



# Backpropagation Algorithmus

Heinrich Mellmann

# Roadmap

---

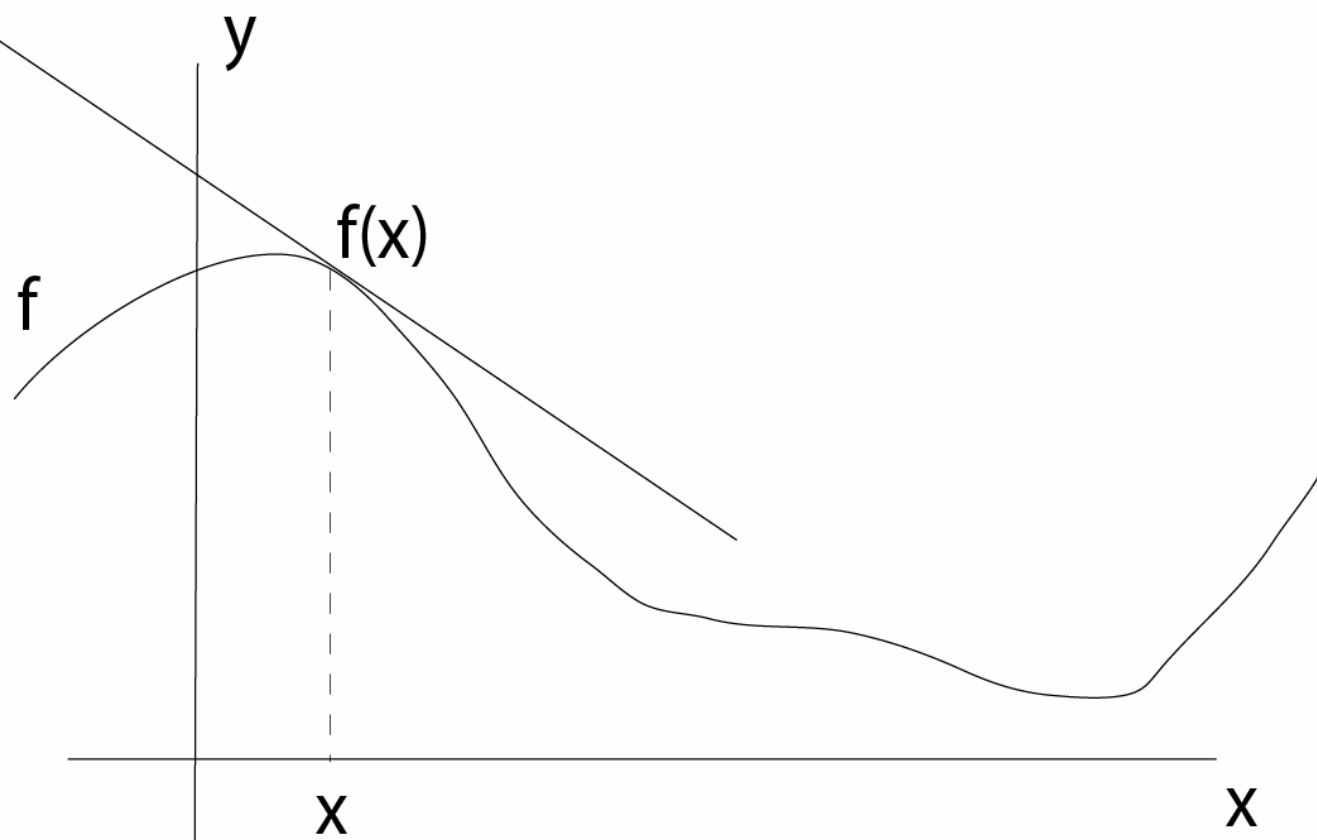


- Grundlagen
  - Gradientenabstieg
  - Kontinuierliche Aktivierungsfunktionen
    - Sigmoid
  - Feed-Forward Netze und Fehlerfunktion
- Backpropagation Algorithmus
  - Ziele (Formal)
  - BP für eindimensionale Netzwerke
  - BP für allgemeine Feed-Forward Netze
- Zusammenfassung, Ausblick, Beispiele

