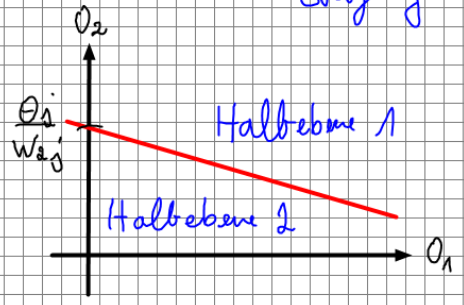
Funktion eines einzelnen Neurons

Behauptung: $o_1 \cdot w_{1j} + o_2 \cdot w_{2j} = \Theta_j$ ist die Gleichung einer Trenngeraden, welche diejenigen Eingangswerte (o_1, o_2) die einen Ausgangswert von 0 ergeben von denjenigen Eingangswerten trennt, die einen Ausgangswert von 1 ergeben.

$$o_2 \cdot w_{2j} = -o_1 \cdot w_{1j} + \Theta_j$$

$$o_2 = \underbrace{-\frac{w_{1j}}{w_{2j}}}_{\text{Steigung}} \cdot o_1 + \underbrace{\frac{\Theta_j}{w_{2j}}}_{o_2\text{-Achsenabschnitt}}$$

⇒ Geradengleichung

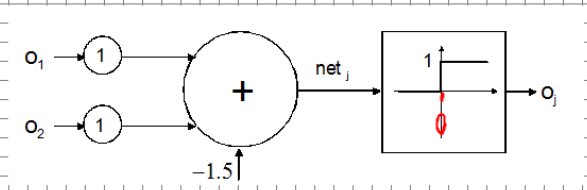


⇒ Das Neuron trennt 2 Halbebenen

⇒ Abhängig davon in welcher Halbebene die Eingangswerte liegen, gibt das Neuron den Wert 0 oder 1 aus.

⇒ Ohne den Biaswert Θ_j würde die Trenngerade immer durch den Ursprung verlaufen.

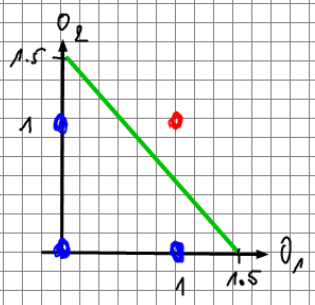
Beispiel: AND



Trenngrade:

$\emptyset = o_1 + o_2 - 1.5$

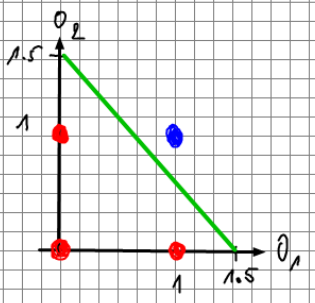
$\Leftrightarrow o_2 = -o_1 + 1.5$



Steigung : -1

o_2-Achsenabschnitt : 1.5

Frage: Was muss man tun, um ein Neuron mit NAND-Charakteristik zu erhalten ?



$\Rightarrow$ gleiche Trenngrade

Behauptung:
Vorzeichen für w_1, w_2, Θ negieren!

Beweis:

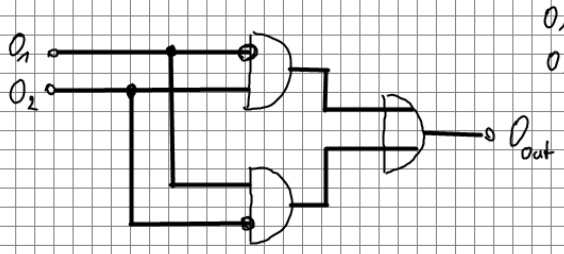
AND $\quad 1 \cdot o_1 + 1 \cdot o_2 - 1.5 \geq 0$ dann wird 1 ausgegeben

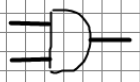
NAND $\quad -1 \cdot o_1 - 1 \cdot o_2 + 1.5 \geq 0 \quad | \cdot (-1)$

