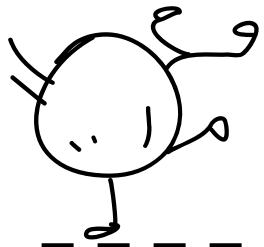


Perceptron

Vorlesung 186.844

26.11.2015

TECHNISCHE
UNIVERSITÄT
WIEN
Vienna University of Technology

Grundgedanke

… dieser Vorlesungseinheit

Bisher:

nicht-parametrische Verfahren	parametrische Verfahren
• Anzahl der Parameter nicht fix • keine Annahme über die Verteilung	• Anzahl der Parameter ist fix • man macht am Beginn eine Annahme über die Verteilung (z.B. Normalverteilung)

Jetzt: Annahme über die Entscheidungsgrenze → parametrisches Verfahren

Überblick

I. Wiederholung (WH): Diskriminantenfunktionen
II. Perceptron
III. Perceptron-Algorithmus
IV. Perceptron als Neuron
V. Generalisierungsfähigkeit des Perceptron

I. Wiederholung: Diskriminantenfunktionen

WH: Spezialfall 1

$$\boldsymbol{\Sigma_i = \sigma^2 I}$$

- Merkmale sind unabhängig $\sigma_{ij} = 0$ für alle $i \neq j$
- alle Merkmale haben dieselbe Varianz $\sigma_{ii} = \sigma^2$

$$g_j(\boldsymbol{x}) = -\frac{1}{2} d_j^2(\boldsymbol{x}) + \left[-\frac{1}{2}\ln|\Sigma_j| + \ln P(\omega_j)\right]$$

$$g_j(\boldsymbol{x}) = \left(-\frac{1}{2}\right)\left(\frac{1}{\sigma^2}\right)(\boldsymbol{x}-\boldsymbol{\mu}_j)^T(\boldsymbol{x}-\boldsymbol{\mu}_j) - \frac{1}{2}\ln|\boldsymbol{\Sigma}_j| + \ln P(\omega_j)$$

$$= \frac{1}{2\sigma^2}\left(\cancel{\boldsymbol{x}^T\boldsymbol{x}} - 2\boldsymbol{\mu}_j^T\boldsymbol{x} + \boldsymbol{\mu}_j^T\boldsymbol{\mu}_j\right) - \cancel{\frac{1}{2}\ln|\boldsymbol{\Sigma}_j|} + \ln P(\omega_j)$$

… für alle Klassen gleich

WH: Spezialfall 1

… man erhält die äquivalente lineare Diskriminanten-funktion

$$g_j(\boldsymbol{x}) = \frac{1}{2\sigma^2}\left(-2\boldsymbol{\mu}_j^T\boldsymbol{x} + \boldsymbol{\mu}_j^T\boldsymbol{\mu}_j\right) + \ln P(\omega_j)$$

$$\Downarrow$$

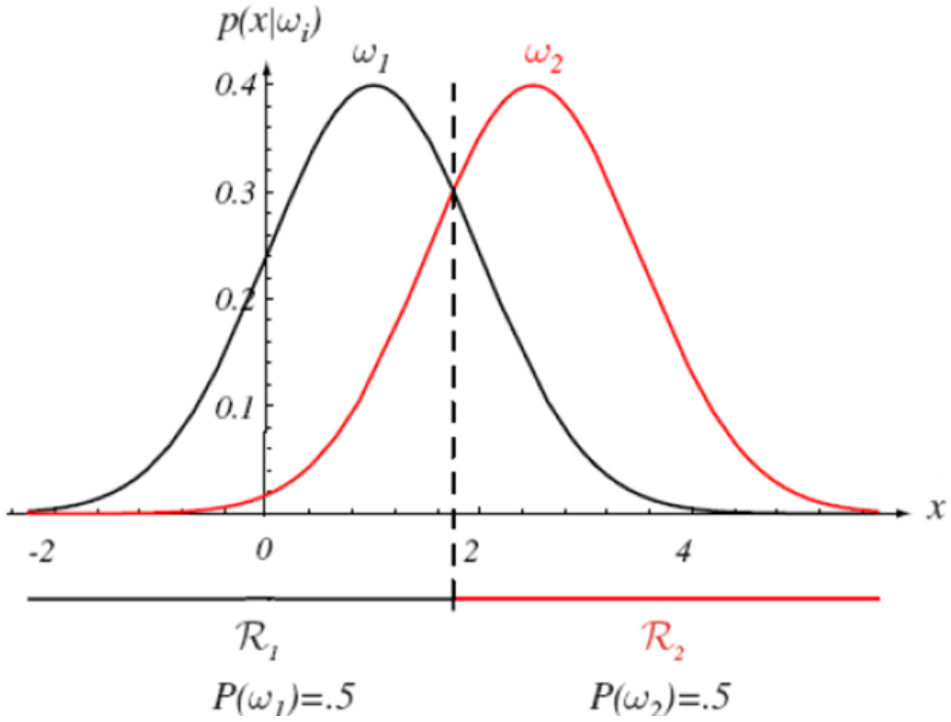
$$g_j(\boldsymbol{x}) = \boldsymbol{w}_j^T\boldsymbol{x} + w_{j0}$$

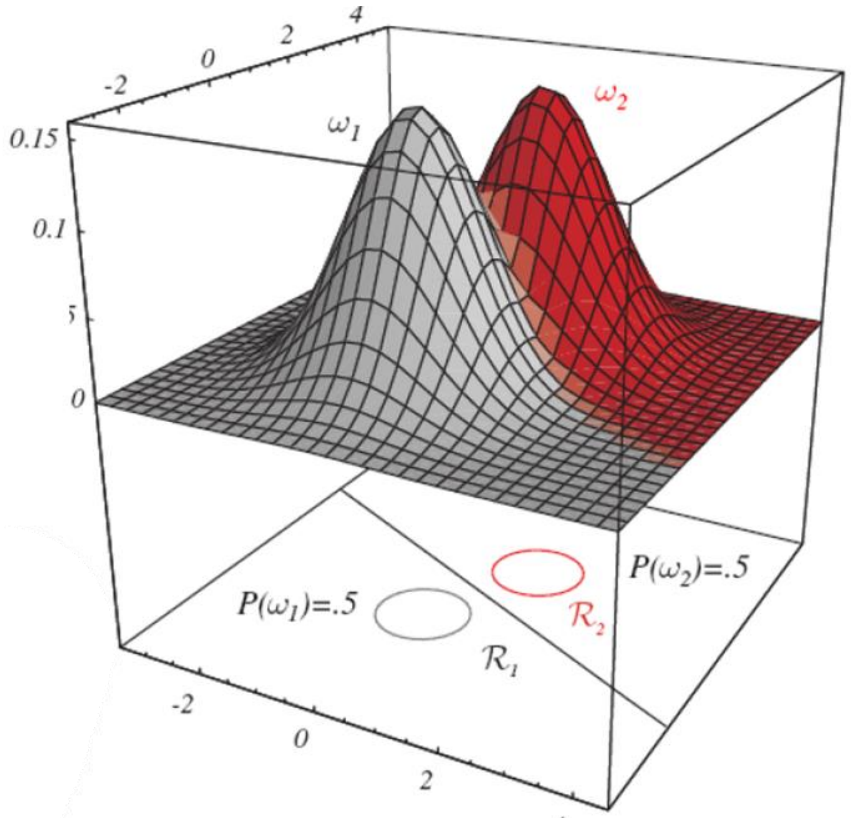
$$\boldsymbol{w}_j = \frac{1}{\sigma^2}\boldsymbol{\mu}_j$$

$$w_{j0} = -\frac{1}{2\sigma^2}\boldsymbol{\mu}_j^T\boldsymbol{\mu}_j + \ln P(\omega_j)$$

[Quelle: Duda et al., 2001]

WH: Spezialfall 1





- univariate (links) bzw. bivariate (rechts) Normalverteilungen mit $\mathbf{\Sigma}_1 = \mathbf{\Sigma}_2 = I\sigma^2$
- Entscheidungsgrenzen sind linear und normal zur Verbindungsstrecke zwischen den beiden Klassenmitteln
- bei gleichen Priors verläuft Entscheidungsgrenze durch $(\mu_1 + \mu_2)/2$
- ansonsten wird sie von Prior der wahrscheinlicheren Klasse wegverschoben.

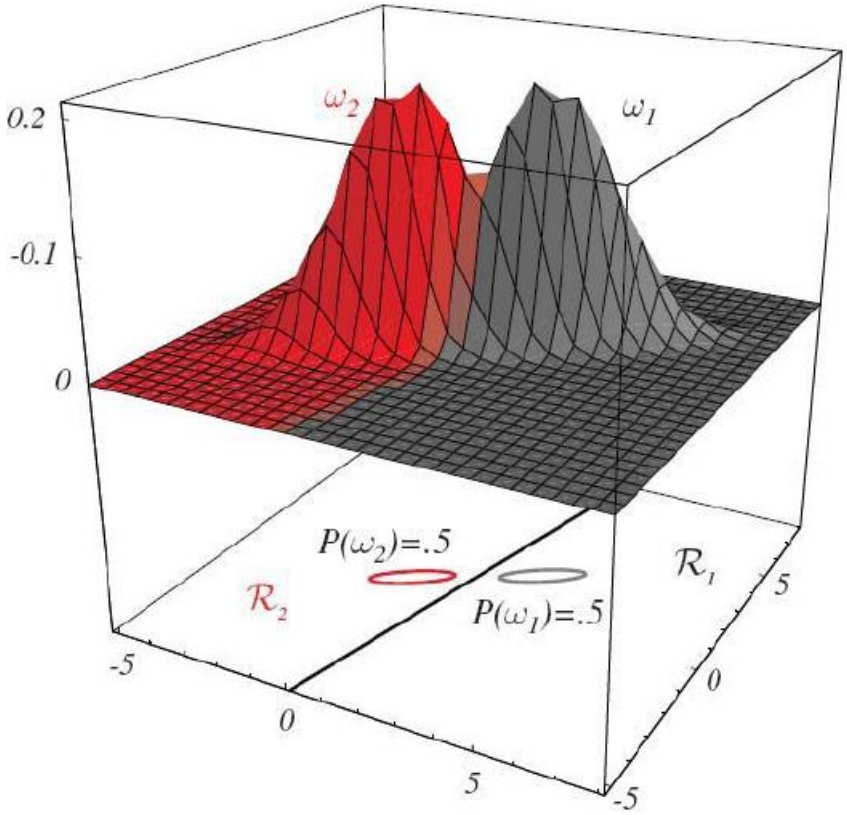
WH: Spezialfall 2

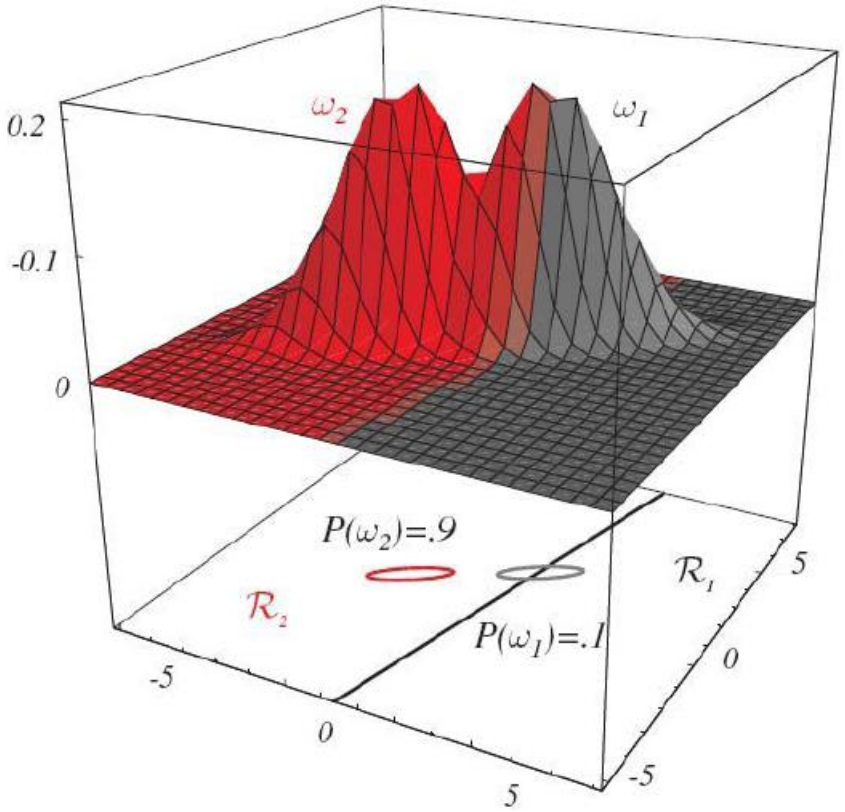
$$\boldsymbol{\Sigma_i = \Sigma}$$

- alle Klassen haben dieselbe Kovarianzmatrix
- Form der Verteilungen ist durch Hyperellipsoide in $\mathcal{R}^p$ gegeben
- Entscheidungsgrenzen sind wieder linear
- jedoch nicht normal zur Verbindungsstrecke zwischen den beiden Klassenmitteln
- Details: siehe Kapitel 2, Duda et al., 2001

[Quelle: Duda et al., 2001]

WH: Spezialfall 2





Spezialfall: 2 Klassen

Im Fall von 2 Klassen ist es üblich **nur eine Diskriminanten-funktion** zu definieren $g(\boldsymbol{x}) \equiv g_1(\boldsymbol{x}) - g_2(\boldsymbol{x})$. Diese Funktion entscheidet für Klasse w_1 wenn $g(\boldsymbol{x}) > 0$; ansonsten für Klasse w_2.