# Gradientenabstieg



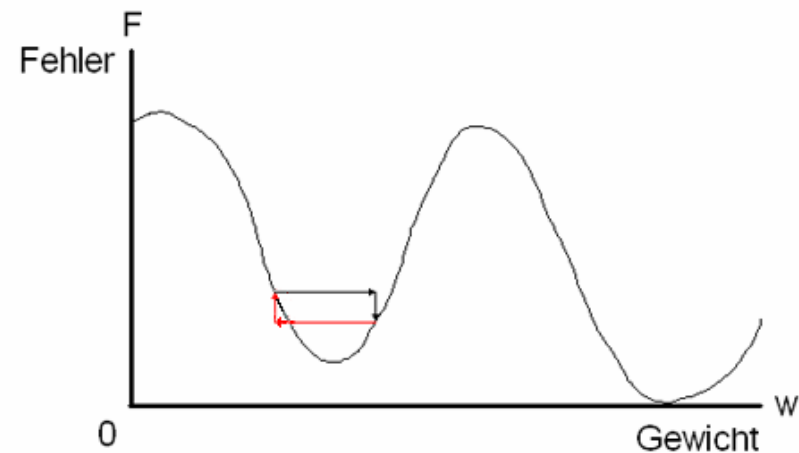
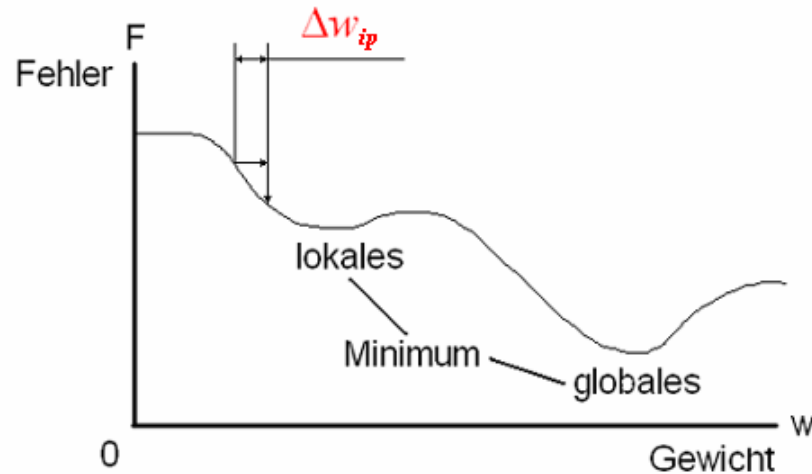
$$f(x) + hf'(x)$$



# Gradientenabstieg (Probleme)



- Findet nur lokale Minima
- Unendliche Schleifen möglich
- Differenzierbare Funktion notwendig

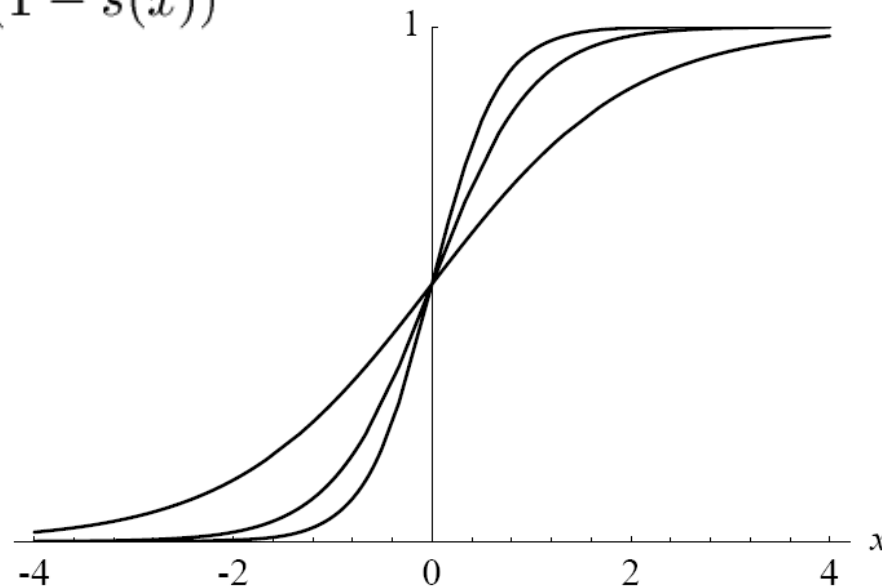


# Sigmoide



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

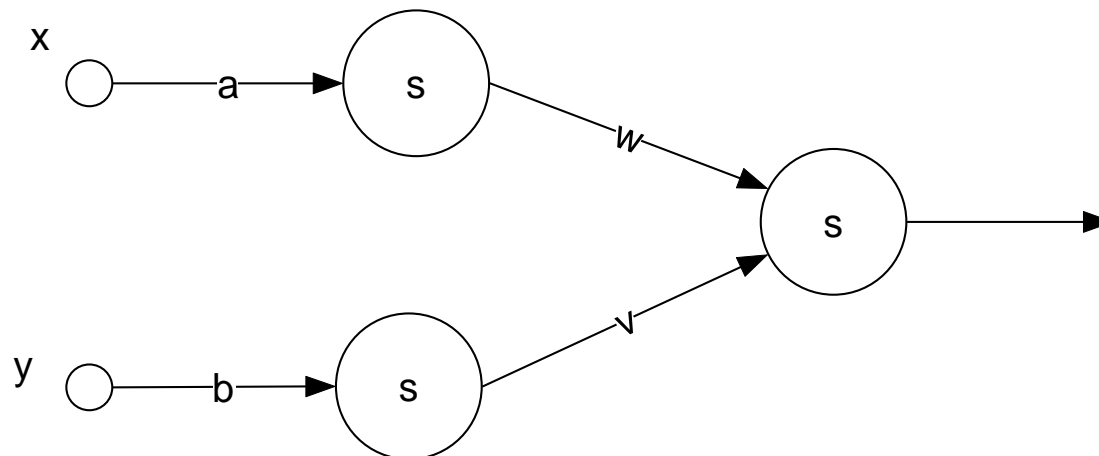
$$\frac{d}{dx}s(x) = \frac{e^{-x}}{(1 + e^{-x})^2} = s(x)(1 - s(x))$$



