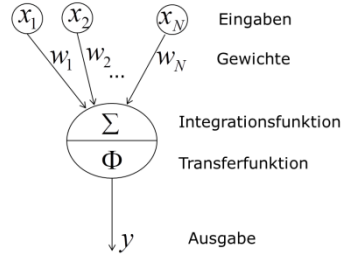

Computational Intelligence

# 3. MLP, Backpropagation

**Prof. Dr. Sven Behnke**

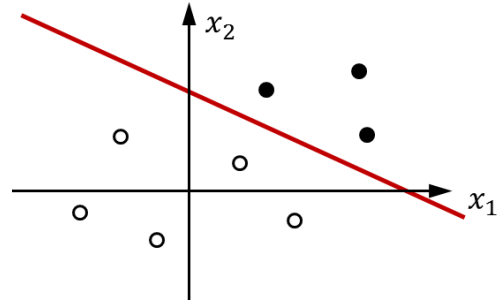
# Letzte Vorlesung

- Realisierung eines Neurons
  - Integrationsfunktion
  - Transfer-/Aktivierungsfunktion




$$y = \Phi\left(\sum_j^N w_j x_j\right)$$

- Überwachtes Training
  - Menge von Trainingsbeispielen $\{(x_1,t_1), \ldots, (x_p,t_p)\}$
  - Perzeptron-Lernalgorithmus

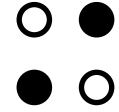


# Korrektheit des Perzeptron-Lernalgorithmus

- Der Perzeptron-Lernalgorithmus kann nun wie folgt interpretiert werden:
  - Auf Z soll das Perzeptron mit 1 antworten, also $\boldsymbol{zw}>0$ für ein geeignetes $\boldsymbol{w}$ gelten. Wir beginnen mit einem beliebigen $\boldsymbol{w_0}$.
  - Ist $\boldsymbol{z} \in Z$ ein Element, das noch nicht korrekt interpretiert wird ($\boldsymbol{zw}<=0$), dann erzeugt die Lern-Regel ein neues $\boldsymbol{w}_{neu} := \boldsymbol{w}_{alt} + \boldsymbol{z}$ und beim nächsten Versuch mit demselben $\boldsymbol{z}$ ist $\boldsymbol{zw}_{neu} = \boldsymbol{zw}_{alt} + \boldsymbol{zz} > \boldsymbol{zw}_{alt}$.
  - Das Verfahren beginnt also mit einem beliebigen erweiterten Gewichtsvektor $\boldsymbol{w_0}$ und addiert (subtrahiert) sukzessive erweiterte Eingaben $\boldsymbol{z}$, bei denen die Ausgabe noch nicht korrekt ist.

# Perzeptron-Konvergenz-Satz

- Konvergenz garantiert, wenn Problem lösbar (Rosenblatt 1962)
- Beispiel für nicht linear trennbare Mengen:



- **Satz:**
  Wenn das Perzeptron eine Klasseneinteilung überhaupt lernen kann, dann lernt es diese mit der Perzeptron-Lernregel in endlich vielen Schritten.
- Problem:
  Wenn der Lernerfolg bei einem Perzeptron ausbleibt, kann nicht direkt erkannt werden, ob
  1. das Perzeptron die Klassifikation prinzipiell nicht lernen kann oder
  2. der Lernalgorithmus mit seinen endlich vielen Schritten
     noch nicht fertig geworden ist,

  denn der Satz als reiner Existenzsatz gibt keinen Anhaltspunkt über eine obere Schranke für die Anzahl von Lernschritten.

# Beweis

- Nach Voraussetzung gibt es ein $\boldsymbol{w}^*$ mit $\boldsymbol{w}^*\boldsymbol{z}>0$ für alle $\boldsymbol{z}\in Z$ und wir beginnen den Algorithmus mit $\boldsymbol{w}_0=0$.
- Sei **a** das Minimum aller $\boldsymbol{w}^*\boldsymbol{z}>0$ und **M** das Maximum aller $\boldsymbol{z}^2$.
- $\boldsymbol{w}_i = \boldsymbol{z}_1 + .. + \boldsymbol{z}_i$, also ist $\boldsymbol{w}^*\boldsymbol{w}_i = \boldsymbol{w}^*\boldsymbol{z}_1 + .. + \boldsymbol{w}^*\boldsymbol{z}_i >= i\cdot a$.
- Wegen $(\boldsymbol{w}^*\boldsymbol{w}_i)^2 <= \boldsymbol{w}^{*2}\boldsymbol{w}_i^2$ folgt $\boldsymbol{w}_i^2 >= i^2\cdot a^2/\boldsymbol{w}^{*2}$
- Andererseits ist für alle k<=i stets $\boldsymbol{w}_{k-1}\boldsymbol{z}_k < 0$
- Also $\boldsymbol{w}_k^2 = (\boldsymbol{w}_{k-1}+\boldsymbol{z}_k)^2 = \boldsymbol{w}_{k-1}^2 + 2*\boldsymbol{w}_{k-1}\boldsymbol{z}_k + \boldsymbol{z}_k^2 < \boldsymbol{w}_{k-1}^2 + \boldsymbol{z}_k^2$
- Also ist $\boldsymbol{w}_i^2 < \boldsymbol{z}_1^2 + ... + \boldsymbol{z}_i^2 < i\cdot M$
- Es folgt $i\cdot M > i^2\cdot a^2/\boldsymbol{w}^{*2}$ und damit $i < M\cdot\boldsymbol{w}^{*2} / a^2$

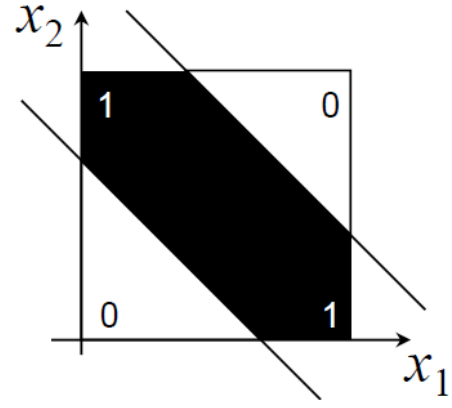
**D.h. der Algorithmus terminiert in endlich vielen Schritten.**

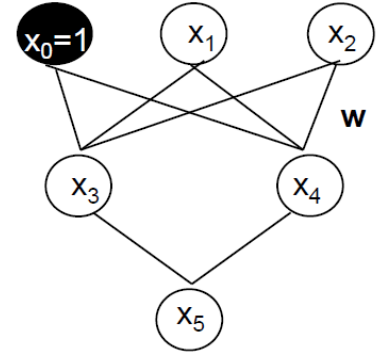
Aber: Ohne Kenntnis von $\boldsymbol{w}^*$ kennen wir weder $\boldsymbol{w}^{*2}$ noch a, also können wir die obere Schranke von i nicht aus dem Beweis bestimmen!

Frage: Wann kann der Algorithmus lange laufen?

# XOR-Problem

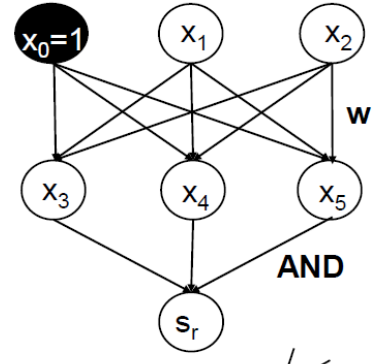
- Die boolesche Funktion XOR stellt ein nichtlinear separierbares Problem dar, weil sich die beiden Klassen auf Diagonalen gegenüberliegen.
- Mit einer Zwischenschicht lassen sich aber zwei trennende Hyperebenen definieren, zwischen denen die eine der Klassen lokalisiert werden kann.
- Komplexere Klassifikationen lassen sich durch Einfügen von Zwischenschichten realisieren.

