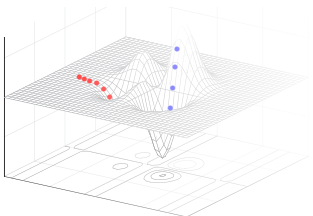
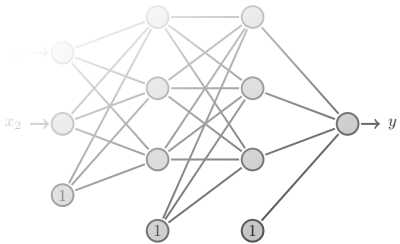
Einstieg Neuronale Netze und einlagiges Perceptron

Ein Kickstart auf der Basis von Keras (Teil 1 von 2)

Prof. Dr. Jörg Frochte

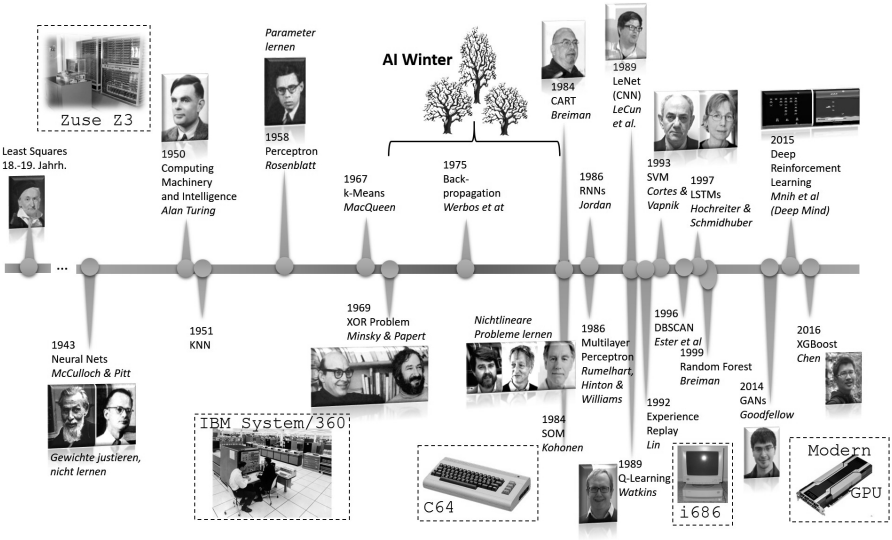
Maschinelles Lernen





- Künstliche neuronale Netze gibt es als Methode schon recht lange. Die Anfänge liegen in den Arbeiten von Warren McCulloch und Walter Pitts aus den 40er Jahren des letzten Jahrhunderts.
- Motiviert wurden und werden diese Ansätze teilweise durch Analogien zu Neuronen im Gehirn.
- Grob kann man zusammenfassen, dass es das Ziel war, ein einfaches mathematisches Modell zu entwickeln, welches das Verhalten der Neuronen im Gehirn erklären sollte.
- Von der ursprünglichen Idee hatten die neuronalen Netze eine Geschichte, die immer wieder von Hype und Desillusion geprägt war.
- Nach einigen Höhen und Tiefen hat sich das Gebiet aktuell wieder konsolidiert und kann immer neue Erfolge vorweisen.
- Um Möglichkeiten und Grenzen der neuronalen Netze zu verstehen, arbeiten wir uns zunächst ein wenig durch die Historie und kommen dann zu komplexeren Strukturen.

Geschichtlicher Abriss: Machine Learning und Neuronale Netze



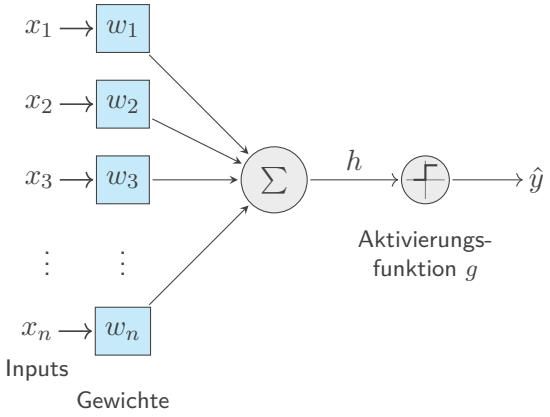
Einfaches mathematisches Modell eines Neurons