- z.B.: Minimierung der Fehlerrate („minimum-error-rate discriminant function“) beim Klassifizieren mit dem Bayes-Theorem → $g(\boldsymbol{x}) \equiv P(\omega_1|\boldsymbol{x}) - P(\omega_2|\boldsymbol{x})$

II. Perceptron

Perceptron

Das **Perceptron** ist ein Trainingsalgorithmus, um für binäre Klassifikationsprobleme (zwei Klassen) eine lineare Entscheidungsgrenze zu finden.

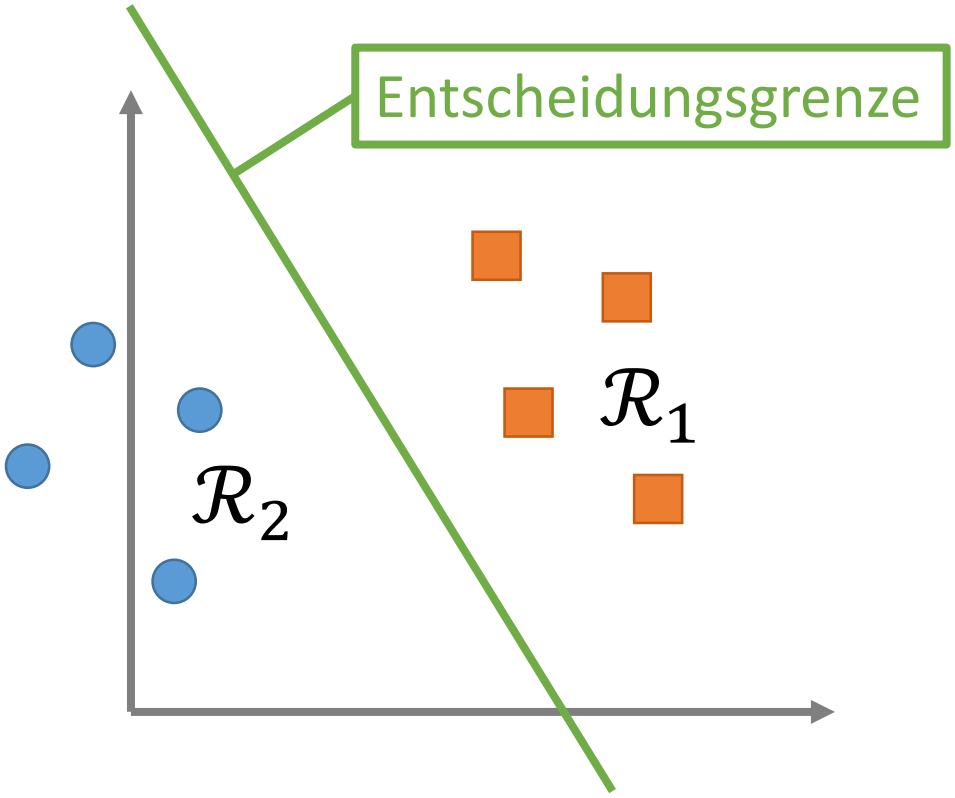
- Annahmen: d-dimensionaler Merkmalsvektor $\boldsymbol{x} \in \mathbb{R}^d$ und zwei Klassen w_1 und w_2 → binäre Klassifikationsproblem

- Ziel: Finde eine Diskriminantenfunktion $g: \mathbb{R}^d \rightarrow \mathbb{R}$ welche die Klassenzugehörigkeit wie folgt kodiert:

$$g(\boldsymbol{x}) > 0 \; falls \; \boldsymbol{x} \in \omega_1$$
$$g(\boldsymbol{x}) < 0 \; falls \; \boldsymbol{x} \in \omega_2$$

Eigenschaften von

… linearen Diskriminantenfunktionen



- können optimal sein, wenn die Datenverteilungen geeignet sind
- schnell und einfach zu trainieren
- geeignet für einen ersten „Probe-Klassifikator“

Perceptron

Der **Absolutbetrag der Diskriminantenfunktion** $|g|$ ist ein Maß für das Vertrauen in die geschätzte Klassenzugehörigkeit eines Merkmalsvektors $\boldsymbol{x}$.

- eine lineare Diskriminantenfunktion hat folgende Form

$$g(\boldsymbol{x}) = \sum_{i=1}^{d} w_i x_i - \theta = \boldsymbol{w}^T \boldsymbol{x} - \theta$$

$\boldsymbol{x} \in \mathbb{R}^d$ ... Merkmalsvektor

$\boldsymbol{w} \in \mathbb{R}^d$ ... Gewichtsvektor

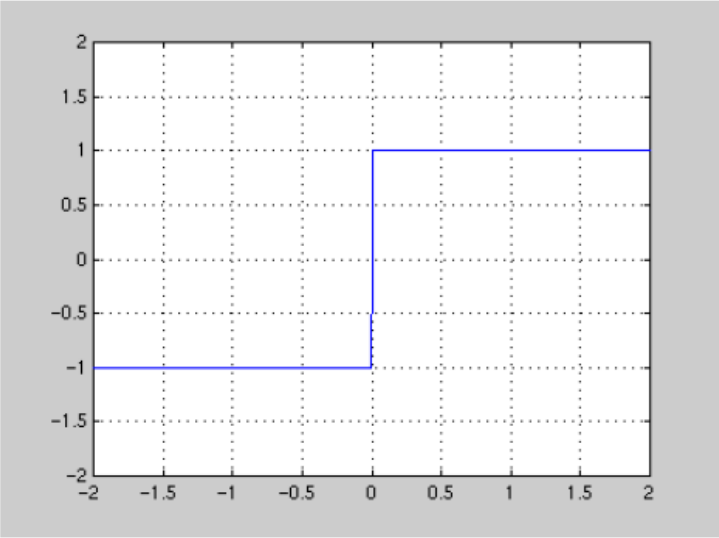
$\theta \in \mathbb{R}$ ... Bias / Schwellwert (Threshold)

Geschichte

- Perceptron-Algorithmus wurde 1957 von Frank Rosenblatt am Cornell Aeronautical Laboratorium erfunden
  - Rosenblatt, Frank (1957), The Perceptron – a perceiving and recognizing automaton. Report 85-460-1, Cornell Aeronautical Laboratory.

Die **Architektur** eines Perceptrons entspricht einer linearen Diskriminantenfunktion mit nachgeschalteter Signumfunktion.

$$sgn(x) = \begin{cases} 1 & \text{falls } x \geq 0 \\ -1 & \text{falls } x < 0 \end{cases}$$



Signumfunktion

- man kann eine lineare Diskriminantenfunktion auch als Signumfunktion anschreiben:

$$o(\boldsymbol{x}) = sgn(\boldsymbol{w}^T\boldsymbol{x} - \theta) = \begin{cases} 1 & \text{falls } \boldsymbol{w}^T\boldsymbol{x} \geq \theta \\ -1 & \text{falls } \boldsymbol{w}^T\boldsymbol{x} < \theta \end{cases}$$

$o(\boldsymbol{x})$ ... ist die Ausgabe des Perceptrons

Ziel des Perceptron-Algorithmus: Bestimme einen Gewichtsvektor $\boldsymbol{w}$ und einen Bias θ, sodass

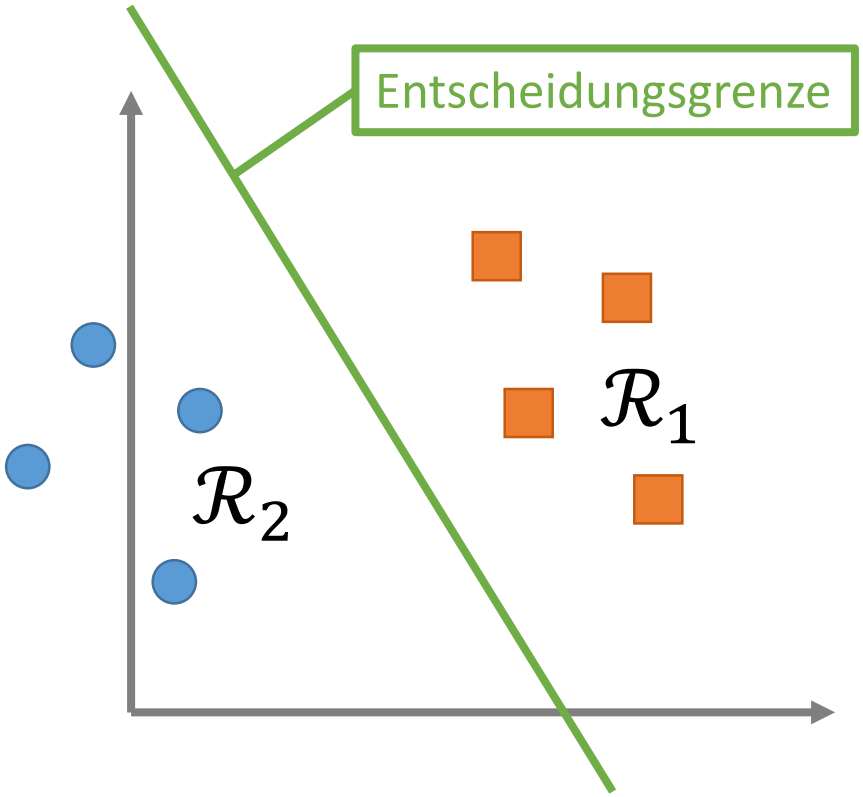
$$o(\boldsymbol{x}) = 1 \quad (\Leftrightarrow \boldsymbol{w}^T\boldsymbol{x} \geq \theta) \quad \text{falls } \boldsymbol{x} \in \omega_1$$
$$o(\boldsymbol{x}) = -1 \quad (\Leftrightarrow \boldsymbol{w}^T\boldsymbol{x} < \theta) \quad \text{falls } \boldsymbol{x} \in \omega_2$$

Geometrische Interpretation

- für $\boldsymbol{x}, \boldsymbol{w} \in \mathbb{R}^d$, legt

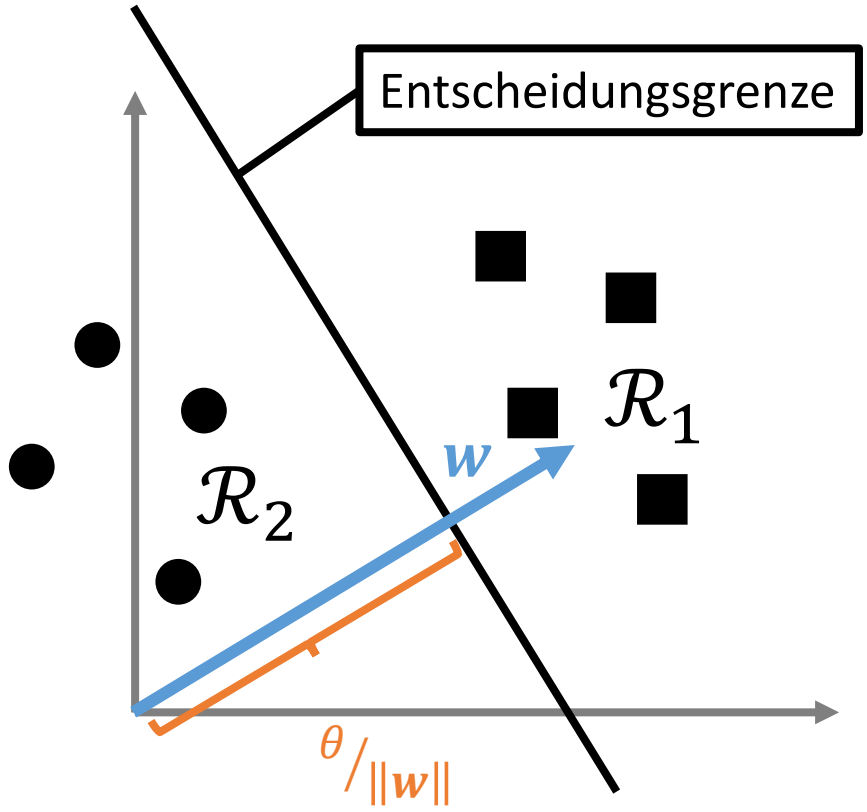
$$\sum_{i=1}^{d} w_i x_i = \boldsymbol{w}^T \boldsymbol{x} = \theta$$

- eine in $\mathbb{R}^d$ eingebettet $(d-1)$-dimensionale Hyperebene fest
  - für $\mathbb{R}^2$ eine Gerade (1D)
  - für $\mathbb{R}^3$ eine Ebene (2D)
  - …