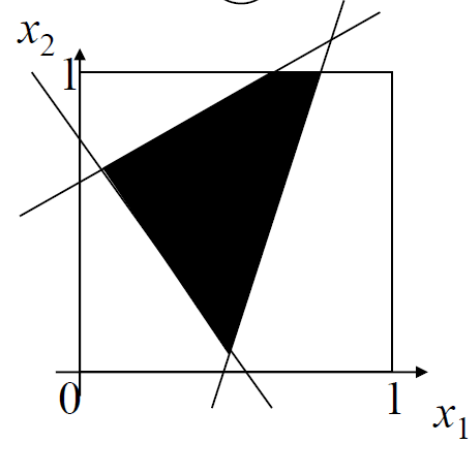


# Zweistufiges Perzeptron

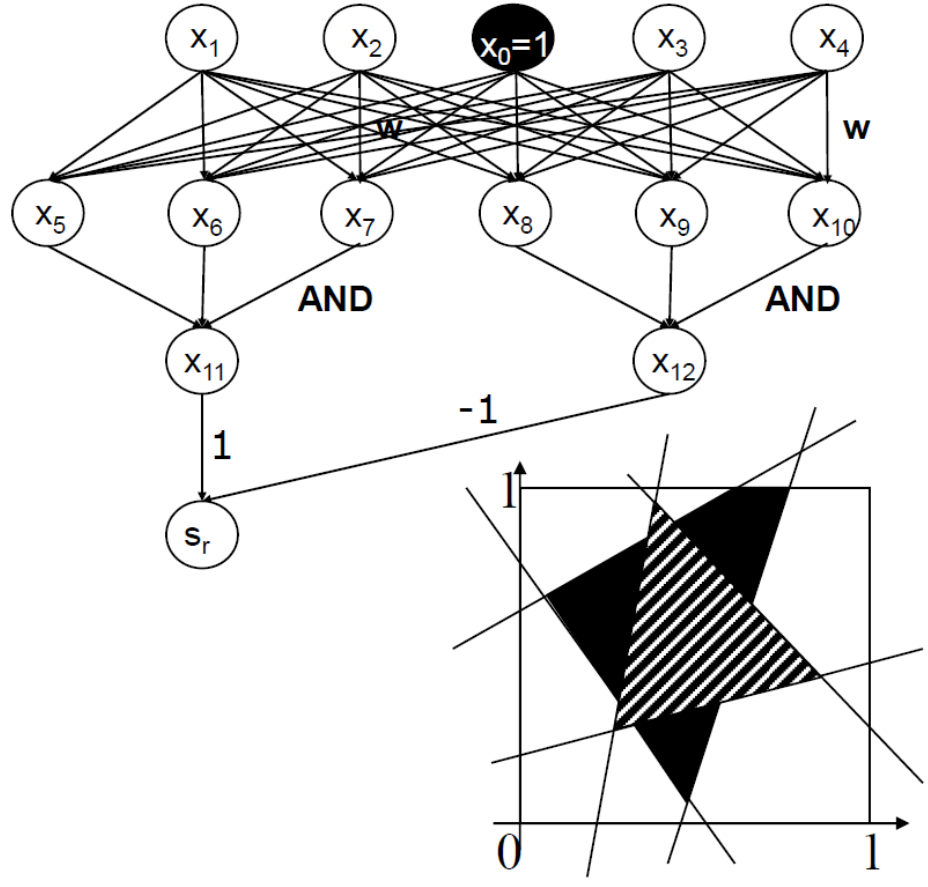
- Wenn man an ein Perzeptron mit M Ausgabe-Neuronen ein neues Ausgabe-Neuron mit AND-Verbindungen (Gewicht 1, Schwellwert M-0,5) anhängt, erhält man ein **konvexes M-flach** (den Durchschnitt von M Halbräumen) als akzeptierten (Ausgabe 1) Bereich.



# Dreistufiges Perzeptron

- Mit einer weiteren Stufe durch AND NOT (Gewicht 1,-1, Schwellwert 0,5) angehängt, kann man sogar nicht zusammenhängende Bereiche als akzeptierten Bereich modellieren. (In dem Beispiel der sichtbare dunkle Bereich)



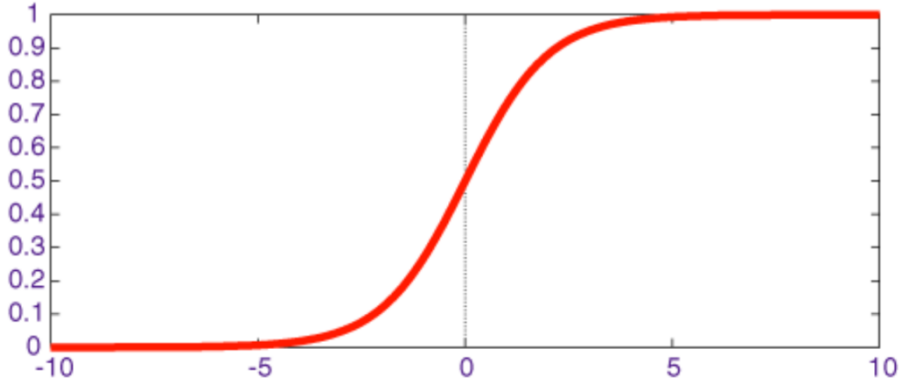
# Problem: Wie trainiert man mehrere Schichten?

- Übersicht
  - Definieren einer Fehlerfunktion *E(w)*
  - Leite Fehler nach Gewichten *w* ab
  - Ändere w so, dass der Fehler *E(w)* kleiner wird
- Zuvor müssen wir  dafür sorgen dass wir ableiten **können:**

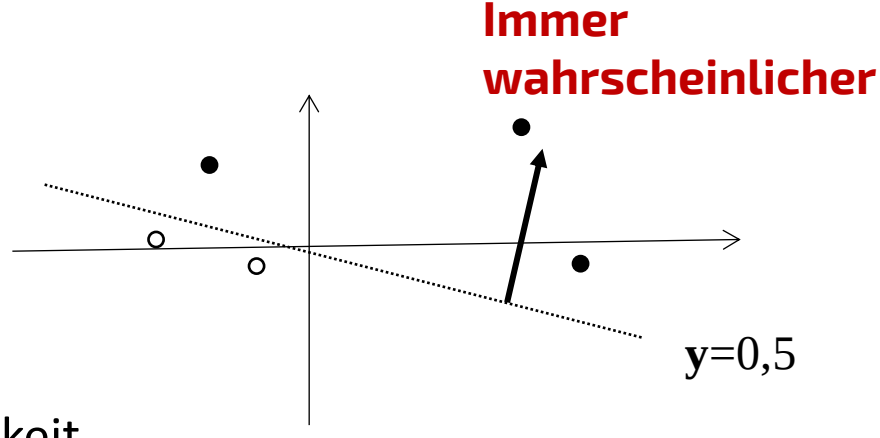
$$y = \text{sign}\left(\vec{w}^T \vec{x} - \theta\right) = \text{sign}\left(\sum_{i=1}^{n} w_i x_i - \theta\right)$$

  - Sign ist nicht stetig! → Approximation mit Sigmoider Funktion

# Sigmoide Transferfunktion




$$y = \Phi(a) = \frac{1}{1+e^{-a}}$$




- Ausgaben begrenzt auf [0,1]
- Quasi-linear um Nettoinput a=0
- Mögliche Interpretation: Wahrscheinlichkeit

# Überwachtes Lernen

**Ziel:** finde für eine durch Beispiele gegebene Funktion y=f(x) die Gewichte $w_{ij}$ so, dass f möglichst gut durch die Netzwerkfunktion approximiert wird

Sei $\{(x_1,t_1), \ldots, (x_p,t_p)\}$ eine Menge von Trainingsbeispielen

Sei *w* der Gewichtsvektor

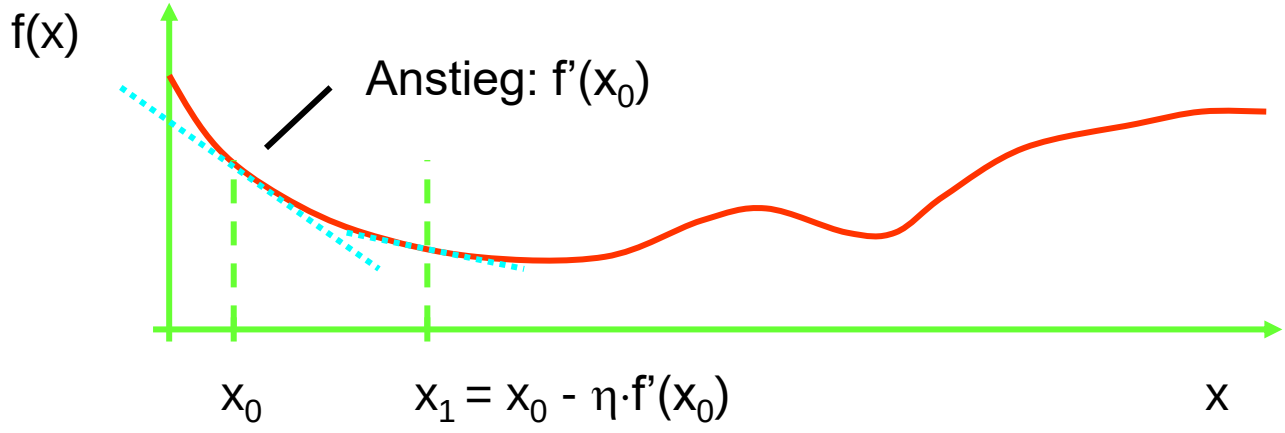
Das Netzwerk berechnet zur Eingabe $x_i$ die Ausgabe $y_i$