Das Modell besteht aus

- einer Menge von Gewichten $w_i \in \mathbb{R}$,
- der Summenfunktion der gewichteten Inputsignale und
- der Aktivierungsfunktion g

$$h = \sum_{i=1}^{n} w_i x_i = \mathbf{w} \cdot \mathbf{x}$$

$$\hat{y} = g(h) = \begin{cases} 1 & \text{für } h > 0 \\ 0 & \text{für } h \leq 0 \end{cases}$$



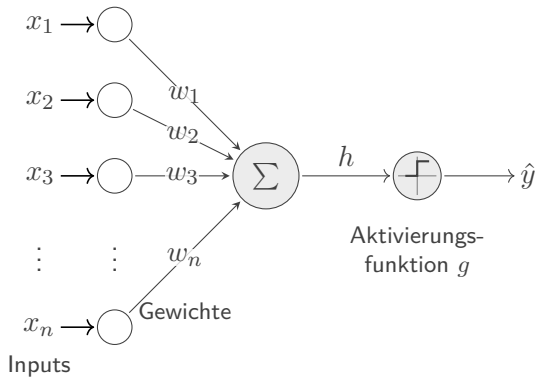
Einfaches mathematisches Modell eines Neurons

Das Modell besteht aus

- einer Menge von Gewichten $w_i \in \mathbb{R}$,
- der Summenfunktion der gewichteten Inputsignale und
- der Aktivierungsfunktion g

$$h = \sum_{i=1}^{n} w_i x_i = \mathbf{w} \cdot \mathbf{x}$$

$$\hat{y} = g(h) = \begin{cases} 1 & \text{für } h > 0 \\ 0 & \text{für } h \leq 0 \end{cases}$$



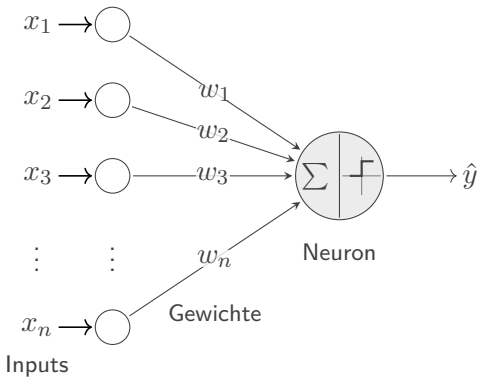
Einfaches mathematisches Modell eines Neurons

Das Modell besteht aus

- einer Menge von Gewichten $w_i \in \mathbb{R}$,
- der Summenfunktion der gewichteten Inputsignale und
- der Aktivierungsfunktion g

$$h = \sum_{i=1}^{n} w_i x_i = \mathbf{w} \cdot \mathbf{x}$$

$$\hat{y} = g(h) = \begin{cases} 1 & \text{für } h > 0 \\ 0 & \text{für } h \leq 0 \end{cases}$$



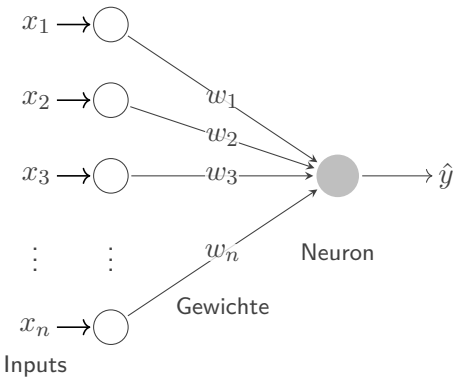
Einfaches mathematisches Modell eines Neurons

Das Modell besteht aus

- einer Menge von Gewichten $w_i \in \mathbb{R}$,
- der Summenfunktion der gewichteten Inputsignale und
- der Aktivierungsfunktion g

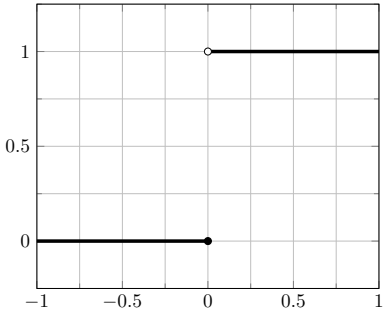
$$h = \sum_{i=1}^{n} w_i x_i = \mathbf{w} \cdot \mathbf{x}$$

$$\hat{y} = g(h) = \begin{cases} 1 & \text{für } h > 0 \\ 0 & \text{für } h \leq 0 \end{cases}$$