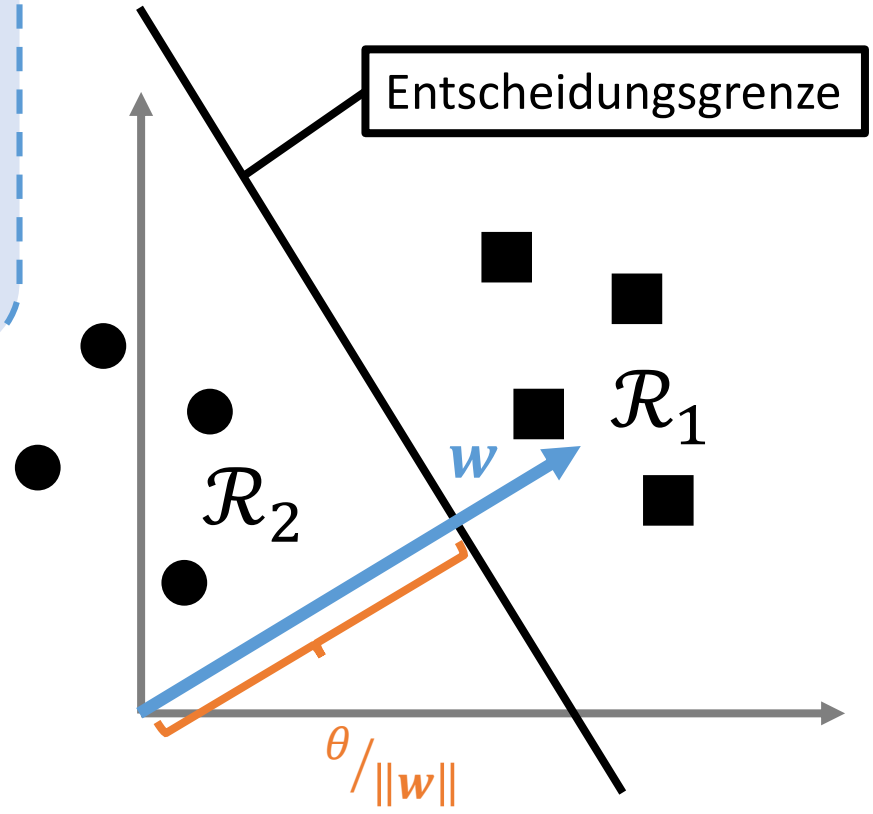
Geometrische Interpretation

- falls $\theta = 0$, dann geht die Hyperebene durch den Ursprung
- ansonsten, ist sie entlang $\boldsymbol{w}$ um den Betrag $\theta/\|\boldsymbol{w}\|$ vom Ursprung verschoben



Geometrische Interpretation

Die **Entscheidungsgrenze** (Hyper-ebene) $\boldsymbol{w}^T\boldsymbol{x} = \theta$ kann durch ihren **Normalvektor $\boldsymbol{w}$** und ihre **Distanz von Ursprung** $\theta/\|\boldsymbol{w}\|$, gemessen entlang $\boldsymbol{w}$, beschrieben werden.



im Beispiel rechts:

- falls $\|\boldsymbol{w}\| = 1$, dann ist $\theta/\|\boldsymbol{w}\| = \theta$
- für alle „quadratischen“ Daten-punkte $\in \omega_1$ gilt $\boldsymbol{w}^T\boldsymbol{x} \geq \theta$ und
- für alle „runden” Datenpunkte $\in \omega_2$ gilt $\boldsymbol{w}^T\boldsymbol{x} < \theta$

Lineare Separierbarkeit

- im Englischen „linear separability“
- $\boldsymbol{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_N) \in \mathbb{R}^{d \times N}$ … N Merkmalsvektoren der Länge d
- $\boldsymbol{y} = (y_1, y_2, \dots, y_N), y_i \in \{1, -1\}$ … N Klassenlabels

$\boldsymbol{X}$ ist genau dann **linear separierbar** (bezüglich $\boldsymbol{y}$), falls es einen Gewichtsvektor $\boldsymbol{w}$ und einen Bias θ gibt, sodass

$$o(\boldsymbol{x}_i) = sgn(\boldsymbol{w}^T \boldsymbol{x}_i - \theta) = y_i, 1 \leq i \leq N$$

Homogene Koordinaten

… für eine kompaktere Schreibweise

- Bias kann durch einen kleinen Kunstgriff in den Gewichtsvektor „hineingezogen“ werden
- Dazu erweitert man den Merkmalsvektor $\boldsymbol{x}$ und den Gewichtsvektor $\boldsymbol{w}$ um eine Dimension:
  - $x_0 = 1$ und $w_0 = -\theta$

$$\breve{\boldsymbol{x}} = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix} \qquad \breve{\boldsymbol{w}} = \begin{bmatrix} -\theta \\ w_1 \\ w_2 \\ \vdots \\ w_d \end{bmatrix}$$

Homogene Koordinaten

Dadurch erhält man:

$$g(\boldsymbol{x}) = \breve{\boldsymbol{w}}^T \breve{\boldsymbol{x}} = \sum_{i=1}^{d} w_i x_i - \theta = \boldsymbol{w}^T \boldsymbol{x} - \theta$$

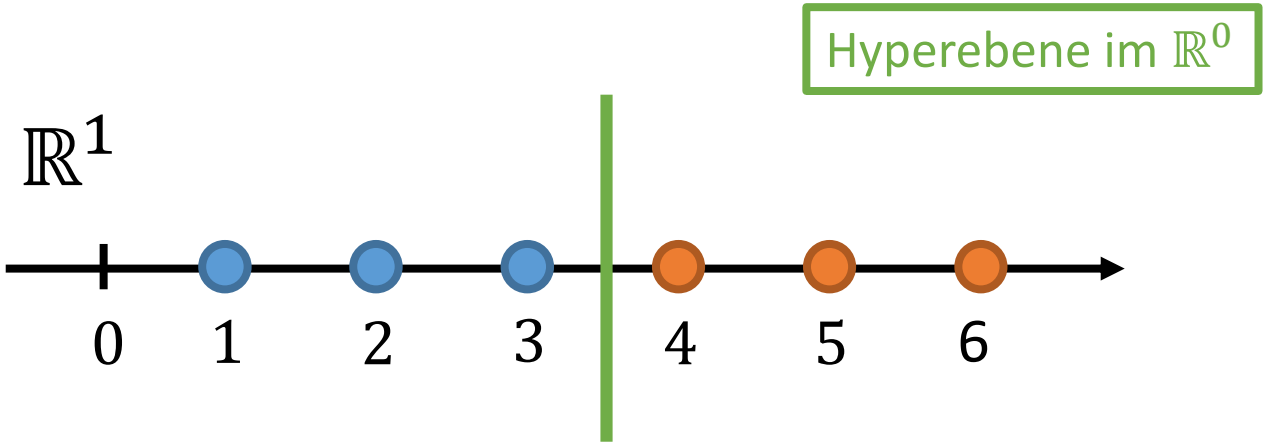
- Transformation in homogene Koordinaten vereinfach Problem indem Dimensionalität um 1 (von d auf $d+1$) erhöht wird
- obige Gleichung definiert eine d-dimensionale Hyperebene im $\mathbb{R}^{d+1}$, welche durch den Ursprung geht
- im folgenden werden stets homogene Koordinaten verwendet → daher schreibt man einfach $\boldsymbol{w}^T \boldsymbol{x}$ für $\breve{\boldsymbol{w}}^T \breve{\boldsymbol{x}}$

Beispiel

- zum Verständnis der homogenen Koordinaten

Trainingsdaten Klasse w_1: $\{1, 2, 3\}$

Trainingsdaten Klasse w_2: $\{4, 5, 6\}$



Beispiel

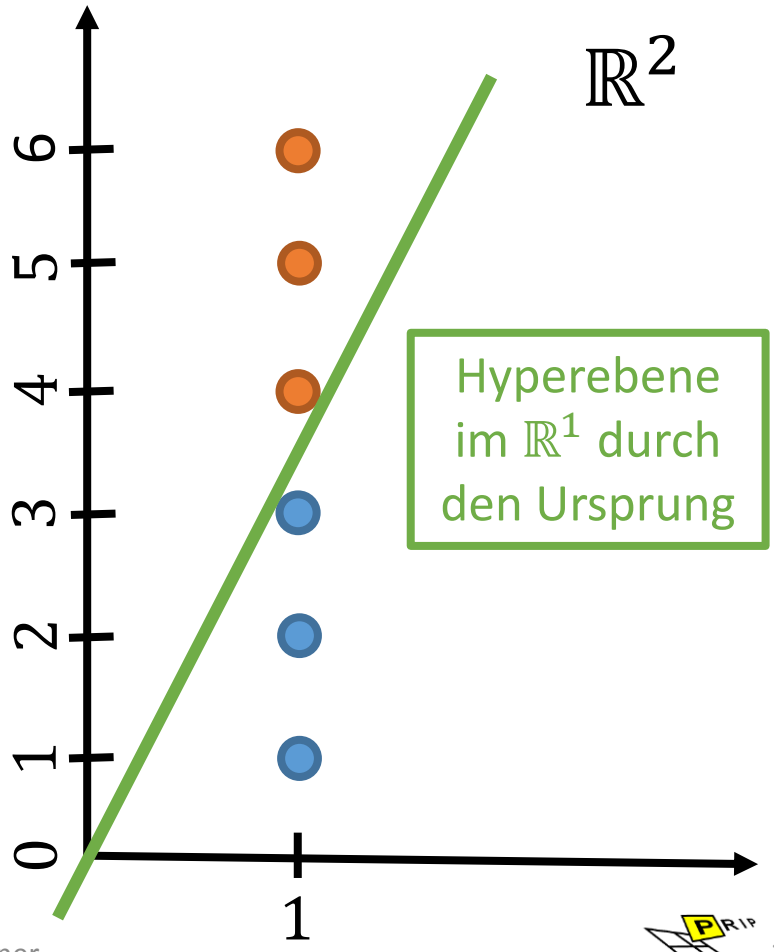
- Transformation in homogene Koordinaten

Trainingsdaten Klasse w_1:

$$\left\{\begin{pmatrix}1\\1\end{pmatrix},\begin{pmatrix}1\\2\end{pmatrix},\begin{pmatrix}1\\3\end{pmatrix}\right\}$$

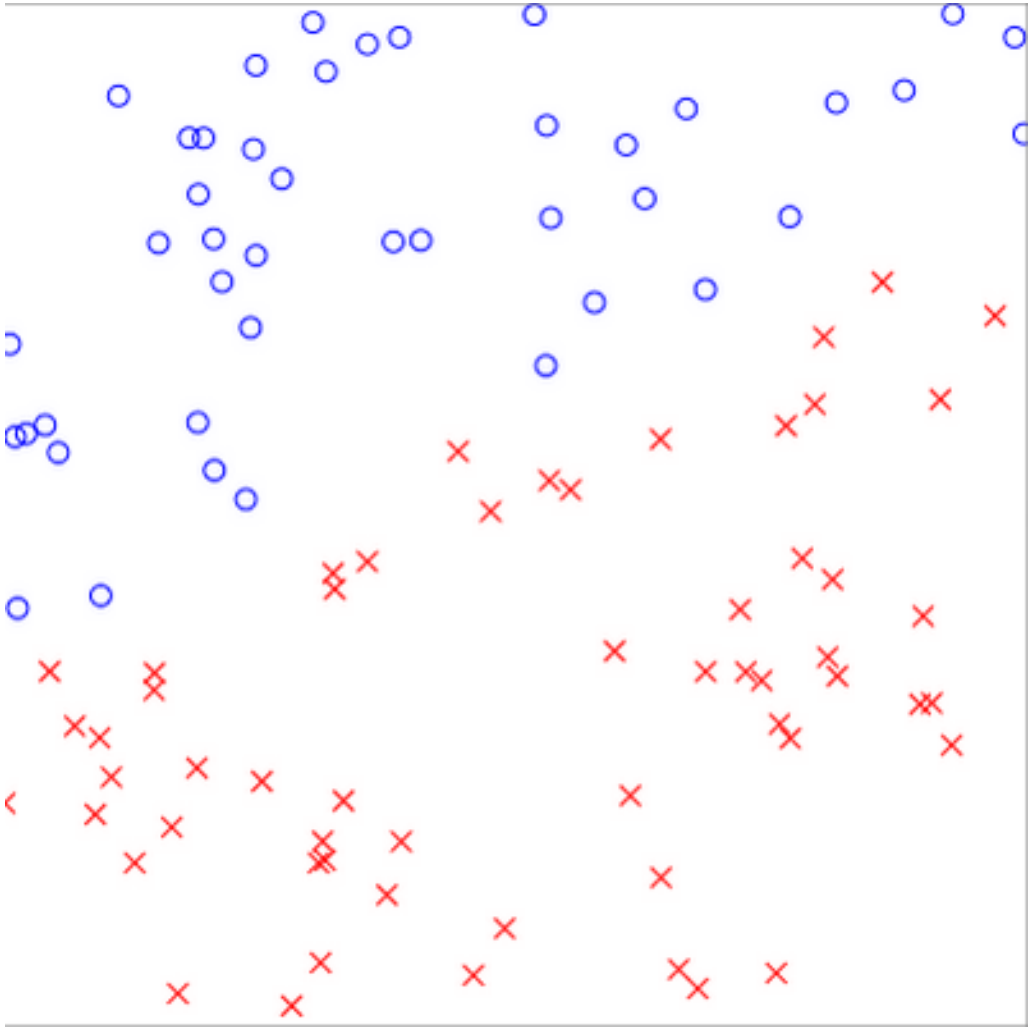
Trainingsdaten Klasse w_2:

$$\left\{\begin{pmatrix}1\\4\end{pmatrix},\begin{pmatrix}1\\5\end{pmatrix},\begin{pmatrix}1\\6\end{pmatrix}\right\}$$



