Fachhochschule
Münster University of
Applied Sciences

Adaptive Systeme

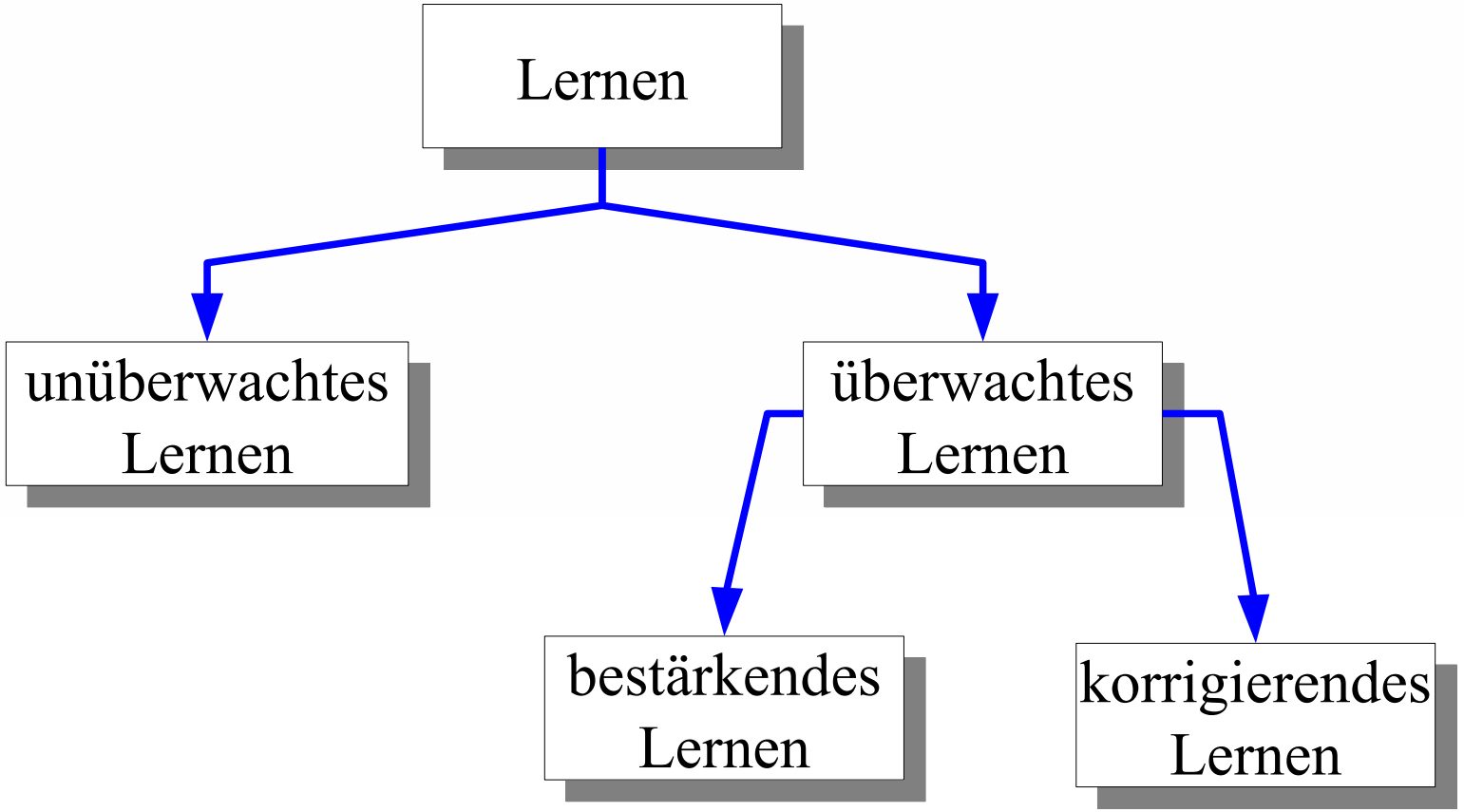
Neuronale Netze: Neuronen, Perzeptron und Adaline