Schwellwertfunktion

- Die Anlehnung an eine echte Nervenzelle kommt daher, dass analog zu dieser Anregungen aus anderen Nervenzellen auf diese übertragen werden.
- Auch hier findet eine Art Summation statt.
- Liegt diese Summe von Anregungen über einem Schwellenpotential (hier: 0), feuert die Nervenzelle ihrerseits.
- Mit der Heaviside-Funktion H als einer der einfachsten Aktivierungsfunktionen g wurde versucht, dieses *Alles-oder-nichts* Prinzip bei der Überschreitung des Schwellenpotentials umzusetzen.



$$\hat{y} = g(h) = H(h) \begin{cases} 1 & \text{für } h > 0 \\ 0 & \text{für } h \leq 0 \end{cases}$$

Was kann so ein Neuron?

- Mathematisch gesprochen ist dieses Neuron eine sehr einfache Sache.
- Jeder Eintrag x_i des Inputvektors $\mathbf{x}$ wird jeweils mit einem Gewicht w_i multipliziert.
- Wir können also w_i erneut zu einem Vektor $\mathbf{w}$ kombinieren.
- Den Effekt erhalten wir durch ein einfaches Skalarprodukt

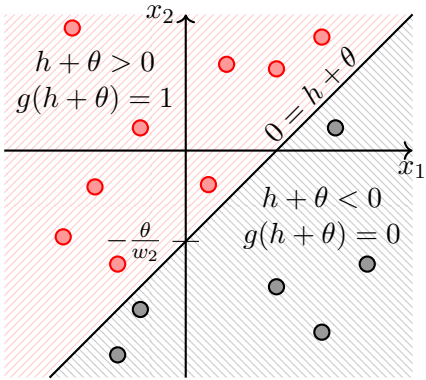
$$h = x_1 \cdot w_1 + x_2 \cdot w_2 + \cdots + x_n \cdot w_n = \sum_{i=1}^{n} x_i \cdot w_i = \mathbf{x} \cdot \mathbf{w}$$

- Bislang schaltet $g(h)$ bei $h = 0$. Wenn der Schwellwert $-\theta$ sein soll, muss θ addiert werden, d. h. $g(h + \theta)$. Die Schwelle liegt nun bei

$$h + \theta = x_1 \cdot w_1 + x_2 \cdot w_2 + \cdots + x_n \cdot w_n + \theta = 0$$

- **Woher kennen Sie diese Form?**

- Mathematisch wird also hierdurch eine (Hyper-)Ebene definiert.
- Durch θ wird die Hyperebene aus dem Ursprung hinaus verschoben.
- Die Aktivierungsfunktion wird dann für alle Werte auf der einen Seite der Hyperebene den Wert 1 haben und auf der anderen 0.
- Eine solche Hyperebene kann man daher als Klassifikator benutzen, wenn zwei Mengen **linear separierbar** sind.

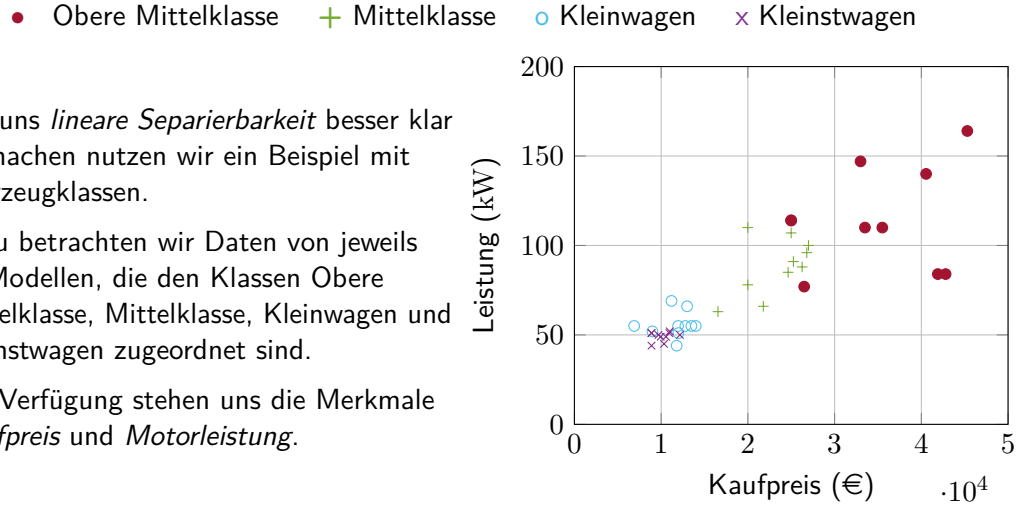


$$h+\theta = x_1 \cdot w_1 + x_2 \cdot w_2 + \cdots + x_n \cdot w_n + \theta = 0$$