$1 \cdot o_1 + 1 \cdot o_2 - 1.5 < 0$ dann wird 1 ausgegeben

XOR mit log. Gattern

$O_{aut} = \overline{O_1} \wedge O_2 \vee O_1 \wedge \overline{O_2}$



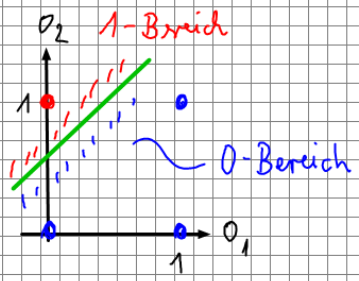

AND


OR

$\triangleq \quad \overline{O_1} \wedge O_2$

O_2 AND NOT O_1

Realisierung von $\overline{O_1} \wedge O_2$



Steigung : 1

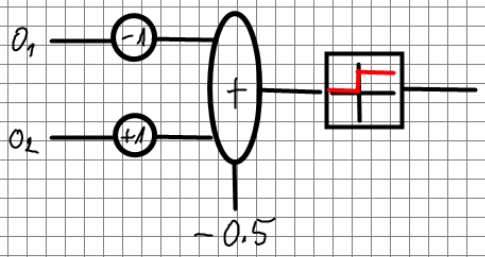
Achsenabschnitt : 0.5

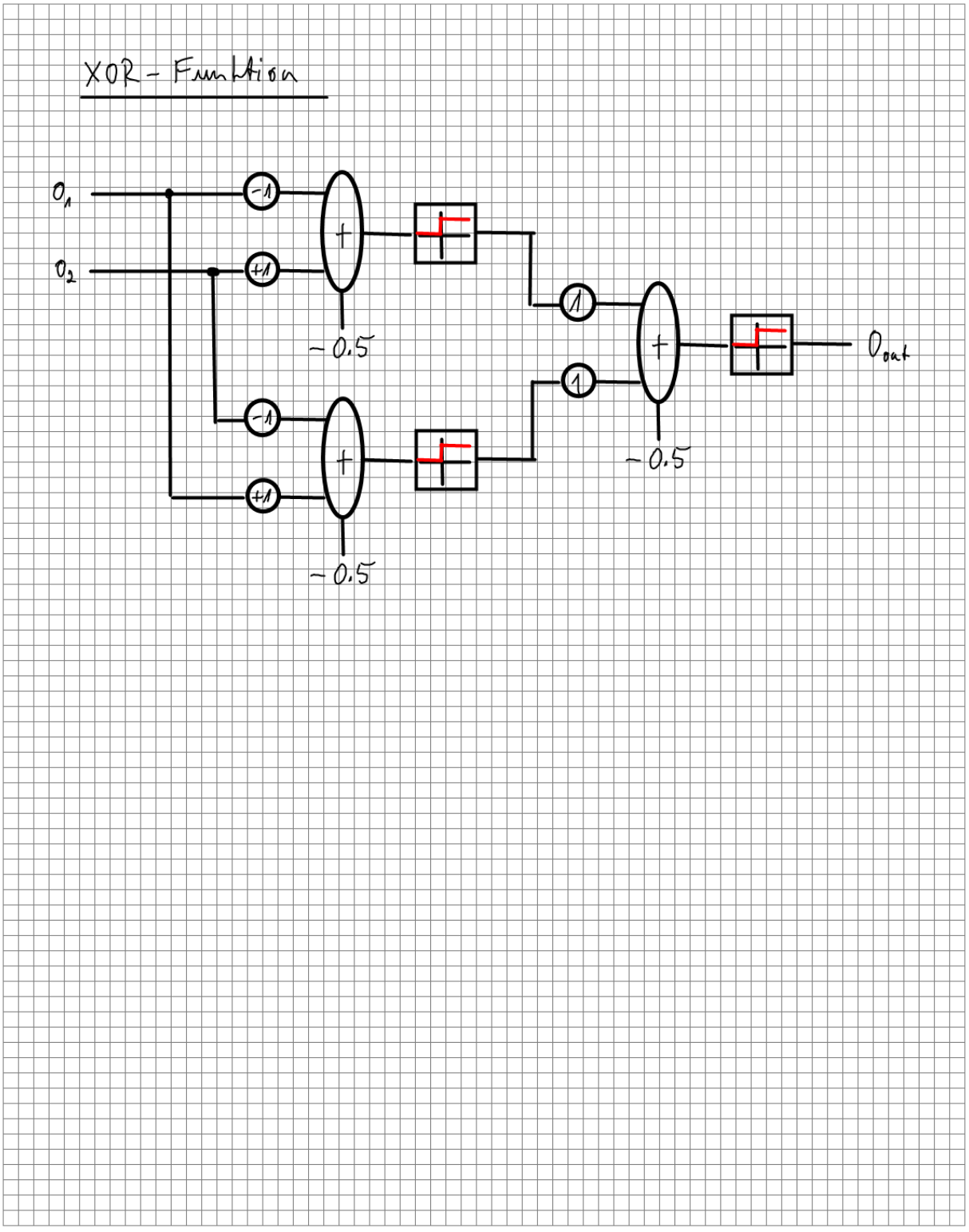
⇓ Grade

$O_2 = 1 \cdot O_1 + 0.5$

$-1 \cdot O_1 + 1 \cdot O_2 - 0.5 = 0$

Gewichte — Achsenabschnitt




XOR - Funktion
o_1
o_2
-1
+1
+
-0.5
1
1
+
o_{out}
-1
+1
+
-0.5
-0.5

Gedankenexperiment mit einem Neuron

Brechstangenansatz = Brute-force-Ansatz

Beispiel: 4 Repräsentanten ($p = 1 \ldots 4$)
2 Eingänge (Gewichte w_1, w_2)

Ansatz 1: Fehlergebirge über der w_1-w_2-Ebene vollständig absuchen (Minimumsuche)

```
Emin = very big number
for w1 = -2.0 to 2.0 step 0.1          // Eing. 1
    for w2 = -2.0 to 2.0 step 0.1      // Eing. 2
        ...
            E = 0
            foreach Trainingmuster p = 1...4
                Ep berechnen
                E = E + Ep
            end foreach
            if E < Emin then
                Emin = E
                W1min = w1 , W2min = w2
            end if
        end for
    end for
```

Diskussion: Mit e : Anzahl der Eingänge

n : Anzahl der Abtastschritte / Gewichte

p : Anzahl der Trainingsmuster

$\Rightarrow$ Anzahl der Fehlerberechnungen $= p \cdot n^{e}$

(n^{e}: Anzahl der Schleifendurchläufe)