Optimiere z.B. **quadratische Fehlerfunktion**

$$E(w) = 1/2 \sum_{i=1}^{p} (y_i - t_i)^2$$

# Gradientenabstieg

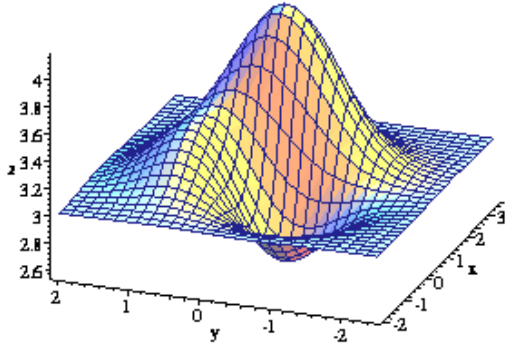
- **1D-Beispiel**: minimiere f(x)

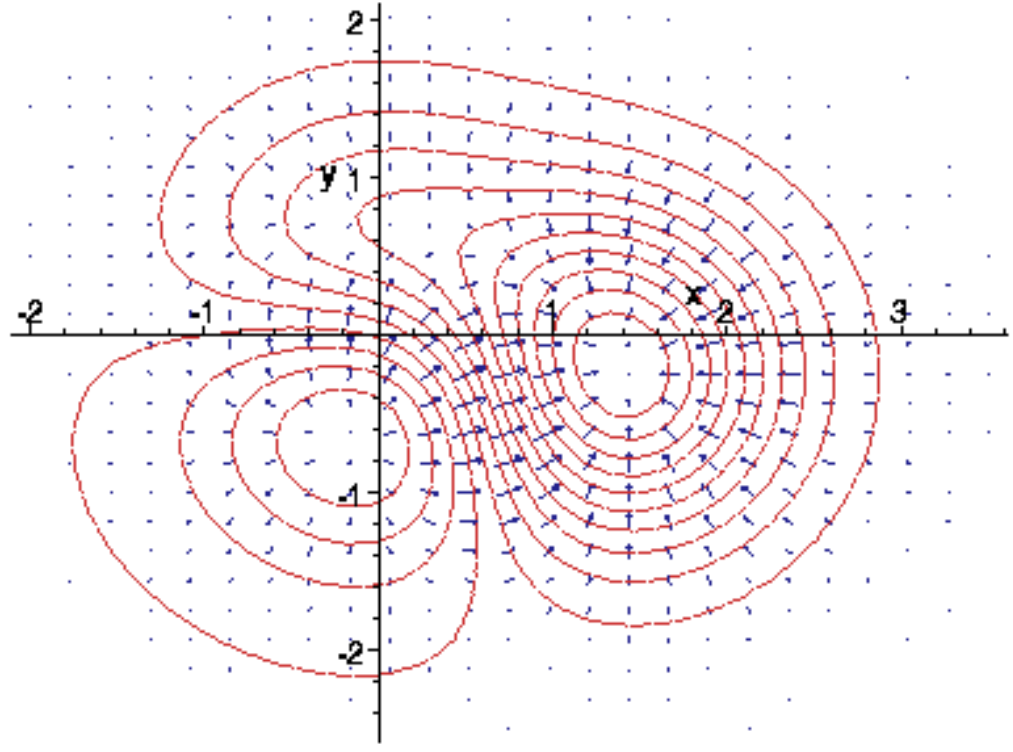



Iteriere bis $f'(x_i)$ verschwindet!

# Gradientenabstieg

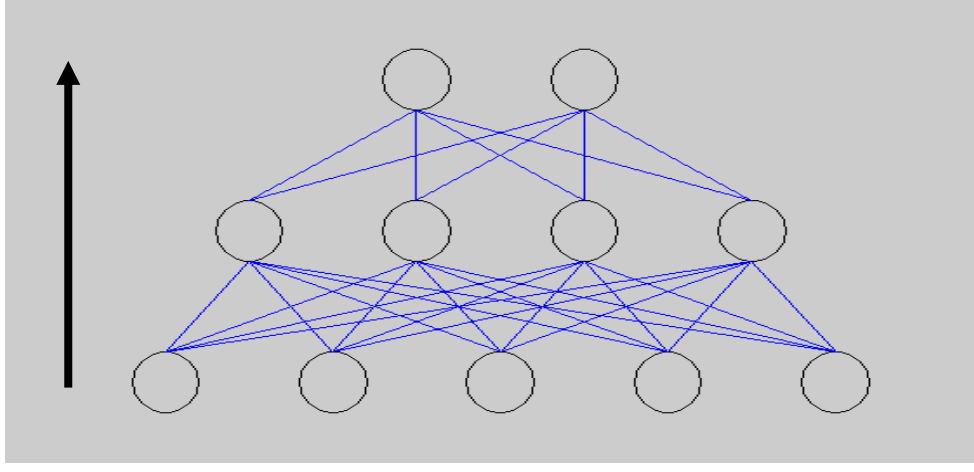
- 2D-Beispiel
- Folge den Pfeilen



# Mehrebenen-Perceptron (MLP)

- Zwei (oder mehrere) Schichten



Ausgabe-Neuronen
(linear oder sigmoid)

Verdeckte Neuronen
(typisch sigmoid)

Eingaben

# Backpropagation als Gradientenverfahren

- Definiere (quadratischen) Fehler (für Muster $l$):

$$E_l = \sum_{k=1}^{m} \left( y_k - t_k \right)^2$$

- Minimiere Fehler, d.h. ändere Gewichte in Richtung des negativen Gradienten

$$\Delta w_{ij} \propto -\frac{\partial E_l}{\partial w_{ij}}$$

(partielle Ableitung der Fehlerfunktion nach dem Gewicht)

- Kettenregel ergibt Backpropagation

# Gradientenabstieg im Gewichtsraum

- Gradient der Fehlerfunktion

$$\nabla E = \left( \frac{\partial E}{\partial w_1}, \ldots, \frac{\partial E}{\partial w_m} \right)$$

- Korrektur der Gewichte macht Fehler kleiner

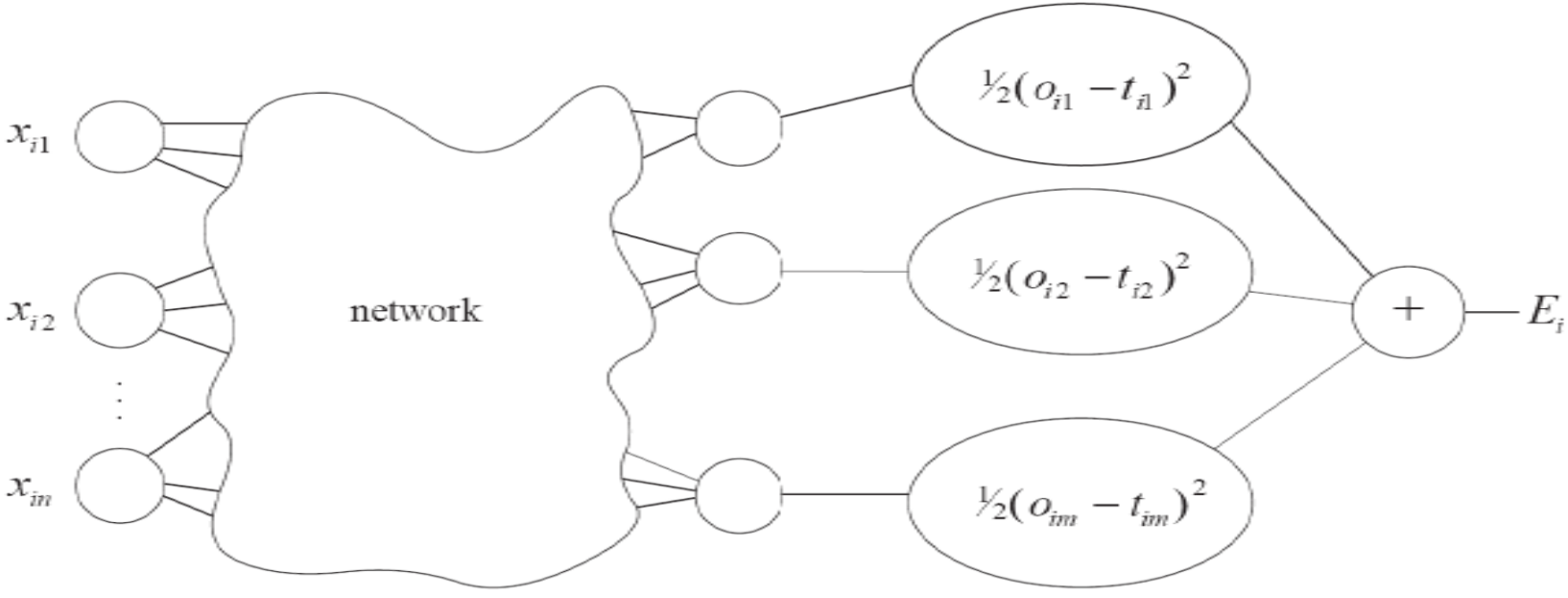
$$w_i^{new} = w_i + \Delta w_i$$

mit

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

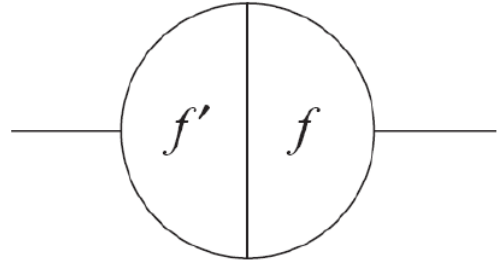
# Berechnung der Fehlerfunktion

Erweitere das Netz:

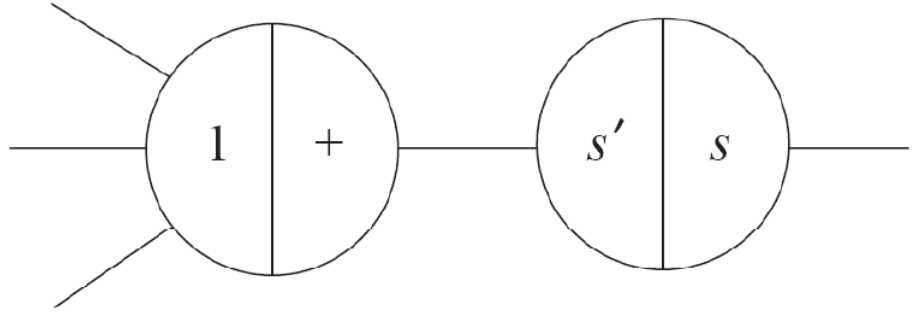


# Berechnung der Ableitung

- Erweiterung der Knoten



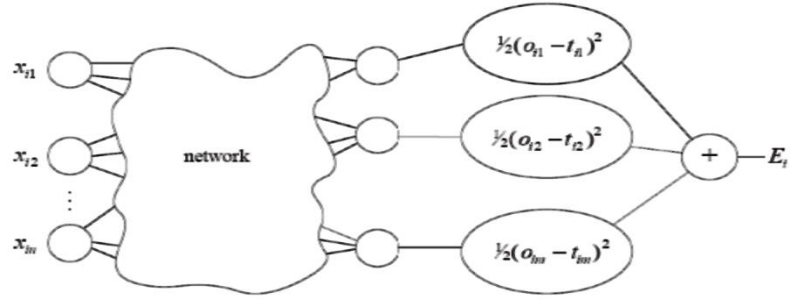

- Trennung von Integrations- und Aktivierungsfunktion



# Dynamische Programmierung

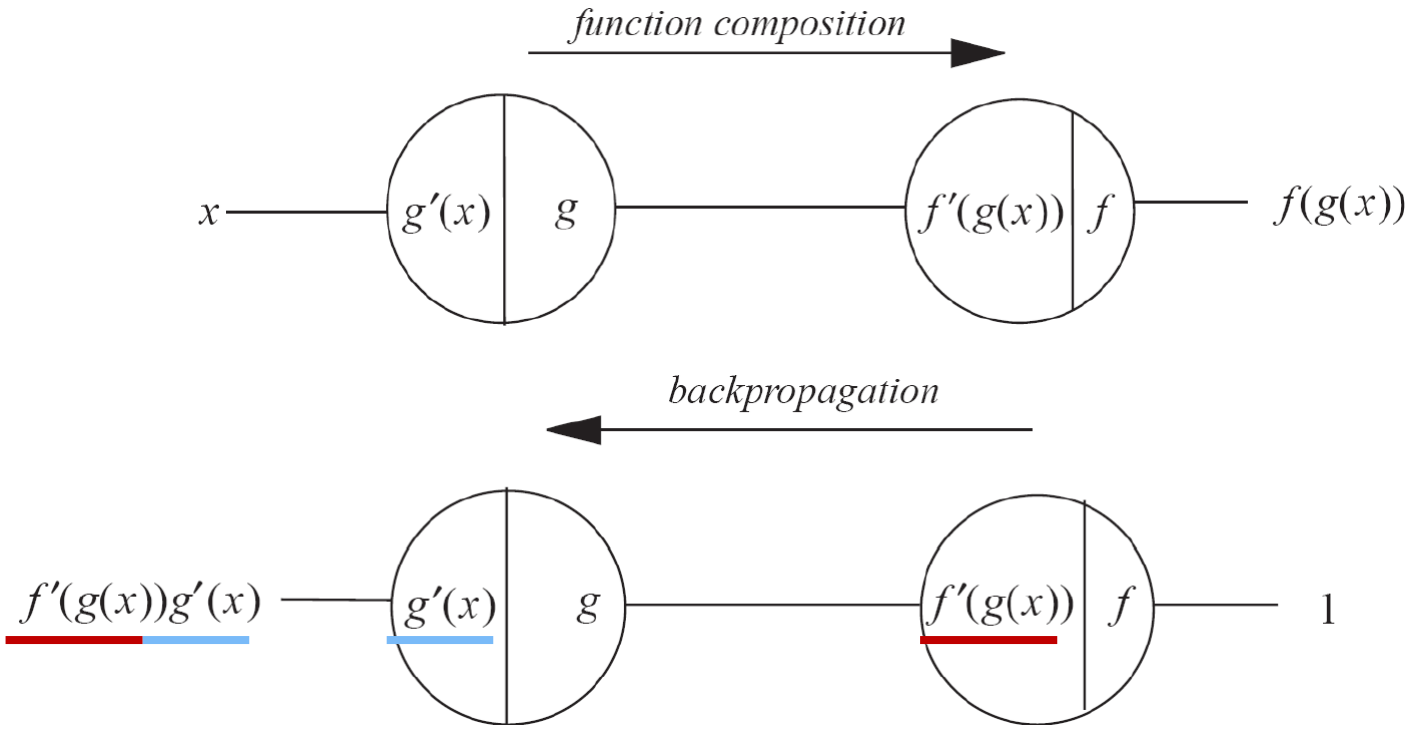
Berechnung erfolgt in zwei Schritten:

- Schritt 1 (Feed-forward):
  - Information fließt von links nach rechts
  - In jedem Knoten wird die Funktion und die Ableitung berechnet und gespeichert
  - Nur der Funktionswert wird weiter gereicht