Linear Separierbar

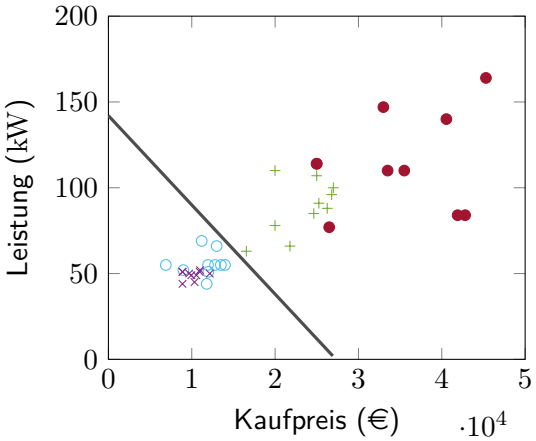
Man bezeichnet Mengen $M_1, M_2 \subset \mathbb{R}^n$ als linear separierbar, wenn eine Hyperebene im $\mathbb{R}^n$ existiert, die diese beiden Mengen im $\mathbb{R}^n$ trennt.

Grundlagen von neuronalen Netzen

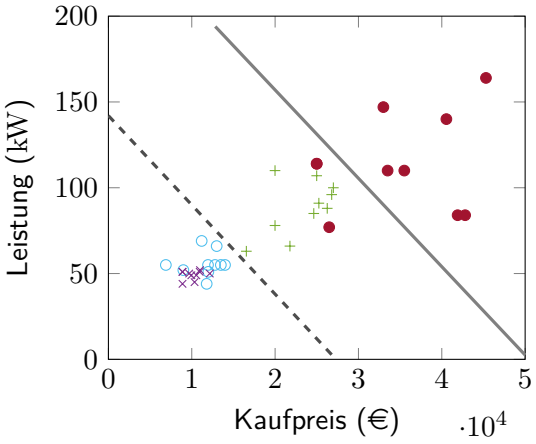


- Um uns *lineare Separierbarkeit* besser klar zu machen nutzen wir ein Beispiel mit Fahrzeugklassen.
- Dazu betrachten wir Daten von jeweils 10 Modellen, die den Klassen Obere Mittelklasse, Mittelklasse, Kleinwagen und Kleinstwagen zugeordnet sind.
- Zur Verfügung stehen uns die Merkmale *Kaufpreis* und *Motorleistung*.

- Das sind jetzt vier Mengen. Wir können sicherlich mit einer Trennlinie nur zwei Mengen unterscheiden.
- Fassen wir also zunächst einmal die Klein- und Kleinstwagen sowie die Mittelklasse und obere Mittelklasse in jeweils eine Menge zusammen.
- Linear separierbar sind die Mengen, wenn Sie jetzt mit einem Lineal und Bleistift einen Strich ziehen können und auf der einen Seite ist nur die Menge A und auf der anderen B.



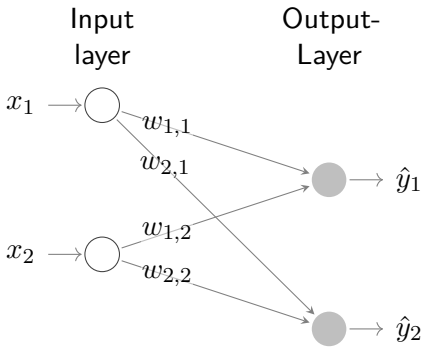
- Das sind jetzt vier Mengen. Wir können sicherlich mit einer Trennlinie nur zwei Mengen unterscheiden.
- Fassen wir also zunächst einmal die Klein- und Kleinstwagen sowie die Mittelklasse und obere Mittelklasse in jeweils eine Menge zusammen.
- Linear separierbar sind die Mengen, wenn Sie jetzt mit einem Lineal und Bleistift einen Strich ziehen können und auf der einen Seite ist nur die Menge A und auf der anderen B.