III. Perceptron-Algorithmus

DEMO



Trainingsdatensatz

Trainingsdatensatz:

$\boldsymbol{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N) \in \mathbb{R}^{(d+1)\times N}$ … N homogene Merkmalsvektoren

$\boldsymbol{y} = (y_1, y_2, \ldots, y_N), y_i \in \{1, -1\}$ … N Klassenlabels

Beispiel:

Wenn man das binäre AND-Problem mit Hilfe eines Perceptrons lösen wollte, hätte der Trainingsdatensatz folgende Form:

$$\boldsymbol{X} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix} \quad \boldsymbol{y} = (-1, -1, -1, 1)$$

Grundidee

Ziel: Finde einen Gewichtsvektor $\boldsymbol{w}$, sodass

$$o(\boldsymbol{x}_i) = sgn(\boldsymbol{w}^T \boldsymbol{x}_i) = y_i, 1 \leq i \leq N$$

Idee:

- falls ein „positiver“ Trainingsvektor $\boldsymbol{x}_i$ (mit $y_i = 1$) falsch klassifiziert wird, d.h.: $\boldsymbol{w}^T \boldsymbol{x}_j < 0$
- dann addiere ein Vielfaches von $\boldsymbol{x}_i$ zu $\boldsymbol{w}$
- dadurch wird die Hyperebene auf den falsch klassifizierten Vektor hinbewegt.

Lernrate

$$(\boldsymbol{w} + \gamma \boldsymbol{x}_i)^T \boldsymbol{x}_i > \boldsymbol{w}^T \boldsymbol{x}_i$$

- $\gamma > 0$ (positiver Faktor) → wird auch Lernrate genannt
- wird verwendet um „Vielfaches“ von $\boldsymbol{x}_i$ zu $\boldsymbol{w}$ zu addieren

Analog zum positiven Fall (siehe oben):

- im Fall eines falsch klassifizierten „negativen“ Trainingsvektors $\boldsymbol{x}_j$ (mit $y_j = -1$), d.h.: $\boldsymbol{w}^T \boldsymbol{x}_j > 0$
- wird ein Vielfaches von $\boldsymbol{x}_j$ von $\boldsymbol{w}$ subtrahiert
- dadurch wird die Hyperebene vom falsch klassifizierten Vektor wegbewegt

Iterativer Prozess

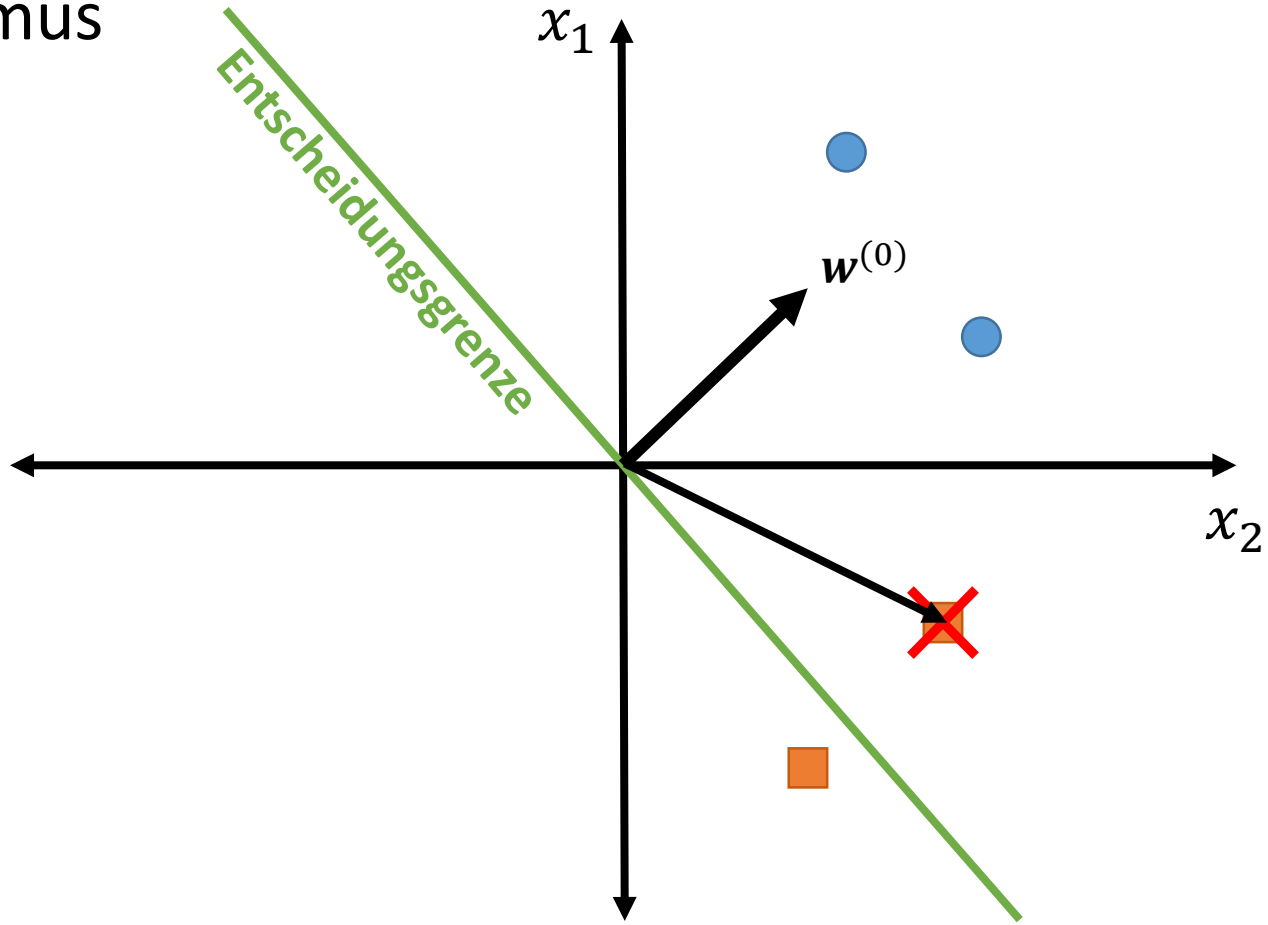
Der Perceptron-Algorithmus ist ein **iterativer Prozess** der die **Parameter** der linearen Diskriminantenfunktion, $\boldsymbol{w}$ und θ, aus einem Trainingsdatensatz bestimmt.

Während des iterativen Trainingsprozesses ist es möglich, dass zuvor richtig klassifizierte Vektoren durch die aktuelle Hyperebene falsch klassifiziert werden → siehe Beispiel auf den nächsten Folien

D.h. dass nicht jede Iteration zu einer Verbesserung führt.