Prof. Dr. rer. nat. Nikolaus Wulff

Neuronale Netze

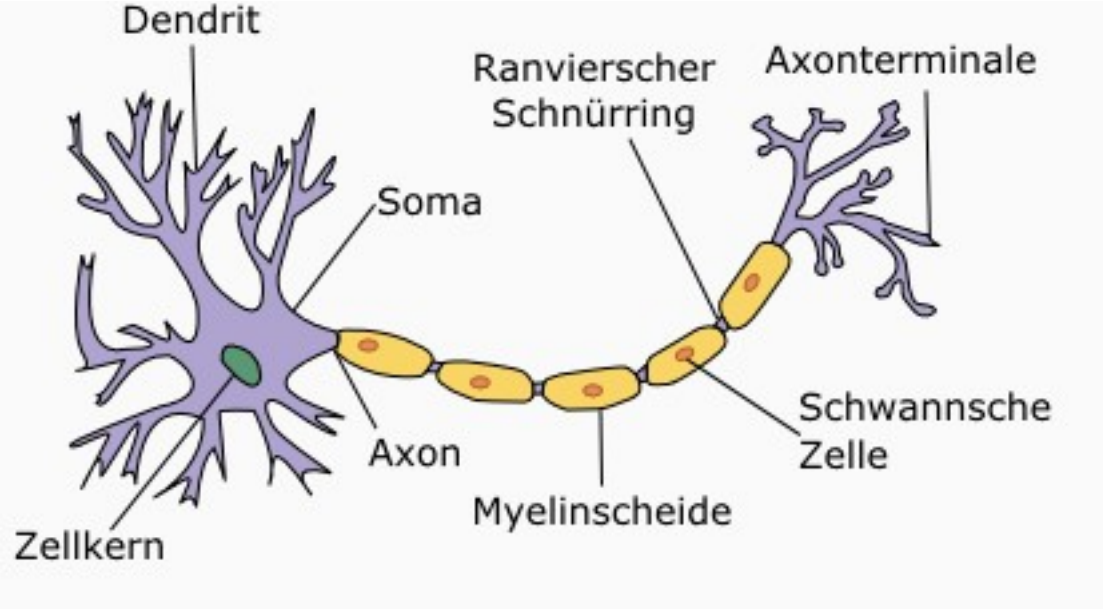
- Das (menschliche) Gehirn ist ein Musterbeispiel für ein adaptives System, dass sich an seine Umwelt anpasst und selbstständig lernt.
- Die Evolution hat das Gehirn flexibel angelegt, um auf (teilweise unvorhergesehene) Situationen in einem gewissen Rahmen flexibel und angemessen reagieren zu können.
- Durch geeignete Stimuli verändert das Gehirn seine Vernetzung der Nervenzellen und lernt neue (Verhaltens)muster.
- Es „programmiert sich quasi“ selbst, ein entscheidener Evolutions- und Selektionsvorteil.

Einordnung des Lernprozesses



- Überwachtes Lernen erfordert einen Trainer, der entweder „die Wahrheit“ kennt oder aber „belohnt“.

Die Nervenzelle



Die Abbildung zeigt den Aufbau einer Nervenzelle.

Die drei biologischen Bausteine Dendritenbaum, Zellkörper und Axon können (sehr stark!) vereinfacht als Eingabe-, Verarbeitungs- und Ausgabeeinheit in der Informatik modelliert werden.

Das Neuron

Die Eingangssignale an den Synapsen des Dendrit führen, beim Überschreiten einer gewissen Schwelle, zu einem Feuern des Axons.

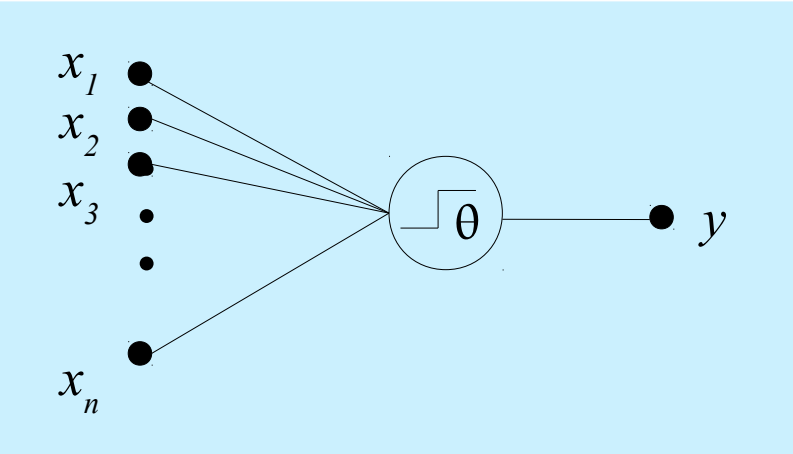
1943 führten W. McCulloch und W. Pitts das Neuron als „logisches Schwellwertelement“ mit mehreren Eingängen und einem Ausgang in der Informatik ein.

