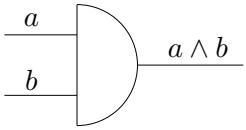
Künstliche Neuronen

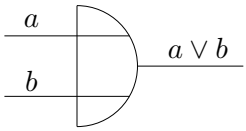
Realisiere die binären Funktionen mit Neuronen.

Und-Gatter *Konjunktion* a und b



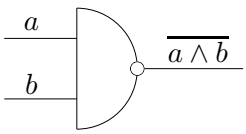
a	b	$a \wedge b$
0	0	0
0	1	0
1	0	0
1	1	1

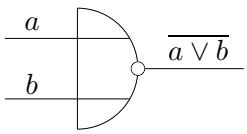
Oder-Gatter *Disjunktion* a oder b



a	b	$a \vee b$
0	0	0
0	1	1
1	0	1
1	1	1

a	b	$a \wedge \overline{b}$
0	0	0
0	1	0
1	0	1
1	1	0





Berechnung logischer Funktionen

```
for a in range(2):
    for b in range(2):
        Neuron
```

Wähle die Gewichte w_1, w_2 und den Schwellenwert S so, dass die Ausgabefunktion

$$f(a,b) = \begin{cases} 0 & \text{falls} \quad w_1 a + w_2 b < S \\ 1 & \text{sonst} \end{cases}$$

mit einer selbstgewählten booleschen Funktion (z. B. $a \wedge \overline{b}$) übereinstimmt.

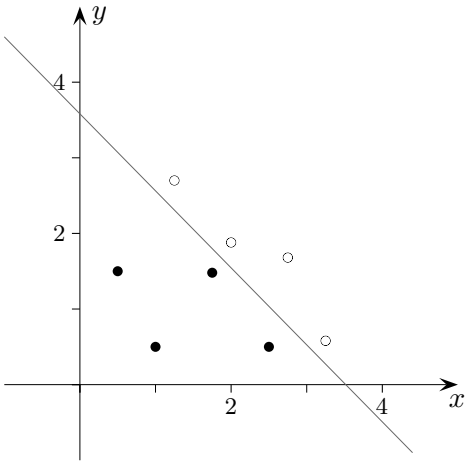
Geometrische Veranschaulichung der Schwellenwertfunktion f

$$f(a,b) = \begin{cases} 0 & \text{falls} \quad w_1 a + w_2 b < S \\ 1 & \text{sonst} \end{cases}$$

Schwellenwert S

$$\begin{aligned} w_1 a + w_2 b &< S \\ w_2 b &< -w_1 a + S \qquad \text{Annahme } w_2 > 0 \\ b &< -\frac{w_1}{w_2} a + \frac{S}{w_2} \end{aligned}$$

vergleiche mit $y = mx + b$

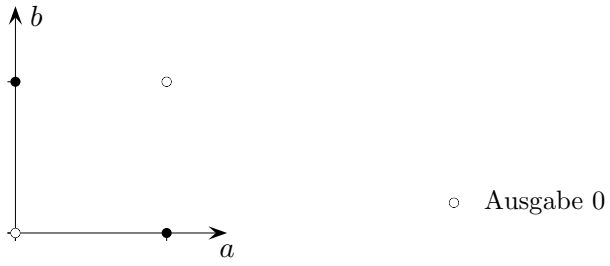


Da jede Gerade die Ebene in zwei Halbebenen trennt (linear separiert), zeigt die Ausgabe 0 oder 1 des Neurons an, in welcher Halbebene der Punkt (a, b) liegt.

Ziel ist es, die Trenngerade zu finden.

Begründe: Für die exklusiv-oder-Funktion $\oplus$ reicht ein Neuron nicht aus.