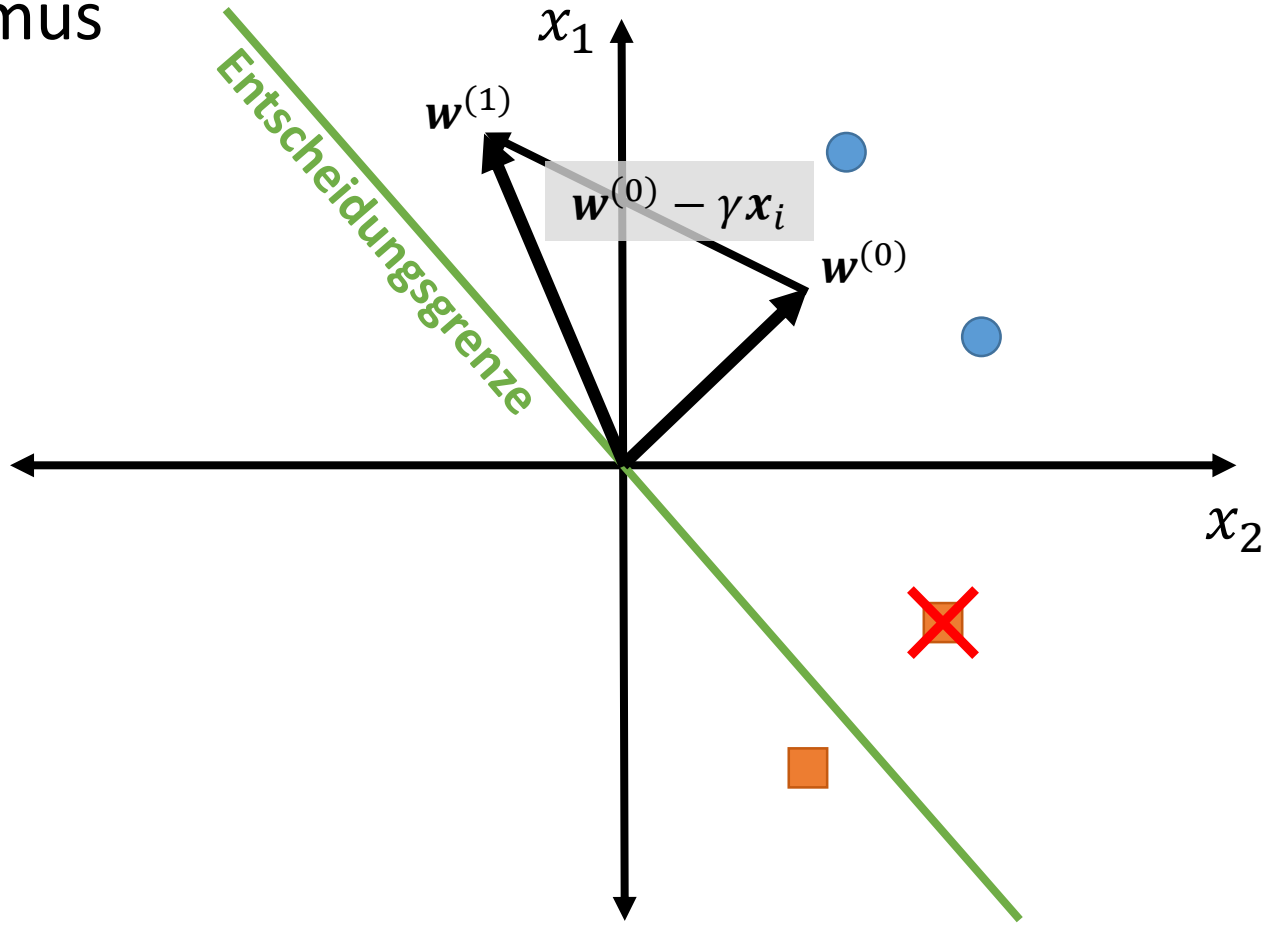
Beispiel

.. des iterativen Trainingsprozesses im Perceptron-Algorithmus



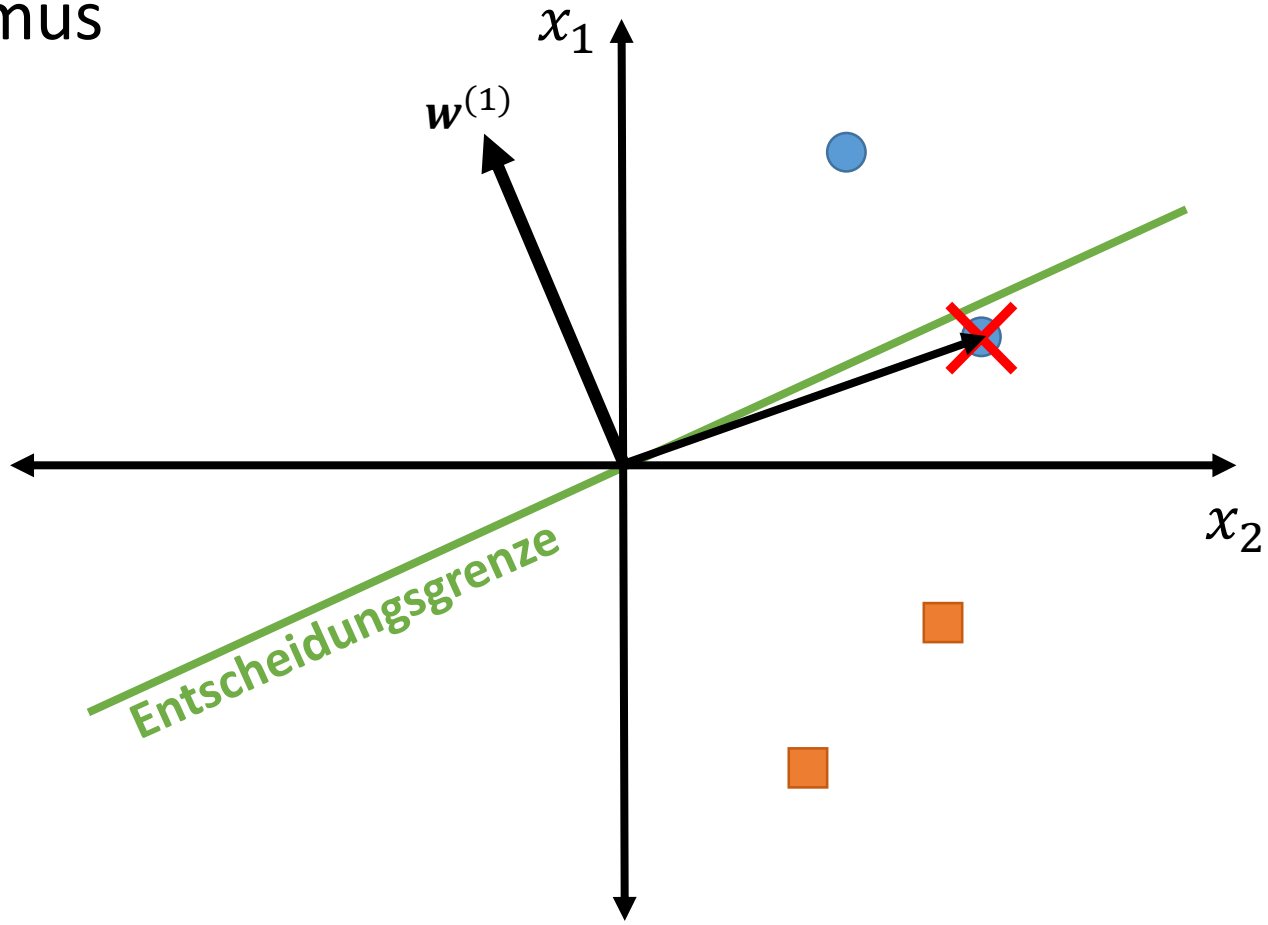
Beispiel

.. des iterativen Trainingsprozesses im Perceptron-Algorithmus



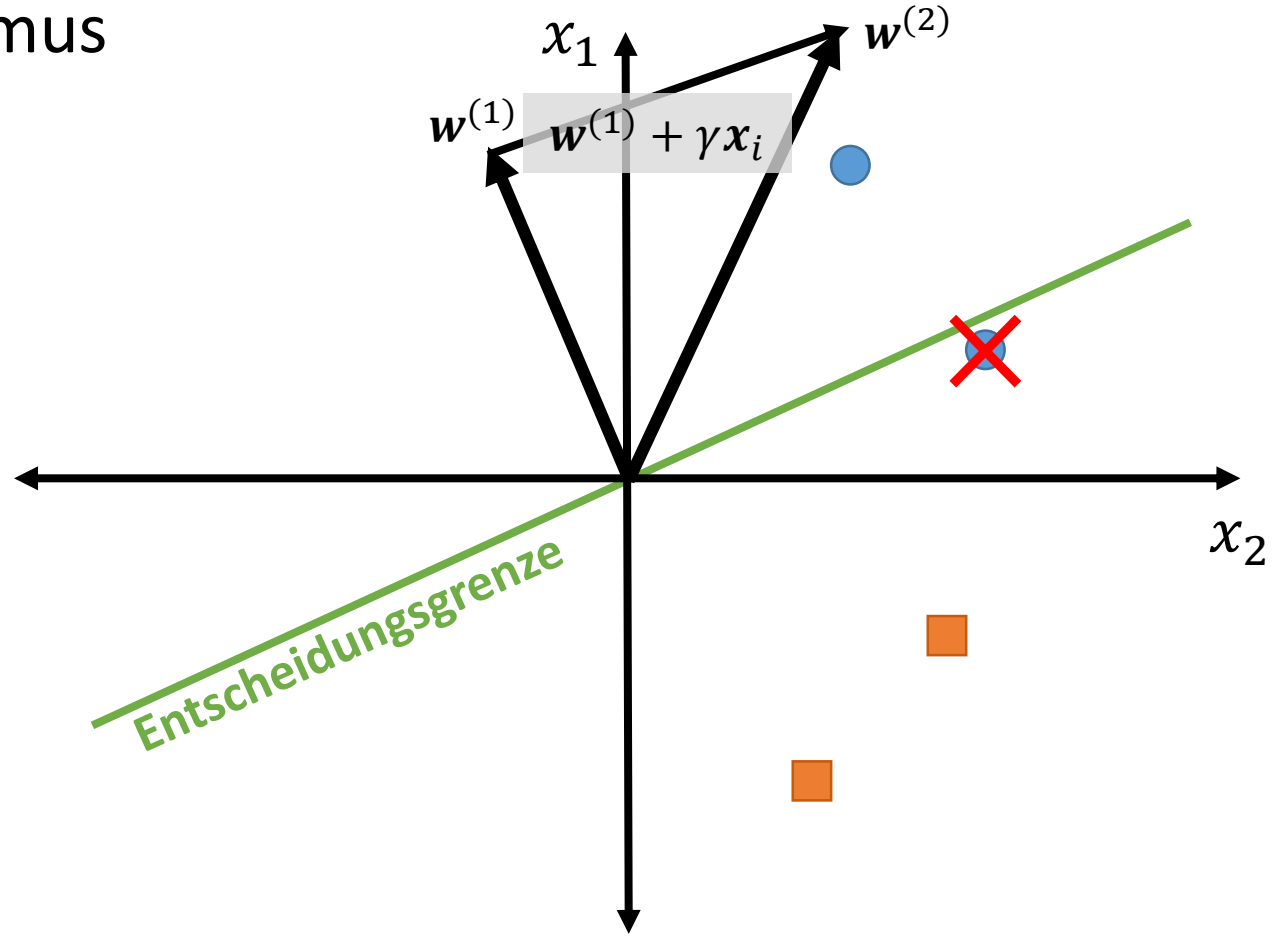
Beispiel

.. des iterativen Trainingsprozesses im Perceptron-Algorithmus



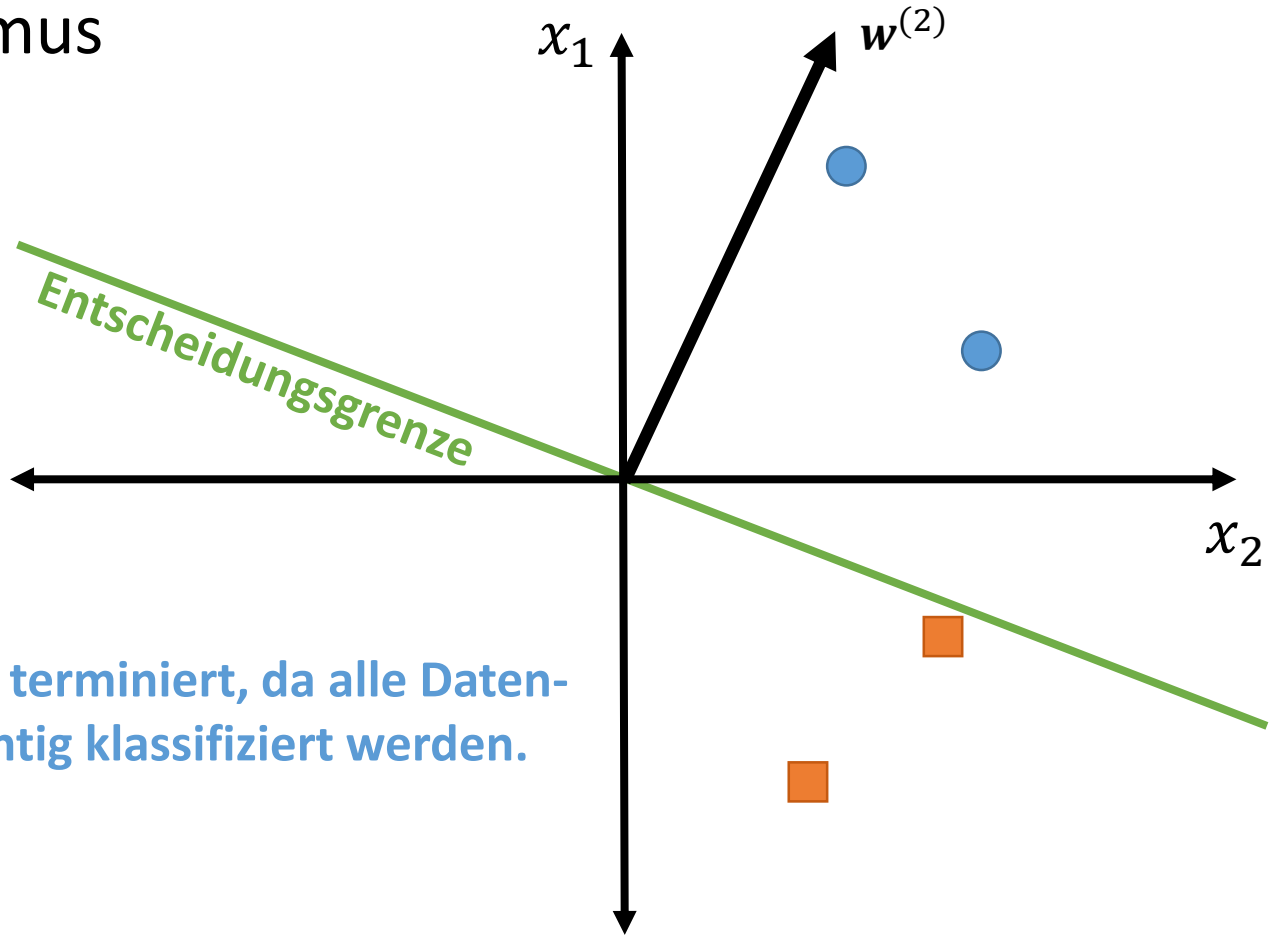
Beispiel

.. des iterativen Trainingsprozesses im Perceptron-Algorithmus



Beispiel

.. des iterativen Trainingsprozesses im Perceptron-Algorithmus



Algorithmus terminiert, da alle Daten-punkte richtig klassifiziert werden.

Vereinfachung

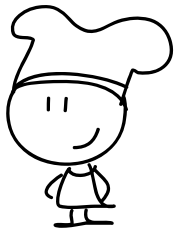
Falls man unabhängig von der Klasse („positive“ oder „negative“, 1 oder -1) immer die gleiche Operation (Überprüfung und Update) durchführen will, dann kann man folgendes tun:

$$sgn(\boldsymbol{w}^T\boldsymbol{x}_i) = y_i \Leftrightarrow sgn(\boldsymbol{w}^T\boldsymbol{x}_i)y_i = 1$$
$$\Rightarrow (\boldsymbol{w}^T\boldsymbol{x}_i)y_i > 0 \Leftrightarrow \boldsymbol{w}^T(\boldsymbol{x}_i y_i) > 0$$

D.h.: Man sucht nun nach einem Gewichtsvektor $\boldsymbol{w}$ der den modifizierten Trainingsdatensatz $\boldsymbol{x}_i y_i, 1 \leq i \leq N$ in die positive Halbebene abbildet.

Perceptron-Algorithmus

1. **Initialisiere** $\boldsymbol{w}$ und γ
2. **do**
3. **for** $i = 1$ to N
4. **if** $\boldsymbol{w}^T(\boldsymbol{x}_i y_i) \leq 0$ (falsche Klassifikation des Vektors $\boldsymbol{x}_i$)
5. $\boldsymbol{w} \leftarrow \boldsymbol{w} + \gamma \boldsymbol{x}_i y_i$ (Update Gewichtsvektor)
6. **end if**
7. **end for**
8. **until** → alle Merkmalsvektoren $\boldsymbol{x}_i$ richtig klassifiziert werden



3. – 7. Schritt: **Epoche**

5. Schritt: **Gewichtsupdate**

Initialisierung: Gewichtsvektor

- häufige Initialisierung: $\boldsymbol{w} = 0$
- dadurch erhält man mit Perceptron-Algorithmus folgenden Gewichtsvektor:

$$\boldsymbol{w}_p = \sum_{i=1}^{N} \boldsymbol{x}_i \, (y_i \gamma k_i) = \gamma \sum_{i=1}^{N} \boldsymbol{x}_i \, (y_i k_i), k_i \in \mathbb{N}_0$$

$\boldsymbol{w}_p$ … linear Kombination aller falsch klassifizierten Merkmalsvektoren $\boldsymbol{x}_i$

k_i … gibt an wie oft $\boldsymbol{x}_i$ falsch klassifiziert wurde

Initialisierung: Lernrate

$$\boldsymbol{w}^T\boldsymbol{x} = \theta \Leftrightarrow (\alpha\boldsymbol{w})^T\boldsymbol{x} = \alpha\theta \left(\forall x \in \mathbb{R}^d\right)$$
$$\boldsymbol{w}^T\boldsymbol{x} \geq \theta \Leftrightarrow (\alpha\boldsymbol{w})^T\boldsymbol{x} \geq \alpha\theta \left(\forall x \in \mathbb{R}^d\right)$$

Obige Gleichungen bedeuten: falls man $\boldsymbol{w}$ und θ mit dem gleichen positiven Faktor $\alpha \in \mathbb{R}^+$ multipliziert, bleiben die Entscheidungs-regionen unverändert.

