur Projektion von x auf w.
- Der Vektor $x - \varphi w$ ist orthogonal zu w.
- Somit bewegt sich w beim Update $w \leftarrow w + \gamma\, \varphi\, (x - \varphi w)$ auf x zu (wenn γ geeignet gewählt ist).



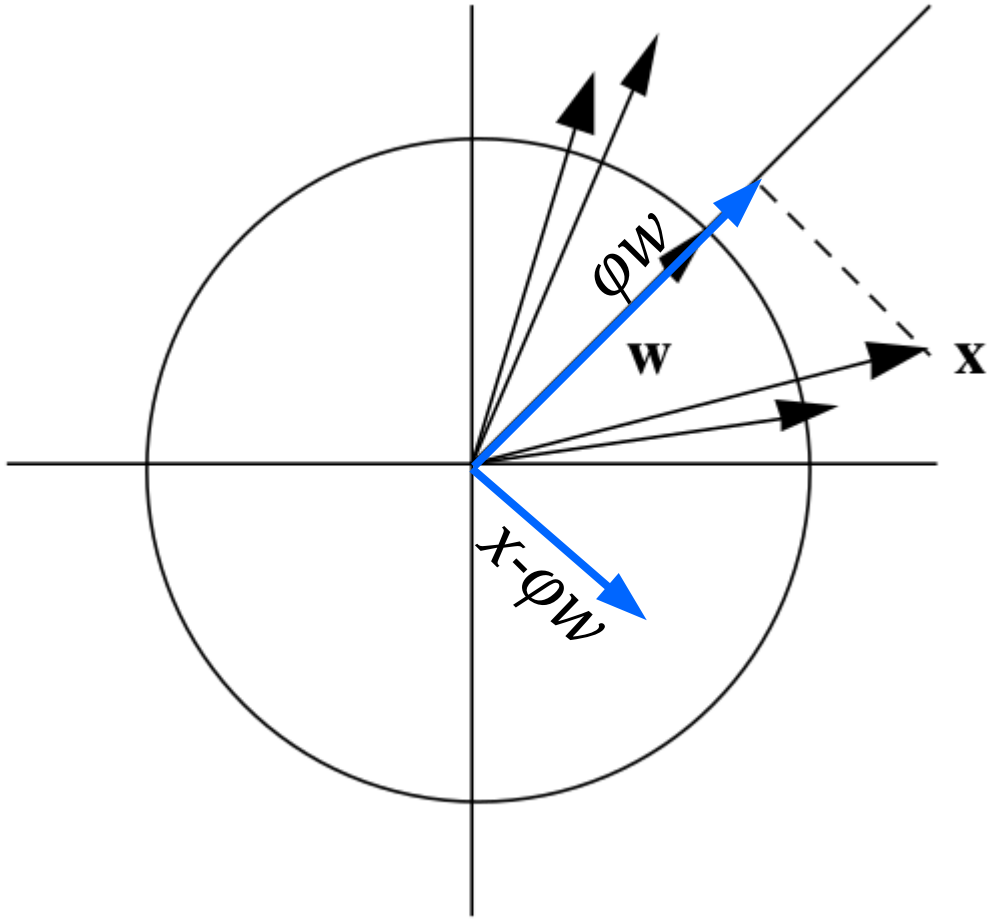
Konvergenz des Lernalgorithmus

- Wir beginnen mit $|w|=1$.
- Der Vektor x wird ausgewählt.
- Das Skalarprodukt $\varphi = x \cdot w$ korrespondiert zur Projektion von x auf w.
- Der Vektor $x-\varphi w$ ist orthogonal zu w.
- Somit bewegt sich w beim Update $w \leftarrow w + \gamma\, \varphi\, (x - \varphi w)$ auf x zu (wenn γ geeignet gewählt ist).



Konvergenz des Lernalgorithmus

- Auch wenn das Skalarprodukt negativ ist wird w in die richtige Richtung gezogen.
- Nach einer Zeit sollte w in das Zentrum aller Vektoren gezogen werden.
- Damit die Korrektur in jedem Schritt nur klein ist sollte w nicht wachsen.