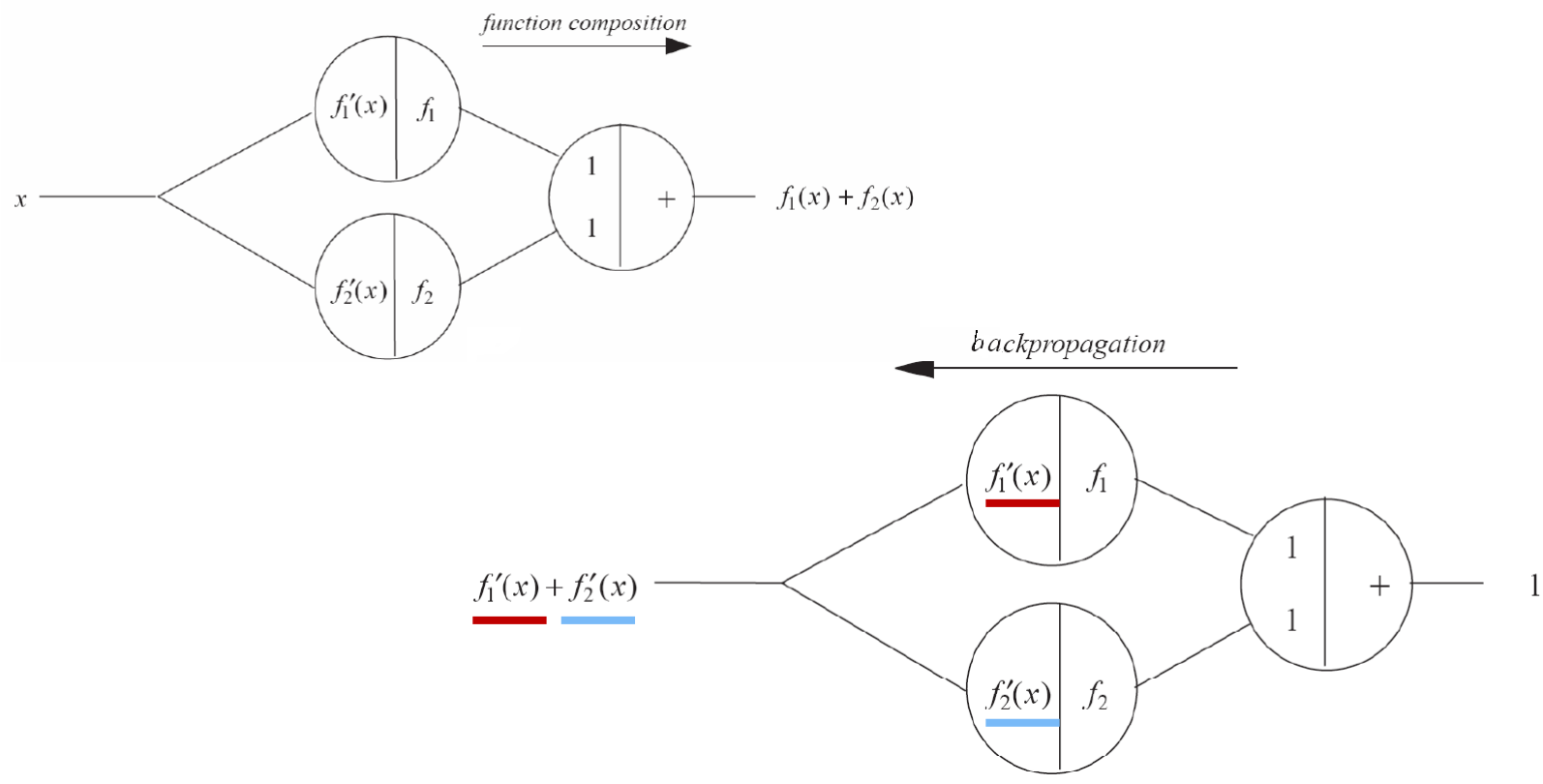

- Schritt 2 (Backpropagation):
  - Das Netzwerk läuft rückwärts und berechnet aus den gespeicherten Teilableitungen die Ableitung der Fehlerfunktion
- Bei beiden Schritten werden Zwischenergebnisse (Aktivitäten, Fehlergradienten) mehrfach wiederverwendet => effiziente Berechnung.

# Backpropagation: Komposition



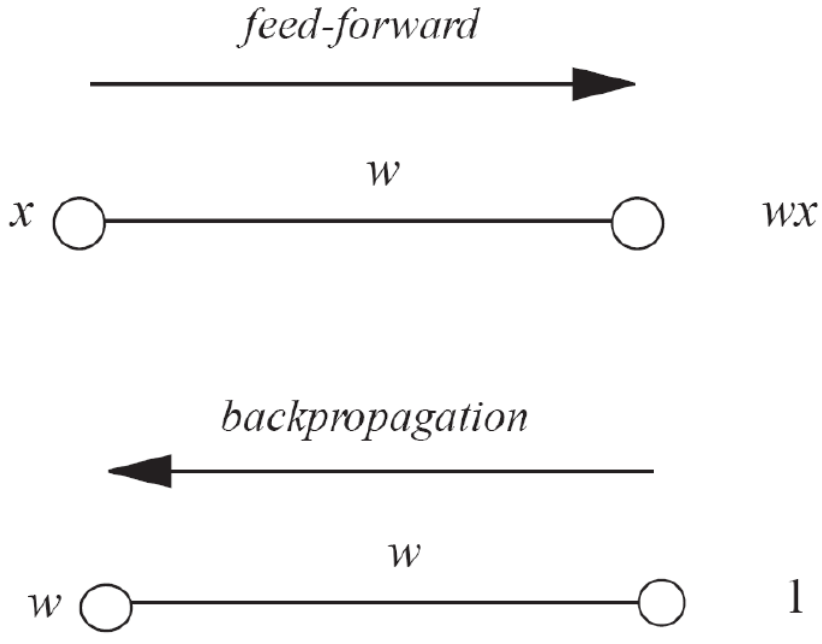

**Kettenregel:** Ableitung verketterter Funktion ist Produkt von innerer und äußerer Ableitung

# Backpropagation: Addition



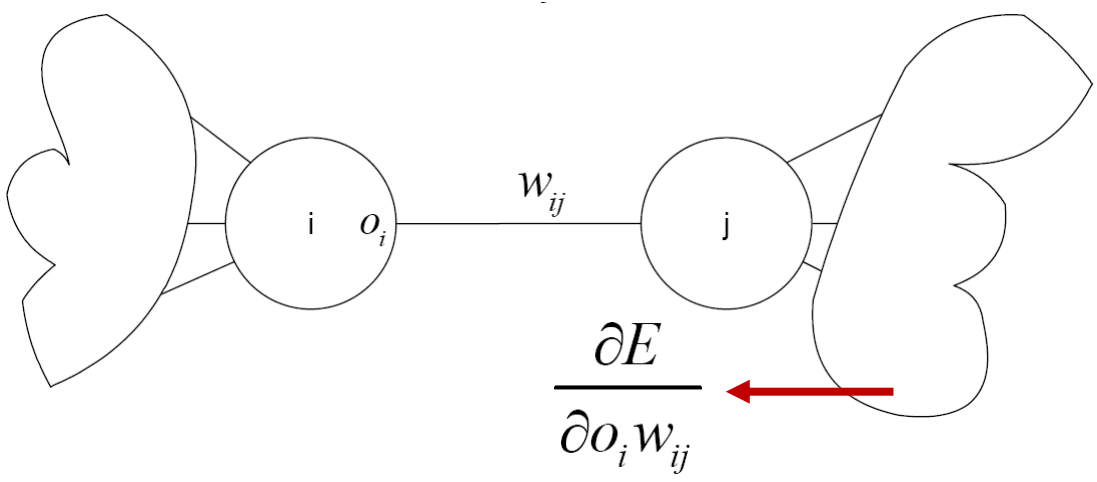

**Summenregel:** Ableitung der Summe von Funktion ist Summe der Einzelableitungen

# Backpropagation: Gewichtete Kanten




**Multiplikation mit einer Konstanten:** Ableitung des konstanten Vielfachen einer Funktion ist Vielfaches der Ableitung der Funktion

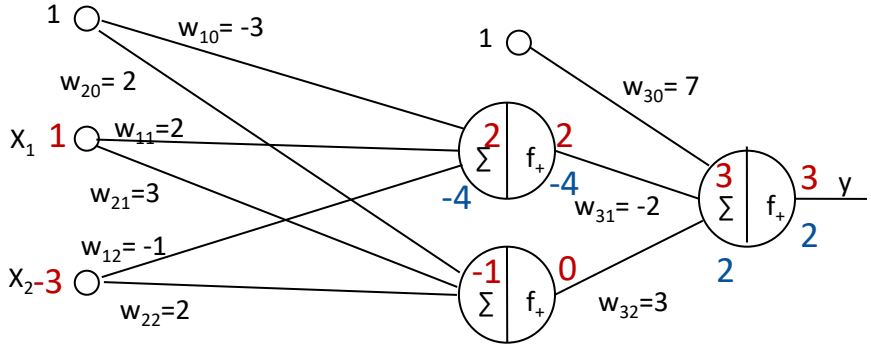
# Ableitung bezüglich Gewicht




- Da $o_i$ konstant ist, gilt: $\frac{\partial E}{\partial w_{ij}} = o_i \frac{\partial E}{\partial o_i w_{ij}}$
- Bei Backpropagation wird der zurückpropagierte Fehler $\delta_j := \frac{\partial E}{\partial o_i w_{ij}}$ zur Wiederverwendung gespeichert

# Beispiel

- Transferfunktion $f_+(a) = \max(0, a)$.
- Eingabe: $x_1=1$, $x_2=-3$
- Berechnung der **Aktivität**
- Gewünschte Ausgabe: t=1
- Propagation des **Gradienten** zu Gewicht $w_{12}$



# Gewichtskorrektur

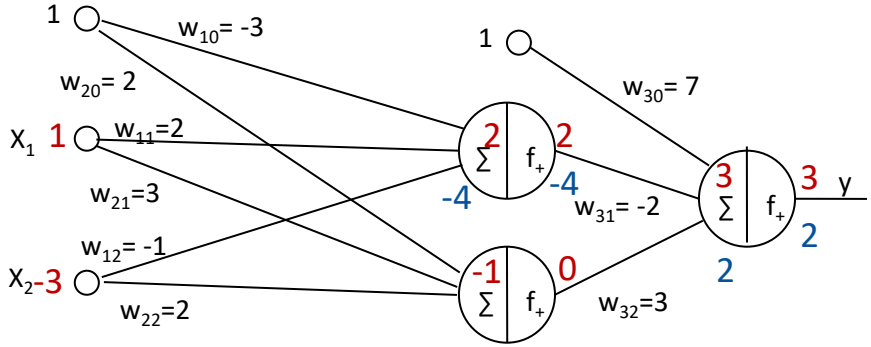
- Die Korrektur eines Gewichts ergibt sich demnach das Produkt aus Eingabe in das Gewicht und zurückpropagiertem Fehler