Daraus folgt: γ ist nur ein Skalierungsfaktor → hat keinen Einfluss auf die Entscheidungsgrenze.

Einfachheitshalber setzt man $\gamma = 1$ für den Perceptron-Algorithmus

$\gamma = 1$ ist nicht für jedes Trainingsverfahren geeignet!

Konvergenztheorem

Das Konvergenztheorem besagt, dass der Perceptron-Algorithmus mit fixer Lernrate γ für jeden linear separierbaren Traings-datensatz mit der Lösung $\boldsymbol{w}_p$ terminiert.

Details für Interessierte:

Pattern Recognition, Sergios Theodoridis, Seite 95 – 97

Pattern Classification, Richard Duda, Seite 229 – 232

MITOpenCourseWare: Machine Learning von Prof. Tommi Jaakkola

Der Perceptron-Algorithmus konvergiert nicht im Falle eines nicht linear separierbaren Trainingsdatensatzes.

Anmerkungen zur Konvergenz

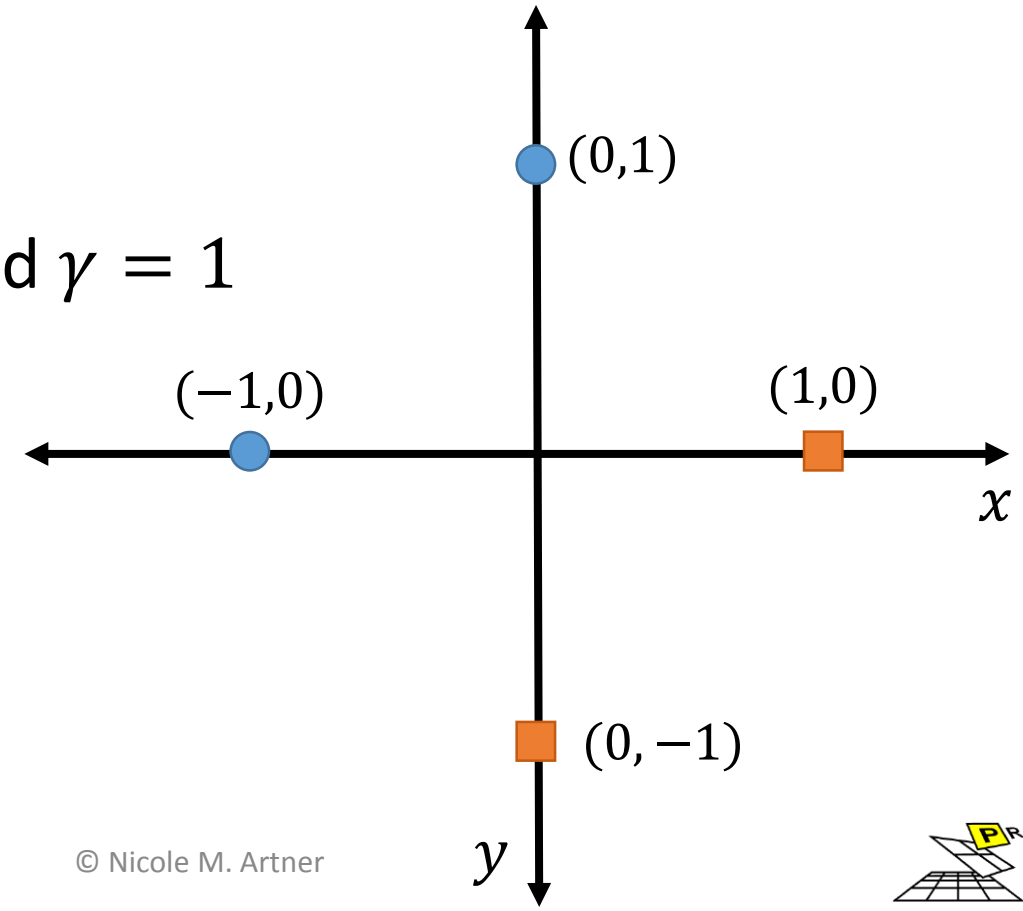
… obwohl Perceptron Algorithmus garantiert eine Lösung für ein linear separierbares Problem findet:

- Anzahl der nötigen Iterationen kann **sehr hoch** werden
- man weiß **erst nach Konvergenz**, dass das Problem linear separierbar ist
- es gibt **viele Lösungen**, welche abhängig von Initialisierung und Reihenfolge der Trainingsdaten sind

Beispiel

- Klasse $w_1 = \{(-1,0,1)^T, (0,1,1)^T\}, y_1 = 1, \ y_2 = 1$
- Klasse $w_2 = \{(0,-1,1)^T, (1,0,1)^T\}, y_3 = -1, \ y_4 = -1$

- $N = 4$
- $\boldsymbol{w} = 0 = (0,0,0)^T$ und $\gamma = 1$



Beispiel

- überprüfe für jeden Vektor $\boldsymbol{x}_i$ ob $\boldsymbol{w}^T(\boldsymbol{x}_i y_i) \leq 0$
- führe Update durch falls notwendig: $\boldsymbol{w} \leftarrow \boldsymbol{w} + \gamma \boldsymbol{x}_i y_i$

Beginn Epoche 1:

$$\boldsymbol{w}^{(0)\,T}(\boldsymbol{x}_1 y_1) = [0 \quad 0 \quad 0] \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} \cdot 1 = 0$$

Update: $$\boldsymbol{w}^{(1)} = \boldsymbol{w}^{(0)} + \gamma \boldsymbol{x}_1 y_1 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}$$

$$\boldsymbol{w}^{(1)\,T}(\boldsymbol{x}_2 y_2) = [-1 \quad 0 \quad 1] \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} \cdot 1 = 1 > 0$$

Beispiel

$$\boldsymbol{w}^{(1)\,T}(\boldsymbol{x}_3 y_3) = [-1 \quad 0 \quad 1] \begin{bmatrix} 0 \\ -1 \\ 1 \end{bmatrix} \cdot -1 = -1$$

Update: $\boldsymbol{w}^{(2)} = \boldsymbol{w}^{(1)} + \gamma \boldsymbol{x}_3 y_3 = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} + \begin{bmatrix} 0 \\ -1 \\ 1 \end{bmatrix} \cdot -1 = \begin{bmatrix} -1 \\ 1 \\ 0 \end{bmatrix}$

$$\boldsymbol{w}^{(2)\,T}(\boldsymbol{x}_4 y_4) = [-1 \quad 1 \quad 0] \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} \cdot -1 = 1 > 0$$

Ende Epoche 1

Beispiel

Beginn Epoche 2:

$$\boldsymbol{w}^{(2)\,T}(\boldsymbol{x}_1 y_1) = [-1 \quad 1 \quad 0]\begin{bmatrix}-1\\0\\1\end{bmatrix}\cdot 1 = 1 > 0$$

$$\boldsymbol{w}^{(2)\,T}(\boldsymbol{x}_2 y_2) = [-1 \quad 1 \quad 0]\begin{bmatrix}0\\1\\1\end{bmatrix}\cdot 1 = 1 > 0$$

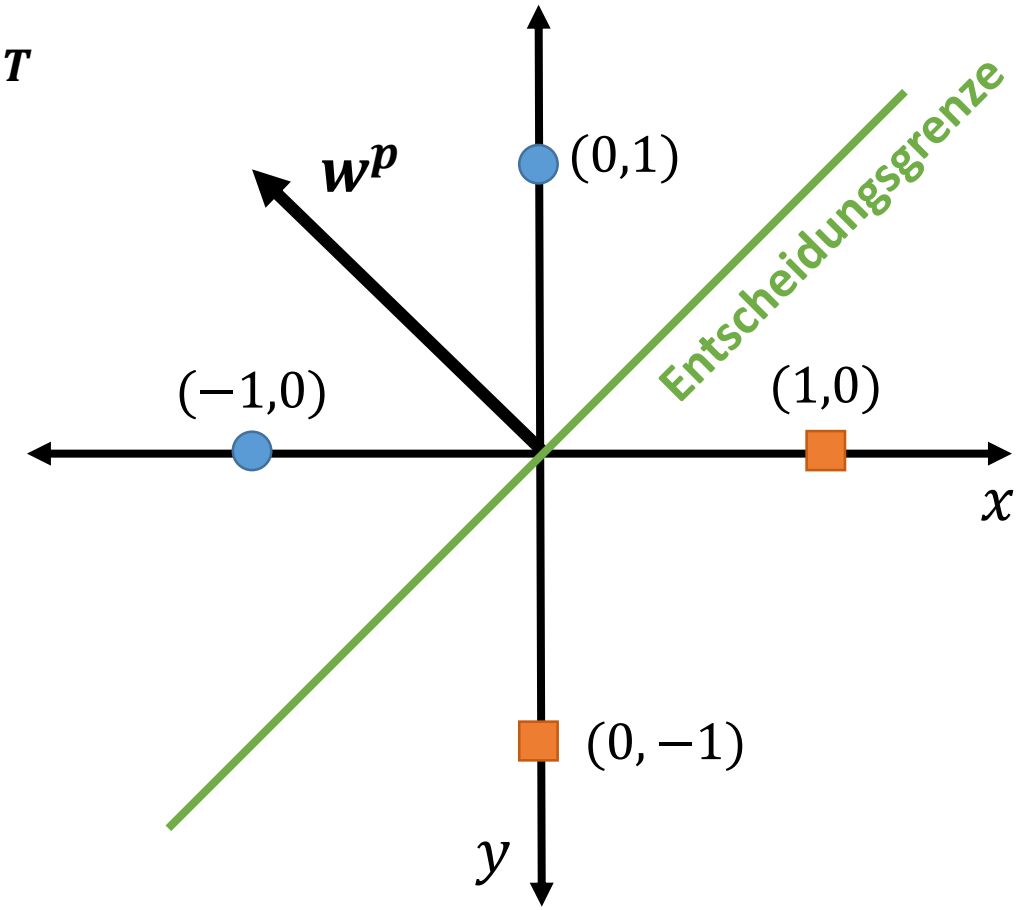
$$\boldsymbol{w}^{(2)\,T}(\boldsymbol{x}_3 y_3) = [-1 \quad 1 \quad 0]\begin{bmatrix}0\\-1\\1\end{bmatrix}\cdot -1 = 1 > 0$$

Ende Epoche 2 → Perceptron-Algorithmus konvergiert

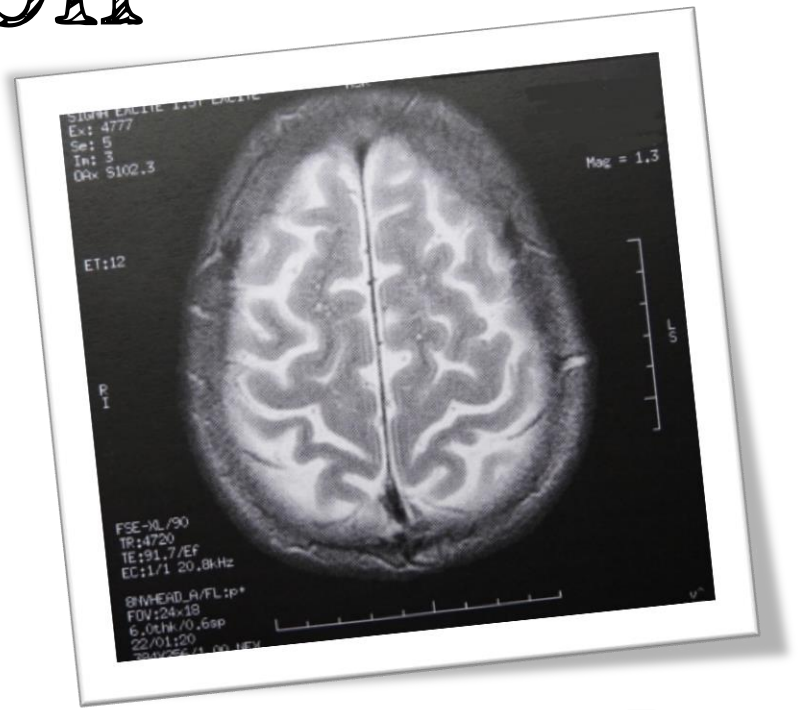
Beispiel

Ergebnis:

$\boldsymbol{w^p} = (\mathbf{-1}, \mathbf{1})^{\boldsymbol{T}}$
$\theta = 0$



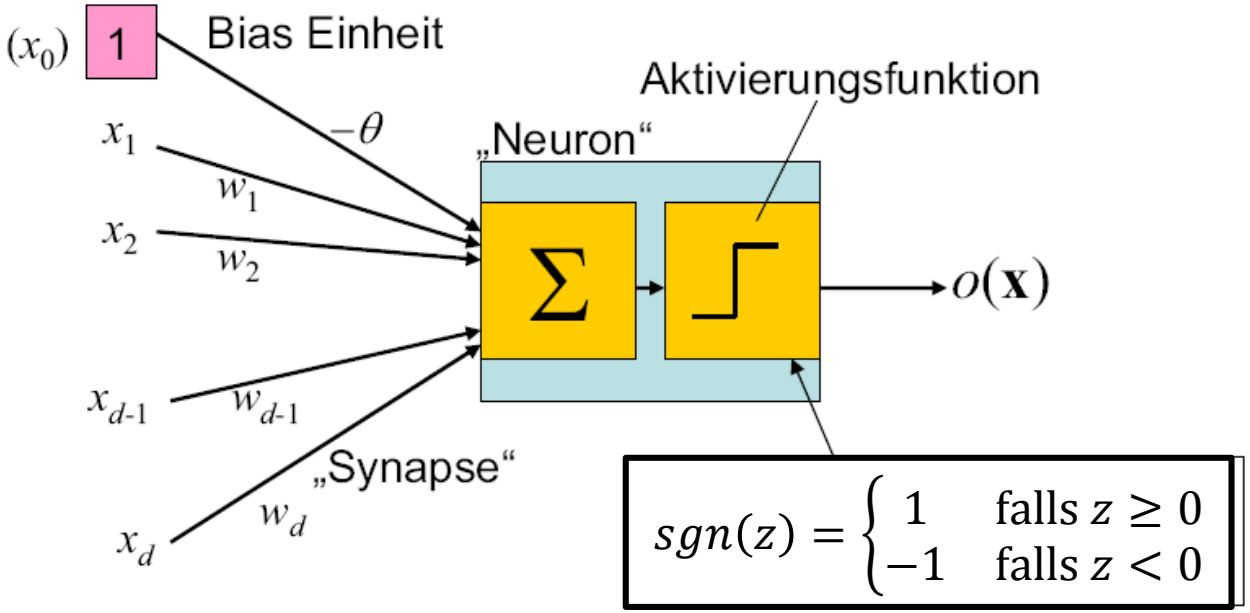
IV. Perceptron als Neuron



Darstellung als Neuron

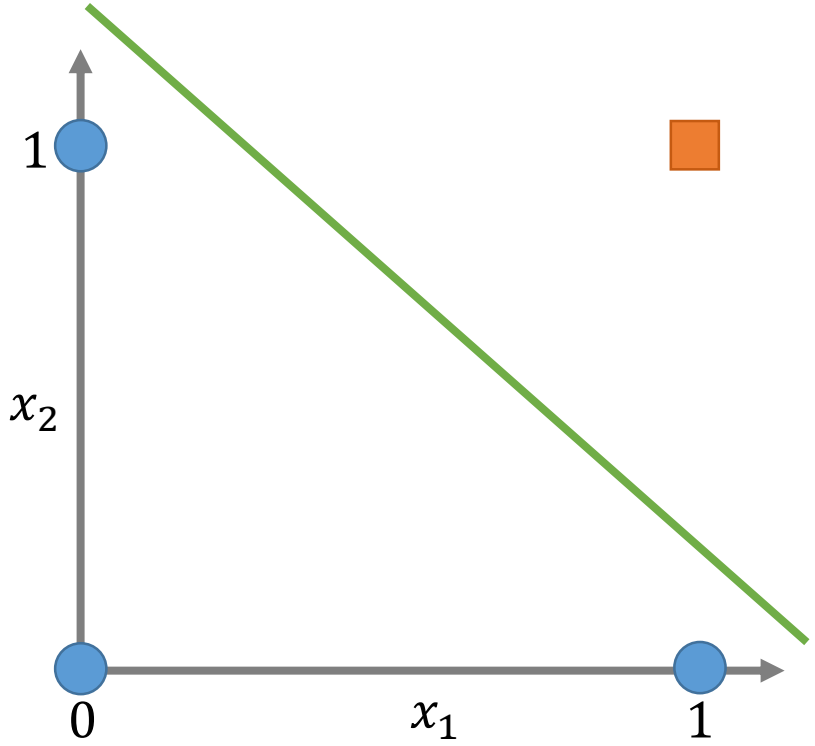
Erinnerung Signumfunktion: $o(\boldsymbol{x}) = sgn\left(\sum_{i=1}^{d} w_i x_i\right)$

Perceptron ist eine gute Annährung an ein biologisches Neuron:



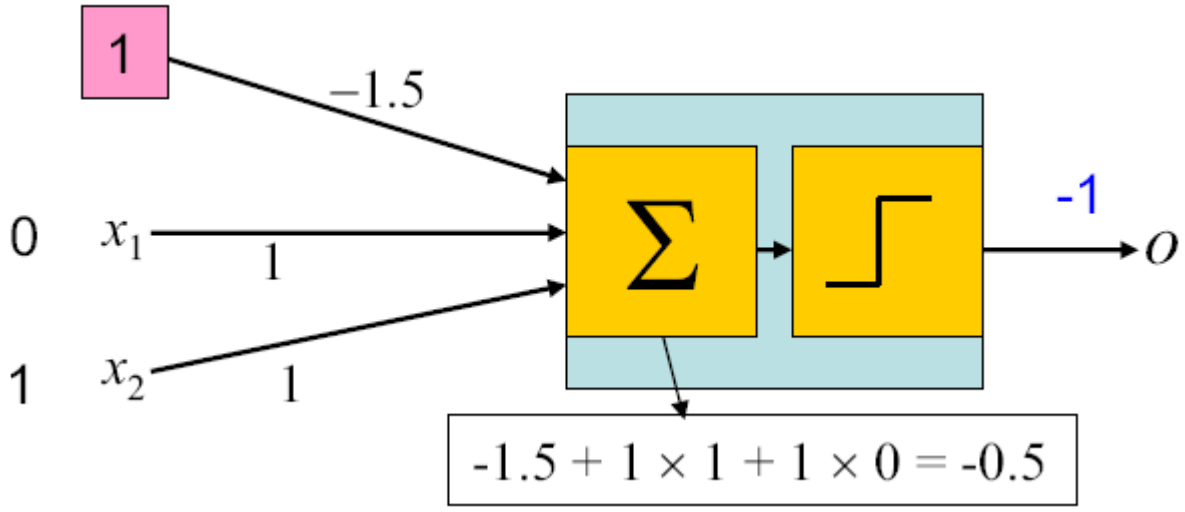
Beispiel: UND

x_1	x_2	$o(x)$
0	0	-1
0	1	-1
1	0	-1
1	1	1



Beispiel: UND

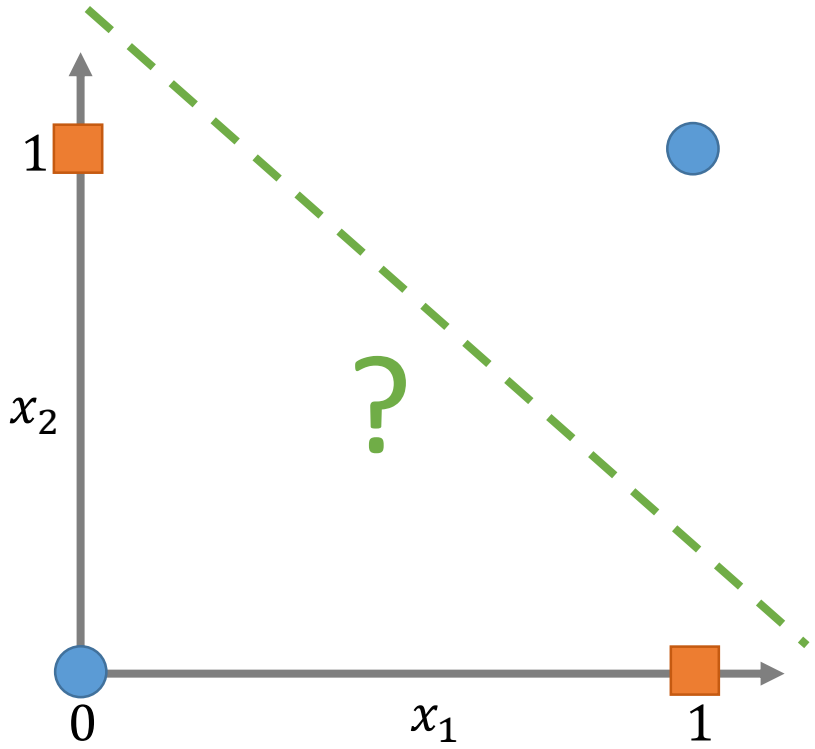
- das UND-Problem kann mit folgendem Perceptron gelöst werden:



Beispiel: XOR

Das XOR-Problem ist nicht linear separierbar!

x_1	x_2	$o(x)$
0	0	-1
0	1	1
1	0	1
1	1	-1



Wichtigster Nachteil

Ein **einstufiges Perceptron** kann nur linear separierbare Mengen, d.h. Mengen die durch eine Hyperebene trennbar sind, fehlerfrei klassifizieren.

Ein bisschen Geschichte …

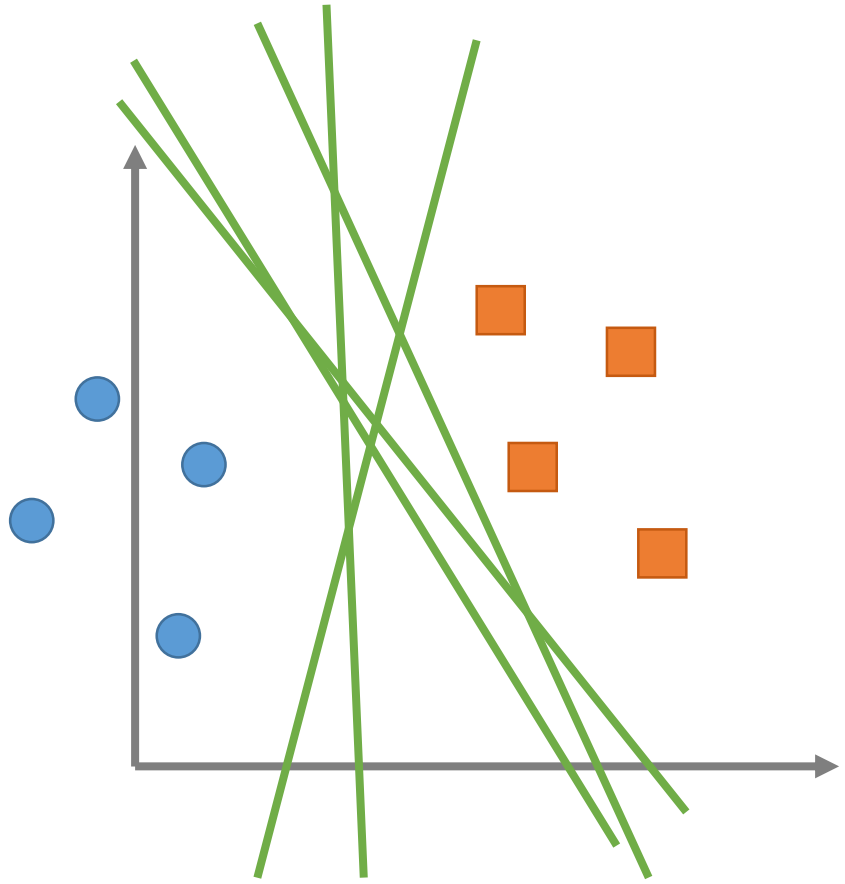
1969: M.L. **Minsky** und S.A. **Papert** zeigen unter anderem diesen Nachteil in ihrem **Buch mit dem Titel „Perceptrons“** auf. Nach der Veröffentlichung dieses Buches war der Enthusiasmus für Forschung auf dem Gebiet der Neuronalen Netze für etwa **15 Jahre gedämpft**.

V. Generalisierungs-fähigkeit des Perceptrons

Lineare Diskriminantenfunktion

Es gibt theoretisch unendlich viele lineare Entscheidungs-grenzen, um dieses Klassifi-kationsproblem zu lösen.

