

Spezialfälle von Funktionen mehrerer Veränderlicher.

$$\textcircled{1} \quad f: \mathbb{R} \ni [a, b] \longrightarrow \mathbb{R}^n$$

$$\textcircled{2} \quad f: \mathbb{R}^n \ni D \longrightarrow \mathbb{R}$$

an $\textcircled{2}$

$$\text{Def. } Df(a) = \left(\frac{\partial}{\partial x_1} f(a), \dots, \frac{\partial}{\partial x_n} f(a) \right) \\ = \text{grad } f(a) \quad \underline{\text{Gradient von } f \text{ in } a}$$

$$Df(a) \begin{pmatrix} u_1 \\ \vdots \\ u_n \end{pmatrix} = \sum_{i=1}^n u_i \frac{\partial}{\partial x_i} f(a)$$

\mathbb{R}^n Einl. Skalarprodukt

$$\langle x, y \rangle = x \cdot y = \sum_{i=1}^n x_i y_i$$

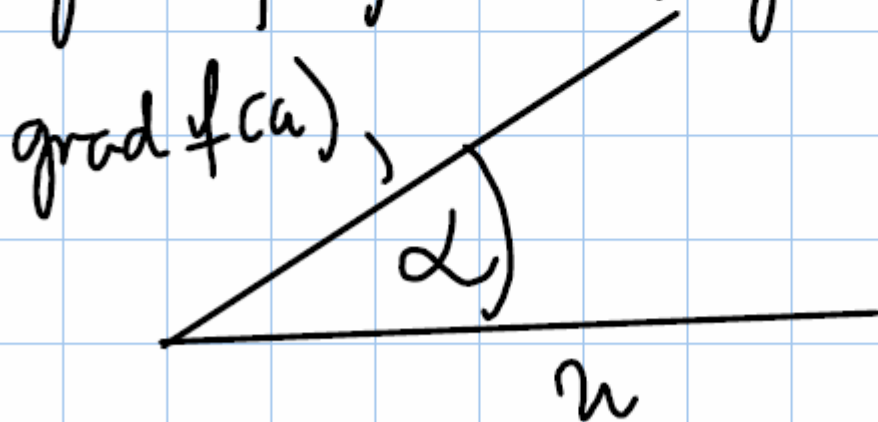
$$Df(a)(u) = \langle Df(a), u \rangle = \langle \text{grad } f(a), u \rangle \\ = \text{grad } f(a) \cdot u$$

Bem. Cauchy-Schwarzsche Ungl. \Rightarrow

$$\frac{\sigma(x, y)}{\|x\| \|y\|} = \cos(\alpha_{x, y})$$

$$> \sigma(x, y) = \|x\| \|y\| \cos(\alpha_{x, y})$$

$$> \text{grad } f(a) \cdot u = \|\text{grad } f(a)\| \|u\| \cos(\alpha)$$



$$\frac{\partial}{\partial u} f(a) = L(u) = Df(a)(u)$$

$$\begin{aligned} \frac{\partial}{\partial u} f(a) &= Df(a)(u) = \text{grad} f(a) \cdot u \\ &= \|\text{grad} f(a)\| \|u\| \cos(\alpha) \end{aligned}$$

o.E.d.A. $\|u\| = 1. \quad \rightarrow$

$$\frac{\partial}{\partial u} f(a) = \|\text{grad} f(a)\| \cos \alpha.$$

Die Richtungsabl. $\frac{\partial}{\partial u} f(a)$ hat den größten Wert für $\alpha = 0 \Rightarrow \cos \alpha = 1$

$$\Rightarrow \frac{\partial}{\partial u} f(a) = \|\text{grad} f(a)\|$$

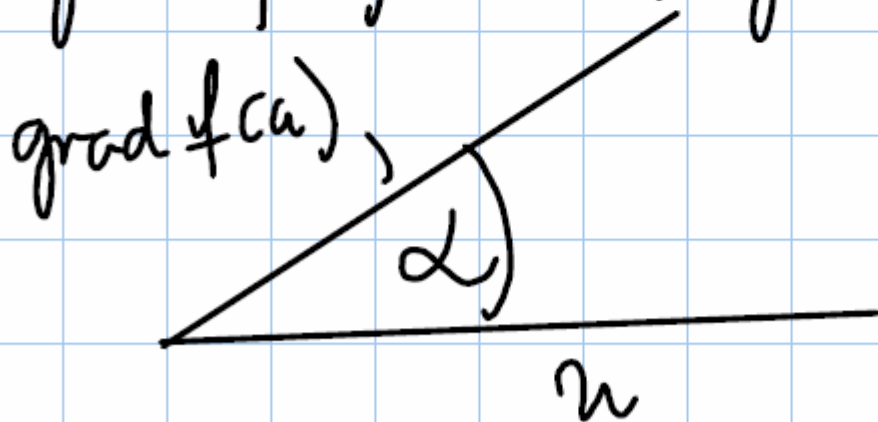
"Der Gradient gibt die Richtung des stärksten Anstiegs an."

Bem. Cauchy-Schwarzsche Ungl. \Rightarrow

$$\frac{\sigma(x, y)}{\|x\| \|y\|} = \cos(\alpha_{x, y})$$

$$> \sigma(x, y) = \|x\| \|y\| \cos(\alpha_{x, y})$$

$$> \text{grad } f(a) \cdot u = \|\text{grad } f(a)\| \|u\| \cos(\alpha)$$

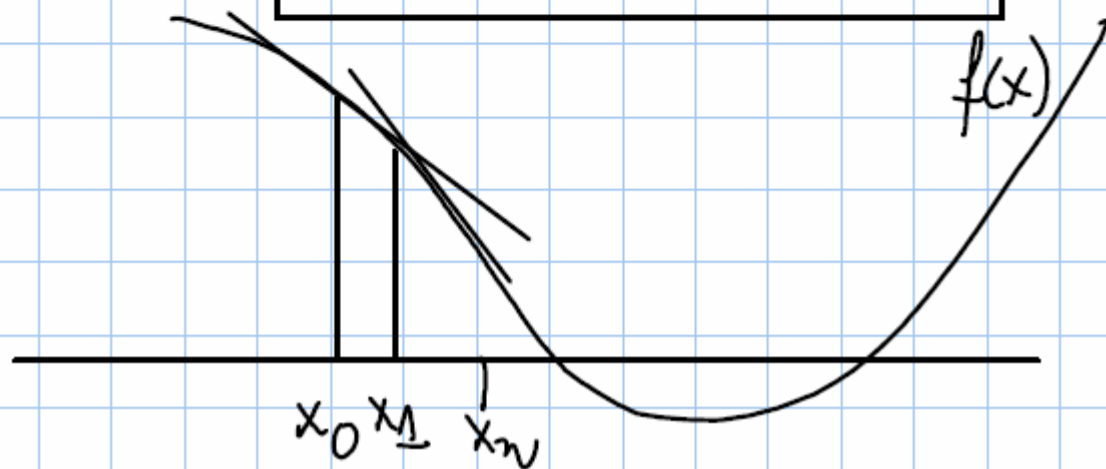


Das Gradientenverfahren zur Bestimmung von Maximum (oder Minimum).

Dies gehend von einem Startwert x_0 bestimmt man in Richtung von $\text{grad } f(x_0)$ (bzw. $-\text{grad } f(x_0)$) einen Punkt $x_1 := x_0 + h \text{ grad } f(x_0)$ mit einer dem Problem angepassten Schrittweite $h > 0$ ($h < 0$). Im Falle $f(x_1) < f(x_0)$ (bzw. $f(x_1) > f(x_0)$) ist man zu weit gegangen. Man versucht es mit halber Schrittweite

$$x_1 = x_0 + \frac{h}{2} \text{ grad } f(x_0).$$

allgemein $x_{n+1} := x_n + h \text{ grad } f(x_n)$

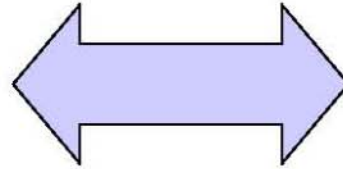


Künstliche Neurale Netze

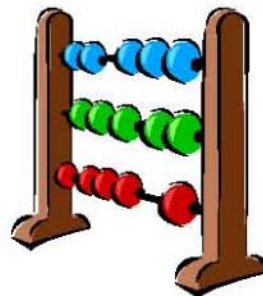
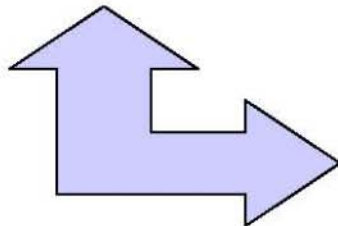
3 Schwerpunktthemen in der Forschung zur Lerntheorie



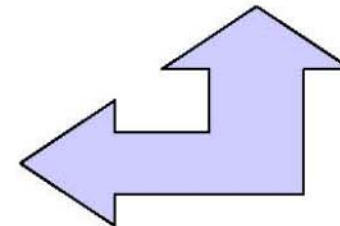
Verstehen des Gehirns
(Physiologie, Psychologie,
Neuroscience, Medizin)



Entwicklung lernender Maschinen
(computer and electronic engineering)



Die Entwicklung maschineller Lernverfahren
(computer and information science)



Maschinelles Lernen (machine learning)

• **Eigenschaften des Gehirns:**

- Erkennen, Klassifikation (pattern recognition)
- hohe Fehlertoleranz (noisy data)
- Lernen von Beispielen
- Generalisierung, Abstraktion
- verteilte Wissenspräsentation
- Komplexität, große Anzahl kleiner Elemente, Ausfalltoleranz

• **Künstliche Neuronale Netze:**

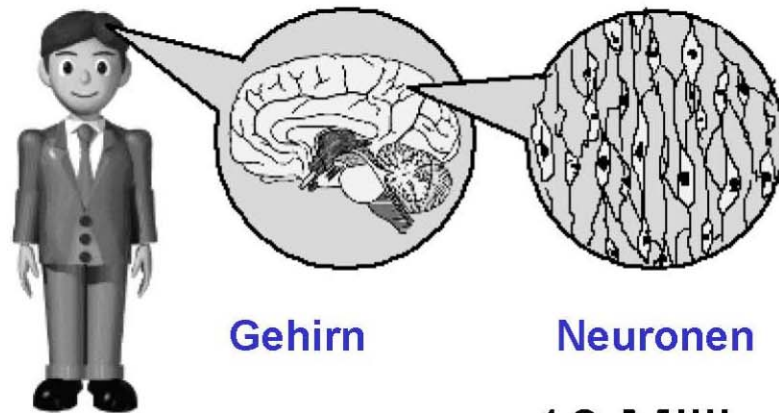
Modelle der Gehirnstrukturen

Modelle der Informationsverarbeitung

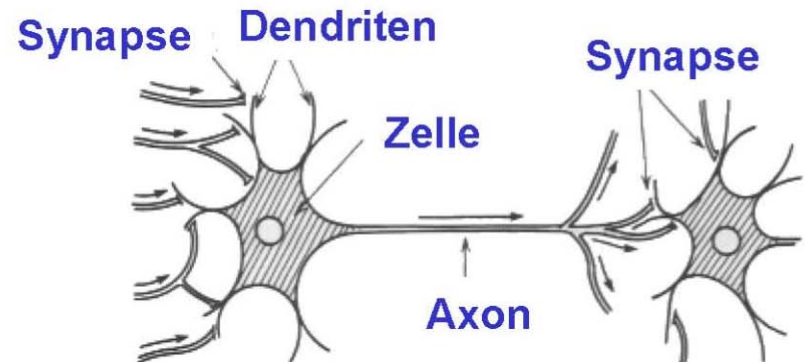
Klassifikatoren: überwachtes Lernen (supervised), unüberwachtes Lernen (unsupervised)

Grundstrukturen des Gehirns

- Structures and activities of our brain have been clarified considerably.
- However, it is not still clear how learning is carried out with a large number of neurons.

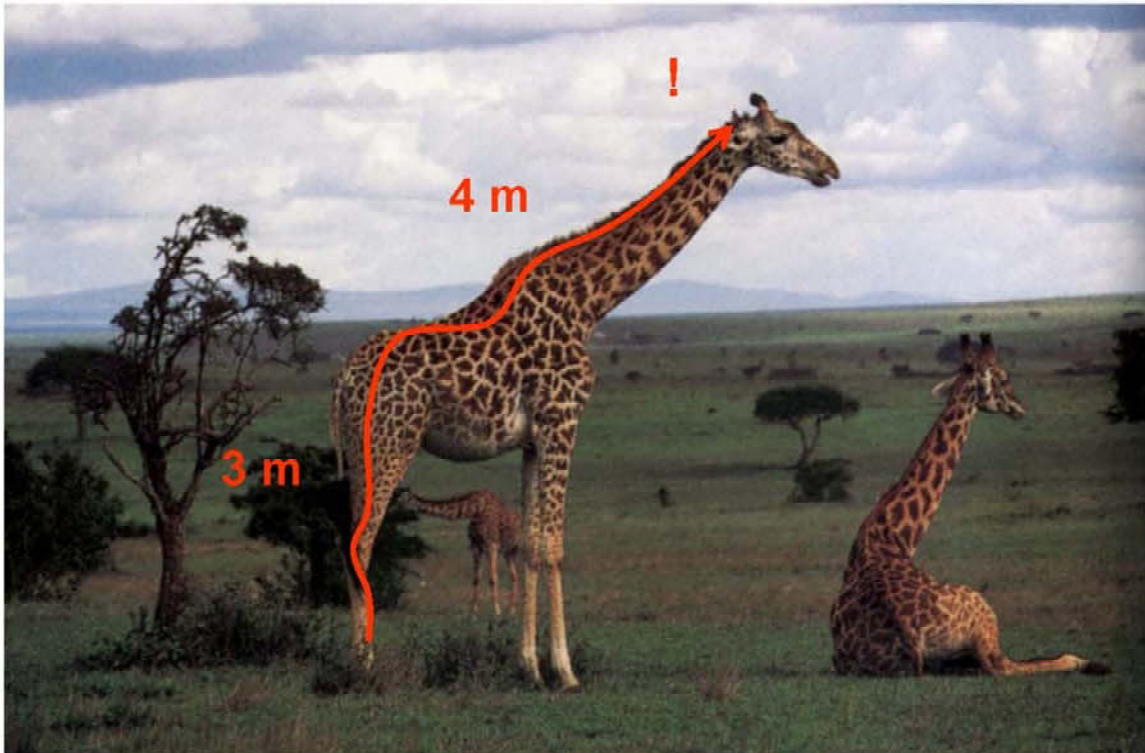


10 Milliarden



Was Elektrokommunikation leisten muss ...