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i} = -\eta o_i \delta_j$$

- Für mehrere Trainingsbeispiele:
  - Jede Korrektur einzeln durchführen (Online-Verfahren) oder
  - Alle Korrekturen gesammelt durchführen (Batch-Verfahren)

# Beispiel cont.

- Transferfunktion $f_+(a) = \max(0, a)$.
- Eingabe: $x_1=1$, $x_2=-3$
- Berechnung der **Aktivität**
- Gewünschte Ausgabe: t=1
- Propagation des **Gradienten** zu Gewicht $w_{12}$



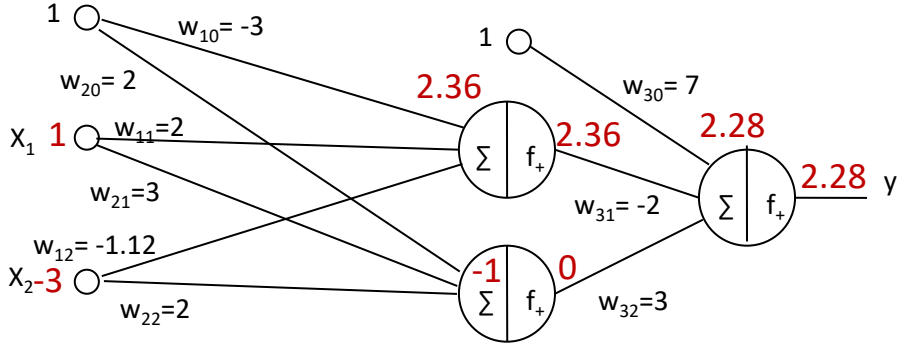

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i} = -\eta o_i \delta_j$$

$$w_{12} \leftarrow -1 - 0.01 \cdot \mathbf{-3} \cdot \mathbf{-4}$$

$$w_{12} = -1.12$$

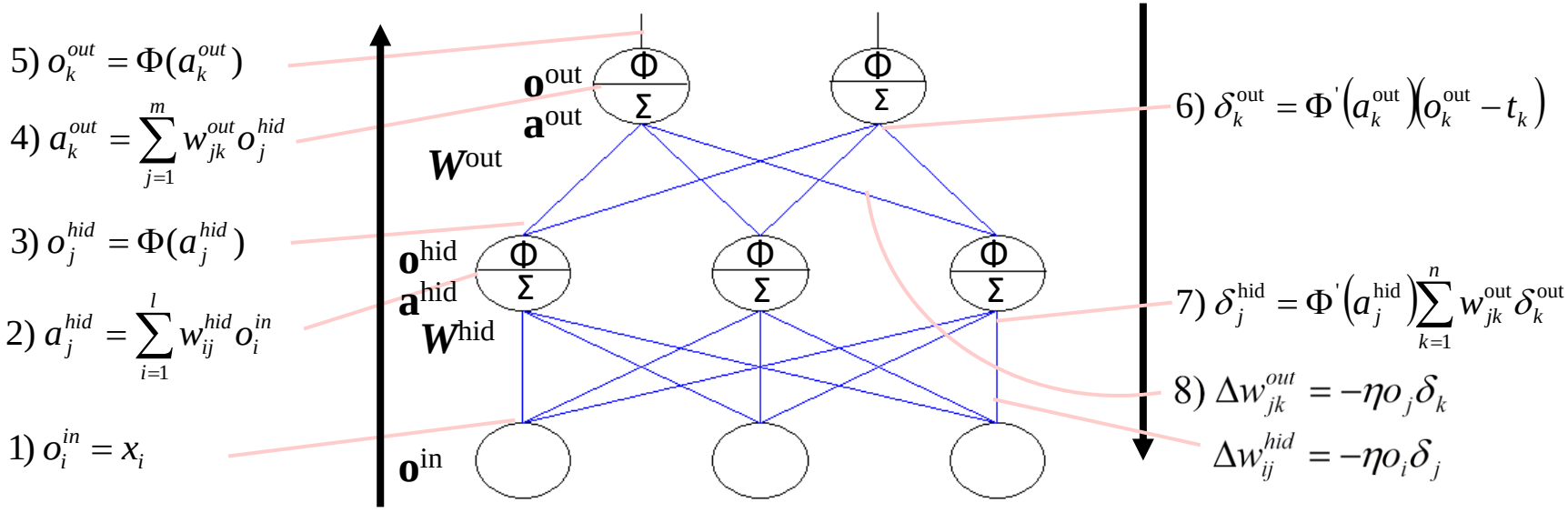
# Beispiel Kontrollrechung

- Transferfunktion $f_+(a) = \max(0, a)$.
- Eingabe: $x_1=1$, $x_2=-3$
- Erneute Berechnung der **Aktivität**
- Gewünschte Ausgabe: t=1
- Ausgabe hat sich in Richtung der gewünschten Ausgabe bewegt



# Backpropagation

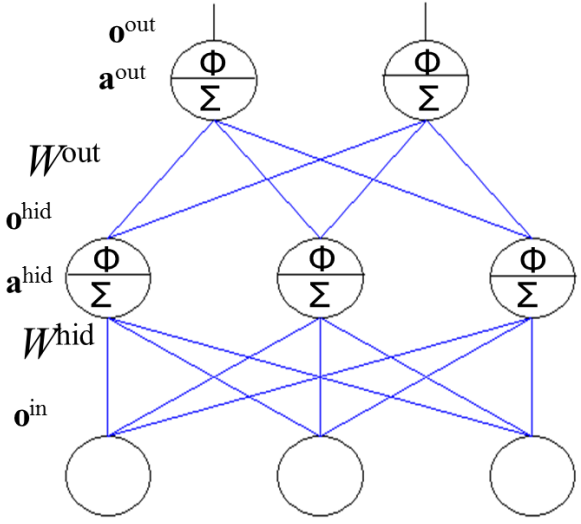
- Vorwärtspropagation von Aktivität
- Rückwärtspropagation von Fehler
- Gewichtsanpassung durch Gradientenabstieg



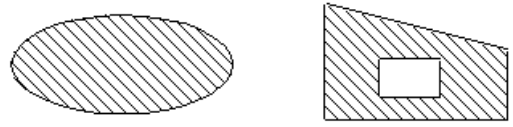

- Zufällige Gewichtsinitialisierung
- Präsentation der Eingabemuster zufällig oder als Block

# MULTI-LAYER PERZEPTRON (MLP)

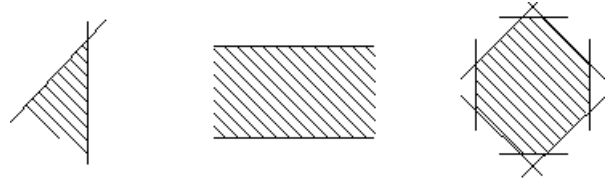
- MLPs können Probleme lösen, die nicht linear separierbar sind
- Verdeckte Knoten bilden Eingaberaum in einen Raum ab, der von den Ausgabeneuronen linear getrennt werden kann




- Ausgabeknoten erzeugen Regionen

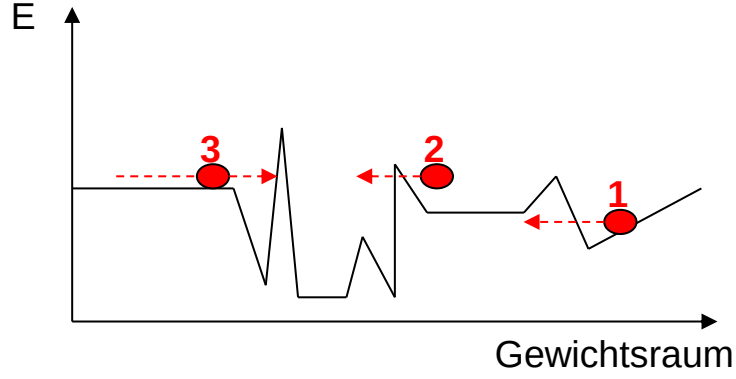


- Verdeckte Knoten erzeugen Trennebenen



# BACKPROPAGATION – VARIANTEN

- Fehlerraum häufig schwierig zu explorieren: Lokale Minima und flache Gebiete




1. Hohe Lernrate: Große Schritte in Richtung des Gradienten

2. Momentum: Addiere Bruchteil des letzten Updates $\Delta w_{ij}^{t} = -\eta o_i \delta_j + \alpha \Delta w_{ij}^{t-1}$

3. Additive Konstante auf Ableitung der Transferfunktion $\delta_k^{\text{out}} = \left(\Phi'\left(a_k^{\text{out}}\right) + \mu\right)\left(o_k^{\text{out}} - t_k\right)$
   => Gradient auch bei Saturierung der Transferfunktion

