Neural Nets, Background Back-Propagation

Neuronale Netze u. Anwendungen, 17. Jan. 2012

Overview

- Backpropagation
  - Implementation
    - Speed, Training
  - Generalization
  - Choice of Architecture
  - Algorithms

Training Time

- Backpropagation is Computer Intenstive during **Training** very efficient during evaluation/testing
- How to Improve Training Time
  - Parallel Hardware (?)
  - Efficient Implementation
  - Faster Gradient Descent Search
  - Selective Choice of Patterns
  - Efficient Architectures

Back-Propagation of Error

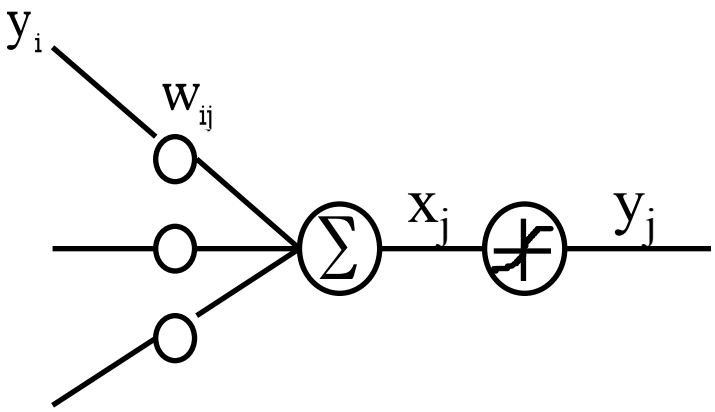
$$E \quad \frac{1}{2}\sum_j (y_j \quad d_j)^2 \qquad y_j \quad \frac{1}{1 \quad e^{\,x_j}} \qquad x_j \quad \sum_i y_i w_{ij}$$

1). $$\frac{E}{y_j} \quad y_j \quad d_j$$

2). $$\frac{E}{x_j} \quad \frac{E}{y_j}\,\frac{y_j}{x_j} \quad \frac{E}{y_j} y_j [1 \quad y_j]$$

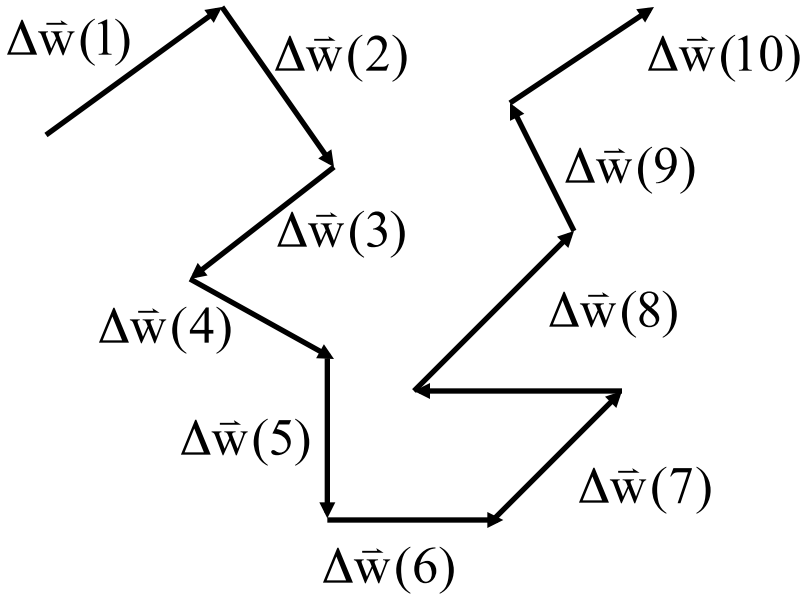
3). $$\frac{E}{w_{ij}} \quad \frac{E}{x_j}\,\frac{x_j}{w_{ij}} \quad \frac{E}{x_j}\, y_i$$

4). $$\frac{E}{y_i} \quad \sum_j \frac{E}{x_j}\,\frac{x_j}{y_i} \quad \sum_j \frac{E}{x_j}\, w_{ij}$$



Momentum

$$\Delta w_{ij}(t) \sim \frac{\partial E}{\partial w_{ij}}$$



$$\Delta w^{*}_{ij}(t+1) = \Delta w_{ij}(t+1) + \alpha \Delta w_{ij}(t)$$

Momentum
0...1

Überspringen von Trainingsbeispielen

- Kein BP für Trainingsbeispiele, die bereits gut gelernt sind

  Vorwärts-Durchlauf des Beispiels X durch das Netzwerk

  Fehler am Ausgang > Schwellenwert?

  Ja: Back-Propagation

  Nein: nächstes Trainingsbeispiel

Trainingsprozedur

A	B	C	D	E	F
A_1	B_1	C_1	D_1	E_1	F_1
A_2	B_2	C_2	D_2	E_2	F_2
A_3	B_3	C_3	D_3	E_3	F_3
⋮					
A_n	B_n	C_n	D_n	E_n	F_n

$(A_1, B_1, \ldots E_n, F_n) \longrightarrow$ 1 weight update

$\longrightarrow$ langsam aber sicher!

Trainingsprozedur

$A_1 \longrightarrow$ 1 weight update

$B_1 \longrightarrow$ 1 " "

$E_n \longrightarrow$ 1 " "

$F_n \longrightarrow$ 1 " "

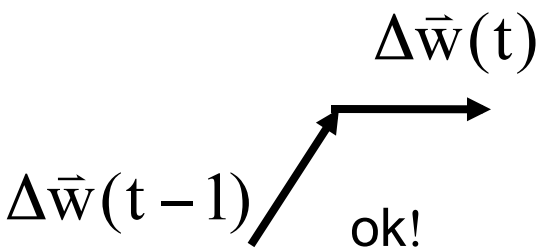
schnell, aber riskant!

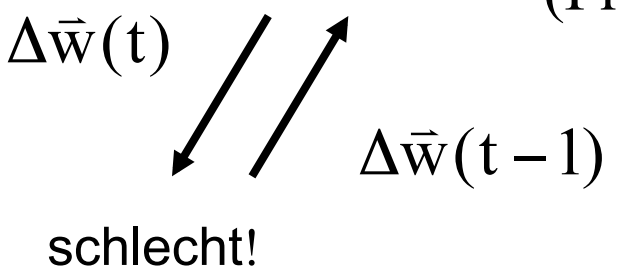
$(A_1 \ldots F_1) \longrightarrow$ 1 weight update

$(A_2 \ldots F_2) \longrightarrow$ 1 weight update

Dynamische Anpassung der Lernrate ε

(Franzini)





$$\cos\Theta = \frac{\sum_{i,j}\left(\Delta w_{ij}(t-1)\cdot\Delta w_{ij}(t)\right)}{\sqrt{\sum_{i,j}\left(\Delta w_{ij}(t-1)\right)^2\cdot\sum_{i,j}\left(\Delta w_{ij}(t)\right)^2}}$$

$\cos\Theta = 1 \quad (\Theta = 0^o, 360^o, ...)$ $\cos\Theta = -1 \quad (\Theta = 180^o)$

$\Rightarrow \varepsilon$ größer machen $\Rightarrow \varepsilon$ kleiner machen

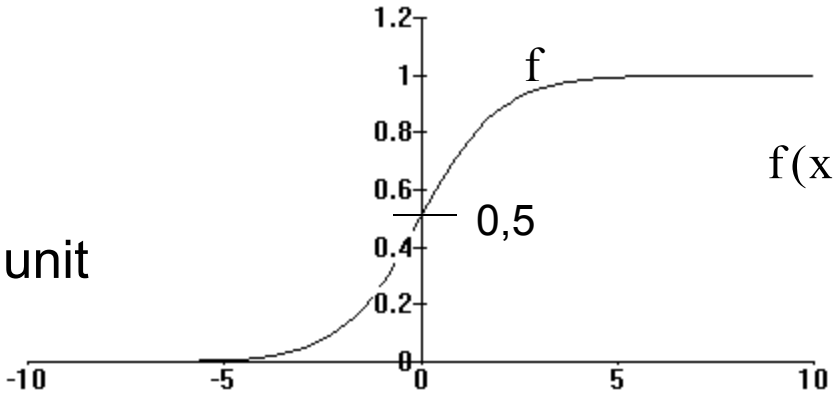
$$\boxed{\varepsilon(t) = \varepsilon(t-1)\cdot const\cdot\frac{\cos\Theta + 1}{2}}$$