... Signalübertragung

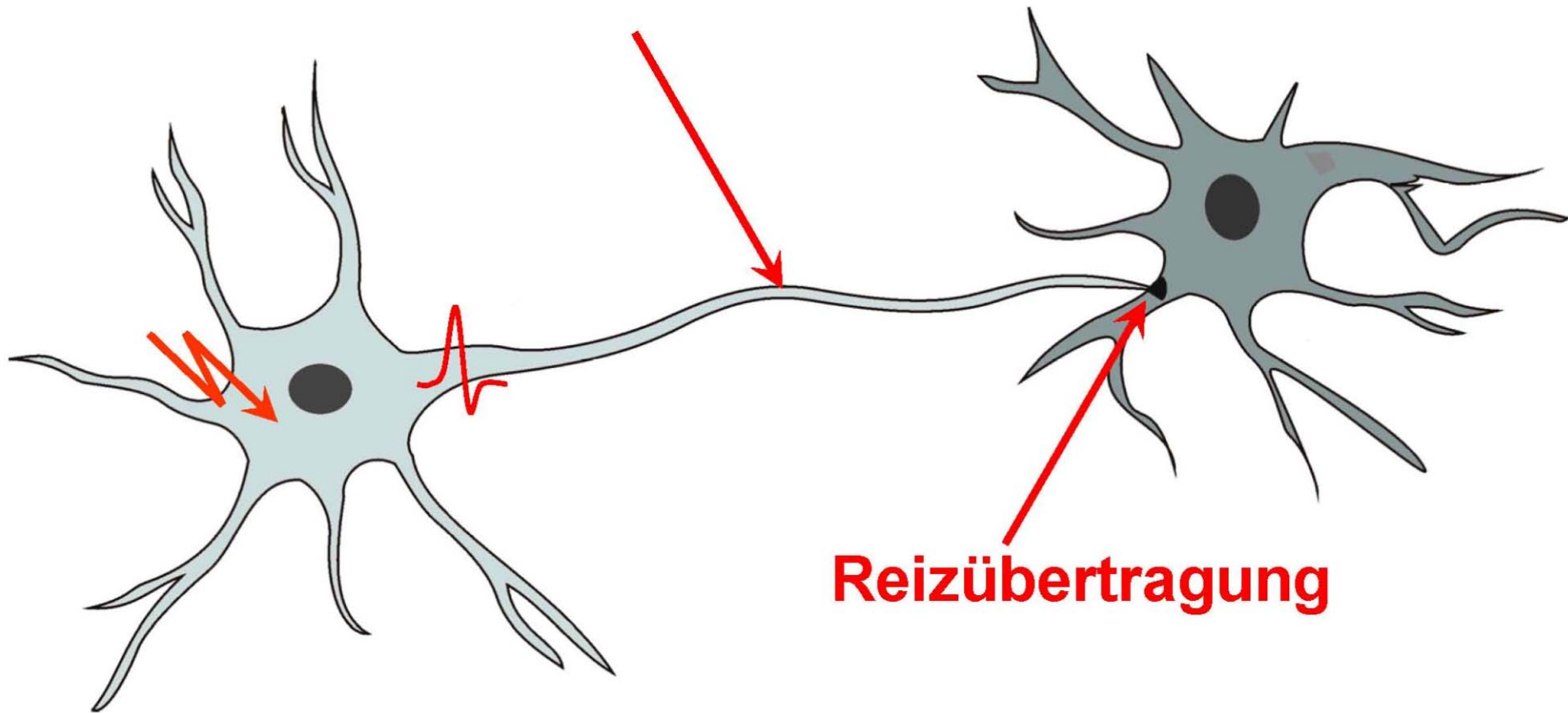


über lange Strecken

schnell

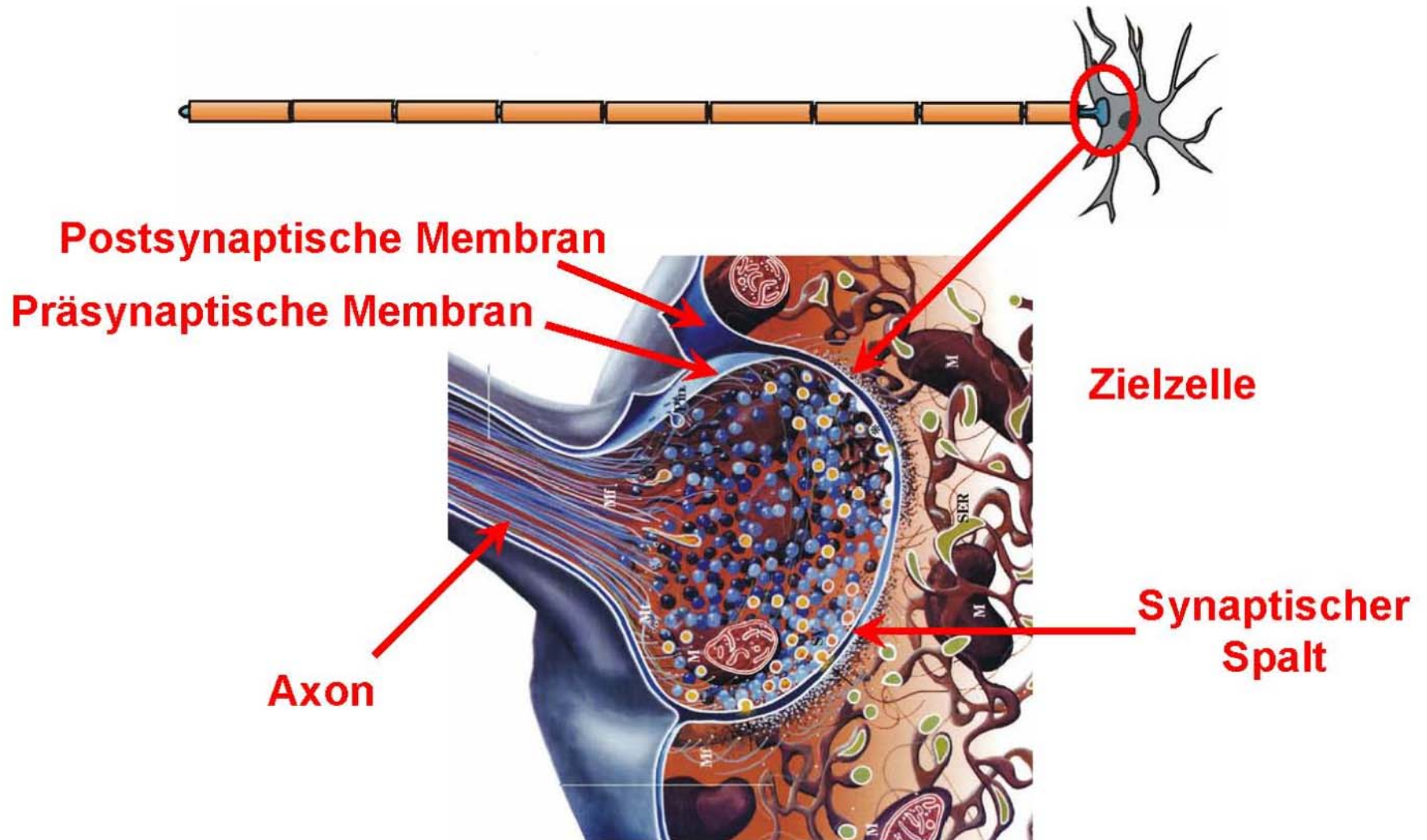
ohne Verlust

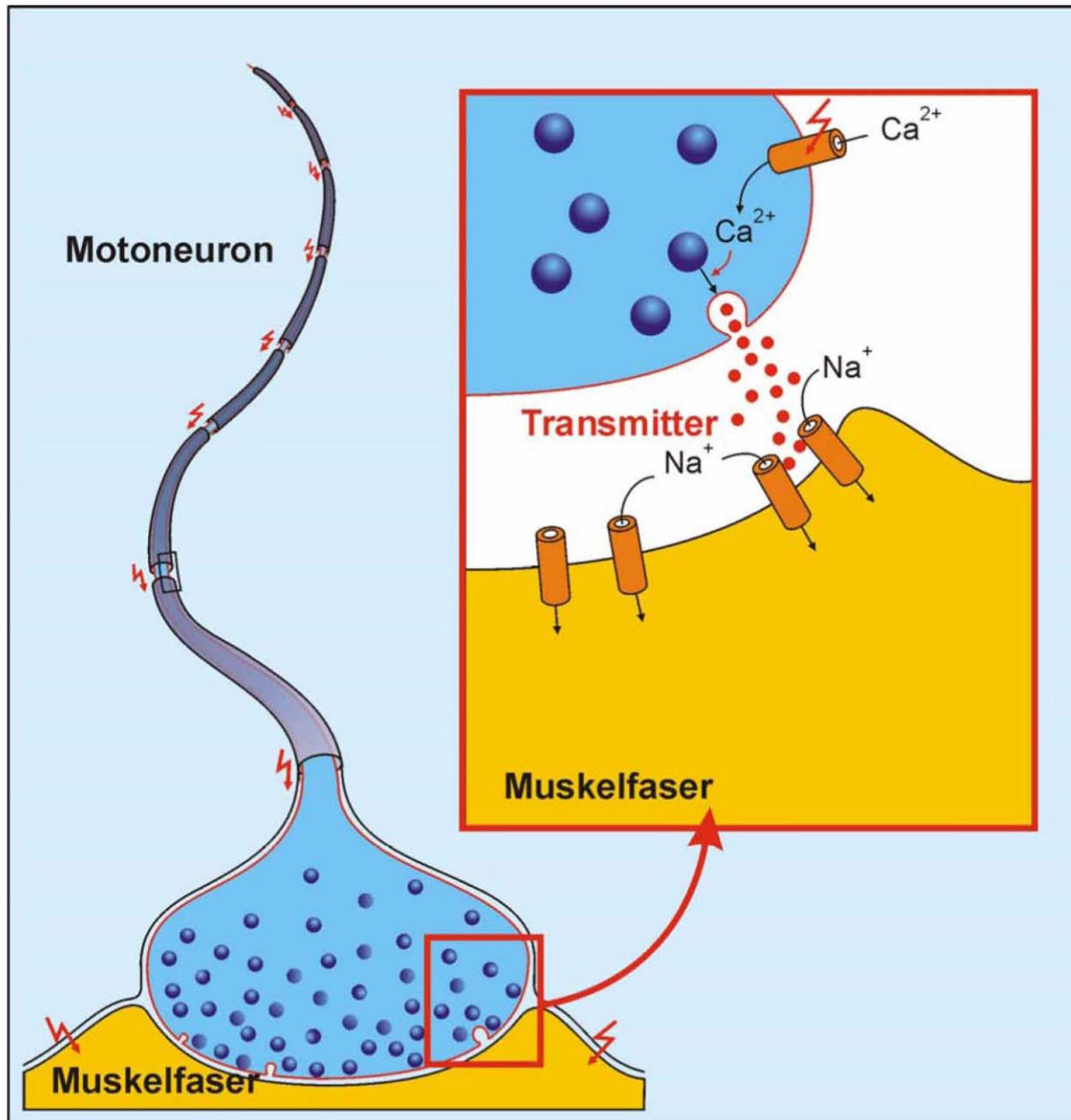
Reizweiterleitung



Reizübertragung

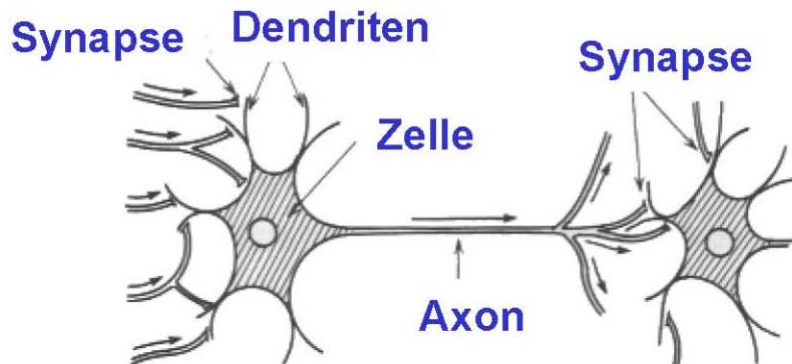
Reizübertragung durch die Synapse



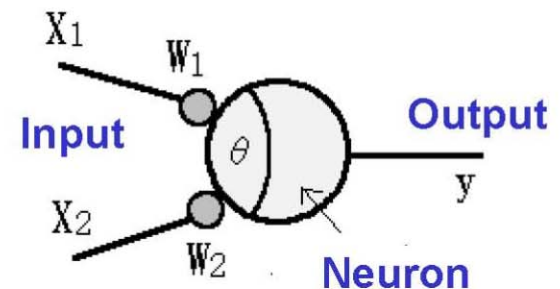
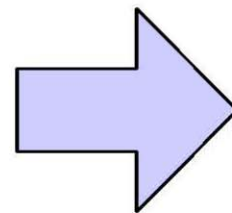


Künstliche Neuronen

- A computer that imitates the information processing carried out in our brains is being developed ([neurocomputer](#)).

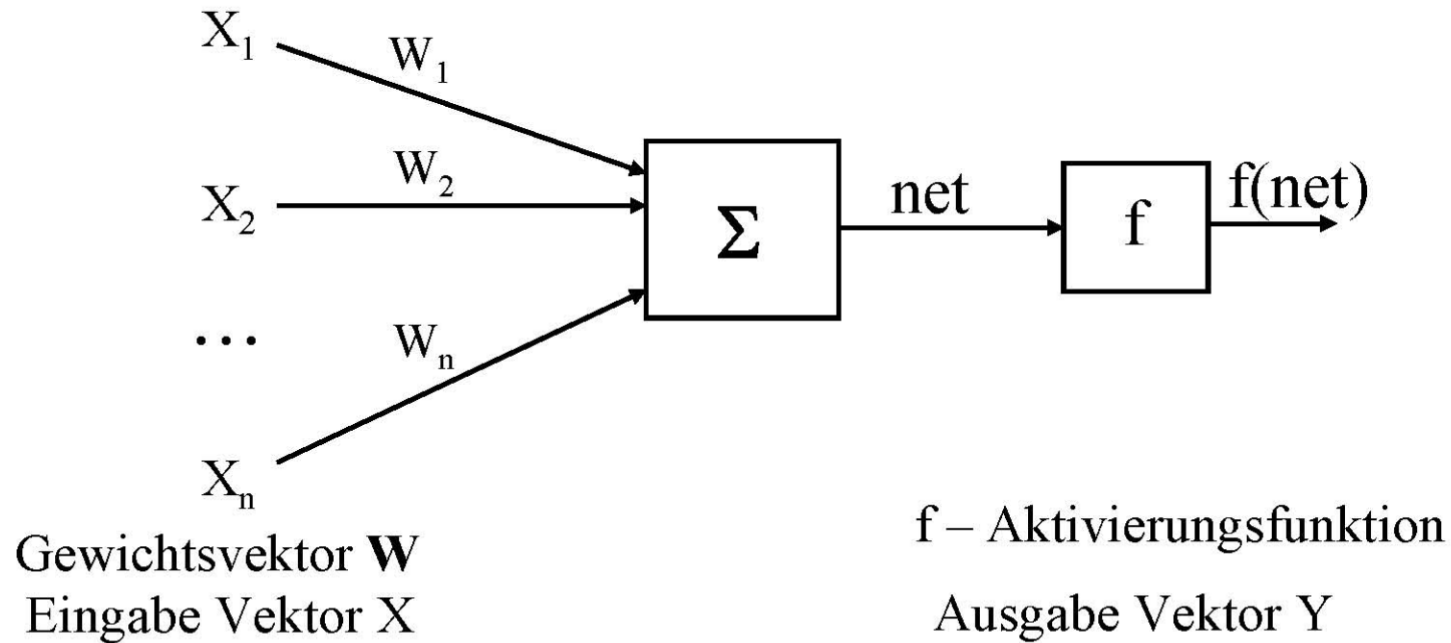


Biologische
Neuronen



Künstliche Neuronen

Einfaches Modell des Neurons



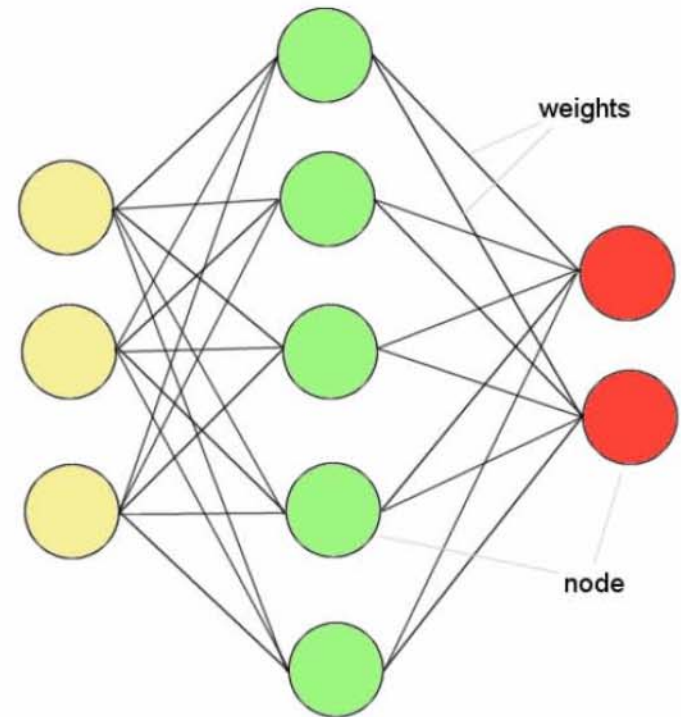
Skalarprodukt: $\mathbf{X} * \mathbf{W} = \sum w_i * x_i = \mathbf{net}(\mathbf{X}) = \mathbf{Nettoeingangssignal}$