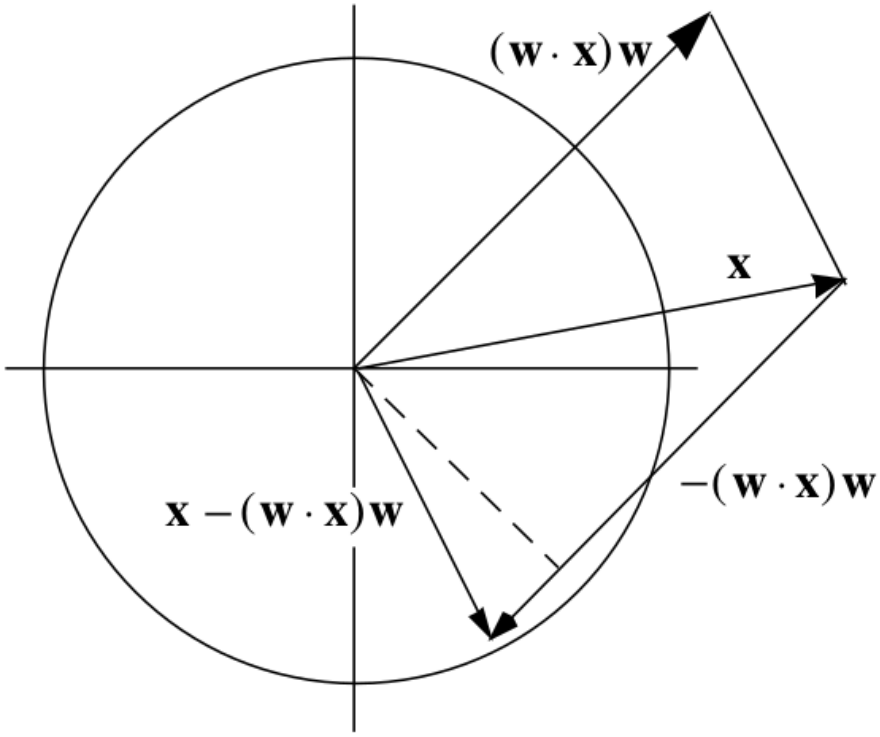
Beschränktheit von w

- Sei $|w|>1$ und $\varphi>0$.
- Dann ist φw länger als die Projektion von x auf w und x-φw hat eine negative Projektion auf w:

$$(x\text{-}\varphi w)\cdot w = (x - (x\cdot w)w)\cdot w$$

$$= x\cdot w - |w|^2\, x\cdot w < 0.$$

- Also wird $|w|$ kürzer beim Update $w \leftarrow w + \gamma\, \varphi\, (x - \varphi w)$!