# Feed-Forward Netzwerk



- Gerichteter, azyklischer Graph
- Knoten berechnen primitive Funktionen
- Kanten sind gewichtet
- Bsp.:



# Roadmap

---



- Grundlagen
  - Gradientenabstieg
  - Kontinuierliche Aktivierungsfunktionen
    - Sigmoide
  - Feed-Forward Netze und Fehlerfunktion
- Backpropagation Algorithmus
  - Ziele (Formal)
  - BP für eindimensionale Netzwerke
  - BP für allgemeine Feed-Forward Netze
- Zusammenfassung, Ausblick, Beispiele

# Training

---



- **Ziel:** finde für eine gegebene Funktion  $f$  die Gewichte  $w_{ij}$  so, dass  $f$  möglichst gut durch die Netzwerkfunktion  $f_i$  approximiert wird
  
- **Problem:** die Funktion  $f$  ist nur anhand von Beispielen (Trainings-Proben) gegeben

# Training (Formal)

---

- Sei  $\{(x_1, t_1), \dots, (x_p, t_p)\}$  eine Menge von Trainings-Beispielen
- Sei  $w$  der Gewichte-Vektor des Netzes
- Das Netzwerk berechnet zur Eingabe  $x_i$  die Ausgabe  $o_i = o_i(w, x_i)$
- Definiere die Fehlerfunktion:

$$E = E(w) = \frac{1}{2} \sum_{i=1}^p \|o_i - t_i\|^2$$

# Training (Formal)

---



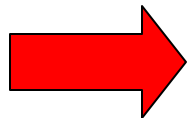
- Formulierung der Lernaufgabe:
  - Minimiere die Fehlerfunktion  $E$  in Abhängigkeit von dem Gewichtevektor  $w$
  
- Ansatz:
  - Gradientenabstieg

# Backpropagation-Alg.

---



- (geschickte) Implementation des Gradientenabstieges zur Minimierung der Fehlerfunktion