Ausgabevektor: $\mathbf{Y} = f(\mathbf{X} * \mathbf{W}) = f(\mathbf{net}(\mathbf{X}))$

KNNs – Grundlagen

KNN enthalten zwei zentrale Begriffe aus der Biologie

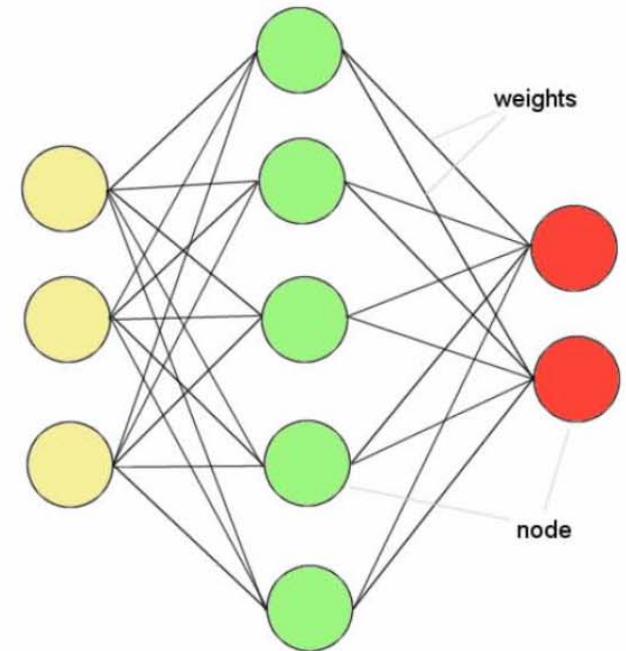
1. Neuronen (Knoten)
2. Synapsen (Gewichte)



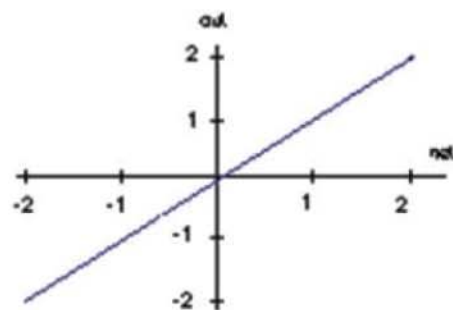
Grundbegriffe, Netzmodell

Netz als Graph mit Knoten und Kanten

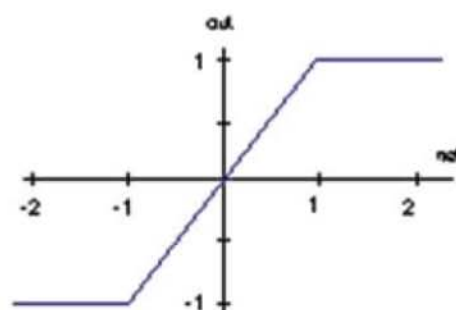
- **Knoten:** Die Neuronen
- **Kanten:** Verbindungen, durch die Aktivierungswerte fließen (Zahlen)
- **Netztopologie** bestimmt die Verarbeitung



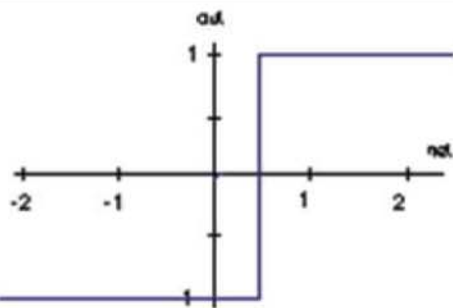
Aktivierungsfunktionen: Beispiele



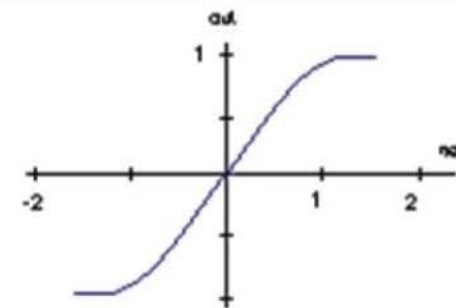
Identität ($o_j = net_j$)



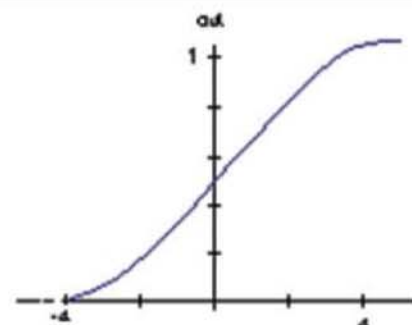
linear bis Sättigung



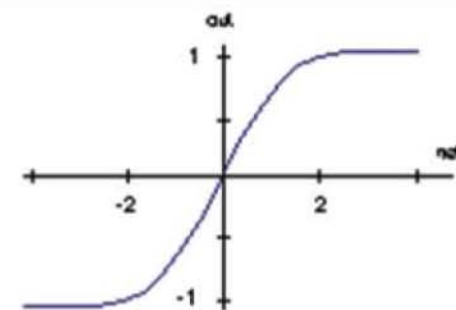
binäre Schwellenwertfkt.



$\sin(x)$ bis Sättigung

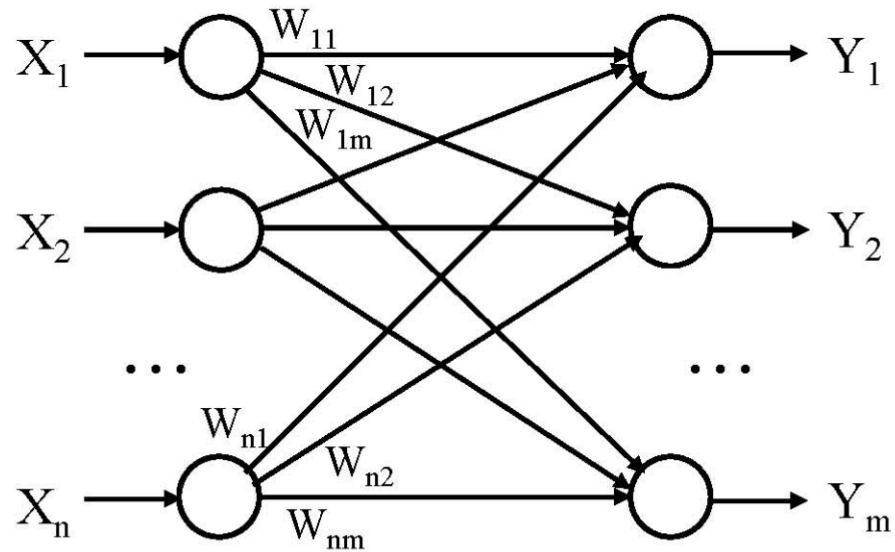


$(1 / (1 + \exp(-x)))$



$\tanh(x)$

Single-layer Neuronales Netzwerk (Perzeptron)



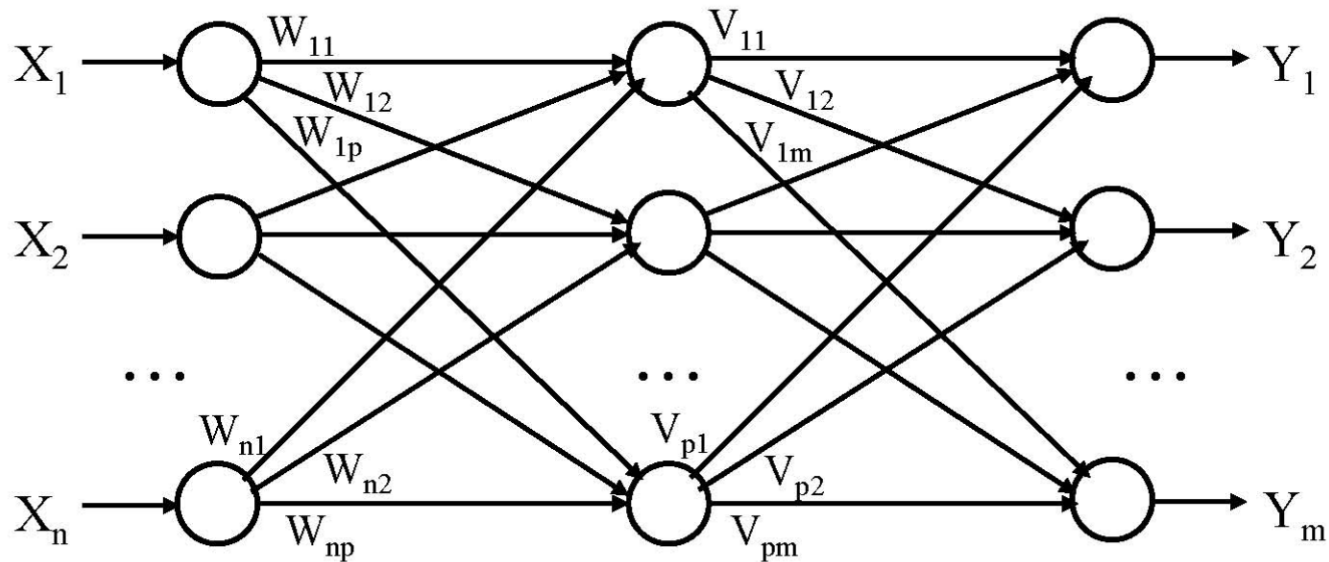
n Eingabeneuronen

m Ausgabeneuronen

Gewichtsmatrix \mathbf{W}

$$N: \mathbb{R}^n \rightarrow \mathbb{R}^m$$

Multi-layer Neuronales Netzwerk



n Eingabeneuronen

p verborgene Neuronen
(Hidden layer)