- Für die Aufteilung vorhin funktionierte es. Ginge es um die Menge *obere Mittelklasse* ● als Menge A und den Rest als Menge B entstünden zwei Fehlklassifikationen.
- Lineare Separierbarkeit ist nichts Selbstverständliches, sondern etwas Besonderes.

Einlagiges Perzeptron

- Ohne θ könnte die Trennungsgerade nur durch den Ursprung gehen.



Schema eines einlagigen Perzeptrons

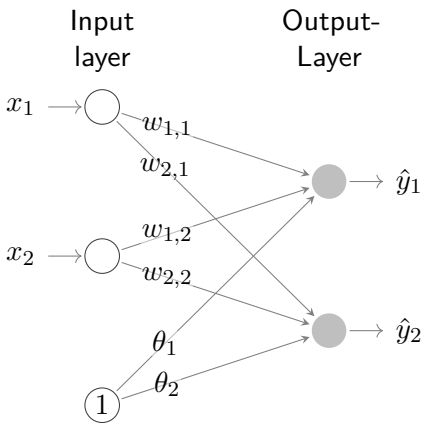
Einlagiges Perzeptron

- Ohne θ könnte die Trennungsgerade nur durch den Ursprung gehen.
- Um θ mitzulernen, fügen wir einen zusätzlichen Input-Wert hinzu, welcher konstant 1 ist.
- Mit der Wahl der Indizes lässt sich das schreiben als

$$\hat{\mathbf{y}} = \mathbf{g}(\mathbf{h}) = \mathbf{g}(\mathbf{W}\,\mathbf{x} + \boldsymbol{\theta}),$$

mit elementweiser Aktivierungsfunktion g und

$$\mathbf{h} = \begin{pmatrix} w_{1,1} & w_{1,2} \\ w_{2,1} & w_{2,2} \end{pmatrix} \mathbf{x} + \begin{pmatrix} \theta_1 \\ \theta_2 \end{pmatrix}$$



Schema eines einlagigen Perzeptrons

Einlagiges Perzeptron

- Ohne θ könnte die Trennungsgerade nur durch den Ursprung gehen.
- Um θ mitzulernen, fügen wir einen zusätzlichen Input-Wert hinzu, welcher konstant 1 ist.
- Mit der Wahl der Indizes lässt sich das schreiben als

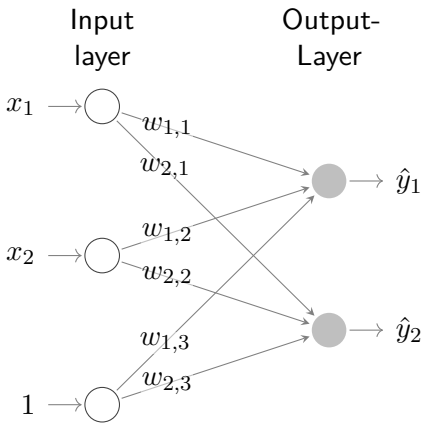
$$\hat{\mathbf{y}} = \mathbf{g}(\mathbf{h}) = \mathbf{g}(\mathbf{W}\,\mathbf{x} + \boldsymbol{\theta}),$$

mit elementweiser Aktivierungsfunktion g und

$$\mathbf{h} = \begin{pmatrix} w_{1,1} & w_{1,2} \\ w_{2,1} & w_{2,2} \end{pmatrix} \mathbf{x} + \begin{pmatrix} \theta_1 \\ \theta_2 \end{pmatrix}$$

oder alternativ, mit 1 im Input-Vektor $\hat{\mathbf{x}}$:

$$\mathbf{h} = \begin{pmatrix} w_{1,1} & w_{1,2} & w_{1,3} \\ w_{2,1} & w_{2,2} & w_{2,3} \end{pmatrix} \hat{\mathbf{x}}$$



Schema eines einlagigen Perzeptrons

- Betrachten wir einmal nur y_1 um die Rolle des Bias-Neurons als Ersatz für θ zu verdeutlichen.

$$h_1(\hat{\mathbf{x}}) = w_{1,1} \cdot x_1 + w_{1,2} \cdot x_2 + w_{1,3} \cdot 1$$