$\Rightarrow$ nicht praktikabel !

Beispiel: Handschriftserkennung im Praktikum

- Eingangsneuron hat 160 Eingänge
- 1500 Trainingsdaten

$\Rightarrow$ pro Eingangsneuron $1500 \cdot n^{160}$ Fehlerberechnungen !

Backpropagation - Lernalgorithmus

Metapher: "Mann im Nebel im Gebirge sucht Tal"

Frage: Wie bekommt man lokal die Richtung des steilsten Abstiegs heraus?

$\Rightarrow$ neg. Gradient

$$-\vec{\nabla} E_p(w_1, \dots, w_n) = -\left(\frac{\partial E_p}{\partial w_1}, \frac{\partial E_p}{\partial w_2}, \dots, \frac{\partial E_p}{\partial w_n}\right)^T$$

Berechnung von $\frac{\partial E_p}{\partial w_i}$

p: Trainingsmusterindex
j: Neuronenindex
i: Eingangsindex

mit den Gleichungen:

$$E_p = (t_{pj} - o_{pj})^2 \qquad (1)$$

$$o_{pj} = f_{act}(net_{pj}) \qquad (2)$$

$$net_{pj} = o_{pi} \cdot w_1 + o_{p2} \cdot w_2 \qquad (3)$$

$$E_p = (t_{pj} - o_{pj})^2 \qquad (1)$$

$$o_{pj} = f_{act}(net_{pj}) \qquad (2)$$

$$net_{pj} = o_{p1} \cdot w_1 + o_{p2} \cdot w_2 \qquad (3)$$

Ableitungen

(1) $$\frac{\partial E_p}{\partial o_{pj}} = -2 \cdot (t_{pj} - o_{pj})$$

(2) $$\frac{\partial o_{pj}}{\partial net_{pj}} = f'(net_{pj})$$

(3) $$\frac{\partial net_{pj}}{\partial w_i} = o_{pi}$$

Kettenregel anwenden

$$\frac{\partial E_p}{\partial o_{pj}} \cdot \frac{\partial o_{pj}}{\partial net_{pj}} \cdot \frac{\partial net_{pj}}{\partial w_i} = \frac{\partial E_p}{\partial w_i}$$

$$\Rightarrow \frac{\partial E_p}{\partial w_i} = -2(t_{pj} - o_{pj}) \cdot f'(net_{pj}) \cdot o_{pi}$$