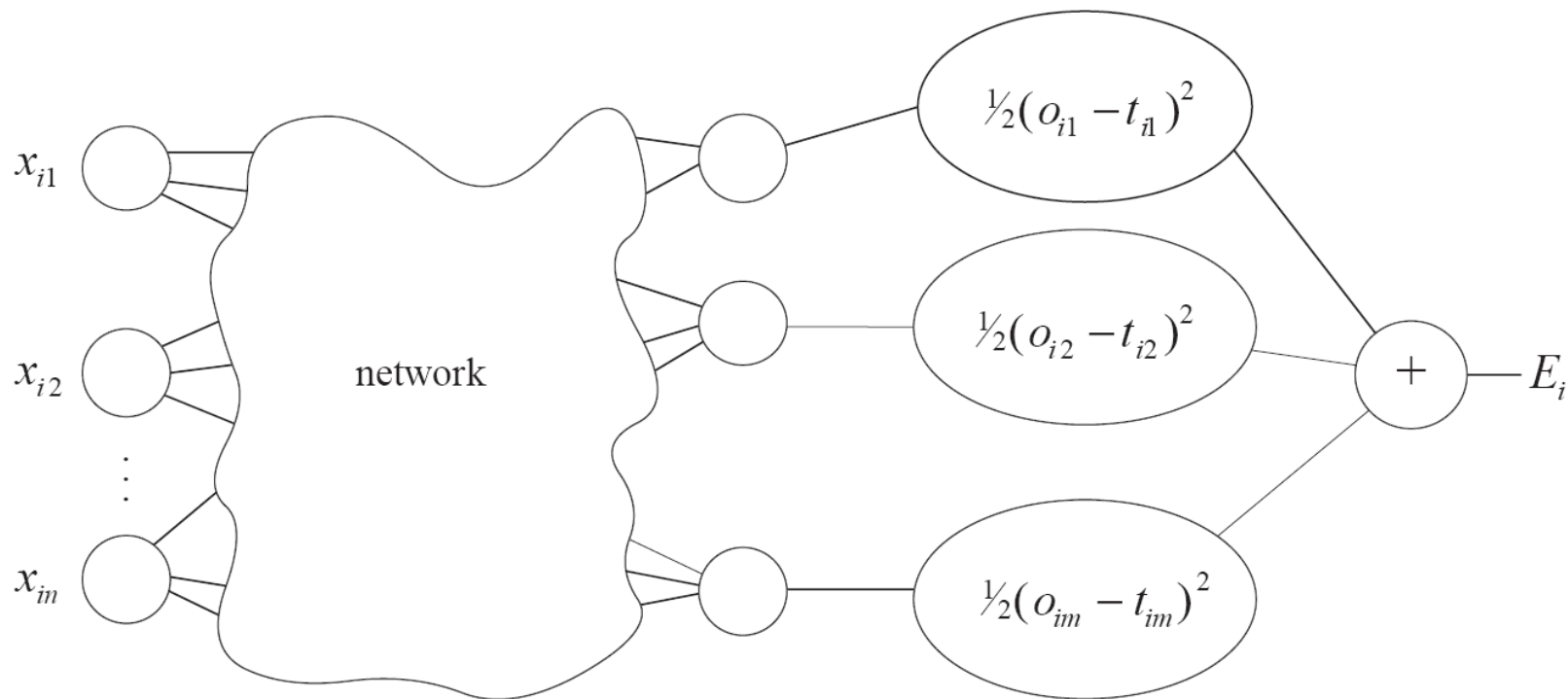


- Topologie des Netzes, sowie Knotenfunktionen werden als gegeben vorausgesetzt
- benötigt differenzierbare Auswertungsfunktionen der Knoten

# Schritt 1



- Erweitere das Netzwerk so, dass es die Fehlerfunktion berechnet:



# Gradientenabstieg (Anwendung)



- Gradient der Fehlerfunktion:

$$\nabla E = \left( \frac{\partial E}{\partial w_1}, \dots, \frac{\partial E}{\partial w_l} \right)$$

- Korrektur der Gewichte in einem Schritt:

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

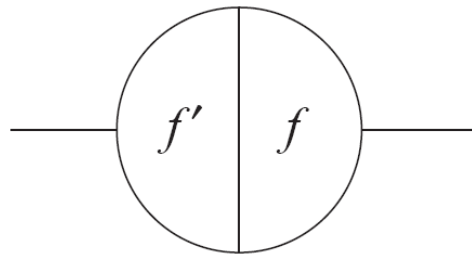
mit

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

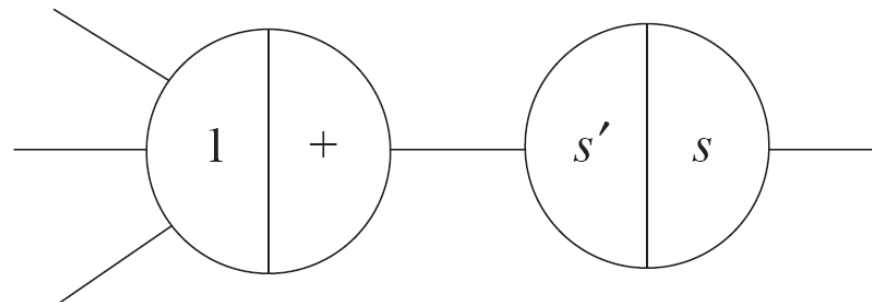
# Berechnung der Ableitung



- Erweiterung der Knoten:



- Trennung der Integrations- und Aktivierungs-Funktion:



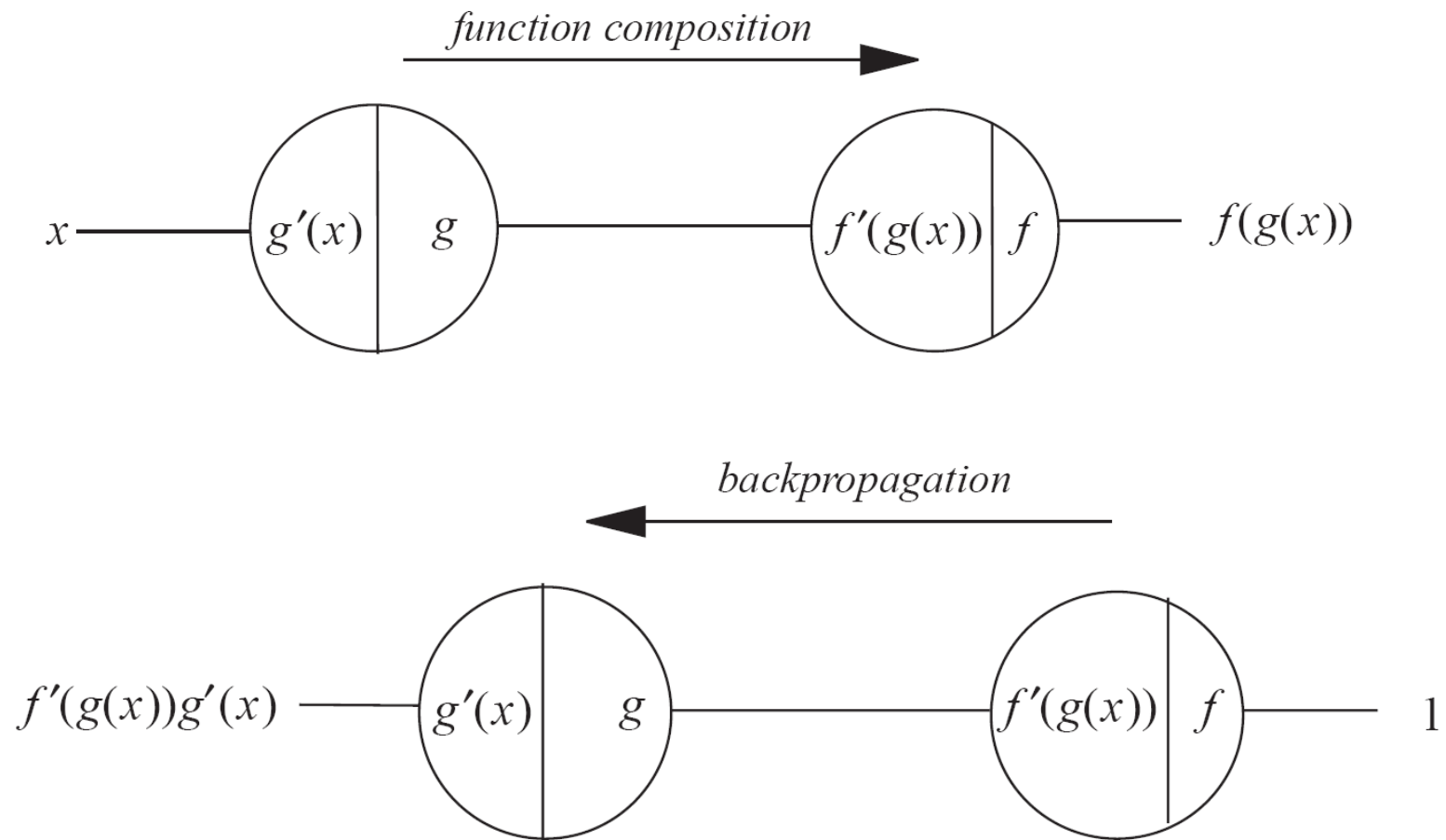
# Berechnung der Ableitung

---

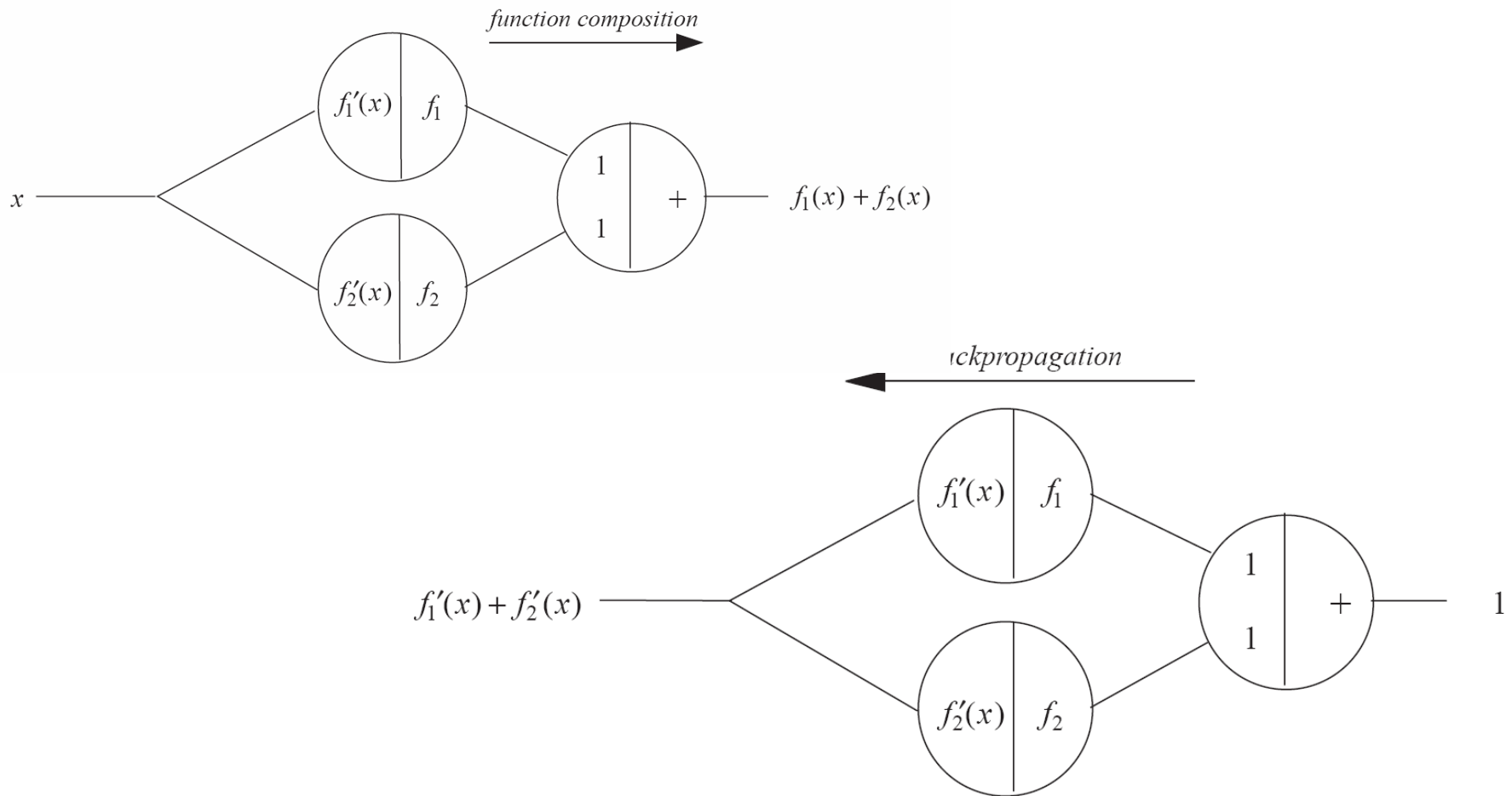


- Berechnung erfolgt in zwei Schritten:
  - Schritt 1 (feed-forward):
    - Information fließt von links nach rechts
    - In jedem Knoten wird die Funktion und die Ableitung berechnet und gespeichert
    - nur der Funktionswert wird weitergereicht
  
  - Schritt 2 (backpropagation):
    - Das Netzwerk läuft rückwärts und berechnet aus den gespeicherten Teilableitungen die Ableitung der Fehlerfunktion

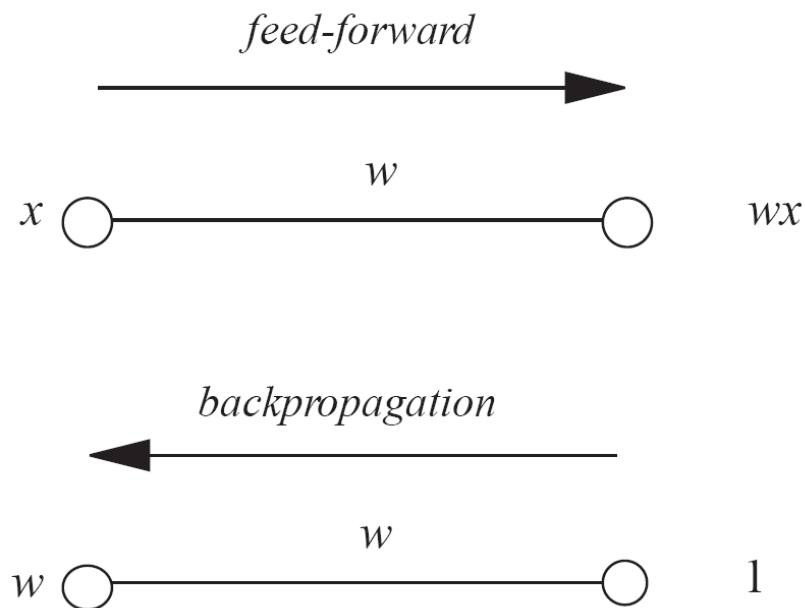