Beschleunigung der Lernphase

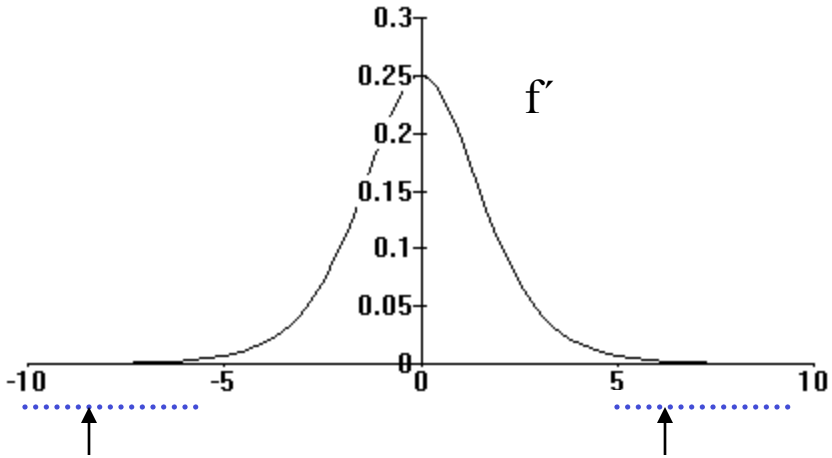
BP: $\Delta w_{ij} = \varepsilon \cdot \delta_j \cdot o_i$

output i-tes unit

$\delta_j \sim f'(\text{input unit}_j)$



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



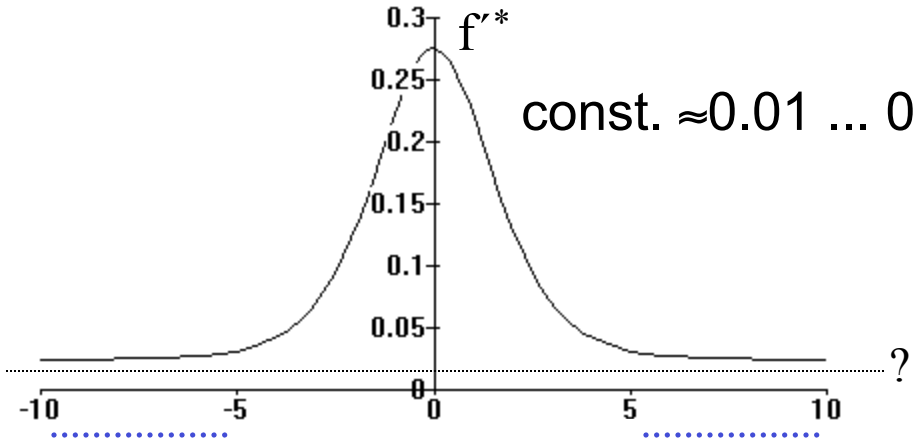
f´ sehr klein

=> Δw_{ij} sehr klein

Beschleunigung der Lernphase

Lösung: (S. Fahlmann)

$$f'^{*} = f' + \text{const.}$$



const. ≈0.01 ... 0.1

- i.A. schnelle Konvergenz
- besonders wirksam bei hochgradig nichtlinearen Klassifizierungsproblemen (XOR)
- manchmal schlechtere Generalisierung auf den Test-Daten

Der Quickprop Algorithmus

Ziel: schnelles Training eines MLP

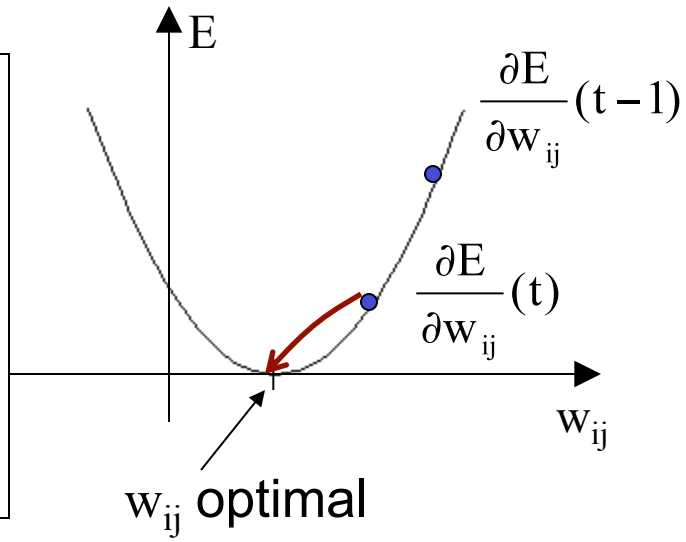
Zwei „riskante“ Annahmen:

1.) die Fehler/weight Kurve hat die Form einer Parabel

2.) die Parabeln für jedes weight w_{ij} sind unabhängig voneinander

$$\Delta w(t) = \frac{s(t)}{s(t-1) - s(t)} \cdot \Delta w(t-1)$$

mit $$s(t) = \frac{\partial E}{\partial w_{ij}}(t)$$

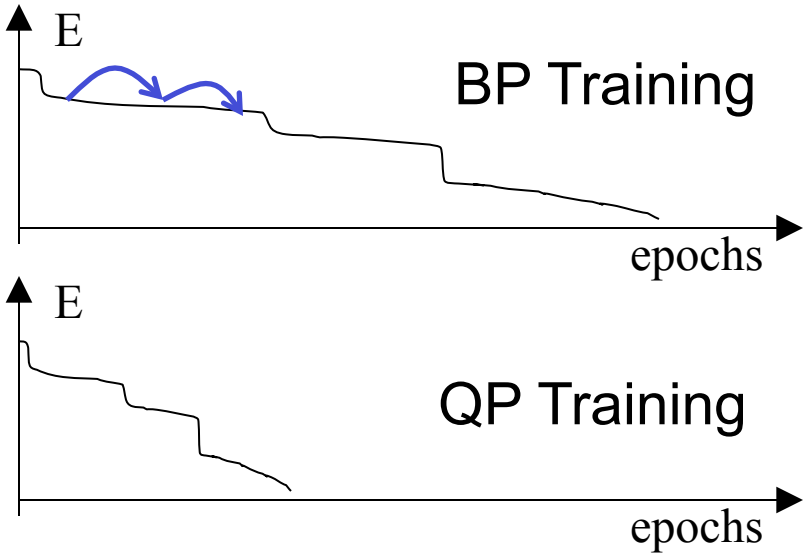


Quickprop

- Annahme 2 ist nicht richtig,

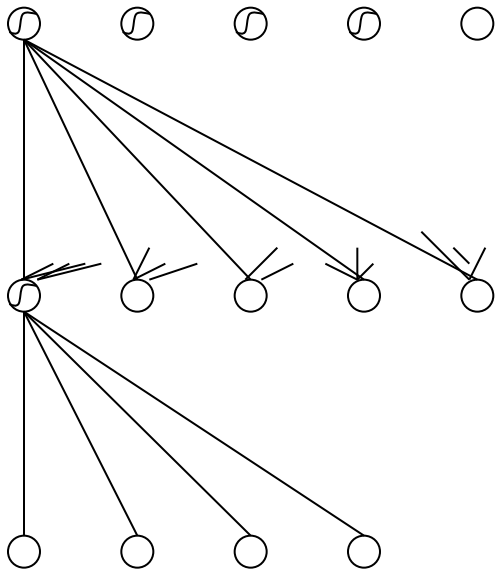
  aber: durch das iterative Einstellen der weights werden die gemachten Fehler aufgehoben

- funktioniert sehr gut für hochgradig nichtlineare Probleme wie z.B. XOR, die sich so verhalten:



Initialisierung mit Zufallszahlen

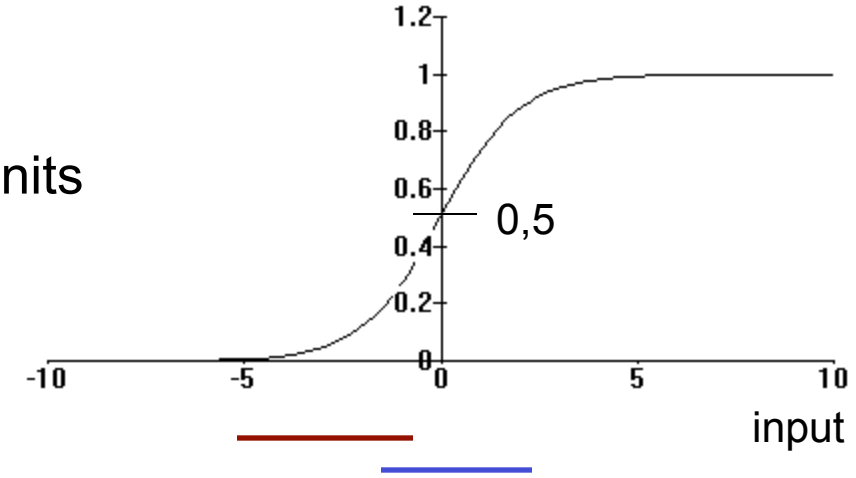
-0.3 ... +0.3



Output units

Hidden units

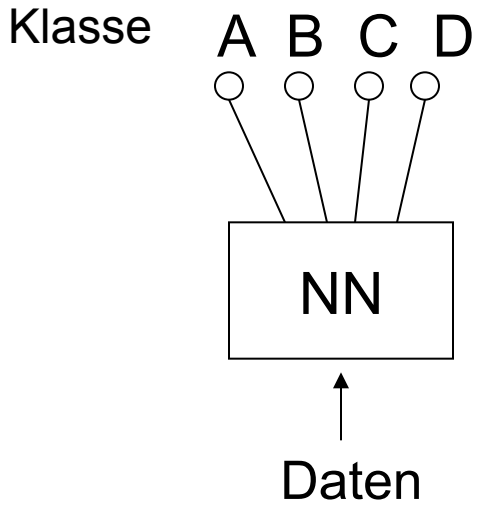
Input units



Klassifikatoren: Ausgangsrepräsentation

Einfachst mögliche Ausgangsrepräsentation:

Klasse A	1	0	0	0
Klasse B	0	1	0	0
Klasse C	0	0	1	0
Klasse D	0	0	0	1

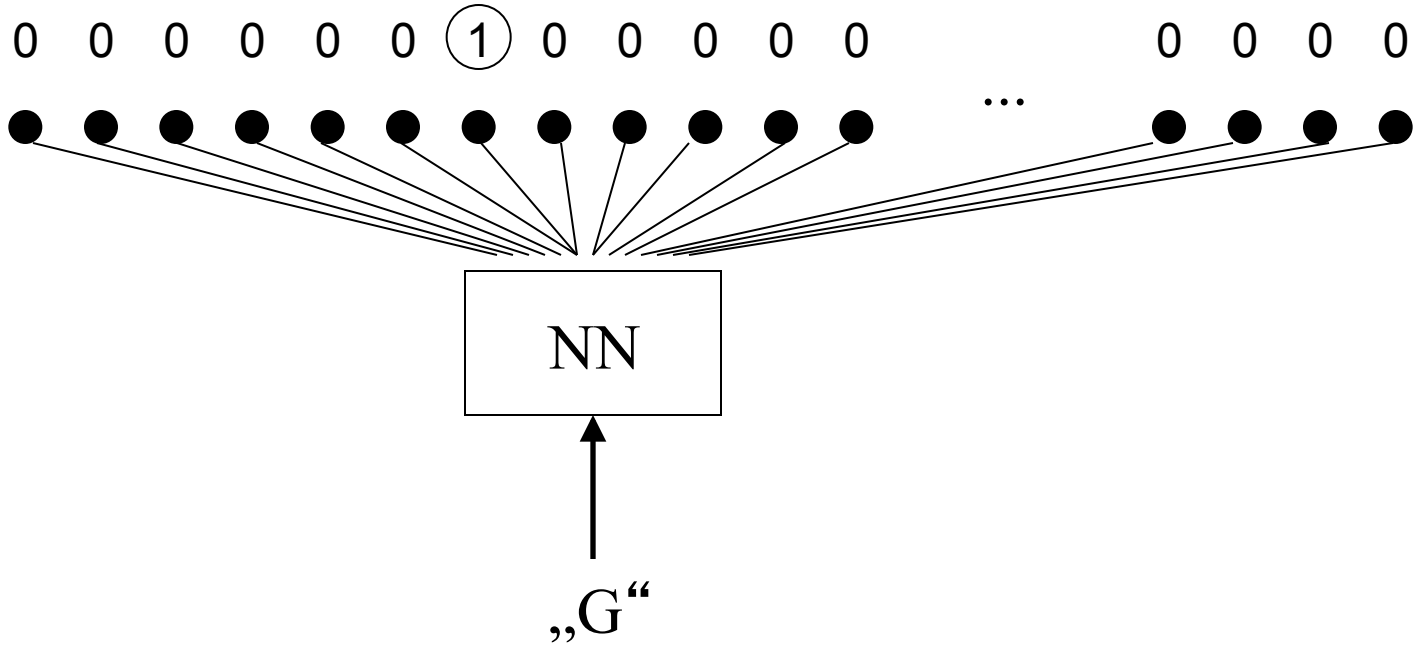


Vorteile: - die wahrscheinlichste Klasse ist leicht zu finden (Maximum suchen)

- Abschätzung der Verläßlichkeit der Klassifikation ist möglich

Probleme bei vielen Klassen

Angenommen: Alphabet-Erkenner
=> 26 Klassen



am einfachsten, wenn alle output units Null gesetzt werden!

Probleme mit dem MSE

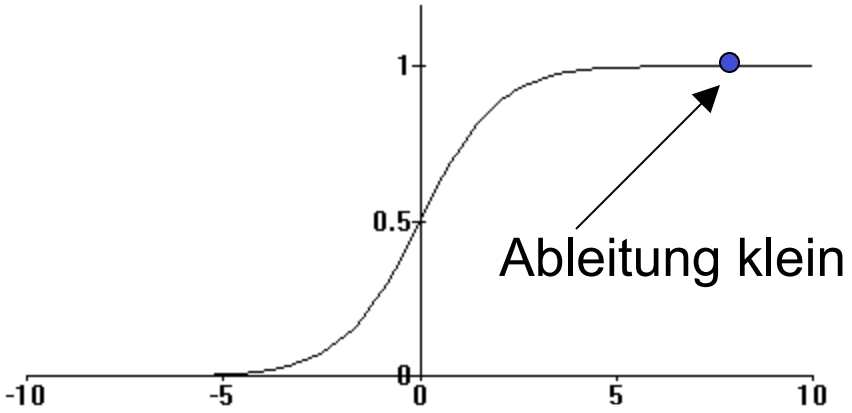
E = MSE = $\sum_j (y_j - target_j)^2$

angenommen: $y_j = 0{,}99$

$target_j = 0{,}0$

=> großer Beitrag zu E

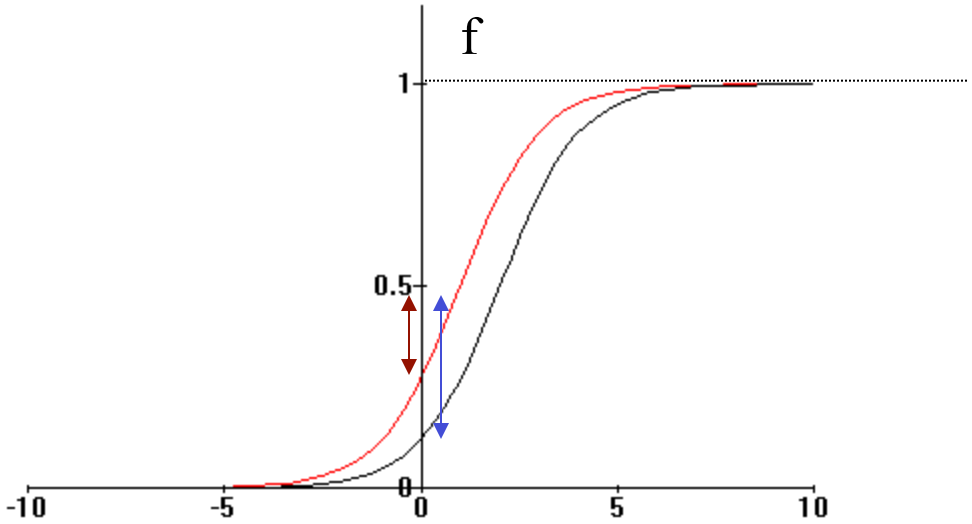
aber:



$\Rightarrow \Delta w$ klein

=> <u>langsames Lernen, obwohl E groß ist</u>

Asymetrische Sigmoid Funktion



input eines units

- stetig
- beschränkt
- Ableitung sollte einfach auszurechnen sein
- monoton

Andere Fehlerfunktionen

$$E = -\sum_j \ln(1-(y_j - target_j)^2)$$

(z.B. Mc Clelland)

$$\frac{\partial E}{\partial y_j} = -\frac{1}{1-(target_j - y_j)} + \frac{1}{1+(target_j - y_j)}$$

Multiplikation mit $f' = y_j(1-y_j)$:

$$\frac{\partial E}{\partial y_j} = \frac{(1-y_j)^2}{y_j - 2}$$ Falls $target_j = 1$ $$\frac{\partial E}{\partial y_j} = \frac{y_j^2}{y_j + 1}$$ Falls $target_j = 0$

=> $\frac{\partial E}{\partial y_j}$ maximal, wenn y_j - $target_j = 1$

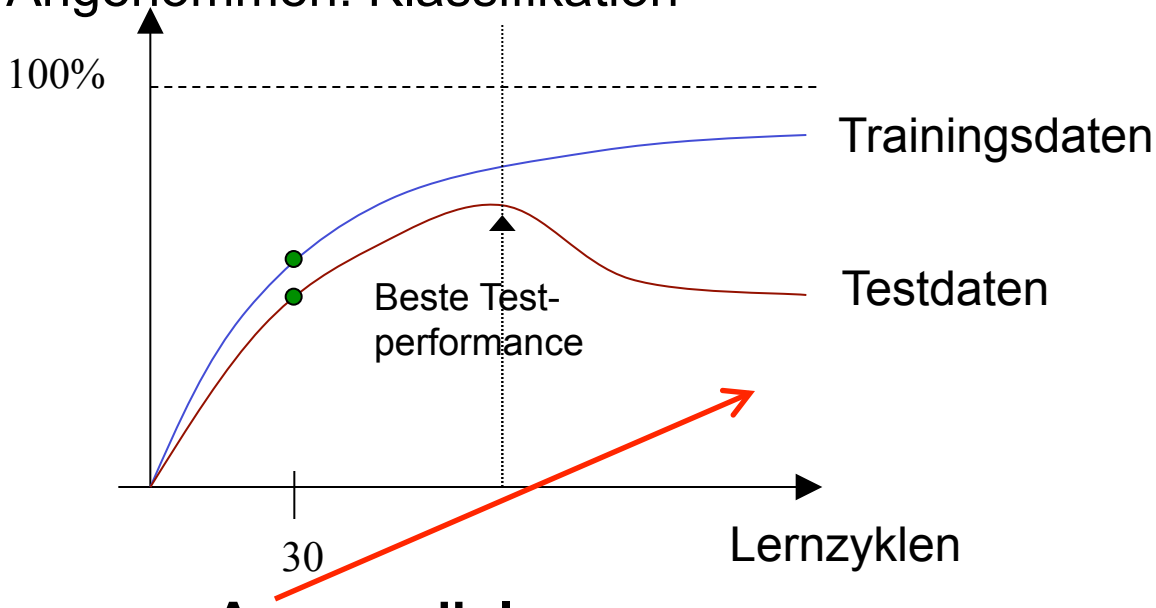
Weitere Fehlerfunktionen

- Backpropagate on Error from Best and Correct output

Generalisierungsfähigkeit:

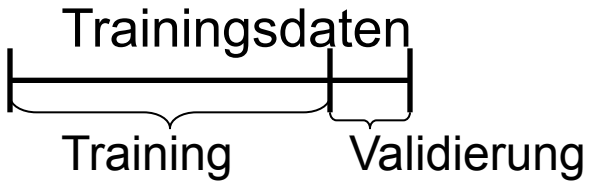
Fähigkeit, das aus den Trainingsdaten Erlernte auf neue Daten (Testdaten) anzuwenden.

Angenommen: Klassifikation



Auswendiglernen
„Overfitting“

→ Eine einfache Lösung: Validation Data

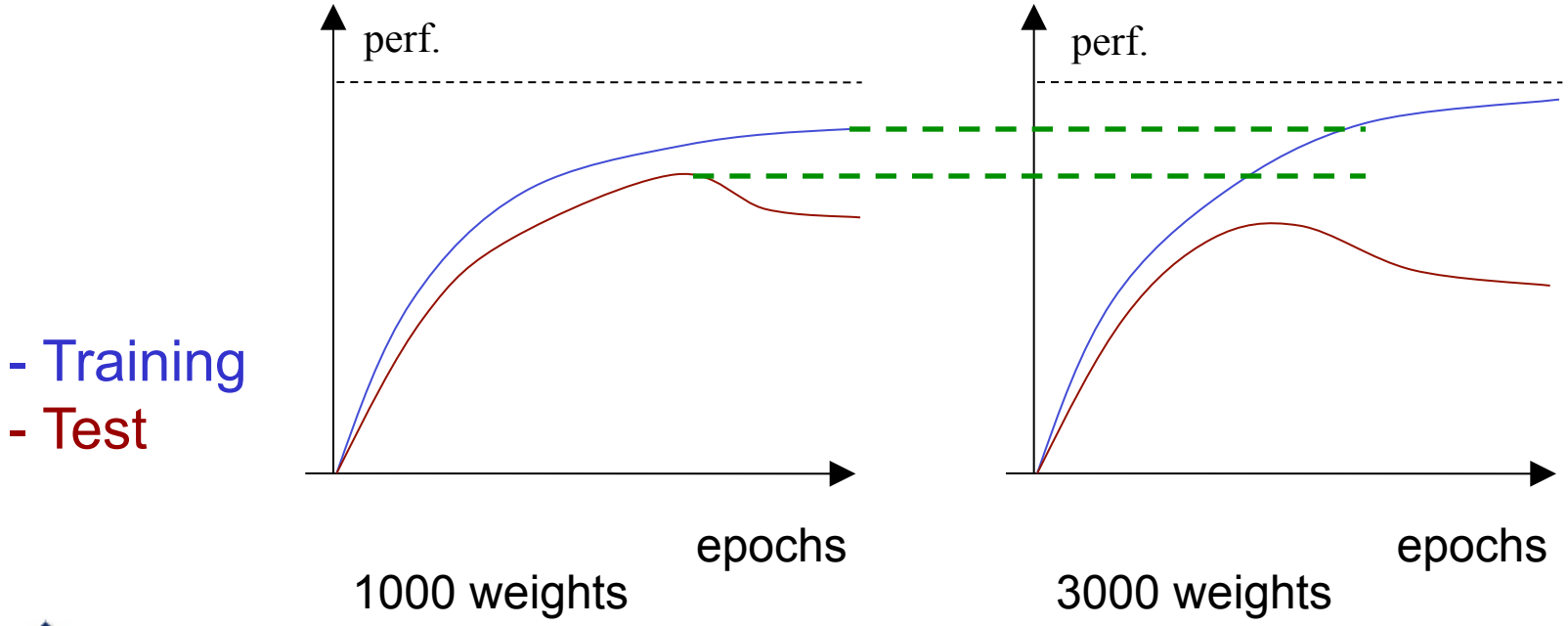


Zwei Gründe für schlechte Generalisierung?