$\Rightarrow$ Komponenten des Gradienten

Vorgehensweise bei Backpropagation-Lernalgorithmus

1. Muster anlegen

2. Steigung im E_p-Gebirge in Richtung $w_1, w_2, \ldots$ berechnen: (= Komponenten des Gradienten)

$$\frac{\partial E_p}{\partial w_1}, \frac{\partial E_p}{\partial w_2}, \ldots$$

3. Schritt in Richtung des steilsten Abstiegs machen

$$\Delta \vec{w} = \eta \left(-\frac{\partial E_p}{\partial w_1}, -\frac{\partial E_p}{\partial w_2}, -\frac{\partial E_p}{\partial w_3}, \ldots \right)$$

η: vorgebbarer Schrittweitenfaktor

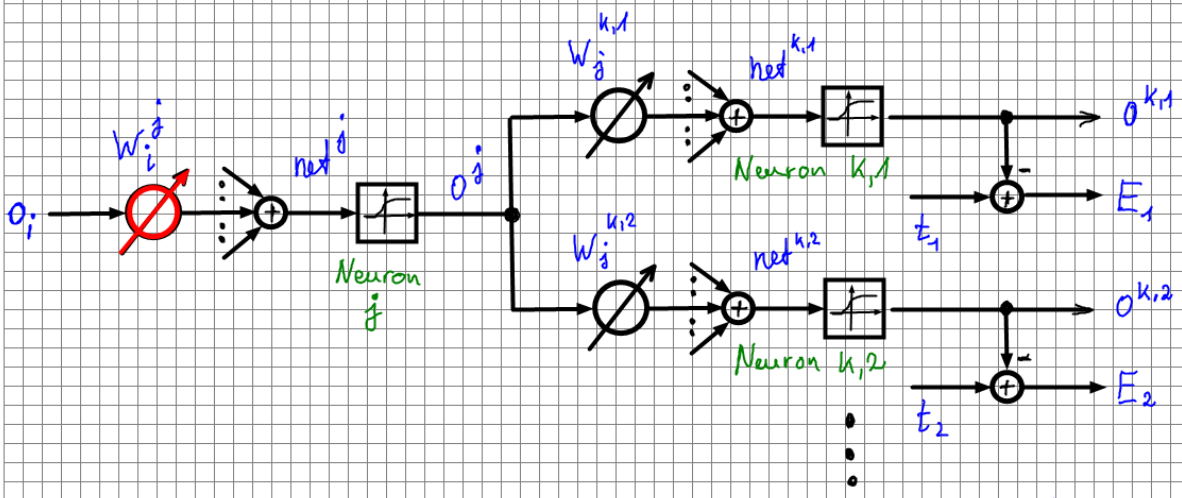
Eigenschaft der Strategie:

Die Schrittweite ist um so größer, je steiler es im Gebirge ist!

$$\vec{\Delta w} = \begin{pmatrix} \Delta w_1 \\ \Delta w_2 \\ \vdots \\ \Delta w_n \end{pmatrix} = \eta \cdot 2 \left(t_{pj} - o_{pj} \right) \cdot f'(net_{pj}) \cdot \begin{pmatrix} o_{p1} \\ o_{p2} \\ \vdots \\ o_{pn} \end{pmatrix}$$

- $\vec{\Delta w}$: Schrittvektor
- t_{pj}: gewünschter Wert (Lehrer)
- o_{pj}: tatsächlich ausgegebener Wert
- f': Ableitung der Aktivierungsfunktion
- net_{pj}: Summe der gew. Eingangssign.
- $\begin{pmatrix} o_{p1} \\ o_{p2} \\ \vdots \\ o_{pn} \end{pmatrix}$: Eingangsvektor