# Backpropagation: Komposition



# Backpropagation: Addition



# Backpropagation: Kanten



- In beide Richtungen dieselbe Funktion

# Backpropagation Algorithmus

---



- Feed Forward:
  - Links wird Eingang  $x$  eingespeist
  - Funktionswerte und Ableitungen werden berechnet
    - Ableitungen werden gespeichert (an jedem Knoten)
    - Funktionswerte (der Knoten) werden weitergereicht

# Backpropagation Algorithmus

---

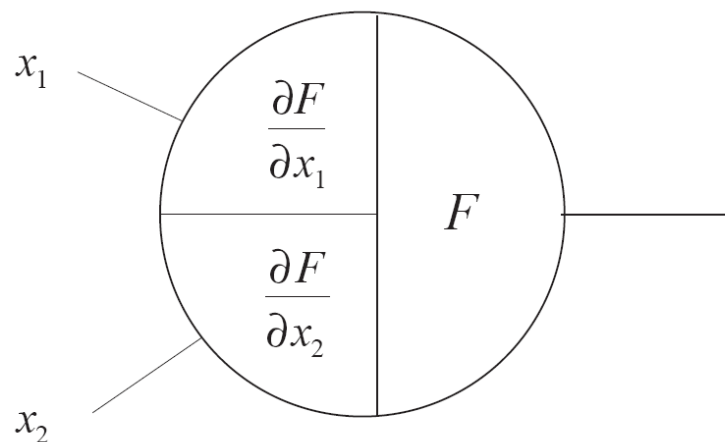


- Backpropagation:
  - Rechts wird die Konstante 1 eingespeist (Netzwerk läuft rückwärts)
  