- Wie man sieht, übernimmt $w_{1,3}$ nun die Rolle von θ_1.
- Wir beschränken uns für unser Startbeispiel mit der Klassifikation der PKW auf den einfachsten Fall mit einem Output-Neuron.
- In diesem Fall vereinfacht sich einiges und wir erhalten eine einzeilige Matrix $\hat{\mathbf{W}} = \begin{pmatrix} w_{1,1} & w_{1,2} & w_{1,3} \end{pmatrix}$, die man natürlich in diesem Fall auch einfacher als Vektor notieren kann: $\mathbf{w} = \begin{pmatrix} w_1 & w_2 & w_3 \end{pmatrix}$
- Jetzt gehen wir daran, unser Netzwerk zu trainieren, was in diesem Fall bedeutet, die Gewichte w_i anzupassen.
- Für unser Einstiegsbeispiel verwenden wir zunächst eine der ältesten und einfachsten Lernregeln. Es handelt sich um die **Hebbsche Lernregel**.

- Grundlage der **Hebbschen Lernregel** ist zunächst der Unterschied zwischen dem gewünschten $\hat{y}$ eines Trainingswertes und dem tatsächlich beobachteten Output y.
- Diesen Fehler notieren wir einfach als:

$$\text{error} = \hat{y} - y$$

- Die Grundidee ist nun, falls $\hat{y} > y$ ist, die Gewichte zu vergrößern, und diese zu verkleinern, falls $\hat{y} < y$.
- Dabei sollen Gewichte, die einen großen Einfluss haben, stärker verändert werden, als Gewichte mit einem kleinen Einfluss.
- Aus dieser Grundidee ergeben sich die folgenden Zuweisungen für die Anpassung der Gewichte:

$$\Delta w_j = \text{error} \cdot x_j$$
$$w_j = w_j + \Delta w_j$$

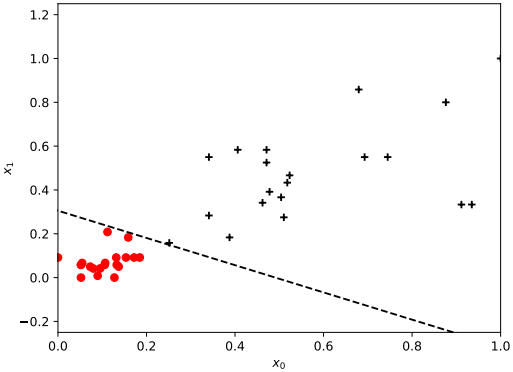
- Der Ansatz ist nicht stabil, da die Werte zu stark springen und nicht konvergieren.
- Um dies zu verhindern, wird die Änderung mit einer Lernrate $\eta \in]0, 1]$ multipliziert, welche dämpfend auf die Änderungsprozesse wirkt.

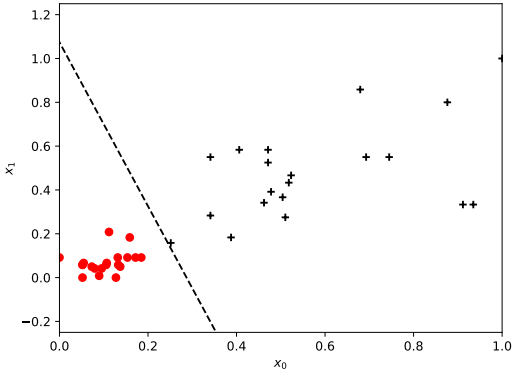
Pseudocode

Gegeben ist eine Menge von Beispielen $D = \{(\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), \dots\}$
$\mathbf{w} \in \mathbb{R}^p$ mit Zufallszahlen im Intervall $[-0.5, 0.5]$ initialisieren
$t = 0$
repeat
    $t = t + 1$
    $(x^{(d)}, y^{(d)}) = \mathrm{RandomSelect}(D)$
    error $= y^{(d)} - H(\mathbf{w}\,\mathbf{x}^{(d)})$ $\{H$ ist die Heaviside-Funktion$\}$
    for $j = 0$ **to** p **do**
        $\Delta w_j = \eta \cdot \text{error} \cdot x_j^{(d)}$
        $w_j = w_j + \Delta w_j$
    end for
until $\|\mathbf{Y} - H(\mathbf{w}\,\mathbf{X}))\| < \varepsilon$ **or** $t > t_{\max}$

Optimale Separation

- Auch, wenn Mengen linear separierbar sind, ist die Trennung nicht eindeutig.





- Welche Trennung finden Sie intuitiv besser? Welche wäre noch besser?
- Eine Lösung liegt z. B. in einer Support Vector Machine (Kapitel 12), die wir aber in der Vorlesung nicht behandeln.