ÜBUNG: Training eines zweilagigen Perzeptrons



Es gelten die Zusammenhänge: insgesamt m Ausgänge

$$E_r = (t_r - o^{k,r})^2$$

$$o^{k,r} = f(net^{k,r})$$

$$net^{k,r} = \ldots + w_j^{k,r} \cdot o^j + \ldots$$

} Layer k, $r = 1, \ldots, m$

$$o^j = f(net^j)$$

$$net^j = \ldots + w_i^j \cdot o_i + \ldots$$

} Neuron j

Frage: Wie muss w_i^j verändert werden, damit $E = \sum_{r=1}^{m} E_r$ kleiner wird.

$$\frac{\partial E}{\partial w_i^j} = \frac{\partial E}{\partial o^j} \cdot \frac{\partial o^j}{\partial w_i^j}$$

$$= \frac{\partial}{\partial o^j}\left(E_1 + E_2 + \dots + E_m\right) \cdot \frac{\partial o^j}{\partial w_i^j}$$

$$= \sum_{r=1}^{m} \frac{\partial E_r}{\partial o^j} \cdot \frac{\partial o^j}{\partial w_i^j}$$

① ②

zu ① $$\frac{\partial E_r}{\partial o^j} = \frac{\partial E}{\partial o^{k,r}} \cdot \frac{\partial o^{k,r}}{\partial net^{k,r}} \cdot \frac{\partial net^{k,r}}{\partial o^j}$$

$$= \underbrace{2\left(t_r - o^{k,r}\right) \cdot f'\left(net^{k,r}\right)}_{\delta^{k,r}\ \text{"Sensitivity"}} \cdot w_j^{k,r}$$

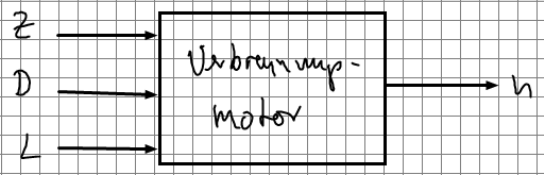
zu ② $$\frac{\partial o^j}{\partial w_i^j} = \frac{\partial o^j}{\partial net^j} \cdot \frac{\partial net^j}{\partial w_i^j} = f'\left(net^j\right) \cdot o_i$$

also insgesamt

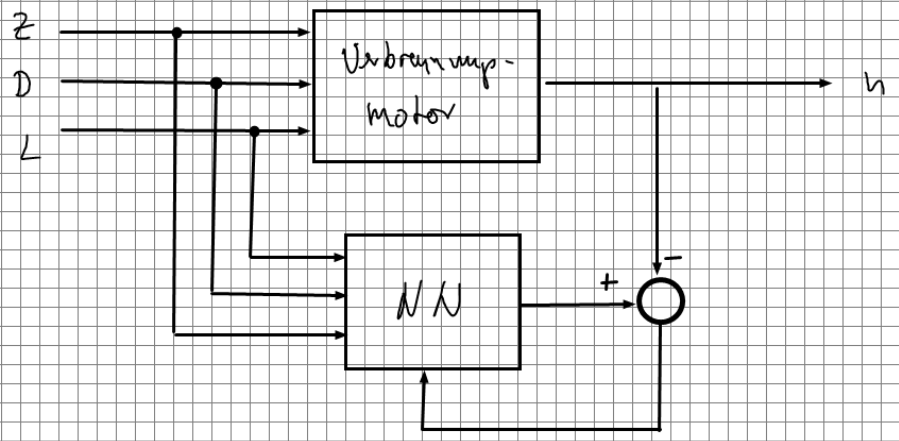
$$\boxed{\frac{\partial E}{\partial w_i^j} = f'\left(net^j\right) \cdot o_i \cdot \sum_{r=1}^{m} \delta^{k,r} \cdot w_j^{k,r}}$$

Erlernen von Prozesskennfeldern

Beispiel: Das Kennfeld eines Verbrennungsmotors soll bestimmt werden.



Z : Zündwinkel
D : Drosselklappenwinkel
L : Lastmoment
n : Drehzahl

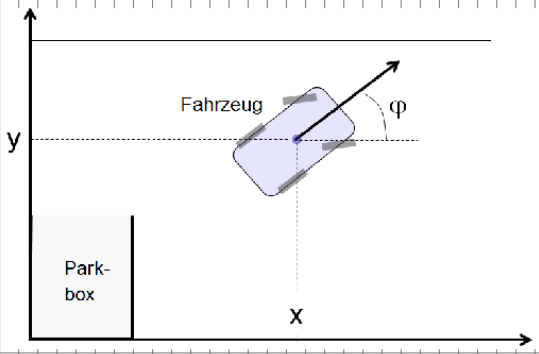


⟹ Das trainierte NN verhält sich wie der Verbrennungsmotor.