1.) Overfitting/Overtraining (zu lange trainiert)

2.) Zu viele Parameter (weights) bzw.
zu wenig Trainingsmaterial

3.) falsche Netzwerkstruktur (neu)



Generalisierung in linearen Systemen

Für lineare Systeme gilt:

(Akaike, 1970: Statistical Predicator Identification
Ann. Inst. Stat. Math., 22:203 oder Barron oder Eubank)

$$\left\langle \varepsilon_{test} \right\rangle_{\xi\zeta'} = \left\langle \varepsilon_{train} \right\rangle_{\xi} + 2\sigma^2 \frac{p}{n}$$

Effektive Varianz des Rauschens

ξ : Training Set
ζ ‘ : Test Set
p : # Parameter
n : # Trainingsbeispiele

Generalisierung in nichtlinearen Systemen

(Moody, 1992)

$$\left\langle \varepsilon_{\text{test}} \right\rangle_{\zeta\zeta'} = \left\langle \varepsilon_{\text{train}}(\lambda) \right\rangle_{\xi\xi'}$$

$$\approx \left\langle \varepsilon_{\text{train}}(\lambda) \right\rangle_{\xi} + 2\sigma^2_{\text{eff}} \frac{p_{\text{eff}}(\lambda)}{n}$$

σ^2_{eff}: effektive Varianz des Rauschens
p_{eff} : effektive Anzahl der Parameter

$p_{\text{eff}} \leq p$

λ: Regulierungsparameter

Methoden zur Verbesserung der Generalisierung

1.) Reduzierung der Kompexität des Netzwerkes
durch Regularisierung
- Weight Decay
- Weight Elimination
- Optimal Brain Damage
- Optimal Brain Surgeon

* Ein großes Netzwerk wird daran gehindert, die Daten auswendig zu lernen.

Methoden zur Verbesserung der Generalisierung

2.) Schrittweises Vergrößern eines zu kleinen Netzes (konstruktiv)

- Cascade Correlation
- Meiosis Netzwerke
- ASO (Automativ Structure Optimization)

* Ein zu kleines Netzwerk wird solange erweitert, bis es die Daten lernen kann.

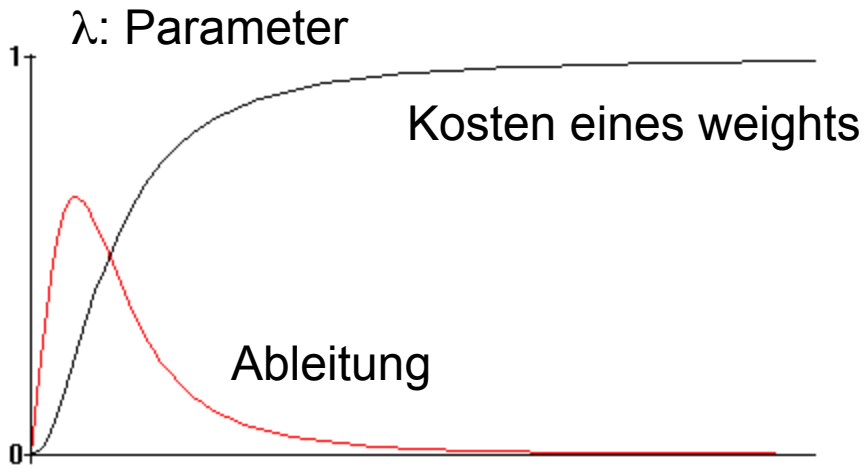
Regularisierung

- soll das „Overfitting“ verhindern

z.B. Weight-Elimination (Rumelhart, Chauvin, Denker, ...)

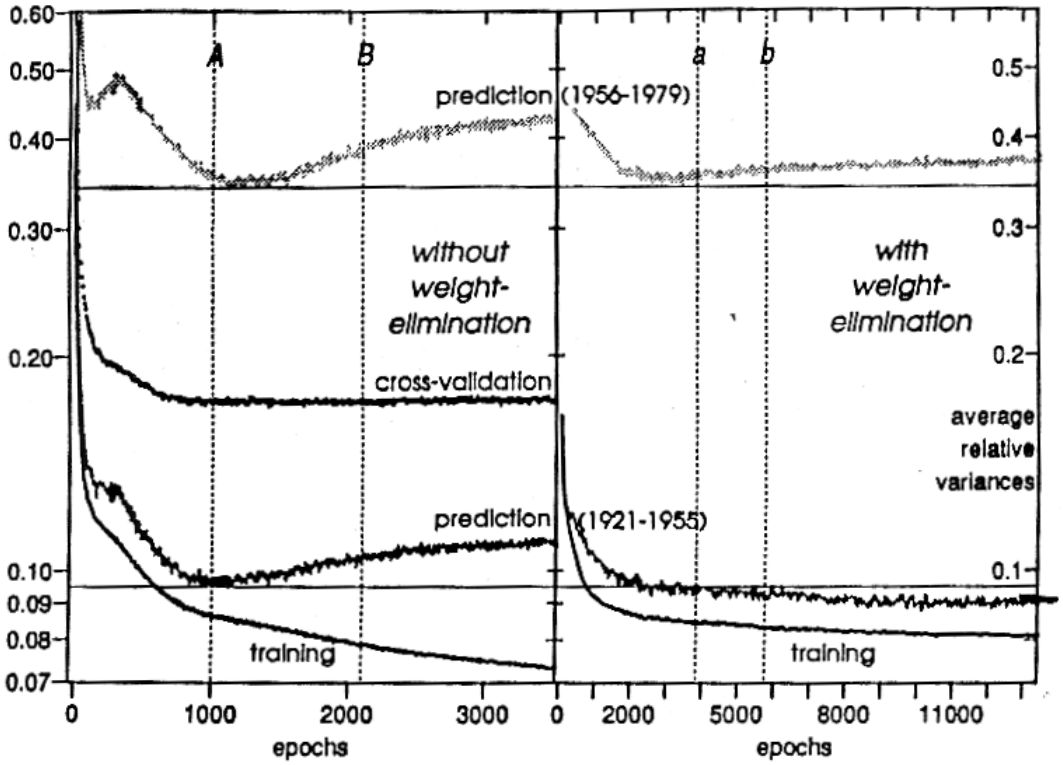
$$E = MSE + \lambda \sum_{i,j} \frac{w_{ij}^2}{1 + w_{ij}^2}$$

„Kosten“ eines weights



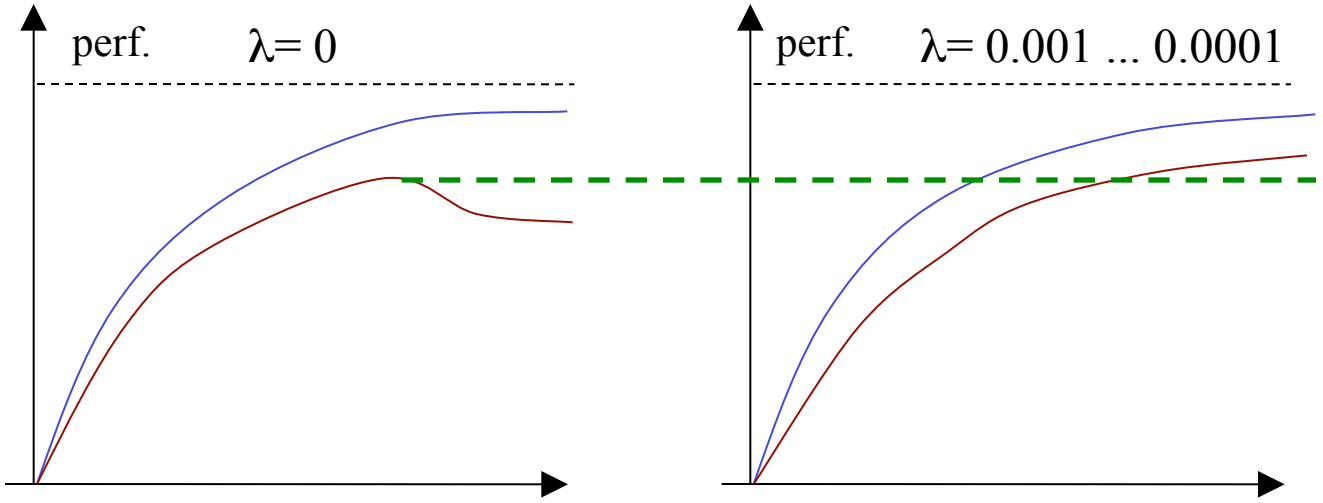
Interpretation : das Netzwerk wird für große |w| bestraft!