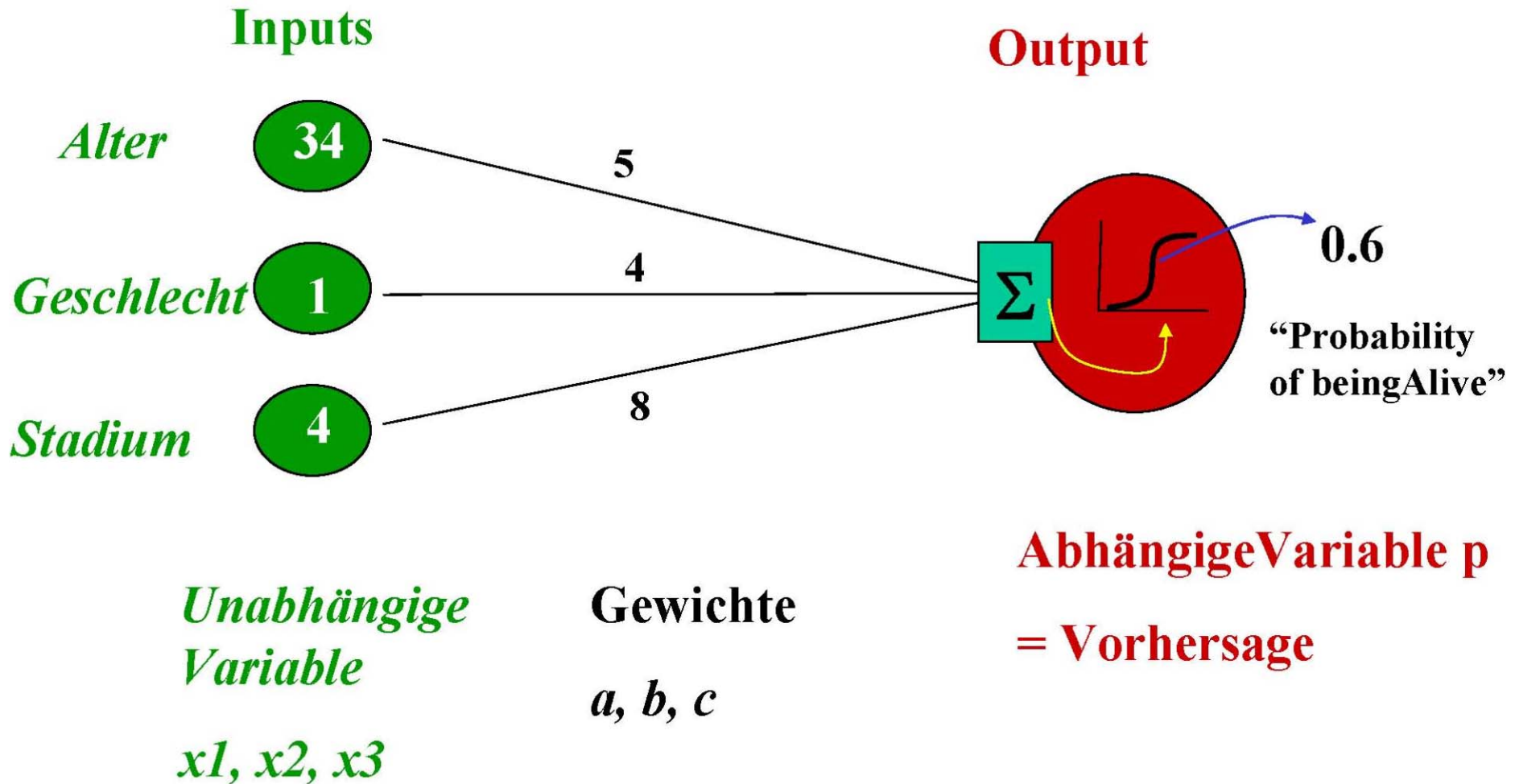
m Ausgabeneuronen

Gewichtsmatrix \mathbf{W}

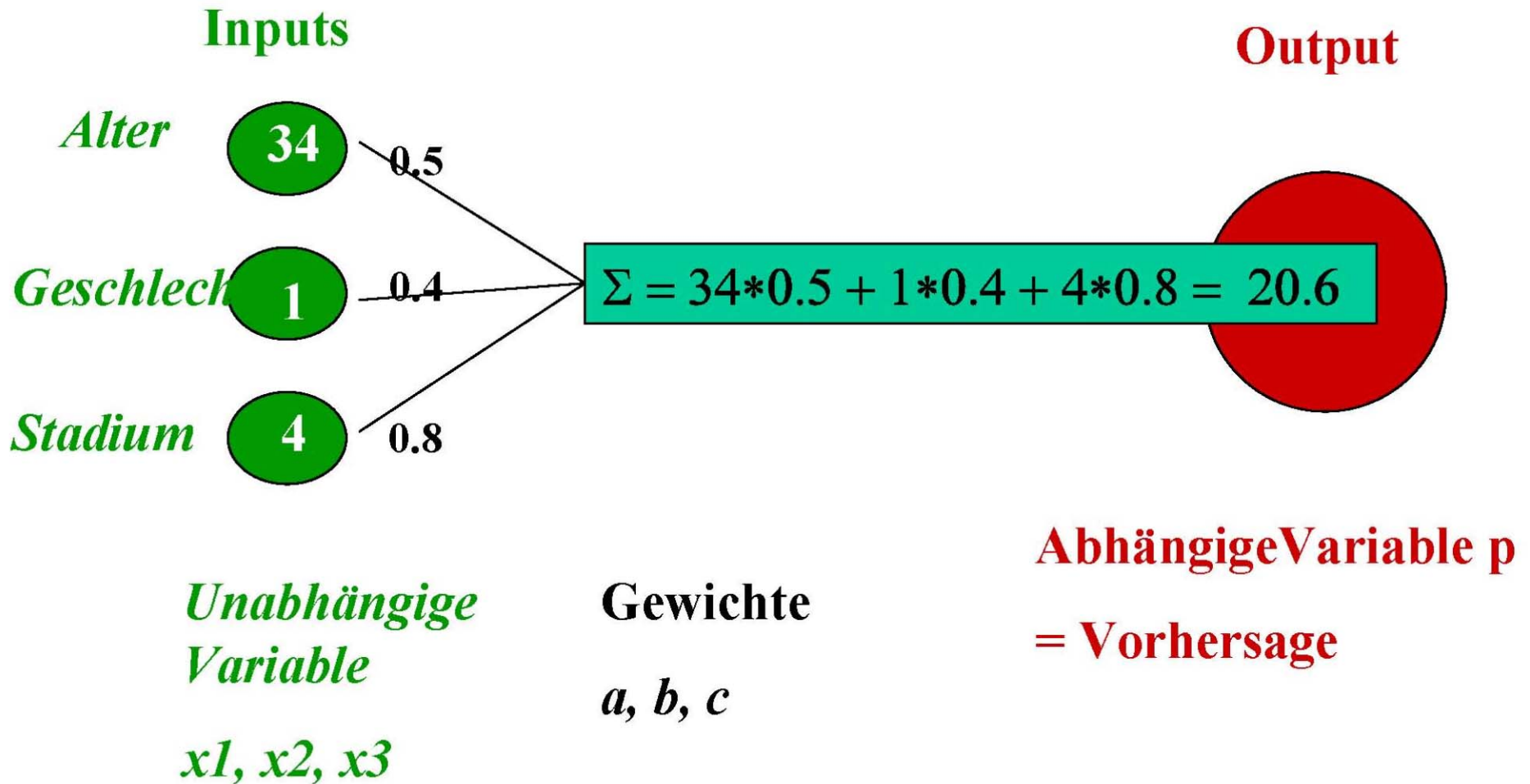
Gewichtsmatrix \mathbf{V}

$$N: \mathbb{R}^n \rightarrow \mathbb{R}^m$$

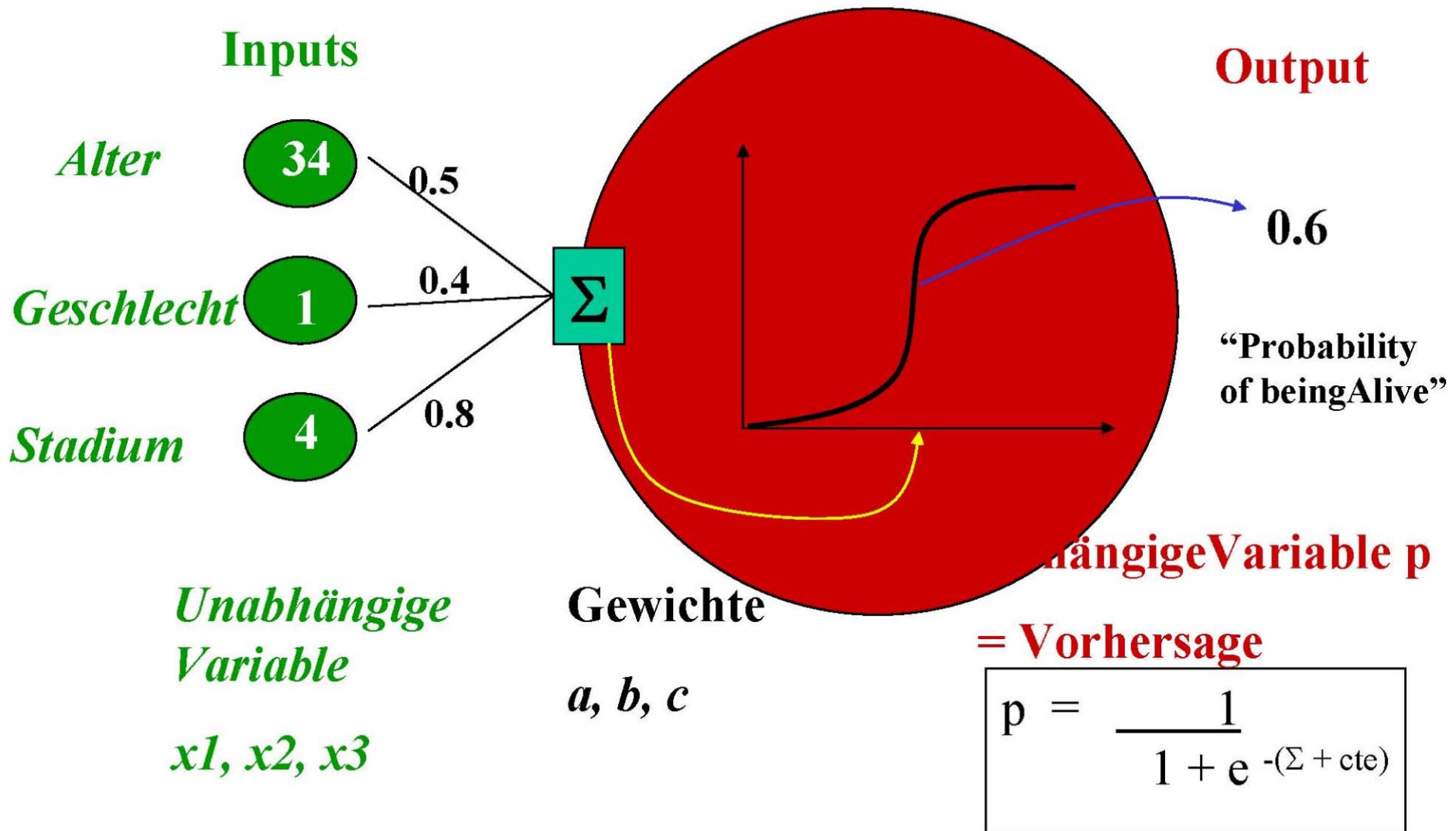
Logistisches Regression Modell



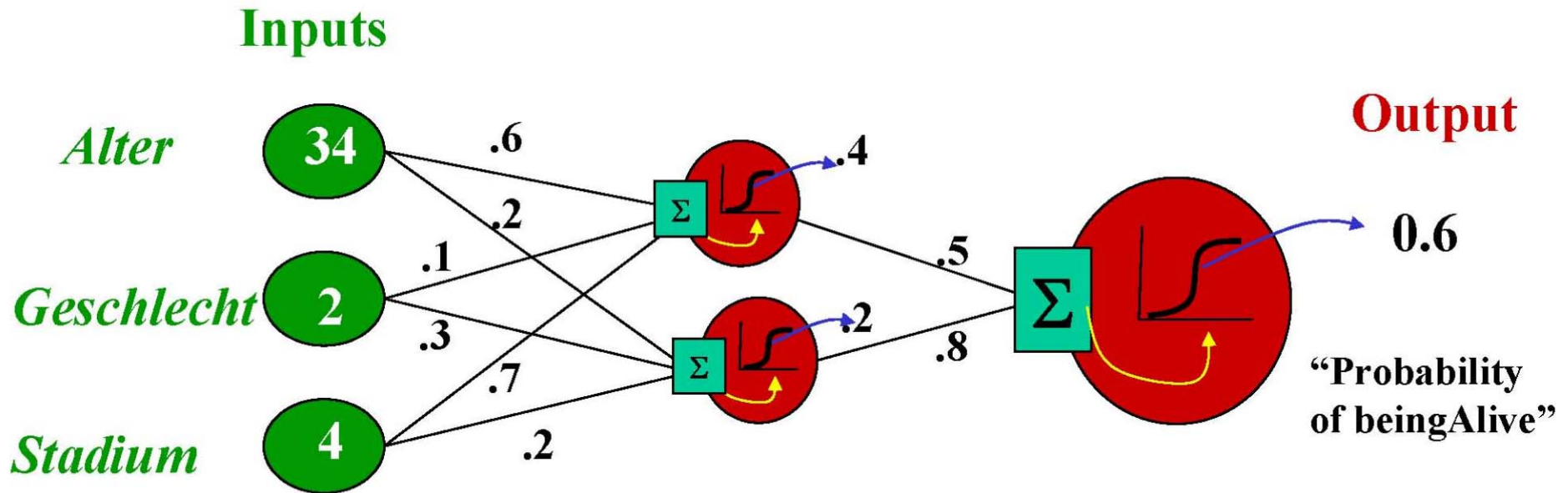
Σ is the sum of inputs * weights



Logistic function



Neurales Netzwerk Modell



Unabhängige Variable

x_1, x_2, x_3

Gewichte

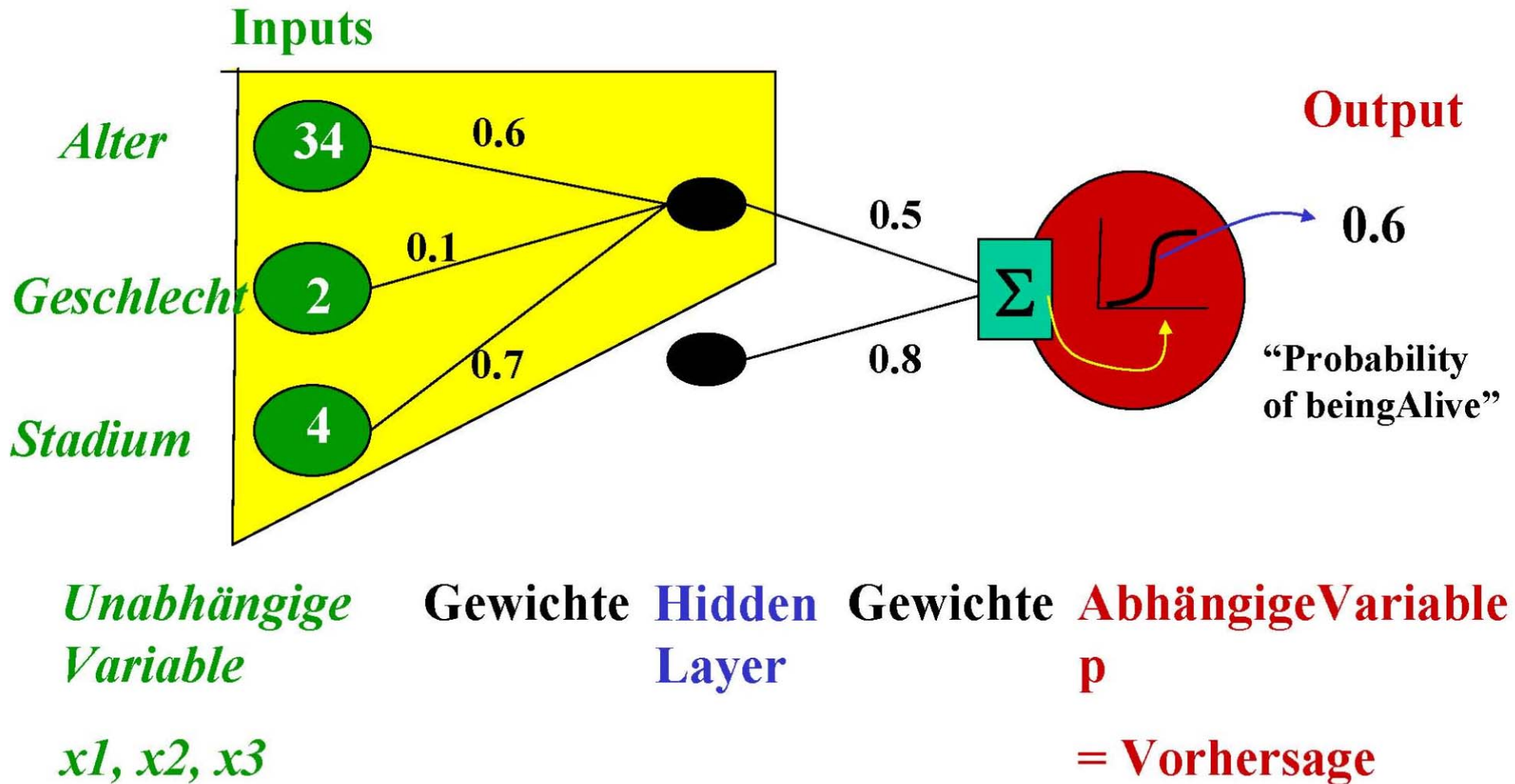
Hidden Layer

Gewichte

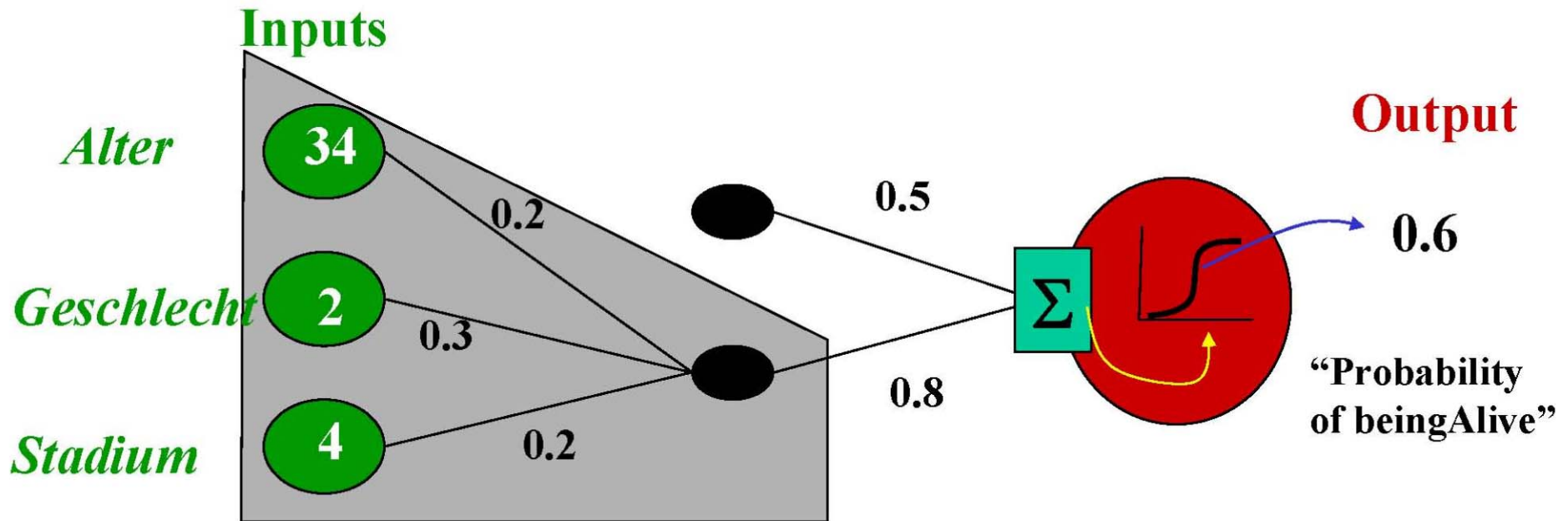
Abhängige Variable p

= Vorhersage

“Kombinierte logistische Modelle”



“Kombinierte logistische Modelle”