# RESILIENT PROPAGATION (RPROP)

- Individuelle Lernrate für jedes Gewicht
- Betrachte Vorzeichen aufeinander folgender Gradienten
- Erhöhe Lernrate langsam bei gleichem Vorzeichen
- Erniedrige Lernrate schnell bei Vorzeichenwechsel

$$\Delta_{ij}^{(t)} = \begin{cases} \eta^+ * \Delta_{ij}^{(t-1)} & , \text{ if } \frac{\partial E}{\partial w_{ij}}^{(t-1)} * \frac{\partial E}{\partial w_{ij}}^{(t)} > 0 \\ \eta^- * \Delta_{ij}^{(t-1)} & , \text{ if } \frac{\partial E}{\partial w_{ij}}^{(t-1)} * \frac{\partial E}{\partial w_{ij}}^{(t)} < 0 \\ \Delta_{ij}^{(t-1)} & , \text{ else} \end{cases}$$

$$\eta^+ = 1{,}2 \ , \ \eta^- = 0{,}5$$

- Gewichtsänderung ignoriert Betrag des Gradienten

$$\Delta w_{ij}^{(t)} = \begin{cases} -\Delta_{ij}^{(t)} & , \text{ if } \frac{\partial E}{\partial w_{ij}}^{(t)} > 0 \\ +\Delta_{ij}^{(t)} & , \text{ if } \frac{\partial E}{\partial w_{ij}}^{(t)} < 0 \\ 0 & , \text{ else} \end{cases}$$

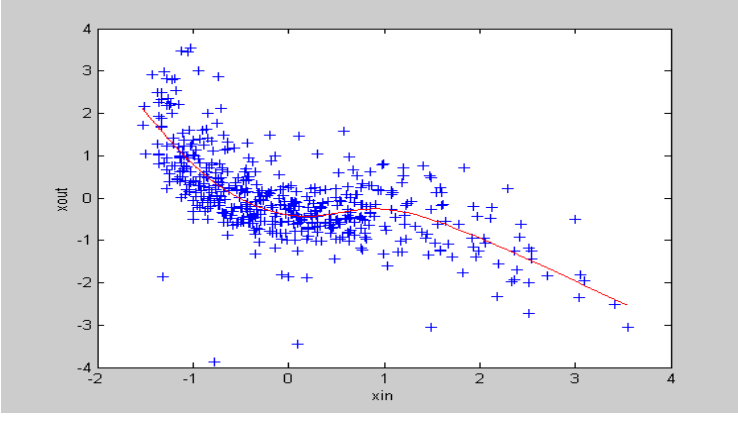
[Riedmiller & Braun, 1992]

# DIE STOCHASTISCHE SICHT DES ÜBERWACHTEN LERNENS

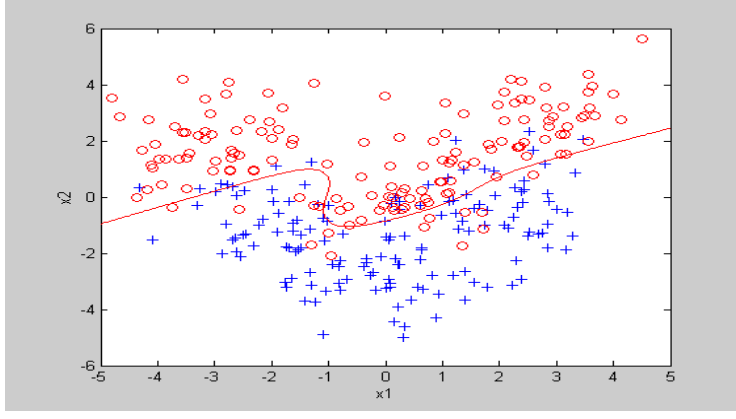
- Realdaten sind stochastisch (mit Rauschen/Streuungen versehen)

- 2 Typen von Problemen:

Regression




Klassifikation



# LINEARE REGRESSION

- Training linearer neuronaler Netze mit quadratischem Fehler ist lineare Regression
- 1D-Fall:

$$t^p = ax^p + b + \varepsilon$$

- Allgemein:

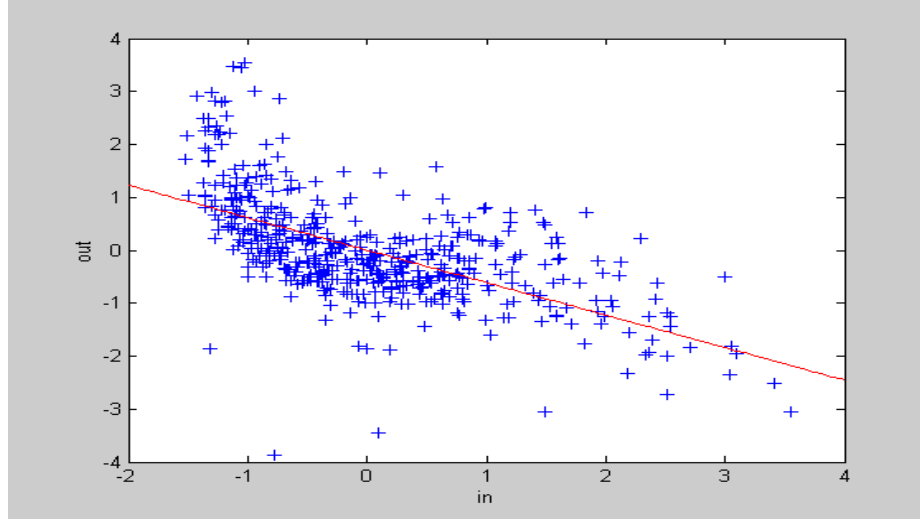
Rauschen

$$t^p = \mathbf{W}\mathbf{x}^p + w_0 + \varepsilon$$

Abhängige Variablen („target“)

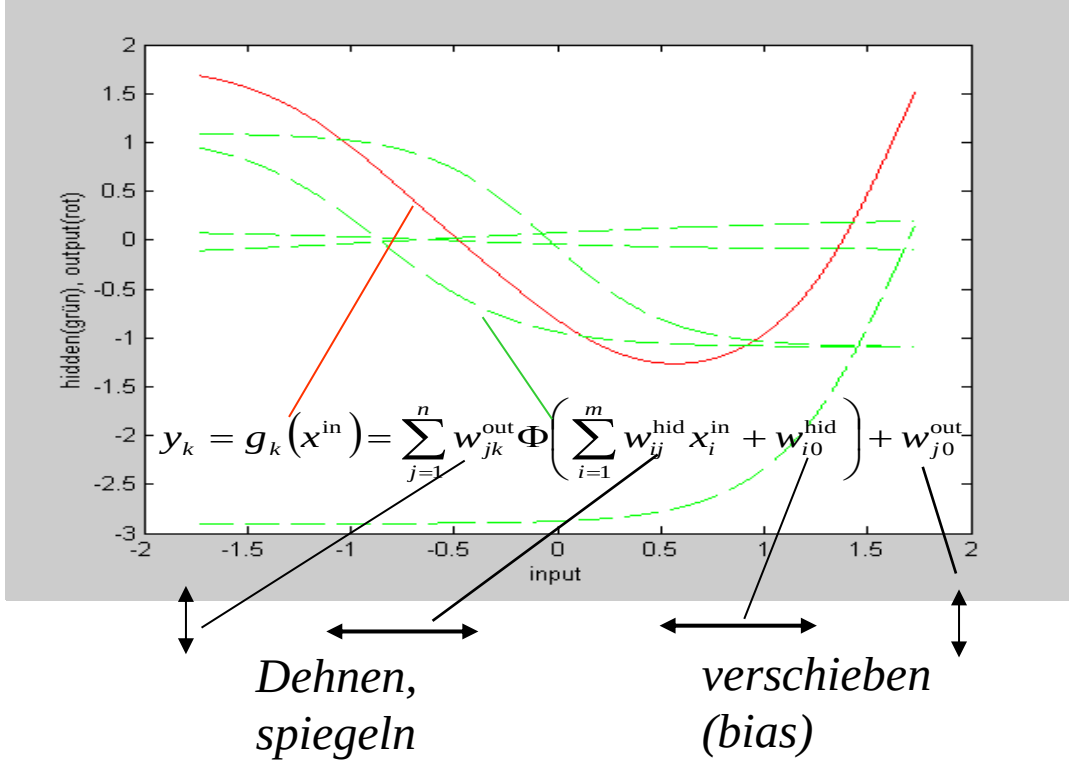
unbhängige Variablen

Wenn $w_0, b = 0$: Korrelation



# MLP ALS UNIVERSALER FUNKTIONSAPPROXIMATOR

- Bsp: 1 Input, 1 Output,
  5 Hidden Units
- MLP kann beliebige Funktionen annähern (Hornik et al. 1991)
- Überlagerung von verschobenen, skalierten Sigmoiden
- Komplexität durch Zusammenspiel vieler einfacher Elemente