Lernkurven eines 12-8-1 Netzwerks



The average relative single-step prediction variance are given for the training sets (solid lines) and the early (low gray lines) and the late prediction set (upper gray lines) (as well as for the cross-validation set for the network trained without weight-elimination on the left side). The vertical lines A, B, a, b) indicate different stopping points

Weight-Elimination : Auswirkungen



- langsameres Lernen
- schlechtere Performanz auf den Trainingsdaten
- bessere Generalisierung

Einfachste Form:

$$\text{Error} = \text{MSE} + \lambda \sum_{i,j} w_{ij}^2$$

„Weight Decay“

Optimal Brain Damage

Idee: bestimmte Verbindungen werden aus dem Netzwerk entfernt um die Komplexität des Netzes zu beschränken und so Overfitting zu vermeiden.

Einfachste Form:

Es werden diejenigen Verbindungen entfernt, die nach dem Training sehr kleine Beträge $|w_{ij}|$ haben.

Richtig:

Es werden diejenigen Verbindungen entfernt, die dabei den geringsten Einfluß auf den Fehler (MSE oder ä.) haben, d.h. die unwichtigsten Verbindungen.

dazu: $\frac{\partial E}{\partial w_{ij}^2}$ berechnen

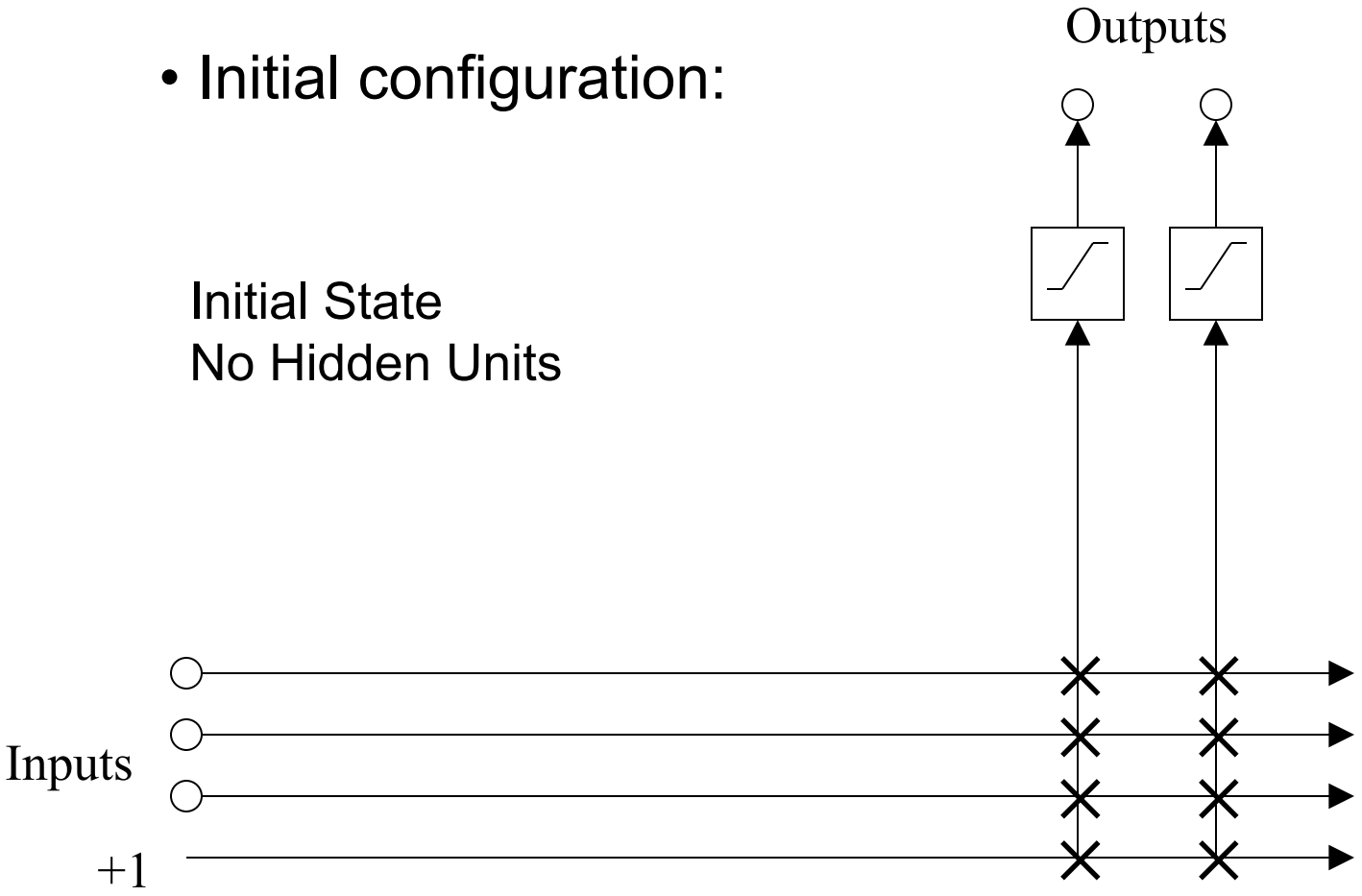
-> zeitaufwendig (schwierig)

Cascade Correlation

(Fahlman, Lebiere)

- Initial configuration:

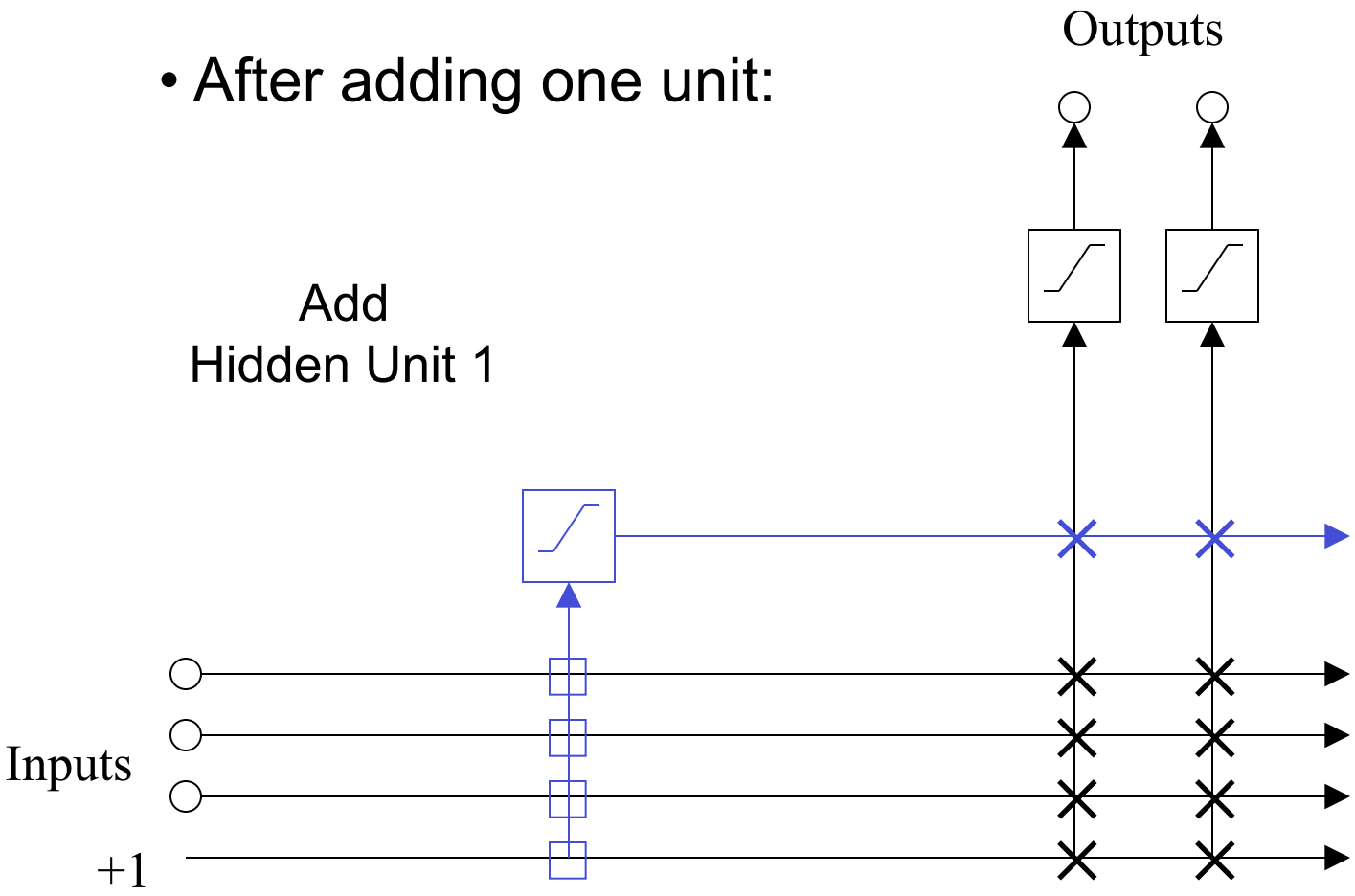
Initial State
No Hidden Units



Cascade Correlation

(Fahlman, Lebiere)