Es verarbeitet als Eingangs- und Ausgangsignale boolsche Variable und realisiert eine Abbildung

$$f: \{0,1\}^n \rightarrow \{0,1\}.$$

Die McCulloch-Pitts-Zellen konnten die UND, ODER und NOT Verknüpfungen realisieren und bildeten eine vollständige Basis der boolschen Algebra.

Modell eines Neuron



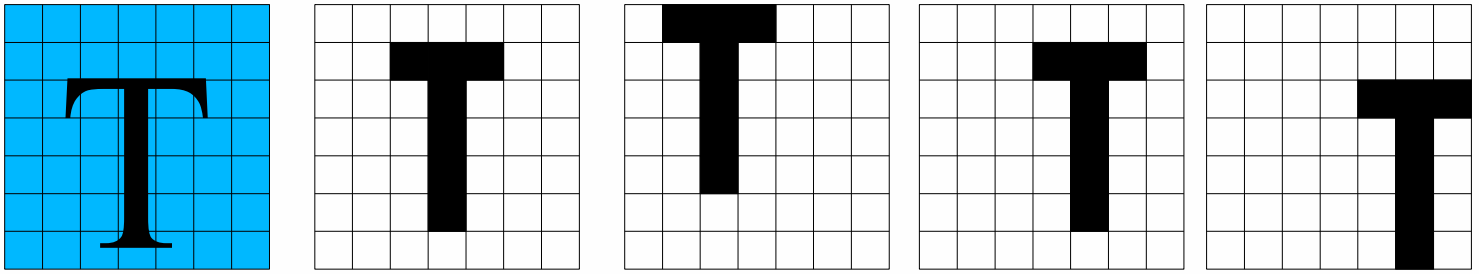
$$y = \sigma\left(\sum_{k=1}^{n} w_k x_k - \theta\right)$$

Die n binären Eingangssignale $x_k \in \{0,1\}$ werden vom Neuron aufsummiert.

Sobald die (gewichtete) Summe der Eingangssignale x_k eine gewisse Schwelle θ übersteigt, „feuert“ das Axon und das Ausgangssignal y geht von 0 auf 1, realisiert durch die Transferfunktion σ.

Mustererkennung

- Neuronen erkennen (trainierte) Muster, z.B. Schrift:

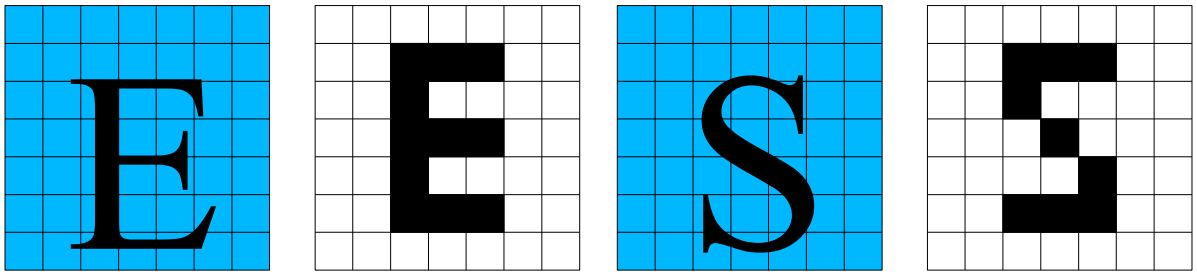


$T_1=(0,0,0,0,0,0,0,0,0,1,1,1,0,0\ 0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0)^T$

$T_2=(0,1,1,1,0,0\ 0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)^T$

$T_3=(0,0,0,0,0,0,0,0,0,0,1,1,1,0,0\ 0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0)^T$

- Damit das Neuron alle Möglichkeiten eines „T“ richtig erkennt, müssen alle „T“ Muster klassifiziert und erlernt und von Anderen unterschieden werden.