Einsatzfeld : Simulation

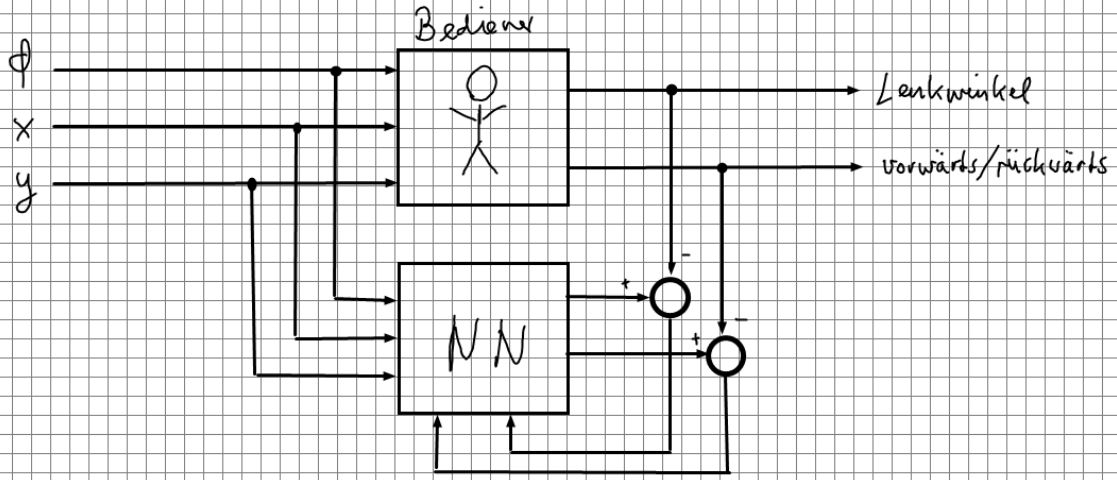
Erlernen von Bedienerverhalten

Beispiel: Anlernen eines Einparkassistenten



Ein erfahrener Fahrer parkt das Fahrzeug aus verschiedenen Posen (x, y, φ) ein.

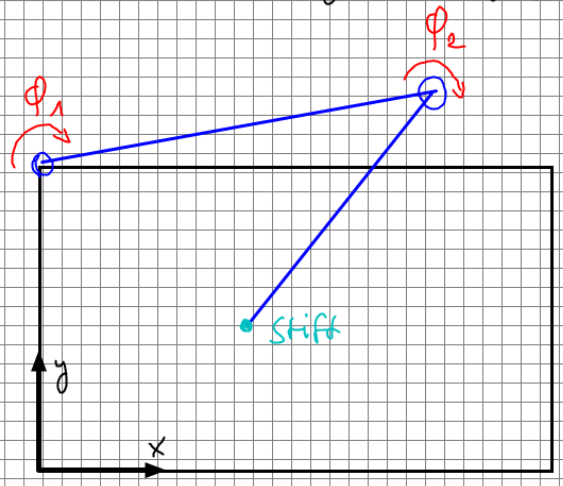
⇒ Trainingsdaten für NN



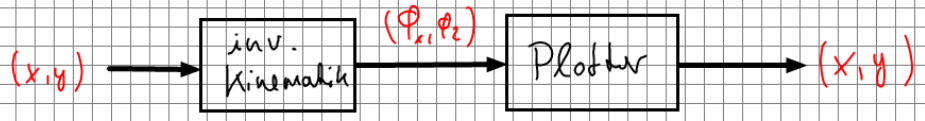
Das trainierte Netz (testen!) parkt anstelle des Fahrers das Fahrzeug ein.