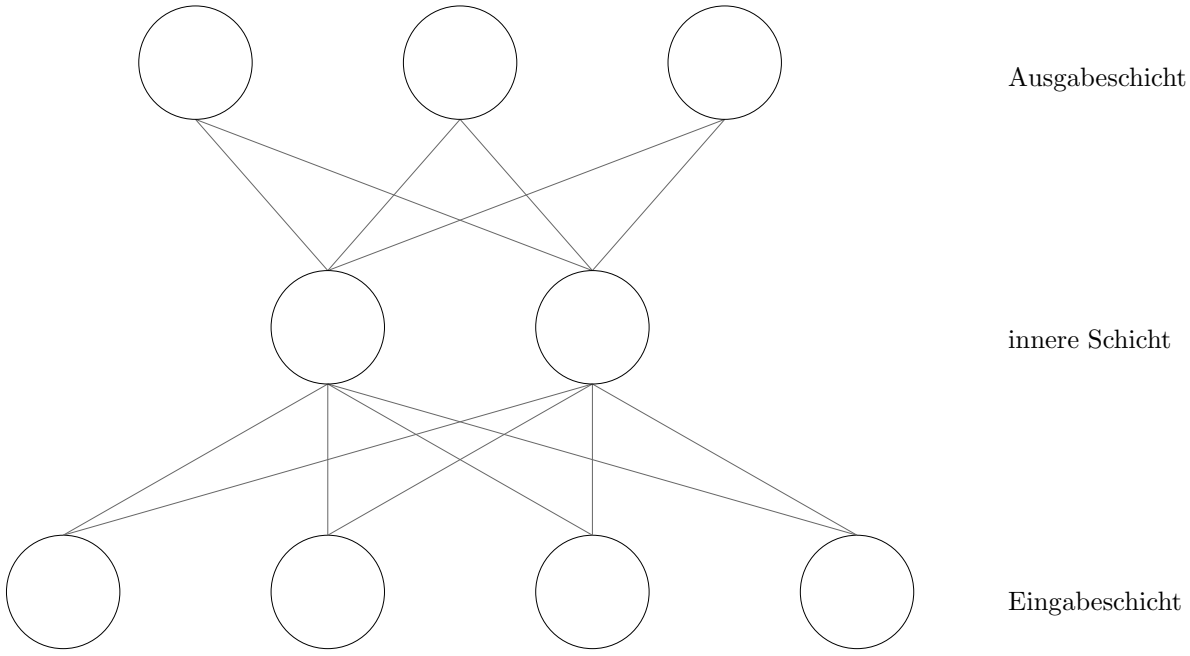
a	b	$a \oplus b$
0	0	0
0	1	1
1	0	1
1	1	0



Es gibt keine Gerade, die die vier Punkte gemäß den Ausgaben 0 und 1 trennt.

Für $n = 3$ Eingabegrößen e_i stellt die Gleichung $w_1e_1 + w_2e_2 + w_3e_3 = S$ eine Ebene dar, die den Raum in zwei Halbräume trennt. Die Überlegung lässt sich für $n > 3$ verallgemeinern, auch wenn die räumliche Anschauung nicht vorhanden ist.

Mehrschichtige Netze



Ab etwa 1985 wurden Lernverfahren für mehrschichtige Netze entwickelt.

Algorithmus für das überwachte Lernen

initialisiere die Neuronengewichte w_1 und w_2 zufällig,

wiederhole

berechne die Ausgabe $A_{\text{ist}} = f(a, b)$ einer zufällig gewählten Eingabe (a, b),

wenn A_{ist} nicht mit der gewünschten Ausgabe A_{soll} übereinstimmt,

dann

$$\Delta = A_{\text{soll}} - A_{\text{ist}}$$
$$w_1^{\text{neu}} = w_1^{\text{alt}} + \alpha \cdot \Delta \cdot a$$
$$w_2^{\text{neu}} = w_2^{\text{alt}} + \alpha \cdot \Delta \cdot b$$

bis für alle Daten die zugehörigen Ausgaben richtig sind.

$$f(a,b) = \begin{cases} 0 & \text{falls} \quad w_1 a + w_2 b < S \\ 1 & \text{sonst} \end{cases} \qquad \text{Schwellenwert } S$$

Die Änderung für jedes Gewicht ist proportional zur Eingabe am zugehörigen Eingang und proportional zur Differenz $\Delta = A_{\text{soll}} - A_{\text{ist}}$. Diese Anpassungsregel heißt Delta-Regel, α wird Lernrate genannt. Es ist sinnvoll, α während der Lernphase allmählich zu verkleinern.

Schreibe ein Programm, bei dem die anfänglich zufällig gewählten Anfangswerte für w_1, w_2 und S iterativ so angepasst werden, dass die Ausgabefunktion f mit einer selbstgewählten booleschen Funktion (z. B. $a \wedge \overline{b}$) übereinstimmt.

```
for a in [False,True]:
    for b in [False,True]:
        print(int(a), int(b), " ", int(a and not b))
```

Programm für das überwachte Lernen

```
from random import *

w1=random()
w2=random()
S=random()
N=10
alpha=0.9

for n in range(1,N):
  for a in [False,True]:
    for b in [False,True]:
            A_soll =int(a and  not b)
            if w1*int(a) + w2*int(b) < S:
                A_ist =0
            else:
                A_ist =1

            Delta = A_soll - A_ist
            w1=w1+Delta*alpha*(int(a))
            w2=w2+Delta*alpha*(int(b))
            S=S-Delta*alpha
            alpha=0.9*alpha
            if n%(N-1)==0:
              print(int(a),int(b), " ", A_soll ,A_ist,w1,w2,S)
```

Beachte:

Mit S=S−Delta*alpha wird die Relation w1*int(a) + w2*int(b) < S in Richtung Gleichheit (=, Geradengleichung) verbessert.