  - Alle an einem Knoten ankommende Werte werden addiert, und das Ergebniss mit der gespeicherten Ableitung (rechte Seite) multipliziert

# Erweiterung



- Erweiterung des Backpropagation-Algorithmus auf Aktivierungsfunktion in mehrerer Variablen:

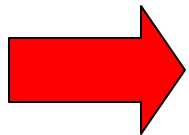


# Erweiterung

---



- Erweiterung auf Netzwerke mit mehreren Input-Variablen:
  - Feed-Forward Schritt wie gewohnt
  - Jeder Eingangsknoten definiert ein Teilnetzwerk bestehend aus allen erreichbaren Knoten

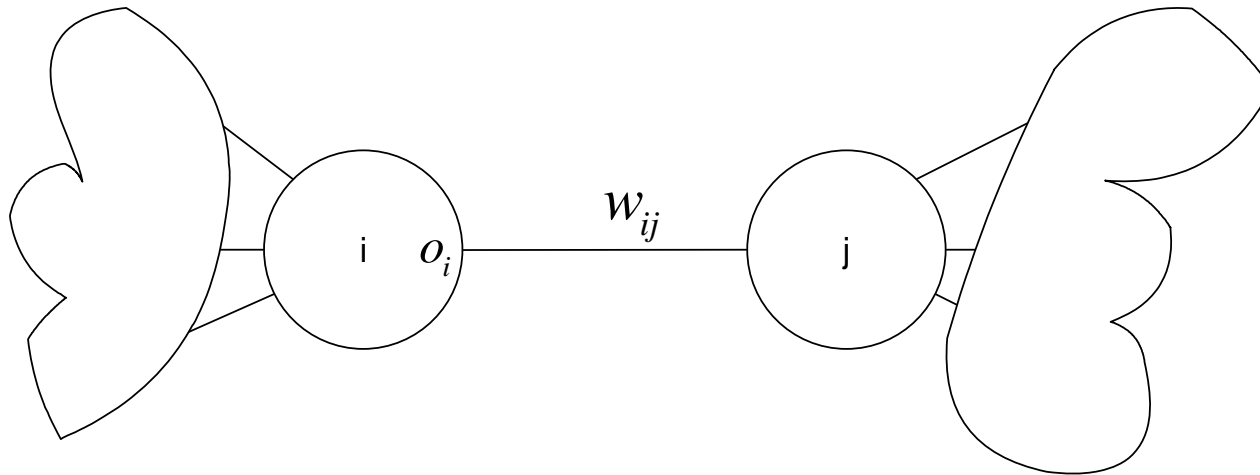


Führe für jedes Teilnetzwerk Backpropagation Schritt aus

# BP als Lernalgorithmus



- Ziel: Bestimmung der Ableitungen bezüglich der Gewichte
- Betrachte Gewicht  $w_{ij}$



# BP als Lernalgorithmus



- Das Teilnetzwerk mit  $o_i w_{ij}$  als Eingabe berechnet also bei dem BP-Schritt die Ableitung

$$\frac{\partial E}{\partial o_i w_{ij}}$$

da  $o_i$  konstant ist, gilt

$$\frac{\partial E}{\partial w_{ij}} = o_i \frac{\partial E}{\partial o_i w_{ij}}$$

# Modifikation des BP



- Bei den Feed-Forward Schritt wird zusätzlich der Funktionswert  $o_i$  gespeichert
- Beim BP-Schritt wird der *backpropagierter Fehler*

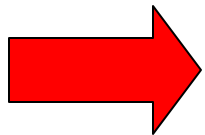
$$\delta_j := \frac{\partial E}{\partial o_i w_{ij}}$$

gespeichert

# BP als Lernalgorithmus



- Zusätzlicher Korrektur-Schritt notwendig:



Korrektur des Gewichtes  $w_{ij}$   
mit dem Term

$$\Delta w_{ij} := o_i \delta_j$$

(Gradientenabstieg-Schritt)

# Fragen?

---



- Grundlagen
  - Gradientenabstieg
  - Kontinuierliche Aktivierungsfunktionen
    - Sigmoid
  - Feed-Forward Netze und Fehlerfunktion
- Backpropagation Algorithmus
  - Ziele (Formal)
  - BP für eindimensionale Netzwerke
  - BP für allgemeine Feed-Forward Netze
- Zusammenfassung, Ausblick, Beispiele