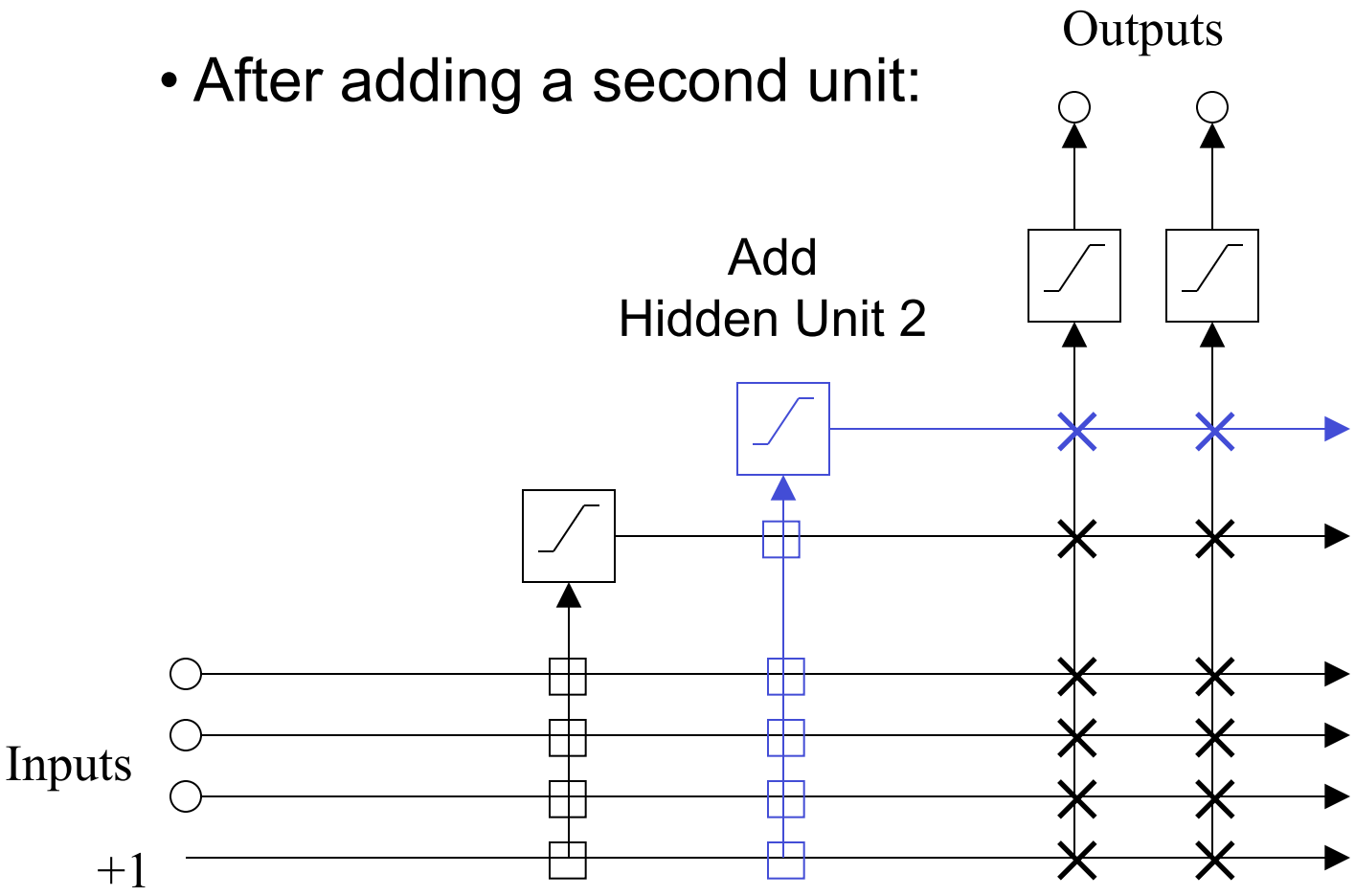
- After adding one unit:



Cascade Correlation

(Fahlman, Lebiere)

- After adding a second unit:



Cascade Correlation

- Hinzufügen eines hidden units:
  - input connections von allen input units
    und allen bisherigen hidden units
  - Zunächst werden nur diese Verbindungen adaptiert.
    Ziel: Maximieren der Funktion

$$S = \sum_{o} \sum_{p} (V_p - \overline{V})(E_{p,o} - \overline{E}_o)$$

p: Trainingsmuster
o: Index der output units

$$\frac{\partial S}{\partial w_i} = \sum_{p,o} \sigma_o (E_{p,o} - \overline{E}_o) f'_p I_{i,p}$$

Vorzeichen der Correlation zwischen der Aktivierung des candidate units und output o