Durch Rasterisierung zum Vektor in $\{0,1\}^{49}$.

Die Transferfunktion

Eine einfache Transferfunktion, die das Feuern einer Nervenzelle darstellt und zugleich $\mathbb{R}$ auf die Menge $\{0,1\}$ abbildet ist die Theta-Funktion:

$$\sigma_\Theta = \Theta(x) = \begin{cases} 1 & x \geq 0 \\ 0 & sonst \end{cases}$$

Da die *Heavyside/Stufenfunktion* im Ursprung weder differenzierbar noch stetig ist, wird häufig eine stetig differenzierbare Funktion gewählt, die nach einer entsprechenden Skalierung einen ähnlichen Verlauf zeigt wie die Theta-Funktion, d. h. ungefähr 0 für negative und ungefähr 1 für positive Argumente ist.

Weitere Transferfunktionen

Stückweise lineare Funktion

$$\sigma_c(x) = \begin{cases} 1 & x > c \\ 1/(2c)(x+c) & -c \le x \le c \\ 0 & x < -c \end{cases}$$

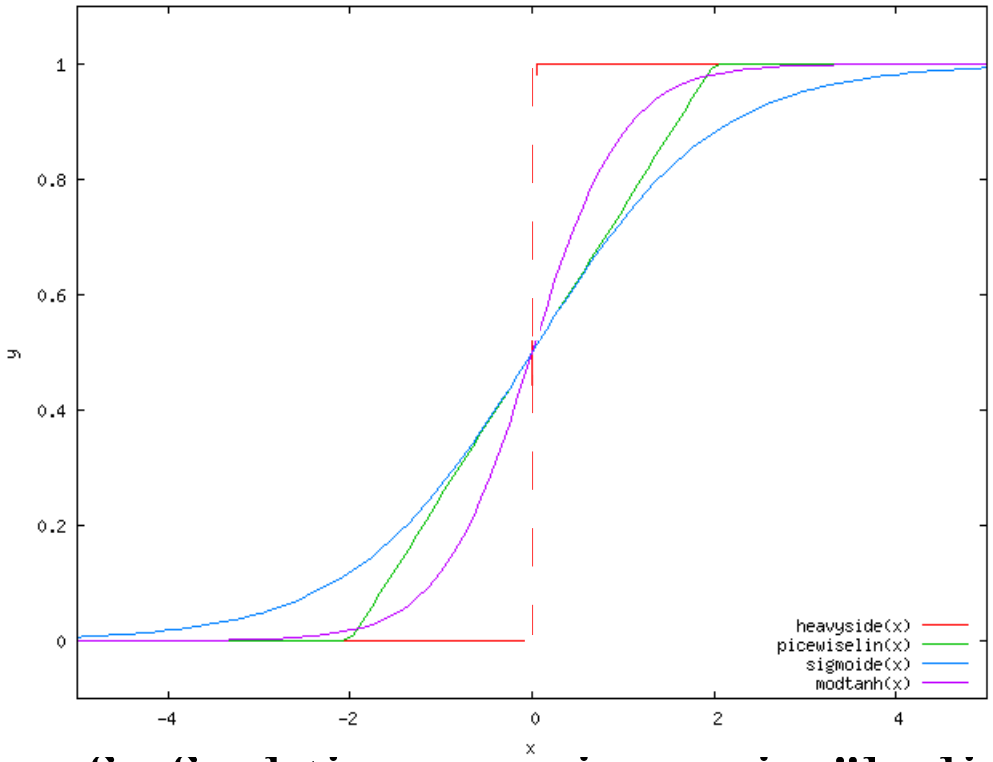
Fermi /sigmoide Funktion

$$\sigma_{fermi}(x) = \frac{1}{1+\exp(-x)}$$

Modified Hyperbolicus Tangens

$$\sigma_{\tanh}(x) = (1+\tanh(x))/2$$

Graph der Transferfunktionen



Alle Transferfunktionen zeigen ein ähnliches Verhalten hinsichtlich ihres „Aktivierungspotentials“.