# MAXIMUM-LIKELIHOOD-TRAINING

- Parametrische Ausgabeverteilung $p(\boldsymbol{y} \mid \boldsymbol{x}; \boldsymbol{\theta})$
- Kostenfunktion $J(\boldsymbol{\theta}) = -\log P(y \mid \boldsymbol{x})$

$$J(\boldsymbol{\theta}) = -\mathbb{E}_{\mathbf{x},\mathbf{y}\sim\hat{p}_{\text{data}}} \log p_{\text{model}}(\boldsymbol{y} \mid \boldsymbol{x})$$

- Normalverteilte Ausgaben

$$p_{\text{model}}(\boldsymbol{y} \mid \boldsymbol{x}) = \mathcal{N}(\boldsymbol{y}; f(\boldsymbol{x}; \boldsymbol{\theta}), \boldsymbol{I})$$

=> Quadratische Kostenfunktion

$$J(\theta) = \frac{1}{2}\mathbb{E}_{\mathbf{x},\mathbf{y}\sim\hat{p}_{\text{data}}} ||\boldsymbol{y} - f(\boldsymbol{x}; \boldsymbol{\theta})||^2 + \text{const}$$

- Angemessen für Funktionsapproximation (kontinuierliche Ausgaben)
- Ungeeignet für sigmoide Ausgabe-Transferfunktion, da Saturierung => Gradient≈0

[Goodfellow et al. 2016]

# BINÄRE KLASSIFIKATION: SIGMOIDE AKTIVIERUNG

- Gewünschte Ausgabe 0 oder 1: $P(y = 1 \mid \boldsymbol{x})$

[Goodfellow et al. 2016]

- Benötigt sigmoide Transferfunktion

$$\hat{y} = \sigma\left(\boldsymbol{w}^\top \boldsymbol{h} + b\right) \qquad z = \boldsymbol{w}^\top \boldsymbol{h} + b$$

- Bernoulli-Verteilung:

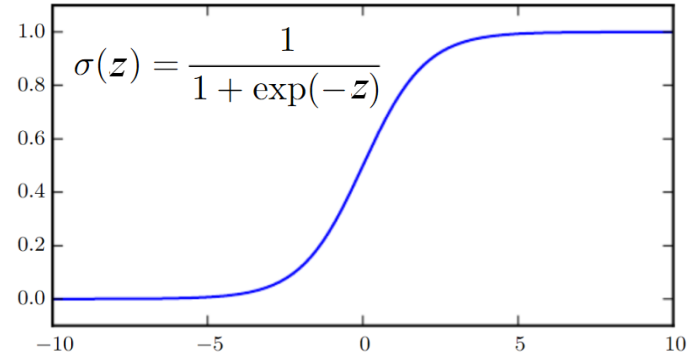
$$P(y) = \frac{\exp(yz)}{\sum_{y'=0}^{1} \exp(y'z)}$$
$$= \sigma\left((2y-1)z\right)$$

- Kostenfunktion:

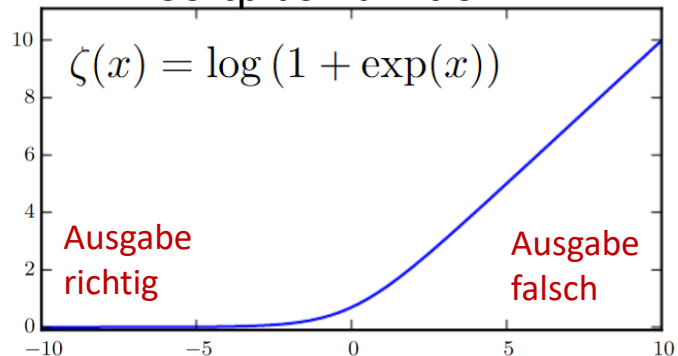
$$J(\boldsymbol{\theta}) = -\log P(y \mid \boldsymbol{x})$$
$$= -\log \sigma\left((2y-1)z\right)$$
$$= \zeta\left((1-2y)z\right)$$

  - Mindert den Gradient für falsche Ausgaben kaum

Sigmoide Transferfunktion

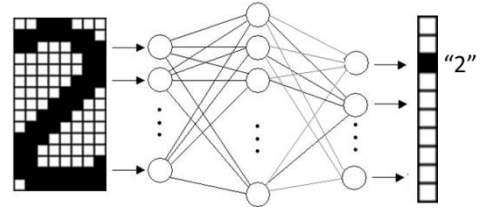



Softplus-Funktion



# N-ÄRE KLASSIFIKATION: SOFTMAX-AKTIVIERUNG

- N Klassenwahrscheinlichkeiten: $\hat{y}_i = P(y = i \mid \boldsymbol{x})$
  - Ausgabe 1 an der richtigen Stelle
  - Alle anderen Ausgaben sollen 0 sein




- Normalisierung der Klassenwahrscheinlichkeiten durch Softmax
  - Unnormalisierte Log-Klassenwahrscheinlichkeiten: $\boldsymbol{z} = \boldsymbol{W}^\top \boldsymbol{h} + \boldsymbol{b}$
  - Normalisierung auf Summe 1:

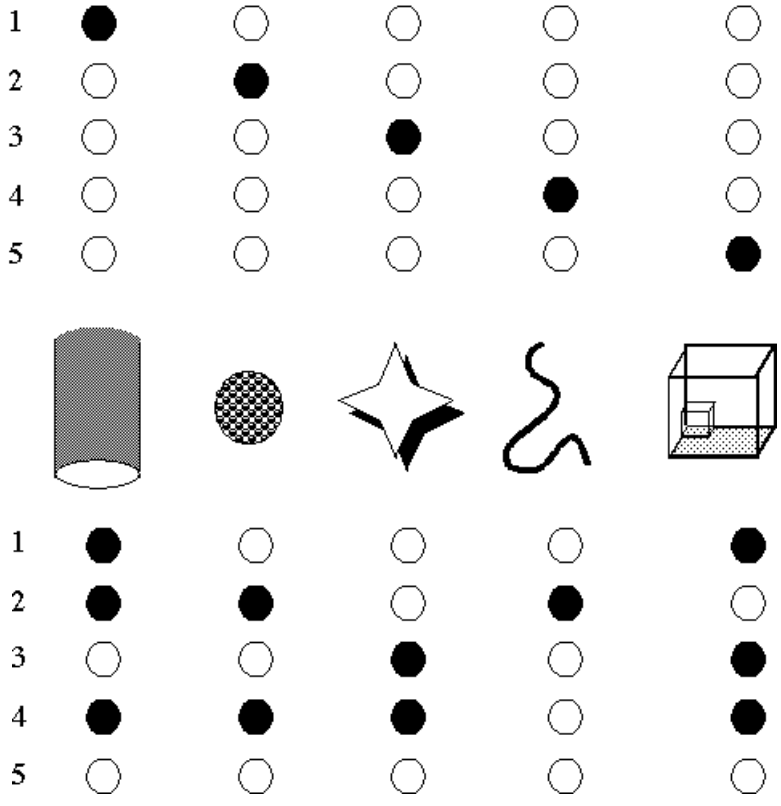
$$\hat{y} = \text{softmax}(\boldsymbol{z})_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

  - Wichtig sind nicht absolute Werte der Nettoaktivitäten, sondern deren Unterschiede (Laterale Hemmung, Winner Takes All)
- Logarithmus liefert direkten Beitrag der Nettoaktivität $z_i$ zur Kostenfunktion:

$$\log P(\text{y} = i; \boldsymbol{z}) = \log \text{softmax}(\boldsymbol{z})_i = \boxed{z_i} - \log \sum_j \exp(z_j)$$

[Goodfellow et al. 2016]

# NEURONALE CODIERUNG




- **Lokal**
  - Ein Neuron codiert ein Objekt
  - Großmutter-Zellen
  - Skaliert nicht
  - Nicht robust

- **Verteilt**
  - Neuronen codieren Merkmale
  - Ein Neuron für mehrere Objekte aktiv
  - Ein Objekt aktiviert mehrere  Neuronen
  - Robust gegen Beschädigungen