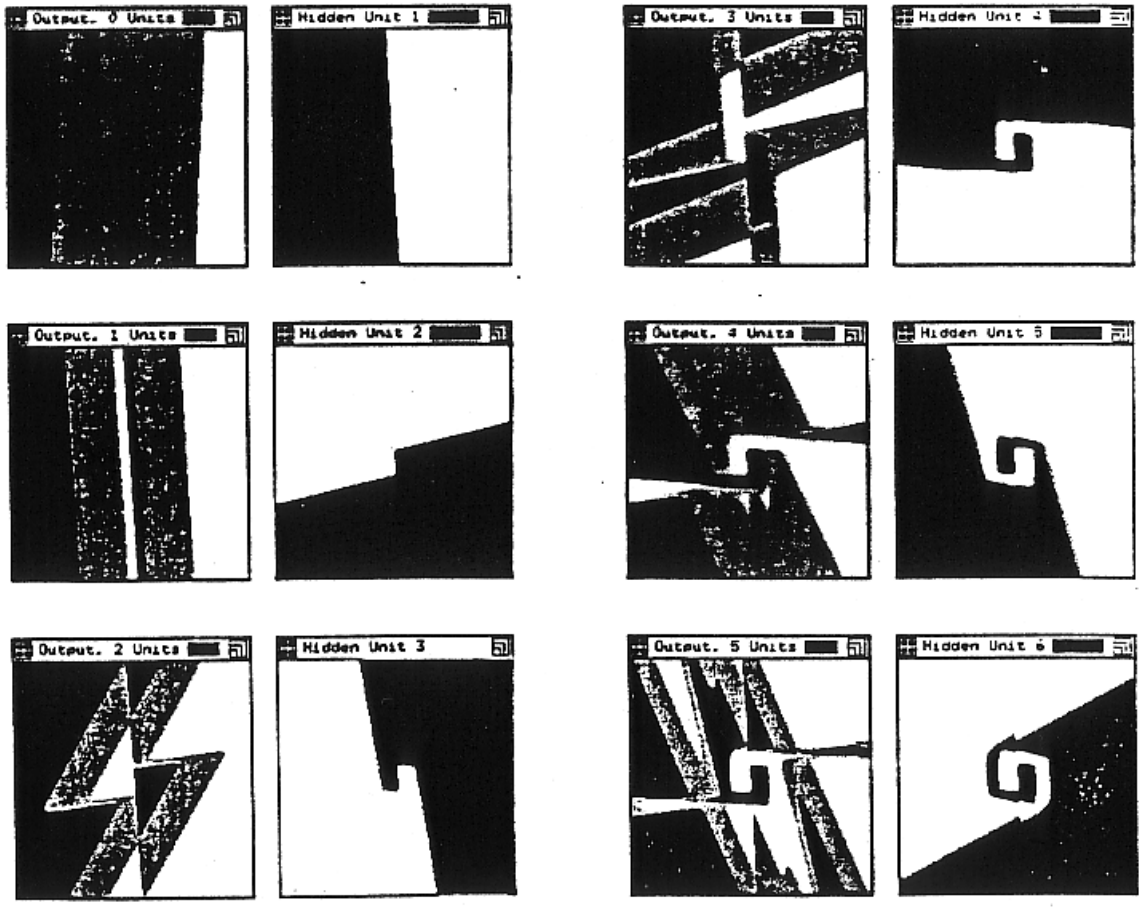
Ableitung der sigmoid Funktion

Input des candidate units

Zusammenfassung Cascade Correlation

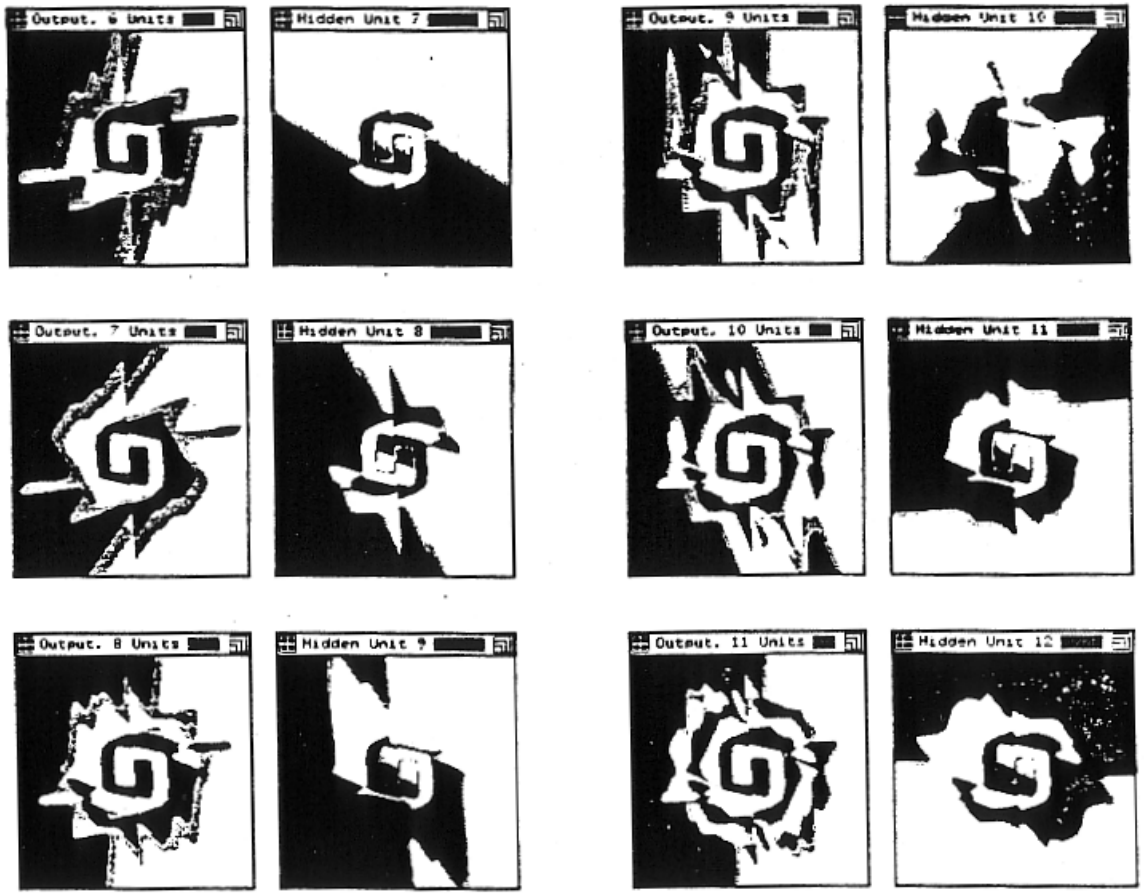
- Nicht noetig, die Anzahl Hidden Units oder Hidden Layers empirisch zu bestimmen.
- Kann tiefe Netze erzeugen ohne dramatischen Slowdown.
- Zu jedem Zeitpunkt wird nur eine Ebene von Verbindungen trainiert.
- Lernt sehr schnell.
- Inkrementelles Training.

Entwicklung einer 12-hidden-unit Loesung fuer das zwei-Spiral-Problem



black: strong negative
white: strong positive

Entwicklung einer 12-hidden-unit Loesung fuer das zwei-Spiral-Problem(kont.)



black: strong negative
white: strong positive

Meiosis Netzwerke

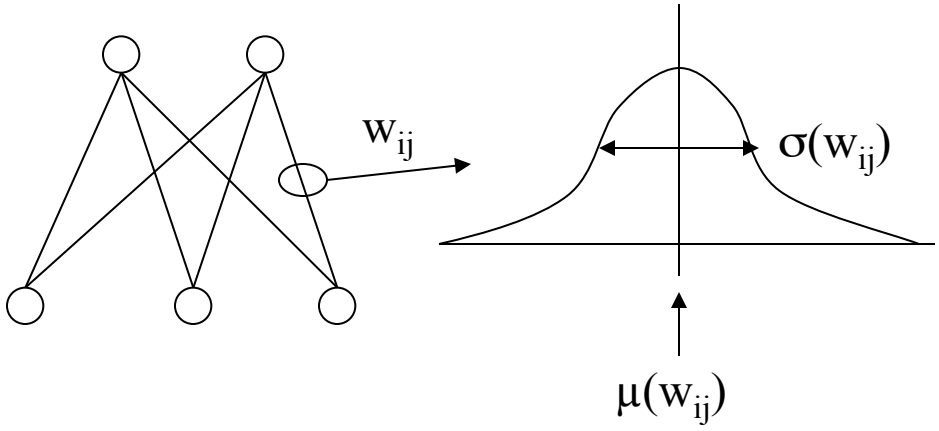
(Hanson, 1990)

Idee: Hinzufügen von hidden units abhängig von der „Unsicherheit“ der weights

$$x_i = \sum_j w_{ij}^* \cdot y_i$$

$$w_{ij}^* = \mu(w_{ij}) + \sigma(w_{ij})\underbrace{\phi(0,1)}$$

Gauss Noise
mit Mittelwert 0 und Varianz 1

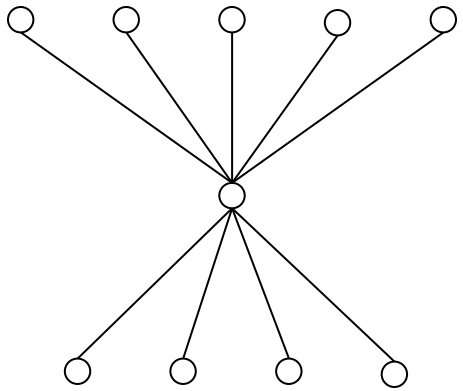


- der Mittelwert eines Gewichts $\mu(w_{ij})$ und die Varianz $\sigma(w_{ij})$ werden gelernt

Meiosis, Teil 2

- Start mit einem hidden unit
- ein hidden unit wird aufgespalten, falls

$$\underbrace{\frac{\sum_i \sigma_{ij}}{\sum_i \mu_{ij}} > 1.0}_{\text{Verbindung zu den Input Units}} \quad \text{und} \quad \underbrace{\frac{\sum_k \sigma_{ik}}{\sum_k \mu_{ik}} > 1.0}_{\text{Verbindung zu den Output Units}}$$



Probleme:

1.) Zusätzliche Parameter der Verbindungen (σ)
2.) Rausch-Generator
3.) Instabilitäten beim Hinzufügen von neuen hidden units