*Unabhängige
Variable*

x1, x2, x3

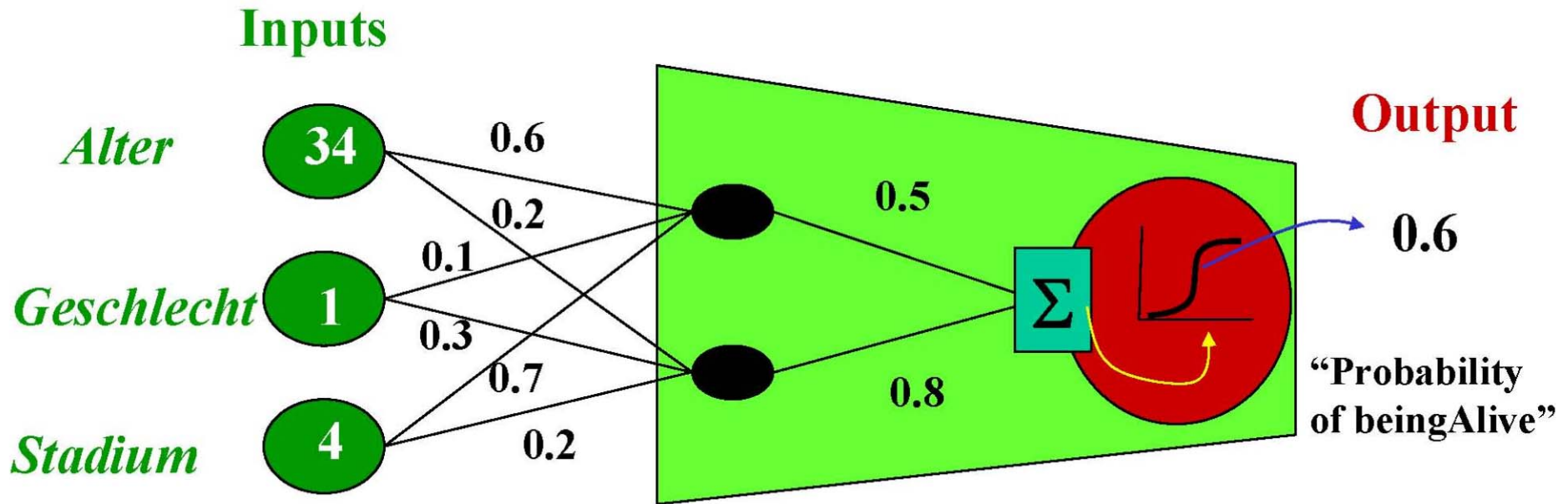
Gewichte **Hidden
Layer**

Gewichte

Abhängige Variable
p

= Vorhersage

“Kombinierte logistische Modelle”



Unabhängige Variable

$x1, x2, x3$

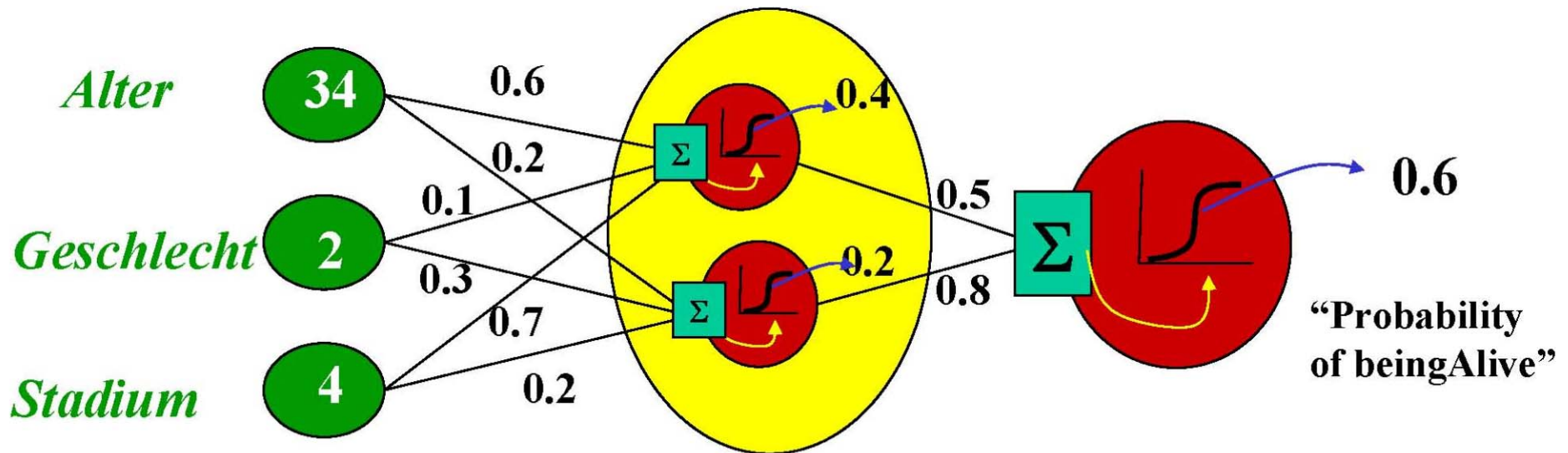
Gewichte **Hidden Layer**

Gewichte

Abhängige Variable
 p

= Vorhersage

“Kombinierte logistische Modelle”