Lediglich die - in den Lernverfahren benötigten -, Ableitungen $\sigma'(x)$ sind unterschiedlich.

AND und OR per Neuron

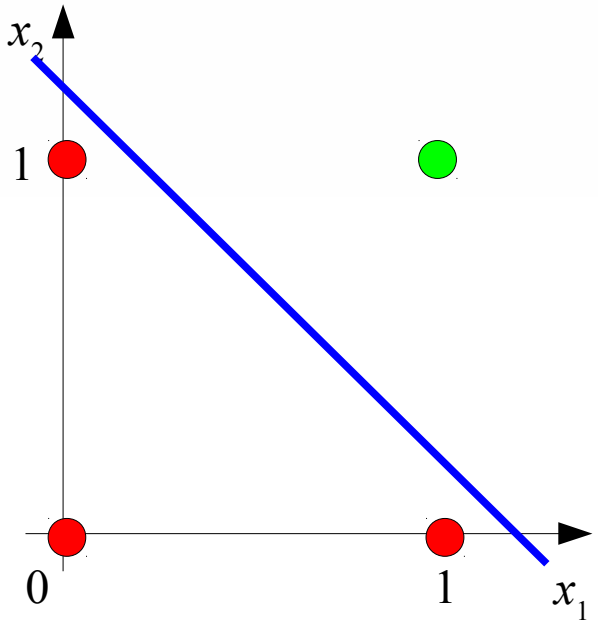
Die Gewichte w_1 und w_2 und das Bias θ zur Realisierung des AND-Gatters, lassen sich durch Lösen des nichlinearen Gleichungssystems für die Schalttabelle finden:

$$\sigma(w_1 \cdot 0 + w_2 \cdot 0 - \theta) = 0 \quad \Rightarrow \quad \theta > 0$$
$$\sigma(w_1 \cdot 0 + w_2 \cdot 1 - \theta) = 0 \quad \Rightarrow \quad w_2 < \theta$$
$$\sigma(w_1 \cdot 1 + w_2 \cdot 0 - \theta) = 0 \quad \Rightarrow \quad w_1 < \theta$$
$$\sigma(w_1 \cdot 1 + w_2 \cdot 1 - \theta) = 1 \quad \Rightarrow \quad w_1 + w_2 > \theta$$

Dieses System hat unendlich viele mögliche Lösungen, eine ist $w_1 = w_2 = ½$ und $\theta = ¾$.

Graphische Veranschaulichung

- Die vier möglichen Eingangsvektoren $(0,0)^T$,$(0,1)^T$, $(1,0)^T$ und $(1,1)^T$ lassen sich für die AND Operation durch eine Gerade g „linear separieren“.



$$g: w_1 \cdot x_1 + w_2 \cdot x_2 = \theta$$

$$\vec{w}^T \cdot \vec{x} = \theta$$

- g ist eine Hyperebene im 2-dimensionalen Raum.

Vereinfachung der Schreibweise

Für viele Berechnungen ist es lästig das Bias θ in den Formlen explizit berücksichtigen zu müssen.

Werden die Vektoren ***x*** und ***w*** um ein *0*-tes Element mit dem konstanten Wert $1 = x_0$ erweitert, so entfällt das Bias mit $w_0 = -\theta$:

$$y = \sigma\left(\sum_{k=1}^{n} w_k x_k - \theta\right) \equiv \sigma\left(\sum_{k=0}^{n} w_k x_k\right)$$

Bzw. in vektorieller Kurzschreibweise:

$$y = \sigma\left(\vec{w}^T \vec{x}\right)$$

Lernalgorithmen

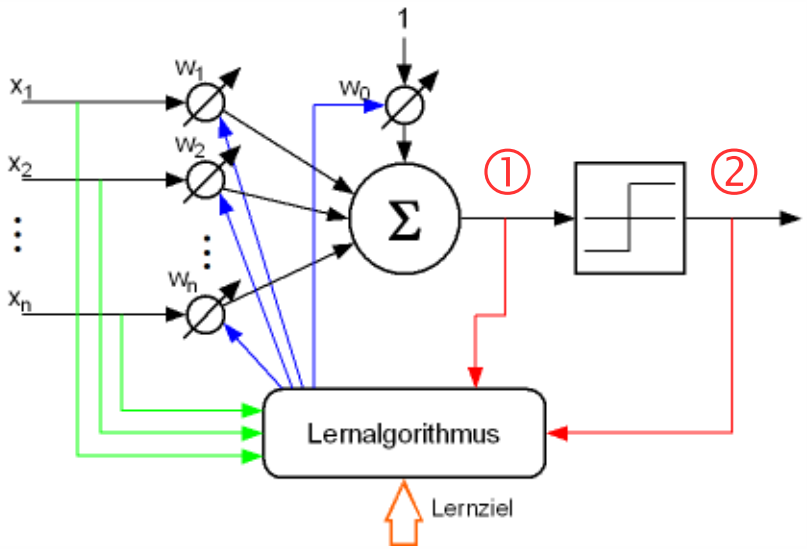
Der Psychologe D. O. Hebb formulierte 1949 die Lernregel, dass Neuronen, die häufig miteinander in Beziehung treten physisch enger verbunden sind „***what fires together, wires together***“:

$$\Delta w_k = \mu \, y(\vec{x}) \cdot x_k$$

Das Perzeptron (F. Rosenblatt, 1958) und Adaline (B. Widrow & M. E. Hoff, 1960) waren mit variablen Gewichten versehene adaptive, lernfähige Neuronen.

Rosenblatt gab einen Lernalgorithmus an, mit dessen Hilfe die Gewichte *w* trainiert werden konnten, so dass das Perzeptron alle mit dem Modell repräsentierbaren Lösungen selbständig erlernen konnte.

Überwachtes Lernen



① Adaline

② Perzeptron

Im Training wird das Neuronale Netz mit zu erkennenden Binärmustern stimuliert.

Ein Lernalgorithmus vergleicht die Antworten des Netz, mit den Erwünschten des Trainingsatzes und verändert die Gewichte so lange, bis alle Musterklassen hinreichend genau erkannt werden.

Perzeptron Lernalgorithmus

Die Hebb'sche Lernregel wurde für das Perzeptron abgewandelt, in dem zu gegebenem Eingangsvektor $\boldsymbol{x}$ die Differenz zwischen Ist- (y) und Sollwert (d) eingeht:

$$\Delta w_k = \mu (d - y) \cdot x_k \qquad \mu > 0$$

Es findet nur dann eine Korrektur mit Lernschritt-weite μ statt, wenn eine Differenz $e=(d-y)$ vorliegt.

Da nur mit 0 und 1 gerechnet wird hat e die Werte 0 oder ± 1 und nur die Gewichte w_k bei denen ex_k von 0 verschieden ist werden um $\Delta w = \pm \mu$ verändert.

Perzeptron optimieren

- Die Bestimmung der Gewichte ***w*** lässt sich auch als klassische Optimierungsaufgabe deuten:

$$\chi^2 = \tfrac{1}{2} \sum_m \left(\sigma\left(\vec{w}^T \vec{x}_m \right) - d_m \right)^2 = \text{Minimal}$$

- Summiert wird über alle *m* Zielmustervektoren und führt zum (nichtlinearen) Gleichungssystem:

$$\nabla_{\vec{w}} \chi^2 = \sum_m \left(y_m - d_m \right) \sigma'\left(\vec{w}^T \vec{x}_m \right) \underbrace{\nabla_{\vec{w}} \left(\vec{w}^T \vec{x}_m \right)}_{\vec{x}_m} = \vec{0}$$

Bemerkung:

– Erforderlich ist eine ableitbare Transferfunktion σ.

Optimierung als Skalarprodukt

- Der Vektor $\vec{x}_m$ gibt für ein iteratives Newton Verfahren à la „Abstieg im Gebirge“ die Richtung der Änderung $\Delta \vec{w}_m$ vor.

$$\Delta \vec{W} \simeq \sum_m \sigma'_m \left(y_m - d_m \right) \vec{x}_m$$

- Die Summe über m kann als Skalarprodukt im m-dimensionalen (Muster)Raum gedeutet werden:

$$\Delta \vec{W} = \left(\sigma' \left(Y - D \right) \right)^T \cdot X$$

- Die Iteration ist beendet falls der Fehler $E = Y - D$ Null oder orthogonal zum Mustervektorensatz X ist.