Erlernen inverser Prozess Kennfelder

Beispiel: inverse Kinematik eines 2-Gelenk-Plotters

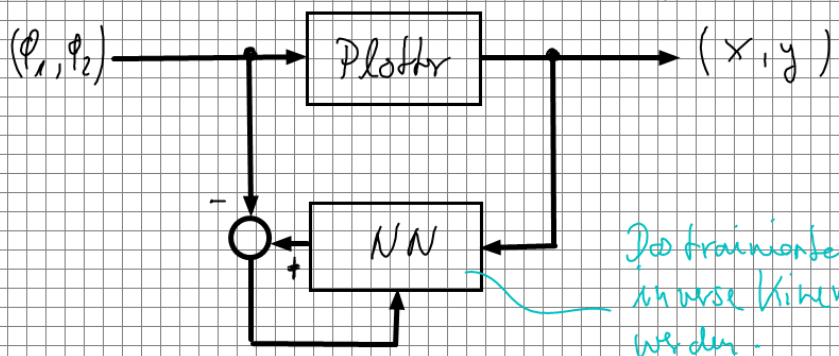


Für die Ansteuerung des Stifts wird die "inverse Kinematik" benötigt.



Training:

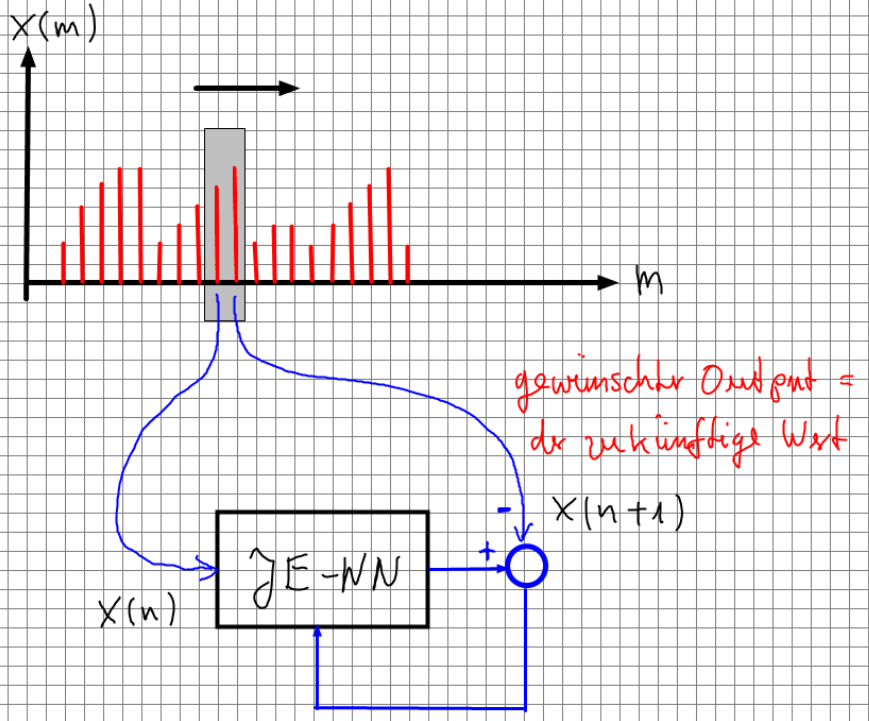
Der Plotter wird in verschiedene Stiftpositionen gebracht.
Die entsprechenden Koordinaten (x,y) sind der Trainingsinput.
Die Gelenkwinkel φ_1, φ_2 sind der gewünschte Output.



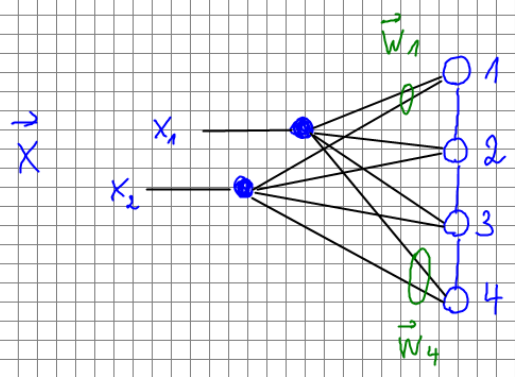
Das trainierte Netz kann als inverse Kinematik verwendet werden.