*Unabhängige
Variable*

x1, x2, x3

Gewichte

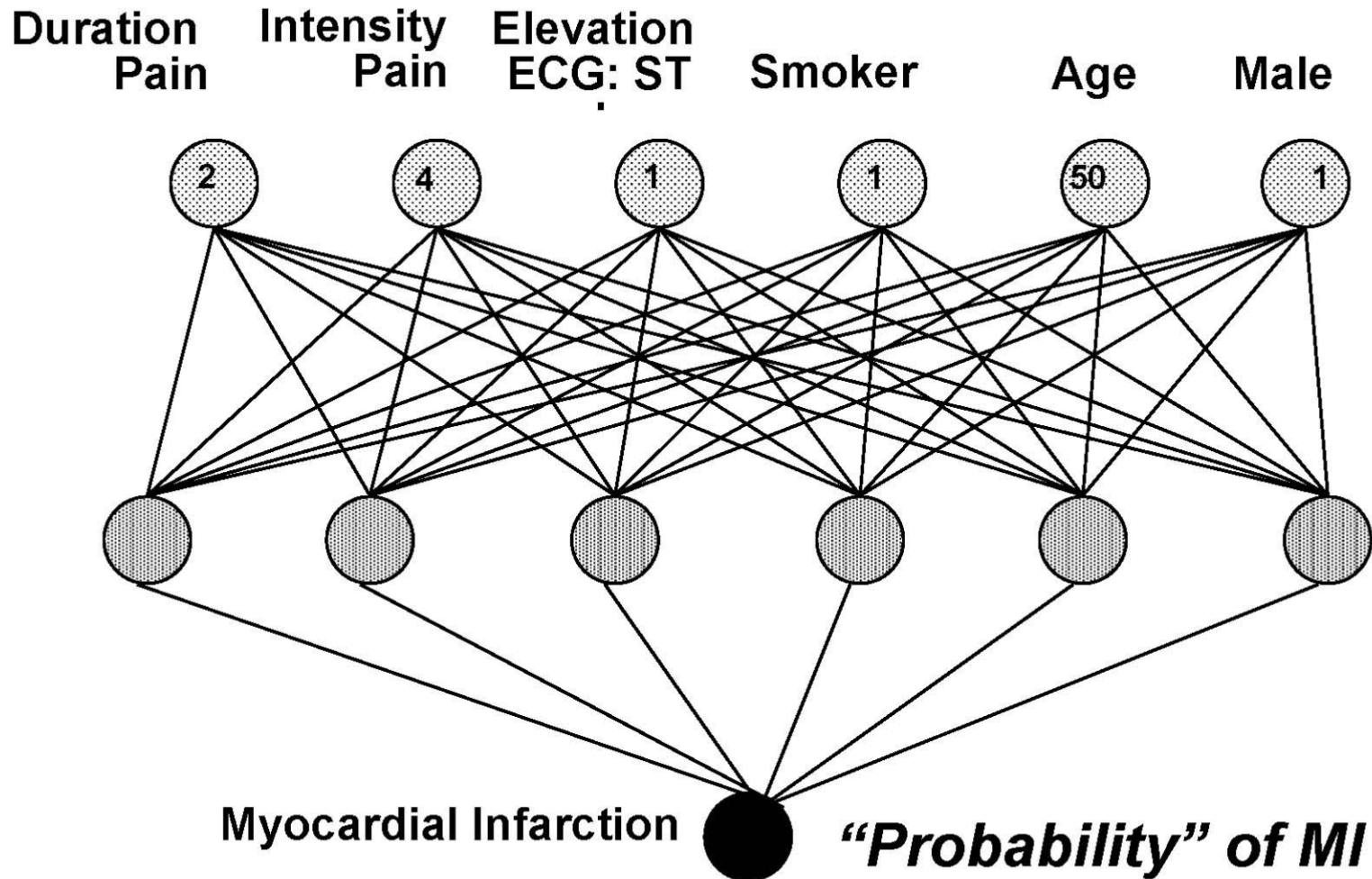
**Hidden
Layer**

Gewichte

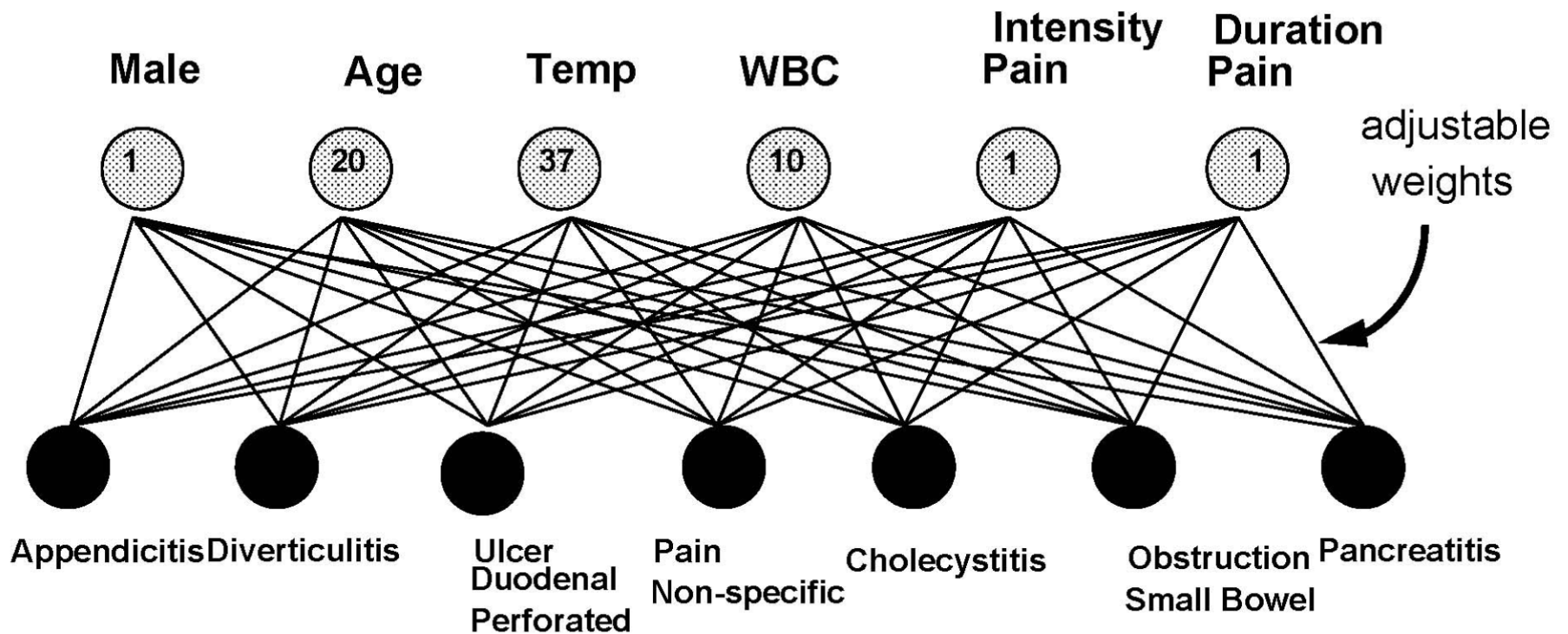
**Abhängige Variable
p**

= Vorhersage

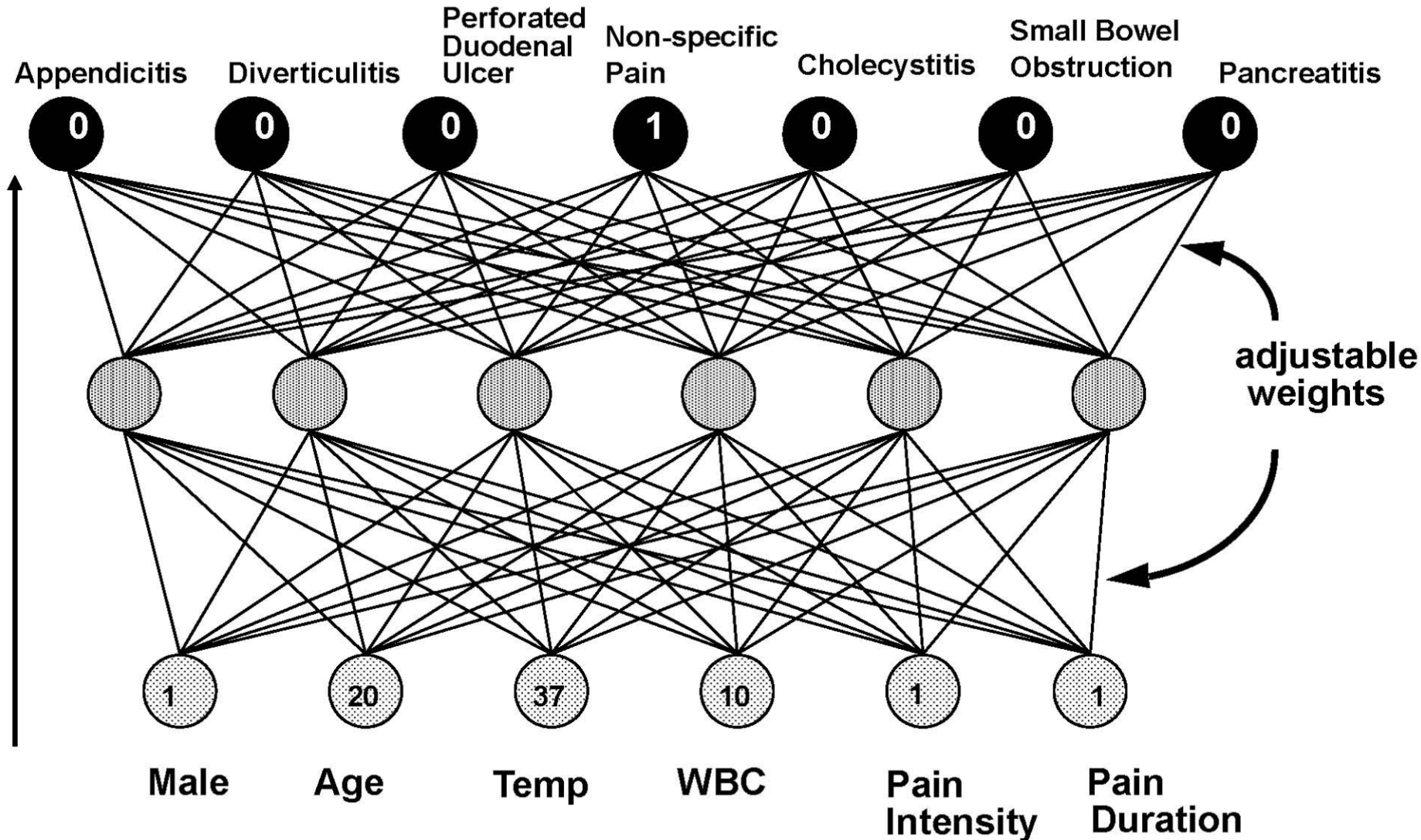
Myocardial Infarction Network

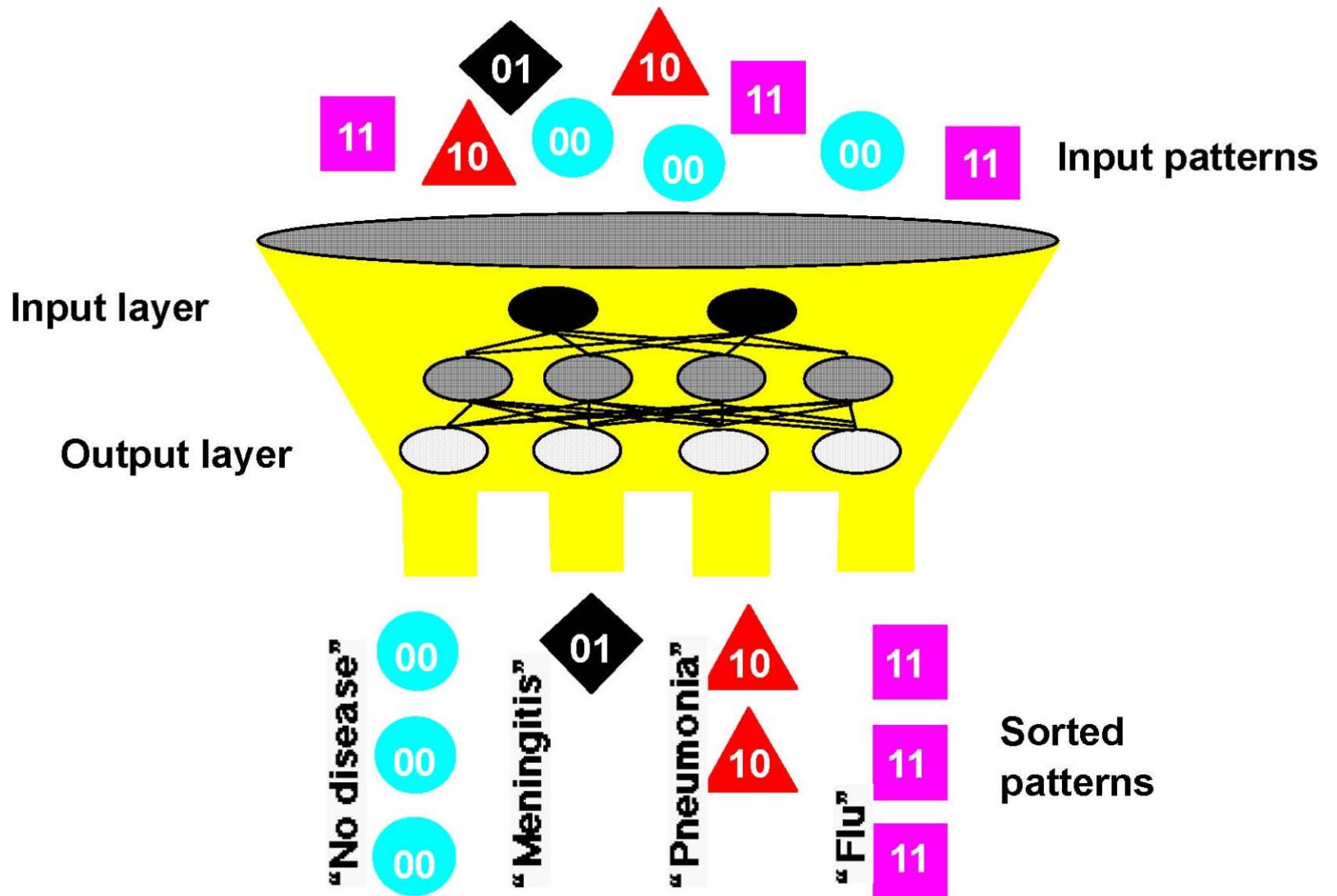


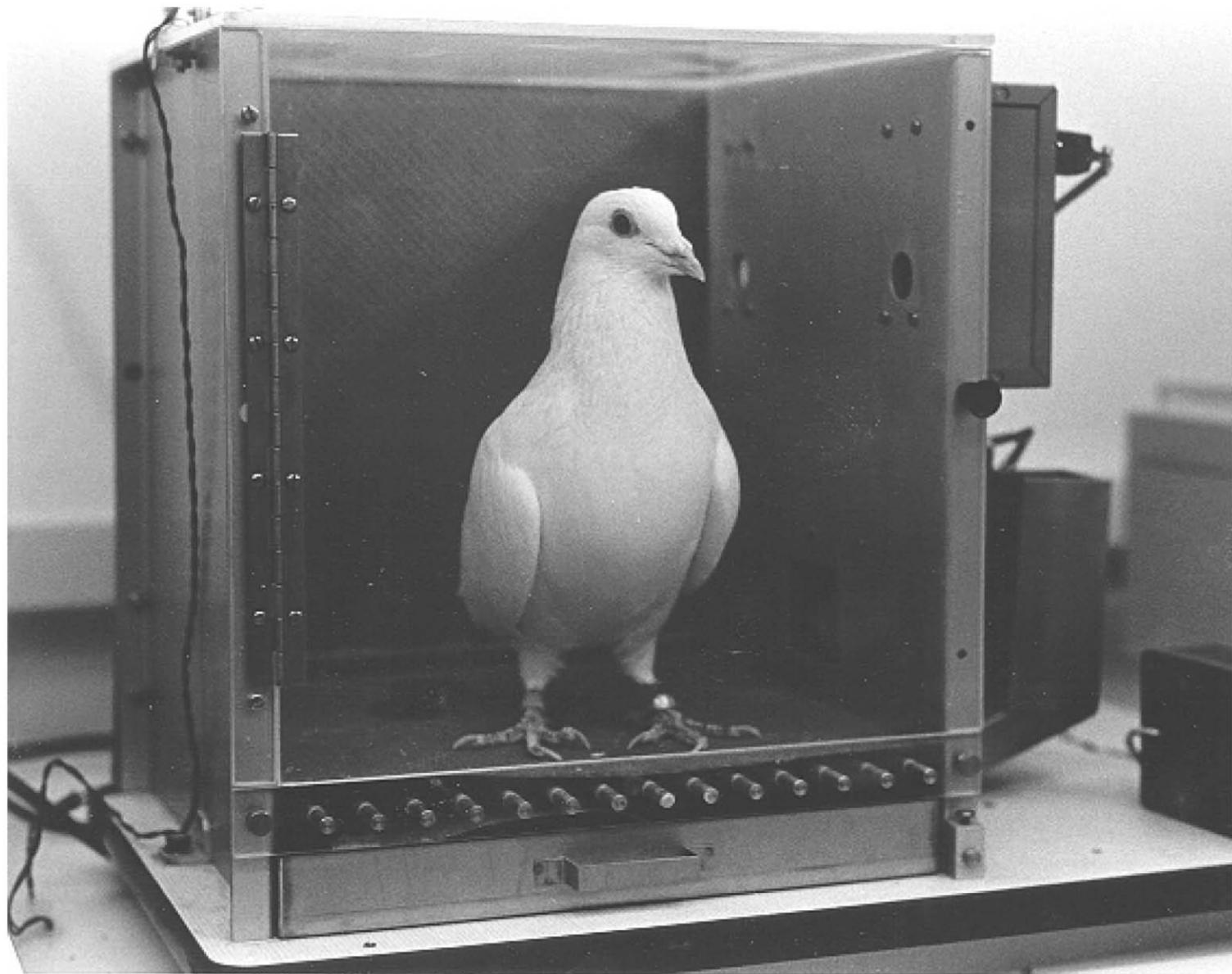
Abdominal Pain Perceptron

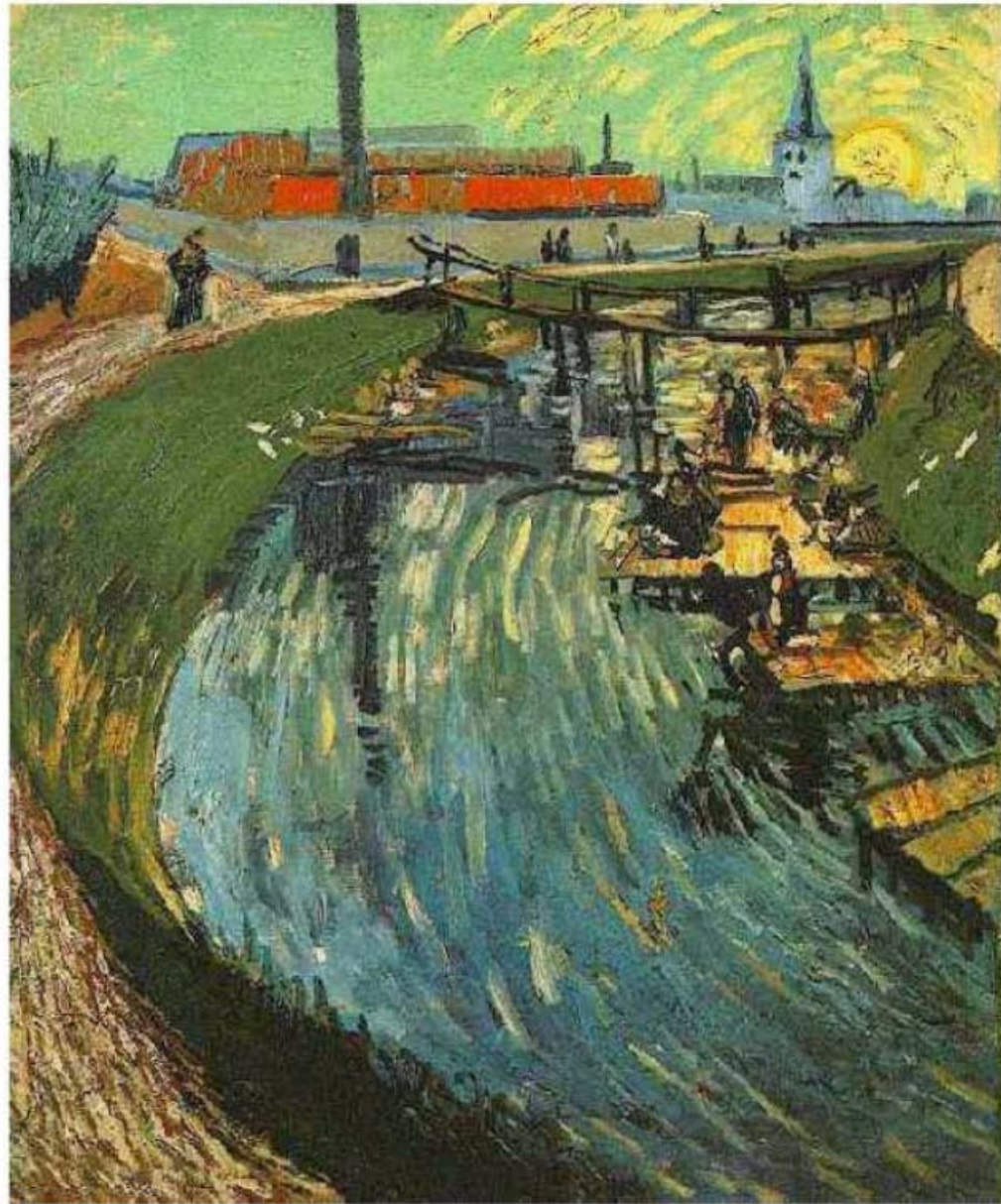


Abdominal Pain





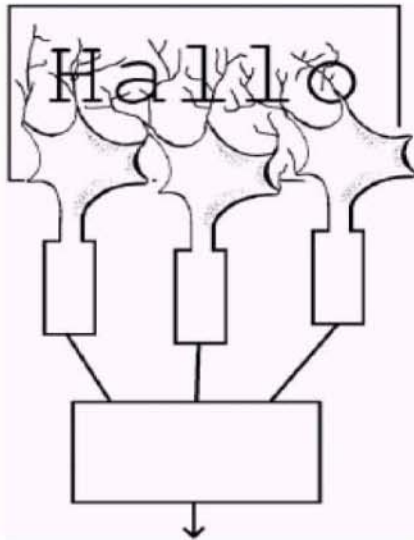






Das Perceptron

Entwickelt von Rosenblatt 1958



Sensorische Elemente:

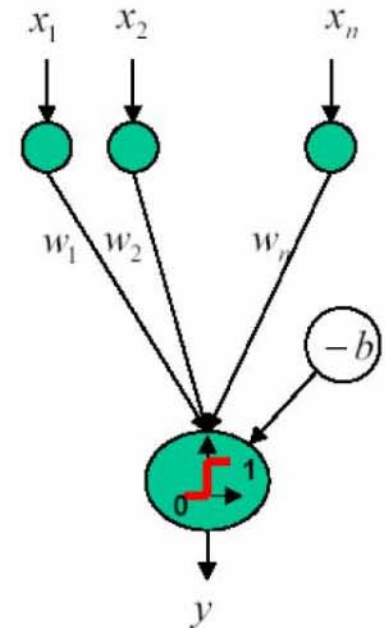
Fotozellen – Netzhaut

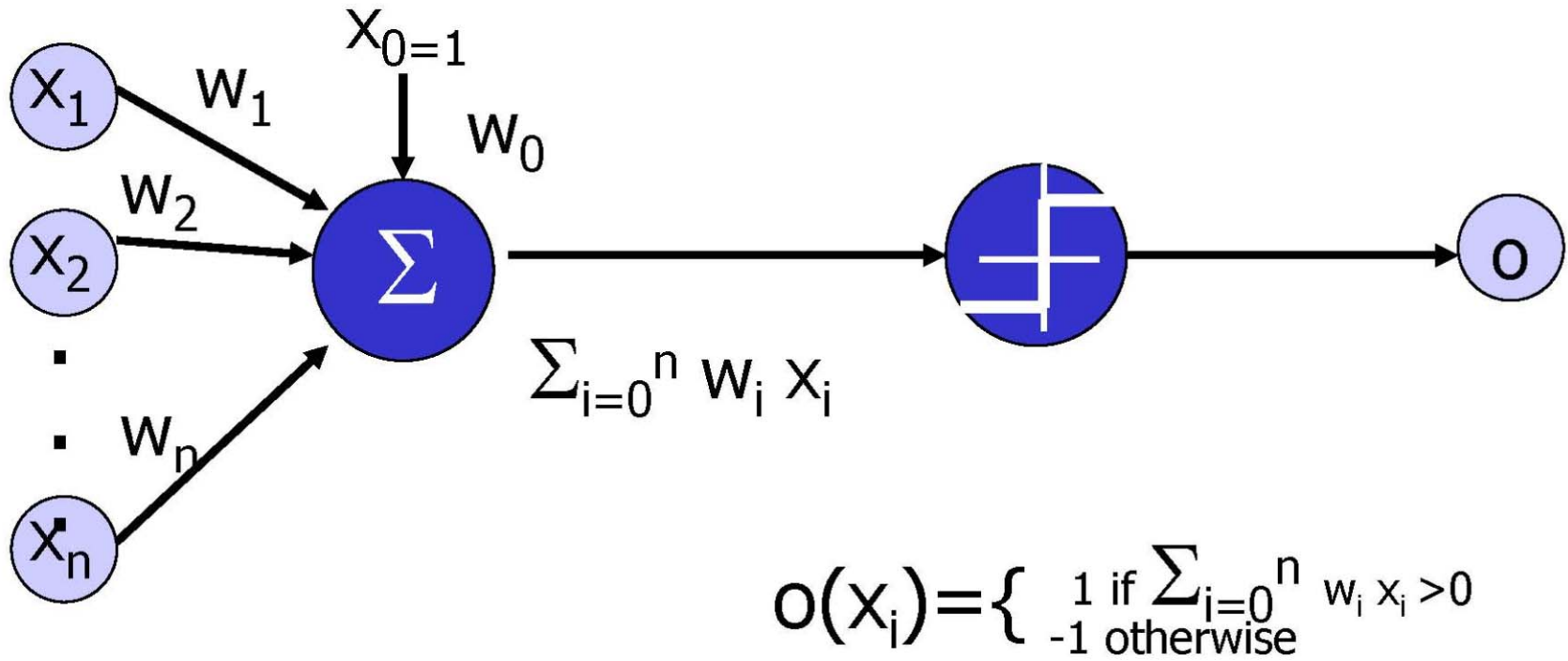
Assoziative Elemente:

Verknüpfung mit
sensorischen Elementen

Ausführende Elemente:

Fassen die Ausgaben der
assoziativen Elemente zu
einer Entscheidung
zusammen



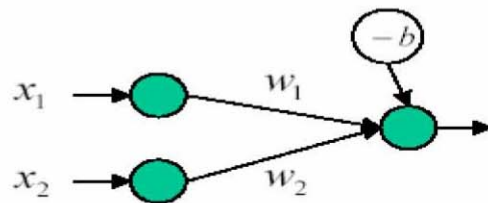


- Input $x = (x_1, \dots, x_n)$
- Gewichte $w = (w_1, \dots, w_n)$
- Output (o)

Das Perceptron

Was macht das Perceptron?