Konvergenz des Perzeptrons

Für den t-ten Lernschritt eines einzelnen Perzeptrons gilt mit $e=\pm 1$: $w_k(t)=w_k(t-1)+\mu\, e\, x_k$

bzw. $\vec{w}_t=\vec{w}_{t-1}+\mu\, e\, \vec{x}$

und somit gilt mit der Schrittweite $\mu=1/\left\|\vec{x}\right\|$:

$$\vec{w}_t^2=\vec{w}_{t-1}^2+\mu^2 e^2 \vec{x}^2+2\mu \underbrace{e\left(\vec{w}_{t-1}^T \vec{x}\right)}_{\leq 0}$$

$$\leq \vec{w}_{t-1}^2+\mu^2 \vec{x}^2=\vec{w}_{t-1}^2+1$$

und als eine obere Schranke gilt: $\left\|w_t\right\|\leq\sqrt{\left\|w_0\right\|^2+t}$

Konvergenz des Perzeptrons (2)

Sofern überhaupt ein optimaler Vektor ω existiert, gibt es ein δ>0 so dass entweder gilt:

$$0 = d(\vec{x}) = \sigma(\vec{\omega}^T \vec{x}) \;\Leftrightarrow\; \vec{\omega}^T \vec{x} < -\delta$$

oder aber

$$1 = d(\vec{x}) = \sigma(\vec{\omega}^T \vec{x}) \;\Leftrightarrow\; \vec{\omega}^T \vec{x} > \delta$$

und somit gilt bei jedem Änderungsschritt für das Sklarprodukt aus der aktuellen Nährung und dem Optimum:

$$\begin{aligned} \vec{\omega}^T \vec{w}_t &= \vec{\omega}^T \vec{w}_{t-1} + \mu\, e\, \vec{\omega}^T \vec{x} \\ &\geq \vec{\omega}^T \vec{w}_{t-1} + \mu\, \delta \\ &\geq \vec{\omega}^T \vec{w}_0 + t\, \mu\, \delta \end{aligned}$$

Konvergenz des Perzeptrons (3)

Letzte Gleichung besagt, dass $\vec{\omega}^T \vec{w}_t$ linear mit der Anzahl der Iterationen anwächst. Da alle Vektoren $\boldsymbol{x}$ maximal lediglich n Einsen enthalten können, ist für alle Vektoren die Norm $\|\boldsymbol{x}\| \leq n$, so dass gilt:

$$\vec{\omega}^T \vec{w}_0 + t\,\delta / n \leq \vec{\omega}^T \vec{w}_t \leq \left\|\vec{\omega}\right\| \cdot \left\|\vec{w}_t\right\|$$

und mit der oberen Schranke der letzten Folie folgt:

$$\vec{\omega}^T \vec{w}_0 + t\,\delta / n \leq \vec{\omega}^T \vec{w}_t \leq \left\|\vec{\omega}\right\| \cdot \left\|\vec{w}_t\right\|$$

$$\vec{\omega}^T \vec{w}_0 + t\,\delta / n \leq \left\|\vec{\omega}\right\| \cdot \sqrt{\left\|w_0\right\|^2 + t}$$

Letzteres kann nur erfüllt werden, wenn nach endlich vielen Schritten $t \leq t_{max}$ der Algorithmus terminiert.

Übung/Praktikum

- Stellen Sie das Gleichungssytem für das Neuron eines OR-Gatters auf und finden Sie eine Lösung.
- Zeigen Sie, dass für das XOR-Gatter keine Lösung bestehend aus einem Neuron existieren kann.
- Implementieren Sie einen Perzeptron Lern-algorithmus, um ein AND- und OR-Gatter zu trainieren.
  - Ohne die Gewichte vorher optimiert zu haben!
  - Wie viele Iterationen benötigen Sie?
  - Wie hängt die Konvergenz vom Lernfaktors μ ab?
- Lassen sich AND und OR auch mit der stückweise linearen Transferfunktion realisieren?