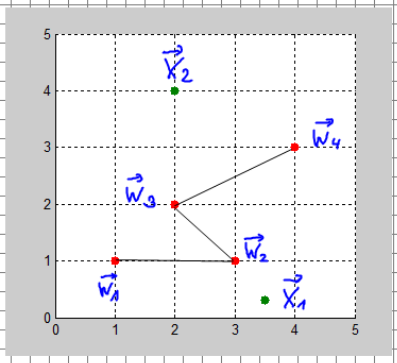
Beispiel: Prognose einer Zeitreihe mit einem Jordan - Elman - Netz

Training



ÜBUNG: Kohonen-Netz





Gegeben sind zwei Eingabewerte:

$\vec{x}_1 = (3.5,\ 0.3)$

$\vec{x}_2 = (2.0\ \ 4.0)$

und ein Nachbarschaftsnetz (eindimensional) mit den initialen Gewichten:

$\vec{w}_1 = (1,\ 1)$

$\vec{w}_2 = (3,\ 1)$

$\vec{w}_3 = (2,\ 1)$

$\vec{w}_4 = (4,\ 3)$

$\sigma = 2,\ \ \eta = 0.5$

a) $\vec{x}_1 = (x_{1_1}, x_{1_2})$ liegt am Gewinner neuron $\Rightarrow$ 2 (durch scharfe Hinsehen)

Topolog. Distanzen

$d_{12} = 1,\ d_{22} = 0,\ d_{32} = 1,\ d_{42} = 2$

$$\tau_{12} = e^{-\frac{d_{12}^2}{2\sigma^2}} = e^{-\frac{1^2}{8}} = 0.88$$

$$\tau_{22} = e^{-\frac{d_{22}^2}{2\sigma^2}} = e^{-0} = 1$$

$$\tau_{23} = \ldots = e^{-\frac{1^2}{8}} = 0.88$$

$$\tau_{24} = \ldots = e^{-\frac{2^2}{8}} = 0.61$$

Iteration :

$$\vec{w}_j(t+1) = \vec{w}_j(t) + \eta \cdot \tau_{ij} \cdot \left(\vec{x} - \vec{w}_j(t)\right)$$

$$\vec{w}_1(t+1) = (1,1) + 0.5 \cdot 0.88 \cdot \left[(3.5, 0.3) - (1,1)\right]$$

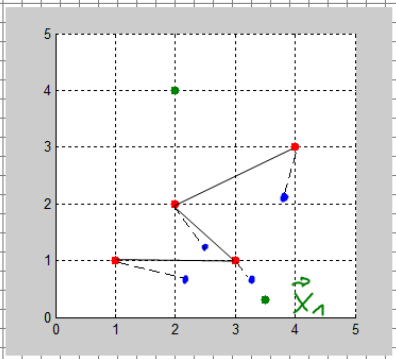
$$= \underline{(2.1,\ 0.692)}$$

$$\vec{w}_2(t+1) = (3,1) + 0.5 \cdot 1.0 \cdot \left[(3.5, 0.3) - (3,1)\right]$$

$$= \underline{(3.25,\ 0.65)}$$

$$\vdots$$

$$\underline{\vec{w}_3(t+1) = (2.66,\ 1.25)} \qquad \underline{\vec{w}_4(t+1) = (3.85,\ 2.18)}$$



b) $\vec{x}_2 = (x_{21}, x_{22})$ liegt am Gewinnerneuron $\Rightarrow$ 4 (durch scharfes Hinsehen)

Topologische Distanzen

$d_{14} = 3$, $d_{24} = 2$, $d_{34} = 1$, $d_{44} = 0$

$\tau_{14} = e^{-\frac{d_{14}^2}{2\sigma^2}} = e^{-\frac{3^2}{8}} = 0.32$

$\tau_{24} = e^{-\frac{d_{24}^2}{2\sigma^2}} = e^{-\frac{2^2}{8}} = 0.61$

$\tau_{34} = \ldots = e^{-\frac{1^2}{8}} = 0.88$

$\tau_{44} = \ldots = e^{-0} = 1.0$

Adaptation : $\vec{w}_j(t+1) = \vec{w}_j(t) + \eta \cdot \tau_{ij} \cdot \left(\vec{x} - \vec{w}_j(t)\right)$

$\vec{w}_1(t+1) = (2.1, 0.692) + 0.5 \cdot 0.32 \cdot [(2,4) - (2.1, 0.692)]$

$= (2.09, 1.23)$

...