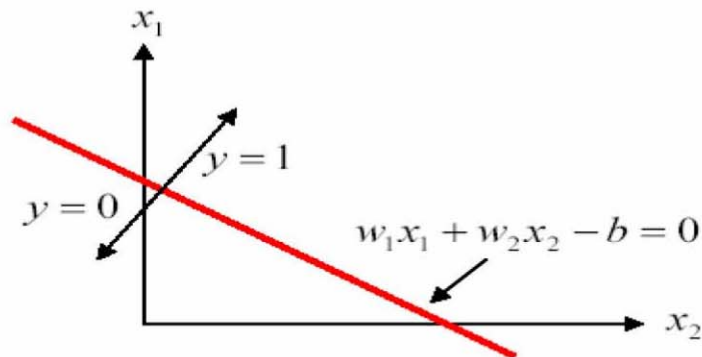
Perceptron mit 2 Eingängen:



$$y = \begin{cases} 1 & \text{falls } w_1x_1 + w_2x_2 > b \\ 0 & \text{falls } w_1x_1 + w_2x_2 \leq b \end{cases}$$

\swarrow
 $= w_1x_1 + w_2x_2 - b \leq 0$

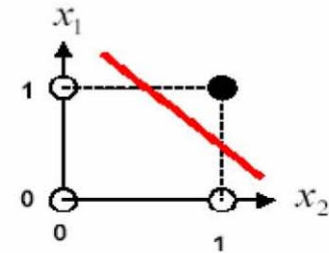
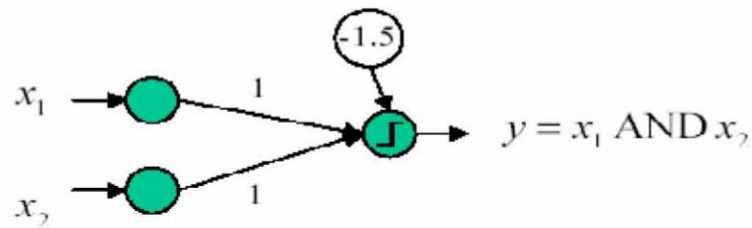
\nwarrow
 $= w_1x_1 + w_2x_2 - b > 0$



Das Perceptron-

Einfaches Beispiel

Berechnung des logischen „AND“



x_1	x_2	$x_1 \text{ AND } x_2$	$y = x_1 \cdot w_1 + x_2 \cdot w_2 - b$
0	0	0	$0 \cdot 1 + 0 \cdot 1 - 1.5 = -1.5 < 0 \Rightarrow y = 0$
0	1	0	$0 \cdot 1 + 1 \cdot 1 - 1.5 = -0.5 < 0 \Rightarrow y = 0$
1	0	0	$1 \cdot 1 + 0 \cdot 1 - 1.5 = -0.5 < 0 \Rightarrow y = 0$
1	1	1	$1 \cdot 1 + 1 \cdot 1 - 1.5 = 0.5 > 0 \Rightarrow y = 1$

Das Perceptron

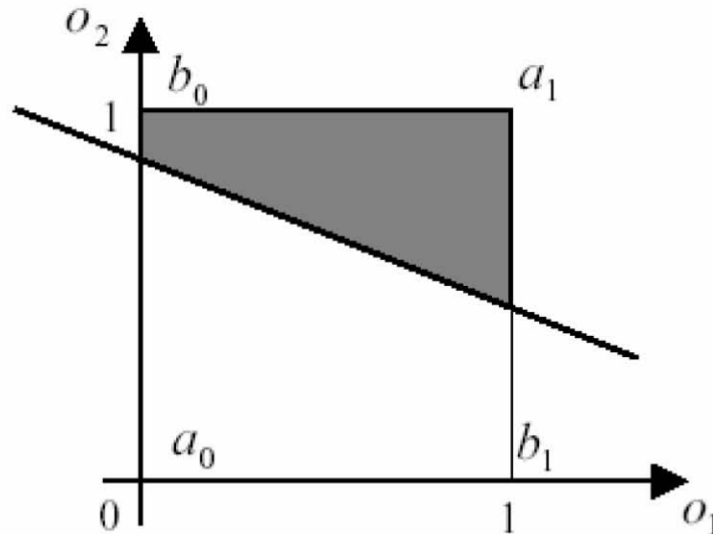
Perceptron-Konvergenz-Theorem:

***Der Lernalgorithmus des Perzeptrons konvergiert
in endlicher Zeit, d.h.
das Perzeptron kann in endlicher Zeit alles
lernen, was es repräsentieren kann.***

Das Perceptron

**Kritikpunkt
von Minsky und Papert:**

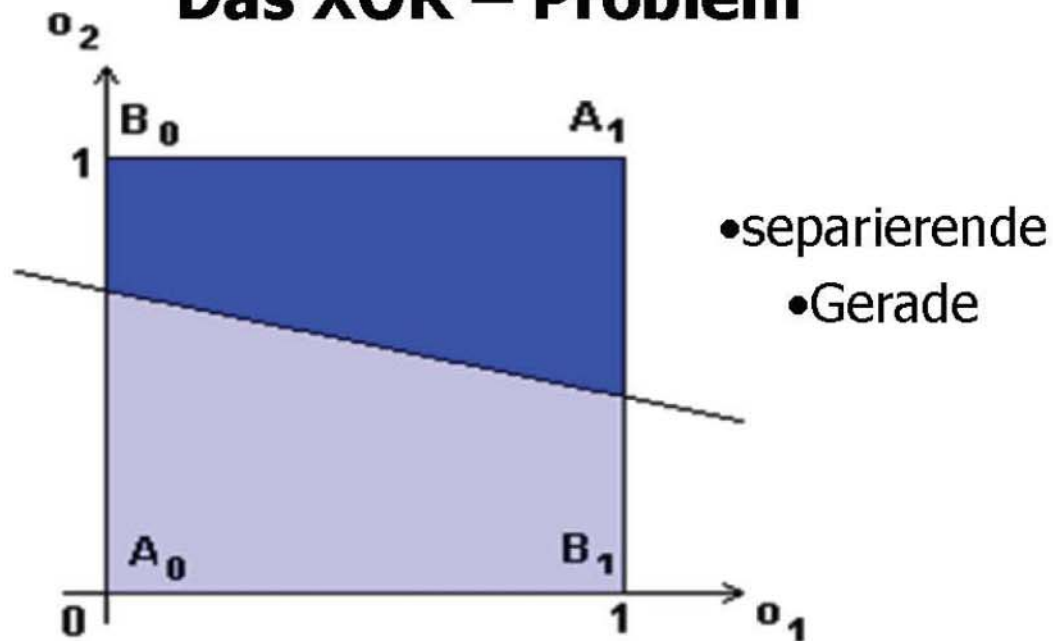
**Ein einstufiges Perzeptron kann nur linear
separierbare Mengen klassifizieren.**



Das Perceptron-

Ein weiteres Beispiel

Das XOR – Problem



Input		Output
00	\Rightarrow	0
01	\Rightarrow	1
10	\Rightarrow	1
11	\Rightarrow	0

läßt sich nicht separieren.

Perzeptronen

Die Prozentzahl der linear trennbaren Funktionen geht gegen 0, wenn die Eingaben gegen unendlich gehen

n	Anzahl der binären Funktionen von n Eingaben	Anzahl der davon linear separierbaren Funktionen
1	4	4
2	16	14
3	256	104
4	65.536	1.772
5	$4,3 \cdot 10^9$	94.572
6	$1,8 \cdot 10^{19}$	5.028.134

Mögliche Lernabläufe in NN

These: Lernen = Veränderung

Beispiele für Veränderungen in Neuronalen Netzen

- Einführen/Löschen von Neuronen
- Entwicklung/Löschen von Verbindungen
- Modifikation der Verbindungs-Gewichtungen
- Modifikation der Neuronen-Schwellenwerte
- Modifikation der Aktivierungs-, Propagierungs- oder Ausgabefunktion

Arten von Lernverfahren

- **Überwachtes Lernen (supervised)**

Lernalgorithmus minimiert den Fehler zwischen berechnetem und gewünschtem Output

- **Unüberwachtes Lernen (unsupervised)**

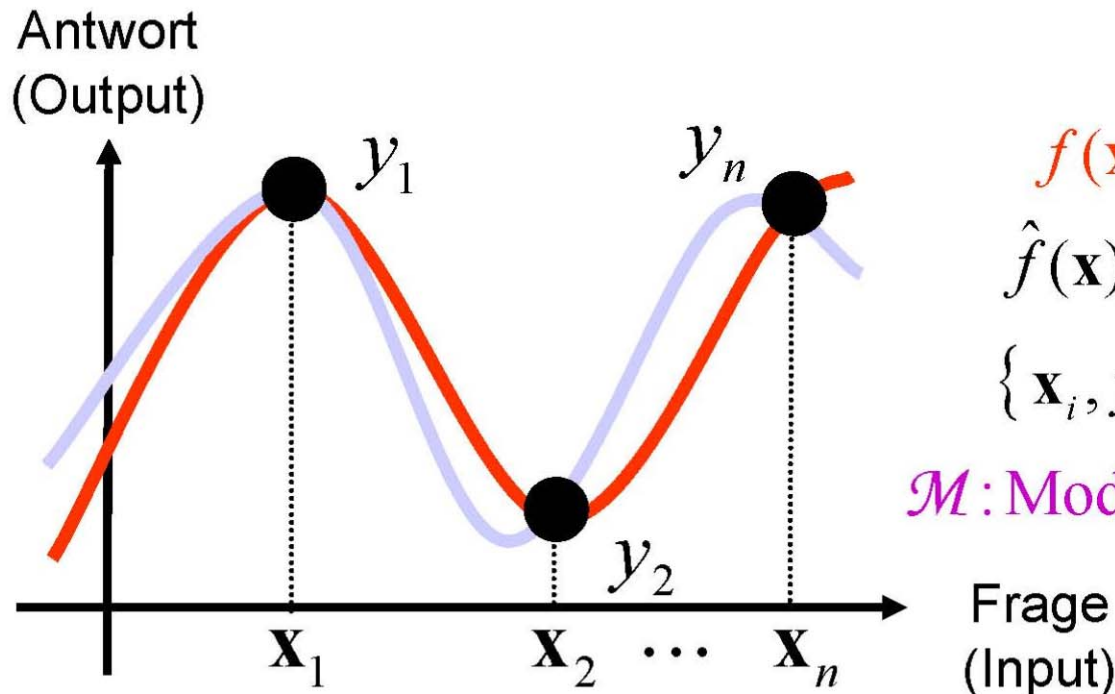
Optimierung von Gewichten basierend auf Kriterien, die das Netz selber hat

- **Bestärkendes Lernen (reinforcement)**

Lernalgorithmus adaptiert Gewichte basierend auf Maximierung einer Belohnung (reward) bzw.

Minimierung einer Bestrafung (punishment), die aufgrund des berechneten Outputs ausgesprochen wird

Überwachtes Lernen als Approximation von Funktionen



$f(\mathbf{x})$: Ziel Funktion

$\hat{f}(\mathbf{x}) \in \mathcal{M}$: Gelernte Funktion

$\{\mathbf{x}_i, y_i\}_{i=1}^n$: Trainings Beispiele

\mathcal{M} : Modell (Menge von Funktionen)

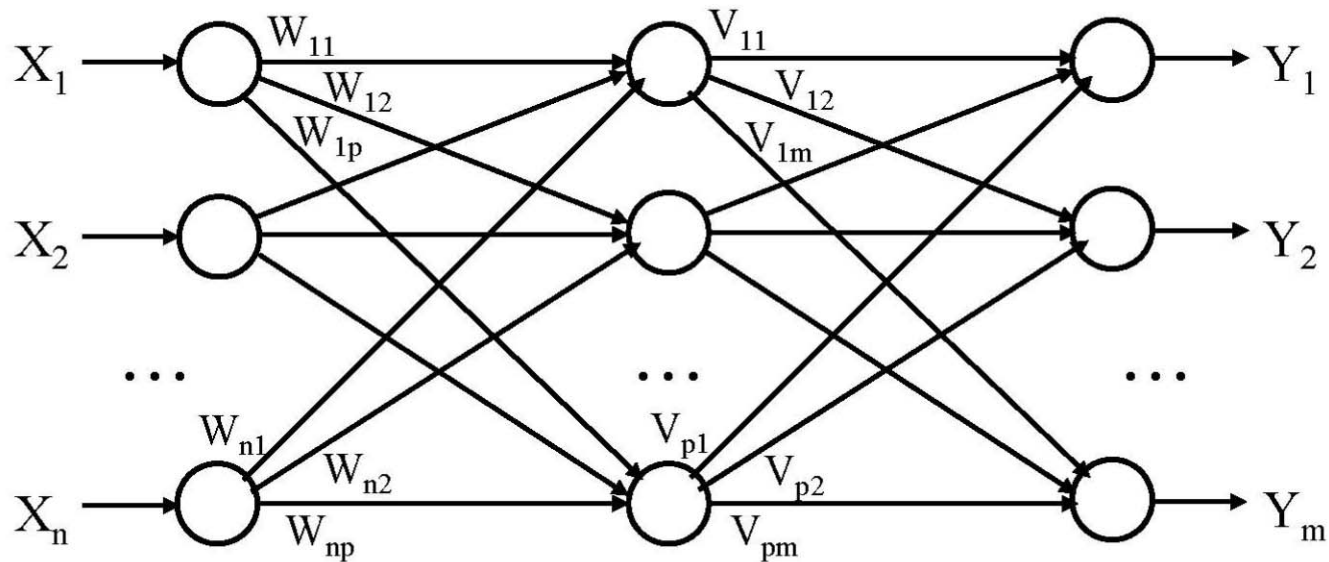
Ausgangspunkt: Beispielmengemenge $\{\mathbf{x}_i, y_i\}_{i=1}^n$
 berechne eine Funktion \hat{f} aus einem Modell \mathcal{M}
 die die Zielfunktion $f(\mathbf{x})$ bestmöglichst approximiert

Multi-layer Neurale Netzwerke

Back-Propagation Learning

Back-Propagation Learning

Überwachtes Lernen für Multi-Layer Feedforward Netze



n Eingabeneuronen p verborgene Neuronen m Ausgabeneuronen
(*Hidden layer*)