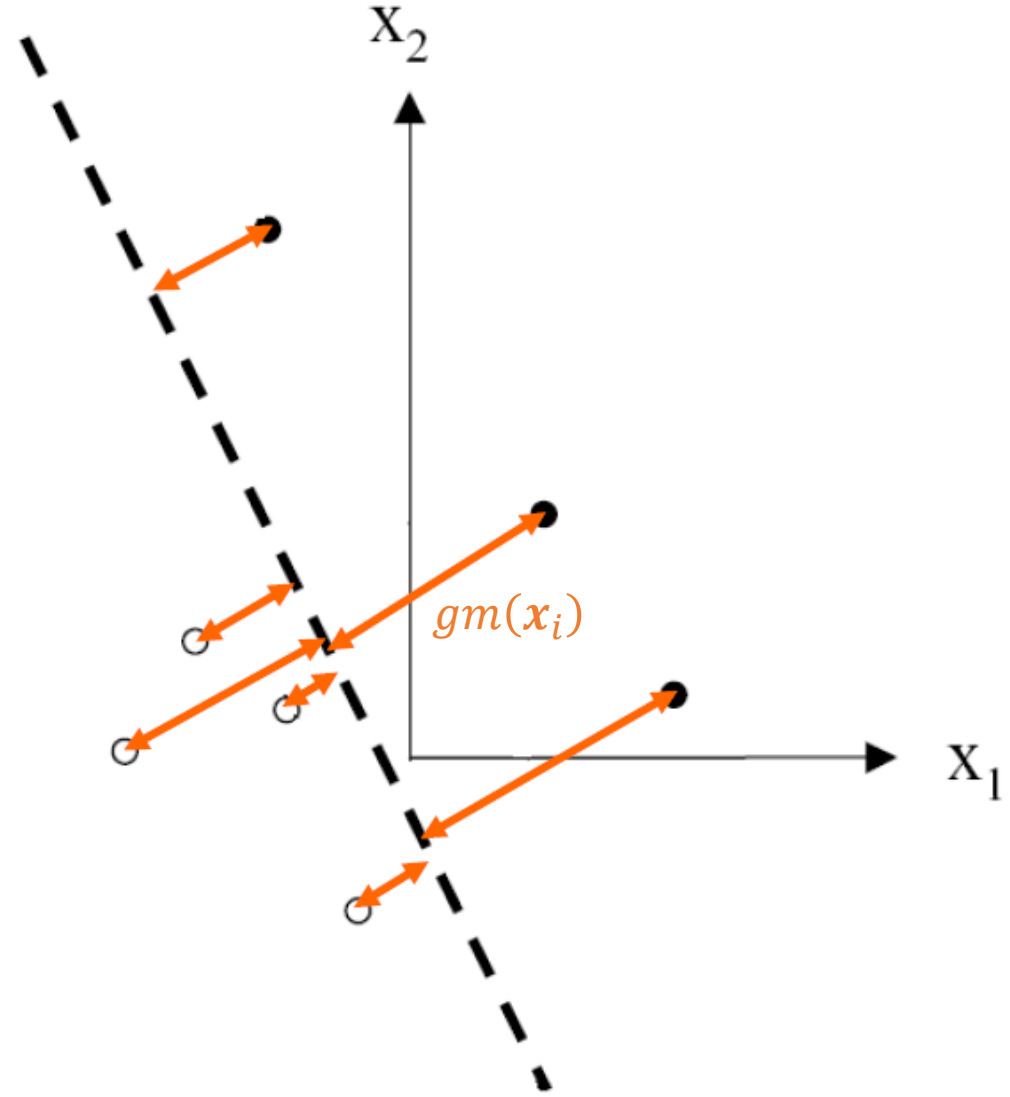
Welche ist die Beste?



Abstand

… Margin (im Englischen)

$gm(\boldsymbol{x}_i)$ …der geometrische Abstand (Margin) vom Vektor $\boldsymbol{x}_i$ zur Entscheidungsgrenze (Hyperebene)

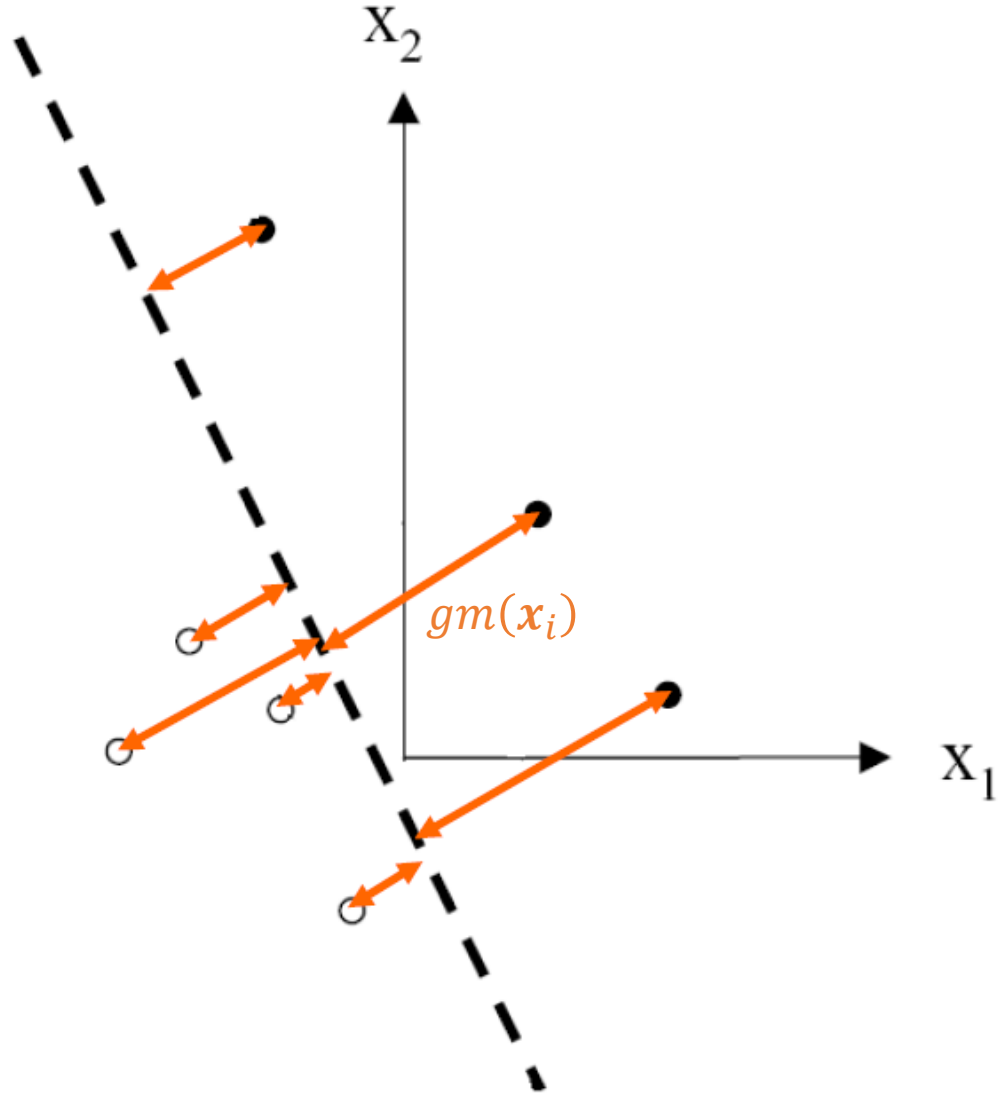


Abstand

… Margin (im Englischen)

$gm(\boldsymbol{w}) = gm(\boldsymbol{x}_j)$ … der geometrische Abstand der Hyperebene bezüglich der Trainingsdaten $\{\boldsymbol{X}, \boldsymbol{y}\}$

$gm(\boldsymbol{x}_j)$ … Vektor mit geringstem geometrischen Abstand: $j = \arg\min_i gm(\boldsymbol{x}_i)$ wobei $1 \leq i \leq N$



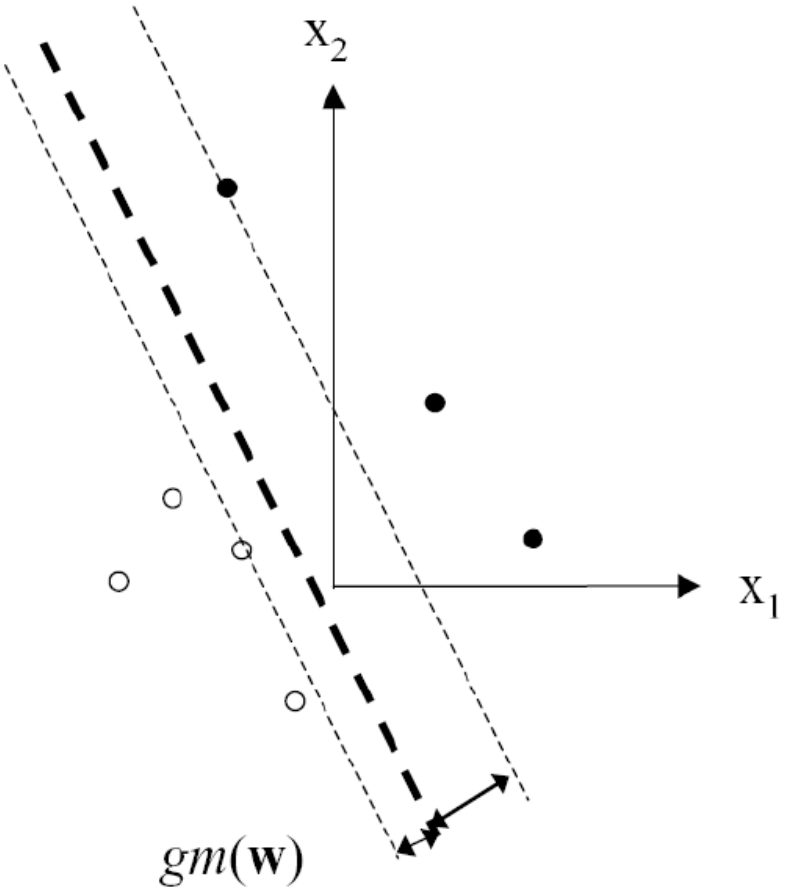
Bezug zum Perceptron

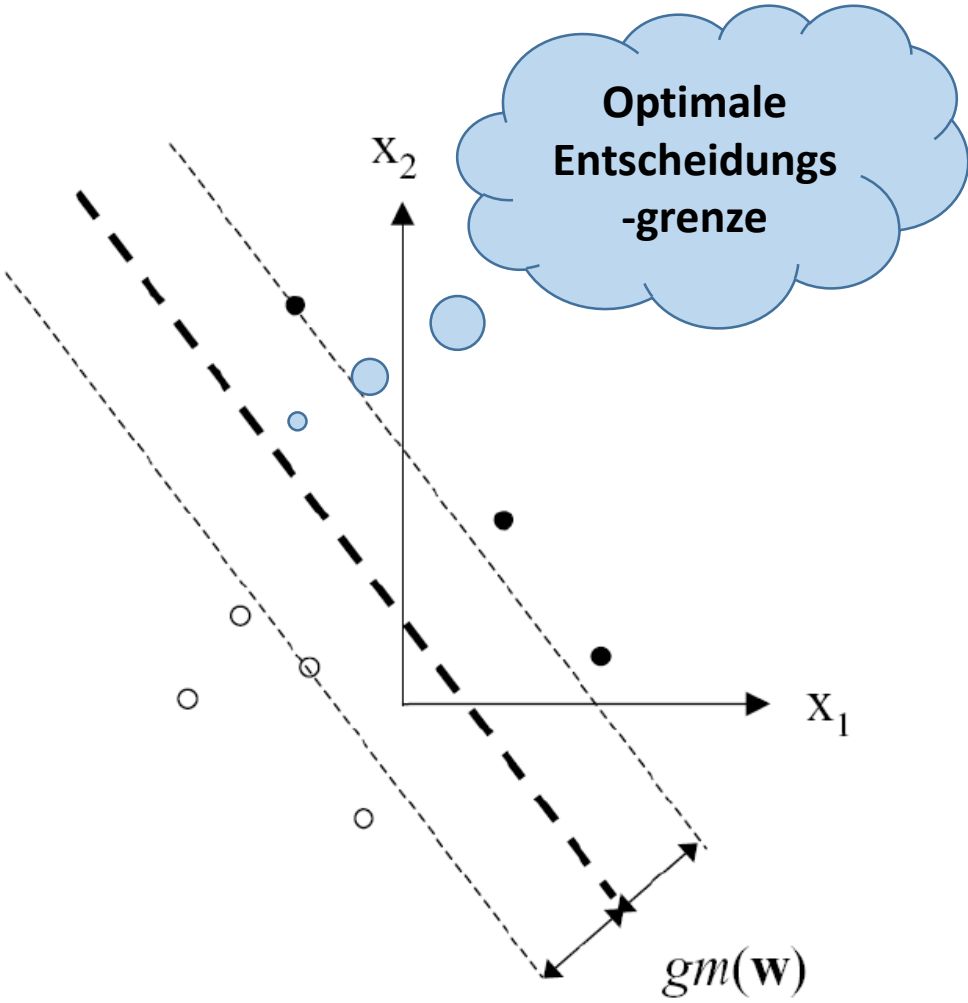
Die **Generalisierungsfähigkeit** des Perceptrons wird umso besser je größer $gm(\boldsymbol{w})$ ist. Es ist auch zu erwarten, dass ein solcher Klassifikator die Klassifikationsfehler minimiert.

Die Idee den Abstand des Trainingsdatensatzes von der Hyperebene zu maximieren ist die Grundidee des beliebten SVM-Klassifikators (SVM=Support Vector Machine).

Solche Klassifikatoren werden auch als „maximum margin classifiers“ bezeichnet.

Beispiel





Verwandte Verfahren

Zwei Nachteile des Perceptrons die die Entwicklung besserer Verfahren motiviert haben:

1. Perceptron-Algorithmus **terminiert nicht** bei nicht linear separierbaren Trainingsdaten → Ho-Kashyap-Algorithmus (siehe Duda et al. ab Seite 249)

1. Perceptron-Algorithmus **findet nicht** unbedingt die optimale Entscheidungsgrenze mit maximalem Abstand zu den Trainingsdaten → SVM