# Lernen mit mehreren Beispielen



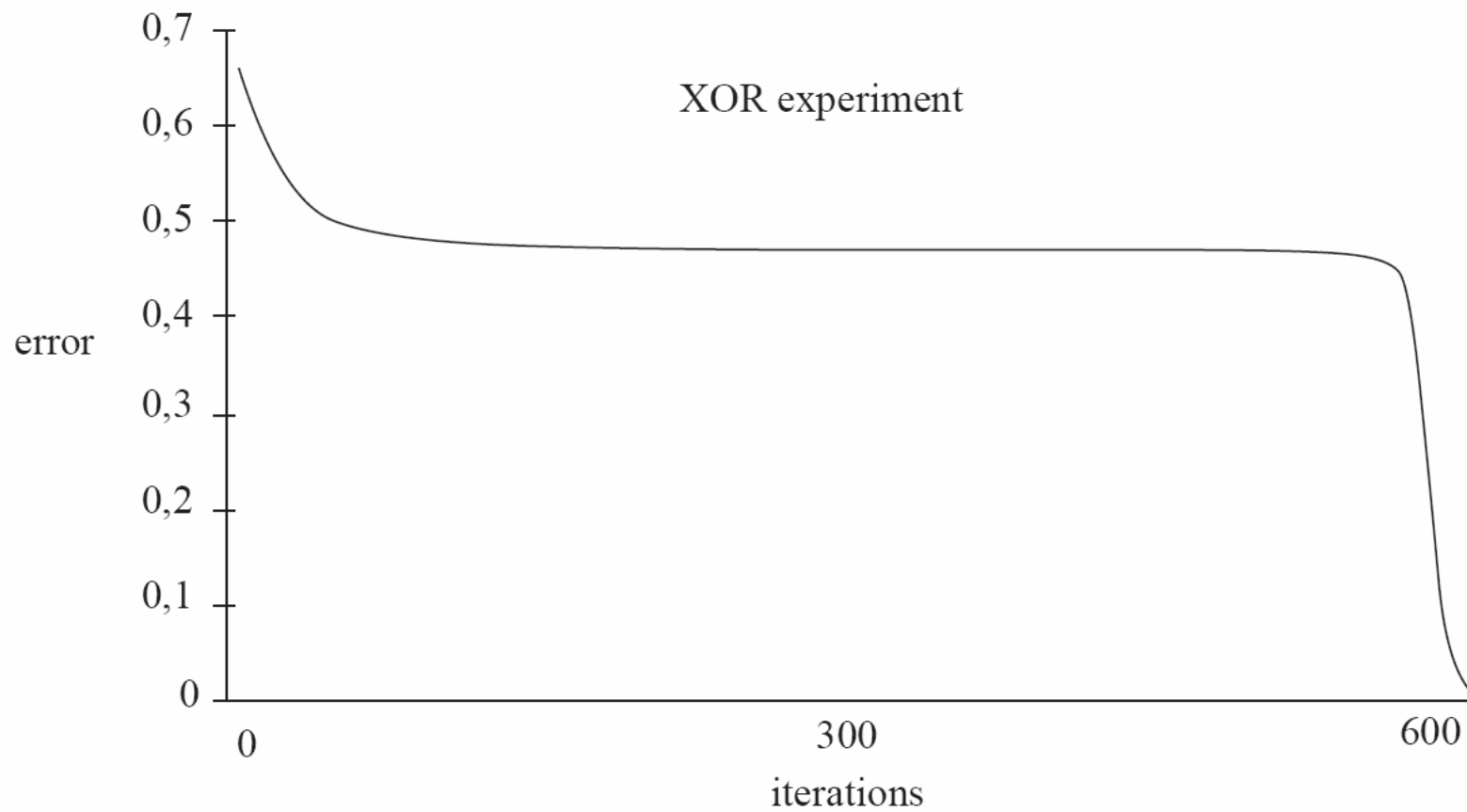
- Für jedes Beispiel  $k$  berechne die Korrekturen  $\Delta w_{ij}^k$  mit BP
- Die gesamte Fehlerfunktion ergibt sich als

$$E = \sum_{i=1}^p E_i$$

- Für die gesamte Korrektur gilt damit

$$\Delta w_{ij} = \sum_{i=1}^p \Delta w_{ij}^k$$

# Problem



# Verbesserungen

---



- Online lernen: die Korrektur wird nach jedem Beispiel vorgenommen
- Zufälliges Rauschen um Lokale Minima und Flächen zu „vermeiden“
- *Gradienten reuse*: wiederverwendung „guter“ Gradienten

# Lokalität

---



- Lokalität bedeutet: bei der Berechnung des Wertes der Kante  $i$  eines Knoten im BP-Schritt werden keine Informationen der Kante  $j \neq i$  benötigt
- Die Addition ist die einzige Integrationsfunktion, die die Lokalität bewahrt