Gewichtsmatrix \mathbf{W} Gewichtsmatrix \mathbf{V}

$$N: \mathbb{R}^n \rightarrow \mathbb{R}^m$$

Backpropagation-Netze

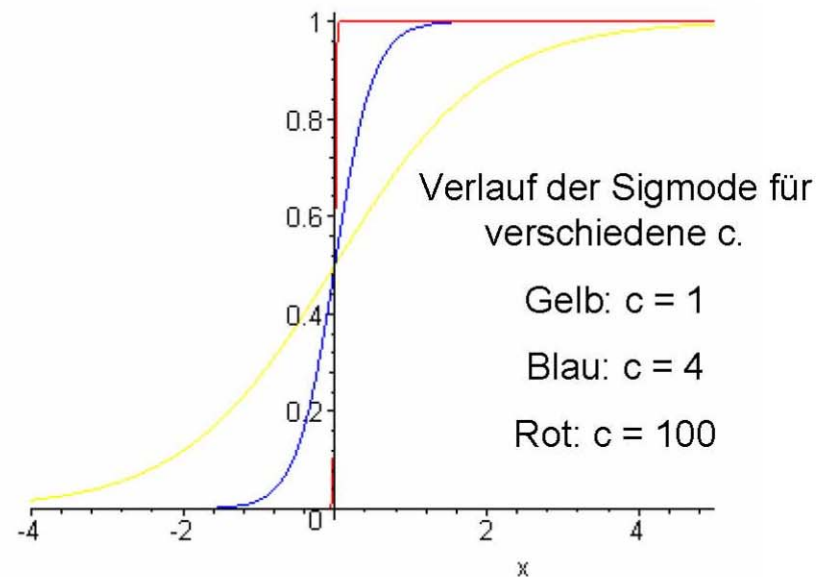
- Sind die in der Praxis am häufigsten eingesetzten Netze
- 1974 von mehreren Wissenschaftlern unabhängig voneinander entwickelt
- Größere Beachtung seit 1986 mit dem Buch „Parallel Distributed Processing: Explorations in the Microstructure of Cognition“

Backpropagation-Netze

- Als Ausgabefunktion wird eine Sigmoidfunktion benutzt, da man eine stetig differenzierbare Funktion braucht
- Hauptvorteil der Sigmoidfunktion ist die einfache Ableitung

$$s(x) = \frac{1}{1 + e^{-cx}}$$

$$s'(x) = s(x)(1 - s(x))$$



Backpropagation-Lernen

- Das Lernproblem
 - n vorgegebene Eingabevektoren sind auf n vorgegebene Ausgabevektoren abzubilden
 - Die Summe der Fehlerquadrate soll minimiert werden
 - Dazu müssen passende Gewichte für die Neuronen gefunden werden

Fehlerfunktion:

$$E = \frac{1}{2} \sum_{i=1}^m \|t_i - y_i\|^2$$

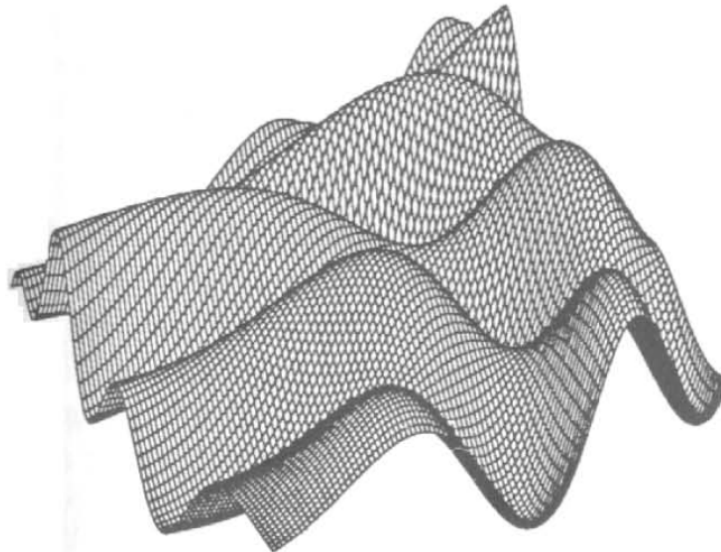
Sigmode:

$$s_c(x) = \frac{1}{1 + e^{-cx}}$$

t_i = vorgegebenen Ausgabevektoren; y_i = im Netz berechneten Ausgaben

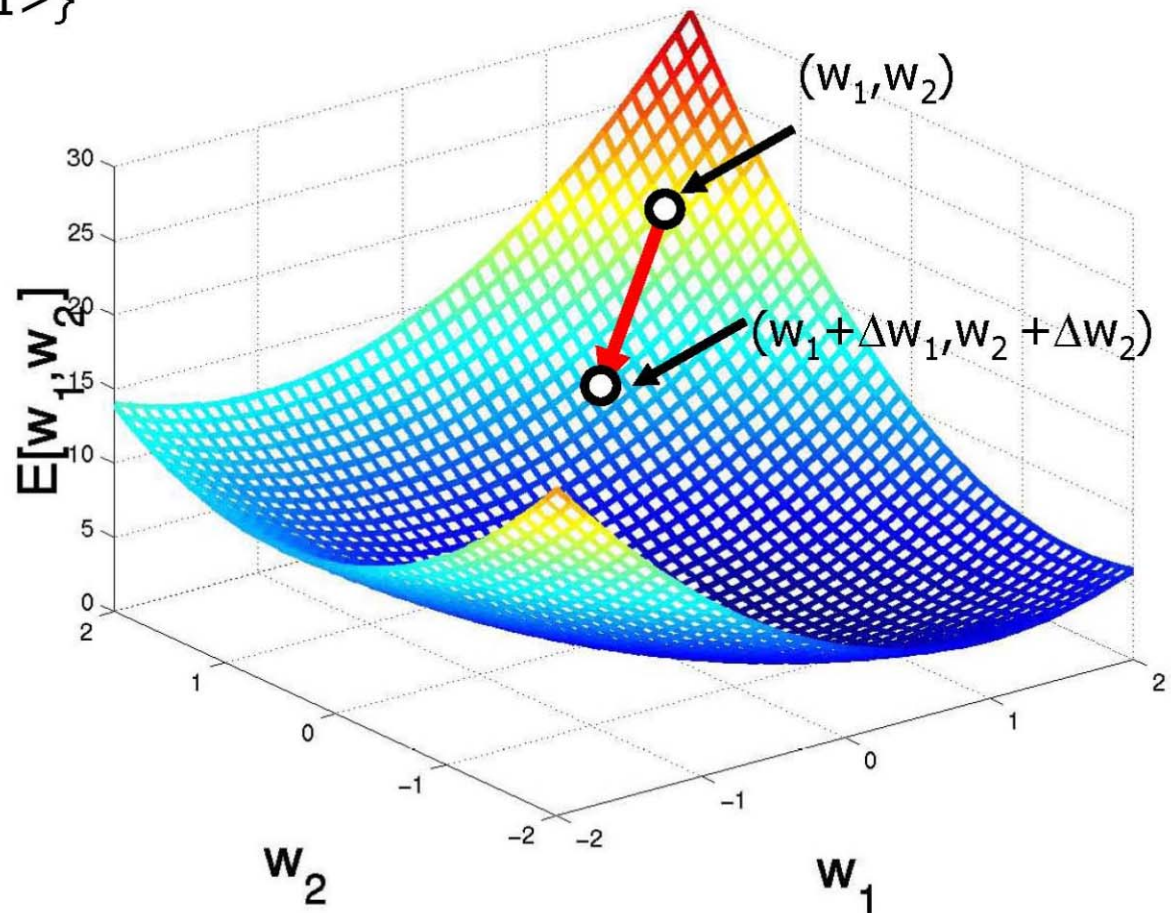
Backpropagation-Prinzip

- Man bildet die Ableitung des Gesamtfehlers nach den Gewichten (Gradient)
- Nun ändert man die Gewichte in umgekehrter Richtung des Gradienten um den Fehler zu verkleinern (Gradientenabstiegsverfahren)
- Ziel: Finden des globalen Minimums



Gradientenabstieg: Das Prinzip

$$D = \{ \langle (1,1), 1 \rangle, \langle (-1,-1), 1 \rangle, \\ \langle (1,-1), -1 \rangle, \langle (-1,1), -1 \rangle \}$$



Gradientenabstieg

- Trainiere die w_i 's so dass der Fehler minimiert wird.

$$\text{Fehlerfunktion: } E[w_1, \dots, w_n] := \frac{1}{2} \sum_{d \in D} (t_d - y_d)^2$$

Änderung der Gewichte: $w_i := w_i + \Delta w_i$.

Ansatz: $\Delta w_i := -\eta \partial E / \partial w_i$

Gradient:

$$\text{grad} E[w] = [\partial E / \partial w_0, \dots, \partial E / \partial w_n]$$

$$\Delta w = -\eta \text{ grad} E[w]$$

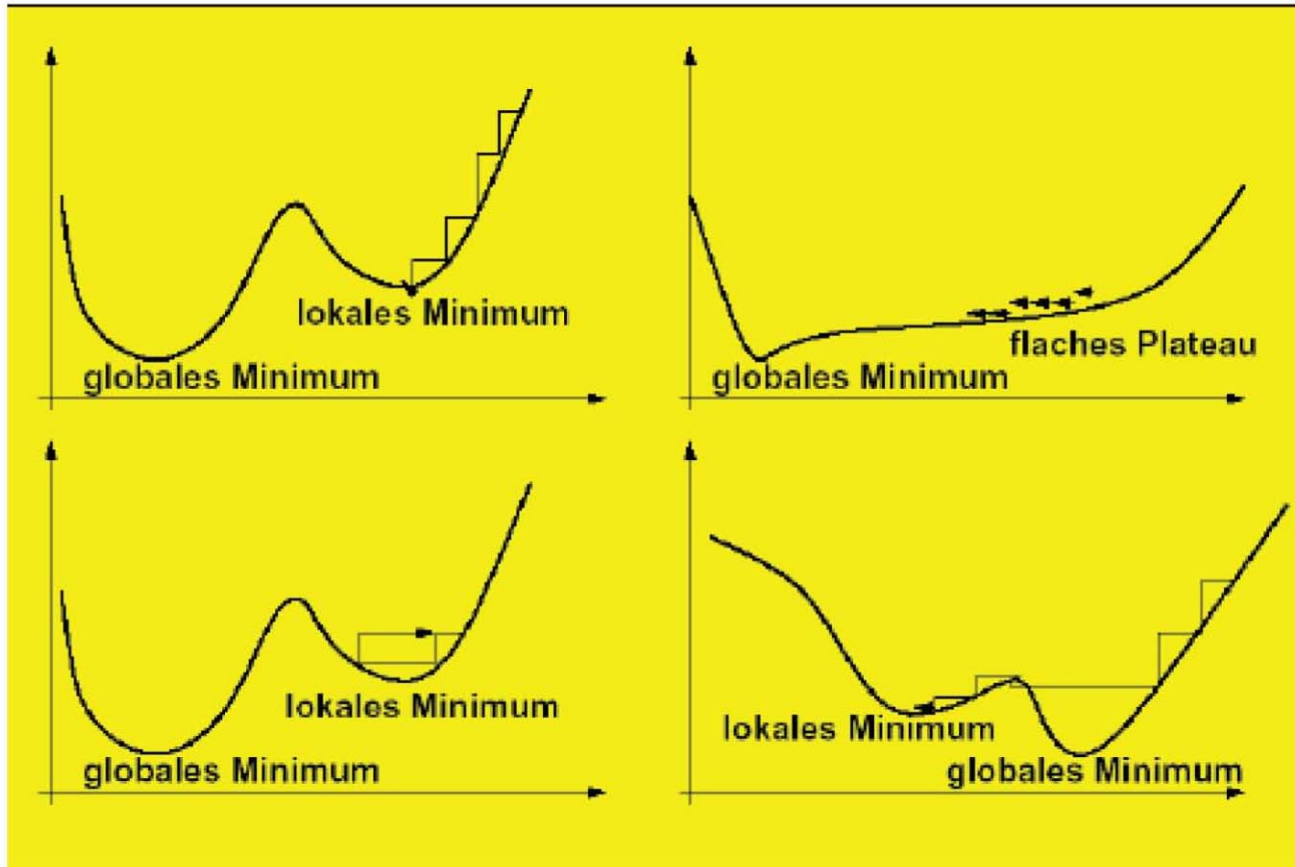
$$\Delta w_i = -\eta \partial E / \partial w_i$$

$$= -\eta \partial / \partial w_i \frac{1}{2} \sum_d (t_d - y_d)^2$$

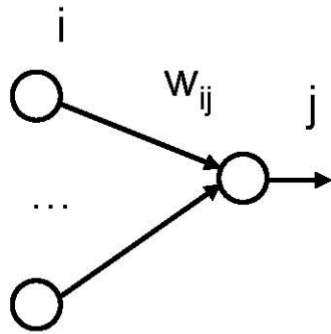
$$= -\eta \partial / \partial w_i \frac{1}{2} \sum_d (t_d - \sum_i w_i x_i)^2$$

$$= -\eta \sum_d (t_d - y_d) (-x_i)$$

Back-Propagation: Konvergenzprobleme



Backpropagation Algorithmus



Fall I: Neuron j -
Ausgabeneuron

Fehlerfunktion:

$$E(x, t) = \frac{1}{2} \sum_i (y_i - t_i)^2,$$

i – Nummer aller Ausgabeneuronen,

y_i – Ausgabe des Neurons i ,

t_i – gewünschte Ausgabe

$$\Delta W = -\eta \nabla E(W), \eta - \text{Lernrate}$$

$$\Delta w_{ij} = -\eta \cdot \frac{\partial E}{\partial w_{ij}}$$

$$y_j = f(\text{net}_j), \quad \text{net}_j = \sum_i w_{ij} y_i$$

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial y_j} \cdot \frac{\partial y_j}{\partial w_{ij}}$$

Falls Neuron j ein Ausgabeneuron ist:

$$\frac{\partial E}{\partial y_j} = \frac{\partial}{\partial y_j} \left(\frac{1}{2} \sum_j (y_j - t_j)^2 \right) = -(y_j - t_j)$$

$$\frac{\partial y_j}{\partial w_{ij}} = \frac{\partial y_j}{\partial \text{net}_j} \cdot \frac{\partial \text{net}_j}{\partial w_{ij}} = y_j \cdot (1 - y_j) \cdot y_i$$

Definieren wir:

$$\delta_j := \frac{\partial E}{\partial y_j} \cdot y_j \cdot (1 - y_j) - \text{Fehlersignal des Neurons } j$$

$$\delta_j = -(y_j - t_j) \cdot y_j \cdot (1 - y_j)$$

Dann:

$$\Delta w_{ij} = \eta \cdot \delta_j \cdot y_i$$

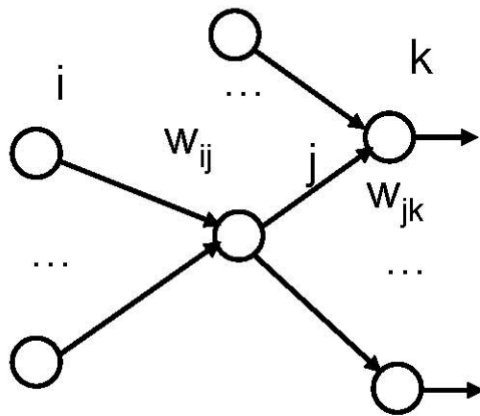
Kettenregel:

$$\frac{\partial f(g(x))}{\partial x} = \frac{\partial f}{\partial g} \cdot \frac{\partial g}{\partial x}$$

$$f_{\text{sigm}}' = f_{\text{sigm}} \cdot (1 - f_{\text{sigm}})$$

Backpropagation Algorithmus

Rekursiv:



Fall II:

j – verborgenes
Neuron

Neuron j – verborgenes Neuron,

Neurone i – seine Vorgänger,

Neurone k – seine Nachfolger, Ausgabeneurone

$$\frac{\partial E}{\partial y_j} = \sum_k \frac{\partial E}{\partial y_k} \cdot \frac{\partial y_k}{\partial net_k} \cdot \frac{\partial net_k}{\partial y_j}$$

$$net_k = \sum_j w_{jk} y_j \Rightarrow \frac{\partial net_k}{\partial y_j} = w_{jk}$$

$$\frac{\partial y_k}{\partial net_k} = y_k \cdot (1 - y_k)$$

$$\frac{\partial E}{\partial y_k} \cdot y_k \cdot (1 - y_k) := \delta_k$$

$$\frac{\partial E}{\partial y_j} = \sum_k \delta_k w_{jk}$$

$$\delta_j := \sum_k \delta_k w_{jk} \cdot y_j \cdot (1 - y_j) - \text{Fehlersignal des verborgenen Neurons } j$$