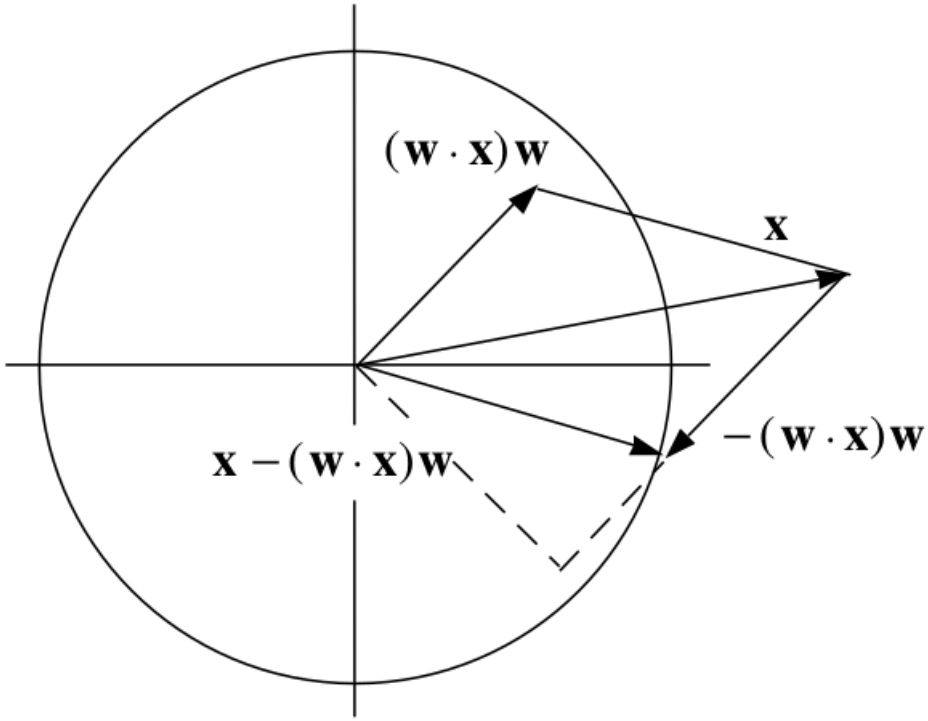
Beschränktheit von w

- Sei $|w|<1$ und $\varphi>0$.
- Dann ist φw kürzer als die Projektion von x auf w und $x-\varphi w$ hat eine positive Projektion auf w:

$$(x-\varphi w)\cdot w = (x - (x\cdot w)w)\cdot w$$

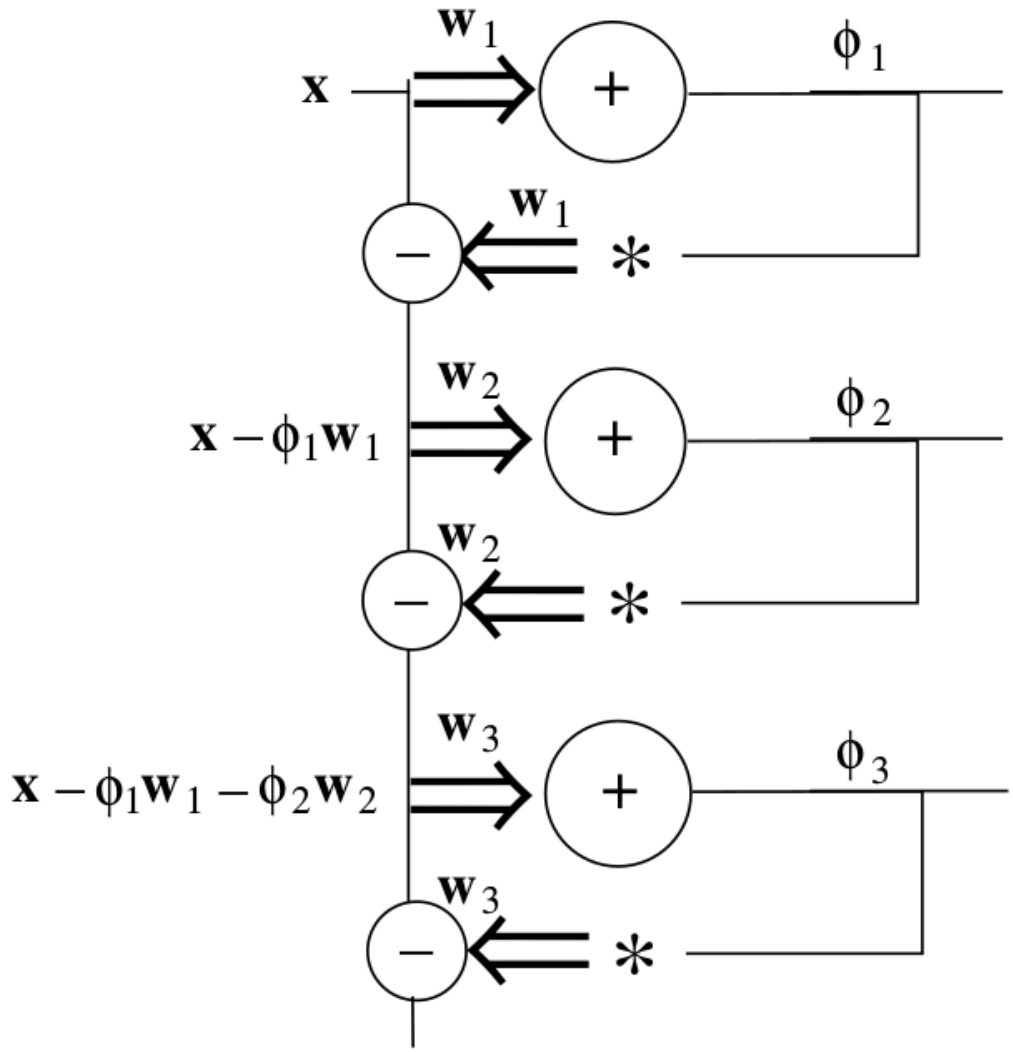
$$= x\cdot w - |w|^2 x\cdot w > 0.$$

- Also wird $|w|$ länger beim Update $w \leftarrow w + \gamma\, \varphi\, (x - \varphi w)$!



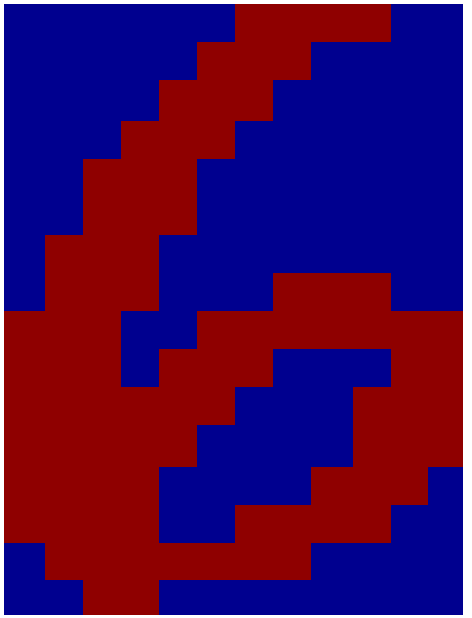
Mehrere Hauptkomponenten

- Spezielles Netz zur Berechnung der ersten 3 Hauptkomponenten.
- Auch die Berechnung der Gewichte kann gleichzeitig mit dem Oja Algorithmus erfolgen.

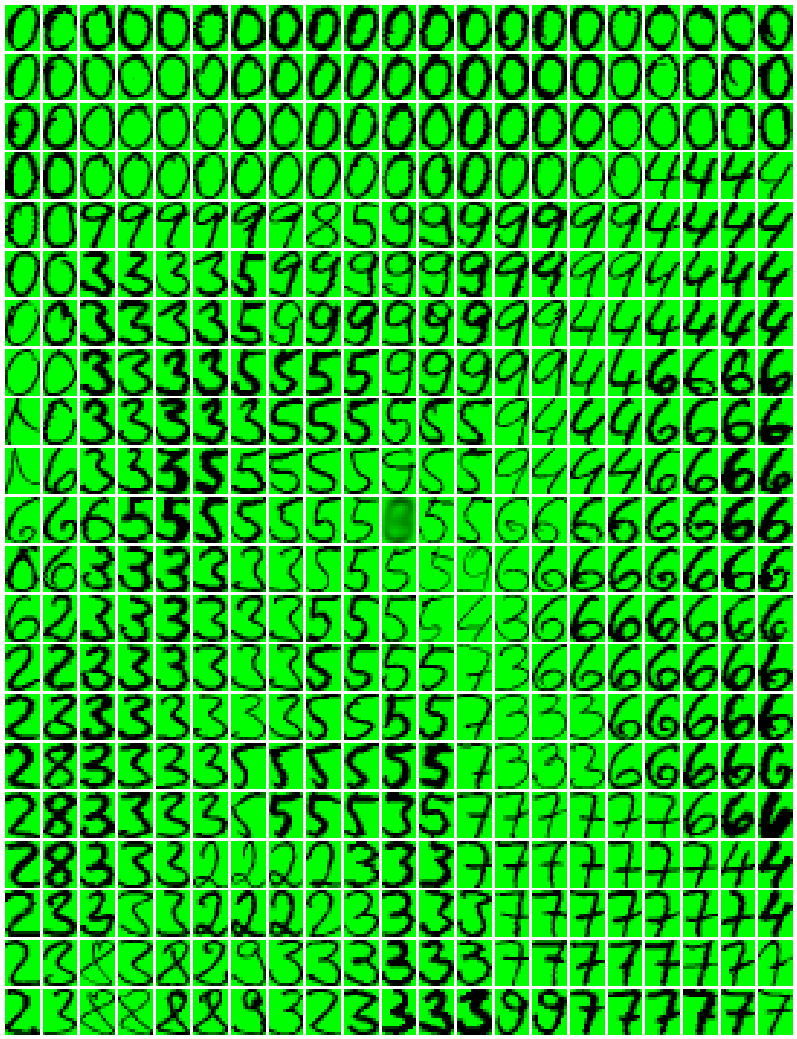


Beispiele unüberwachtes Lernen

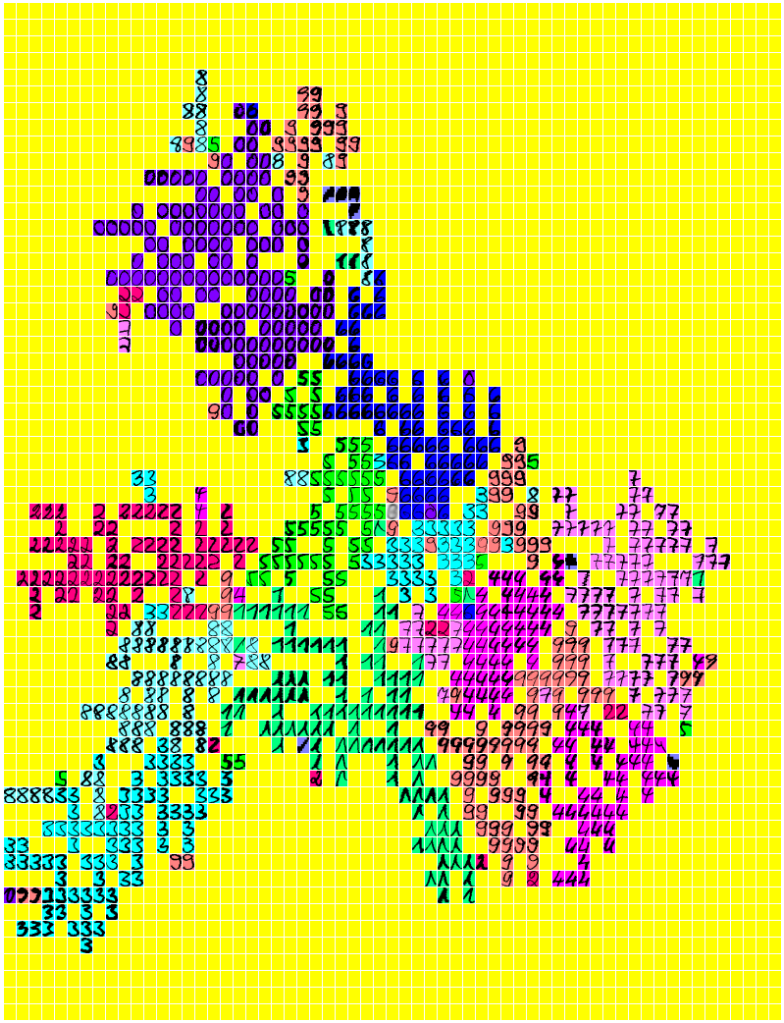
- **Mustererkennung**. Handgeschriebene Ziffern werden durch Vorverarbeitung zentriert uns skaliert. **Bipolare** Vektoren kodieren die Pixel (1=schwarz,-1=weiß). Es sind 10 Cluster zu erkennen (Ziffern 0-9).



Ähnlichkeiten der Ziffern



Bester Nachbar vom Zentrum wachsend



Beste Ziffer an besten Kristallisationspunkt

Aufgaben

1) Implementiere einen optischen Ziffernerkenner:
   - Wähle zehn Gewichtsvektoren geeignet oder zufällig.
   - Benutze den Konkurrenz-Lernalgorithmus, um die Gewichte an die Zifferndaten anzupassen (ohne die Klassifizierung zu benutzen). Die Datensätze sind auf Stud.IP.
   - Wie hoch ist der Fehler für die Trainingsmenge, bzw. welche Ziffern werden welchem Cluster zugeordnet. Dazu soll die Klassifizierung benutzt werden. Die Ergebnisse können in einer Tabelle / Zuordnungs-Matrix gespeichert werden.
   - Wie hoch ist der Fehler für die Testmenge?
   - **Zusatz:** Erhöhe die Anzahl der Cluster. Wie ändert sich die Zuordnung? Unterscheide Trainings- und Testmenge.
2) **Verständnisfrage:** Wie können tote Knoten in einem Netz konkurrierender Knoten vermieden werden. Gebe zwei oder drei unterschiedliche Vorgehensweisen an.