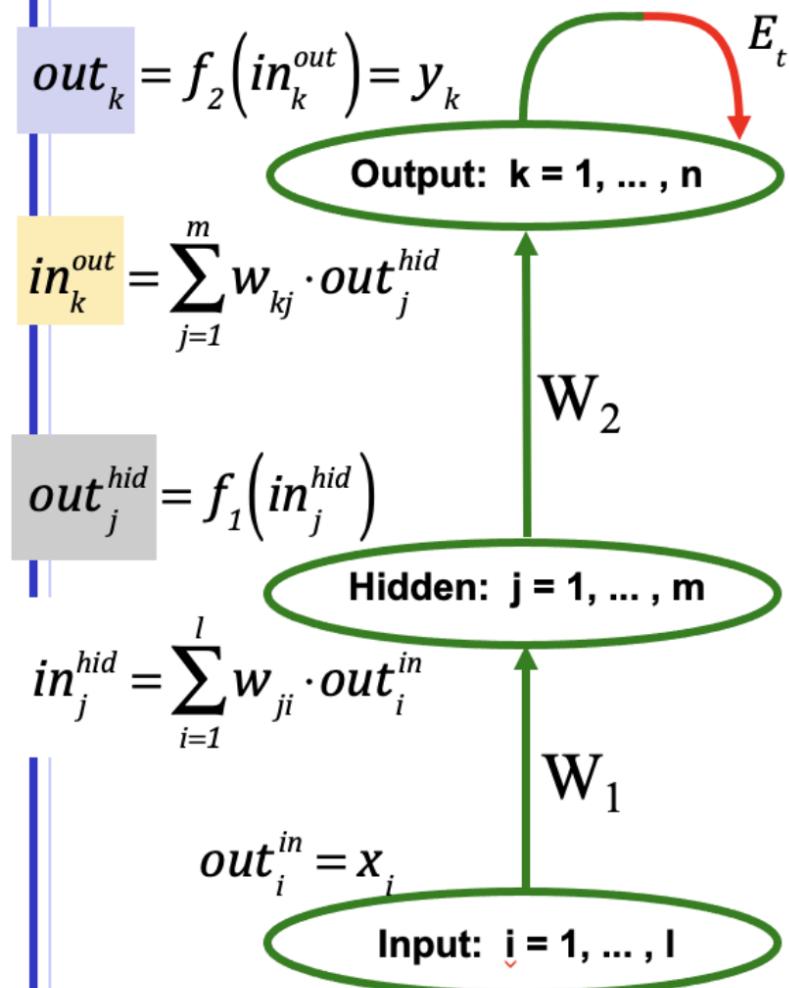


Erinnerung**5.3.2) Die Berechnung des Fehlergradienten**

$$E_t = \sum_{k=1}^n \frac{1}{2} (out_k - tar_k)^2$$

$$\frac{\partial E_t}{\partial w_{kj}} = (out_k - tar_k) \frac{\partial out_k}{\partial w_{kj}}$$

$$f'_2(in_k^{out}) \frac{\partial in_k^{out}}{\partial w_{kj}} \rightarrow out_j^{hid}$$

mit  $\delta_k^{out} := (out_k - tar_k) f'_2(in_k^{out})$

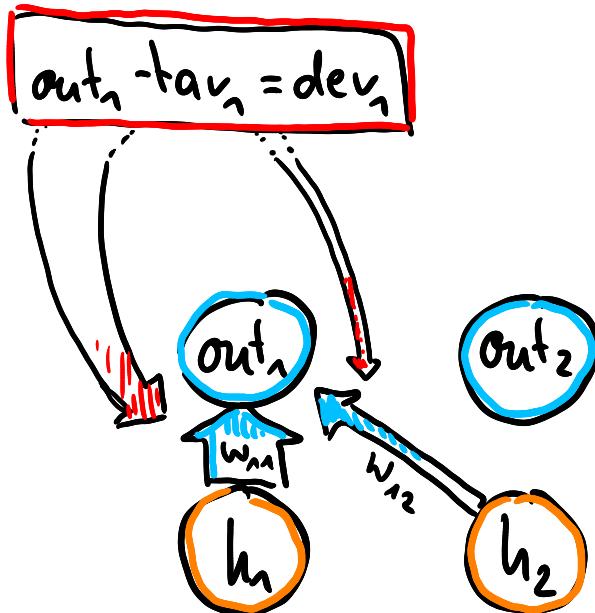
$$\frac{\partial E_t}{\partial w_{kj}} = \delta_k^{out} out_j^{hid}$$

$$\frac{\partial E_t}{\partial w_{ji}} = \sum_{k=1}^n (out_k - tar_k) \frac{\partial out_k}{\partial in_k^{out}} \frac{\partial in_k^{out}}{\partial out_j^{hid}} \frac{\partial out_j^{hid}}{\partial in_j^{hid}} \frac{\partial in_j^{hid}}{\partial w_{ji}}$$

$$\frac{\partial E_t}{\partial w_{ji}} = \delta_j^{hid} out_i^{in}$$

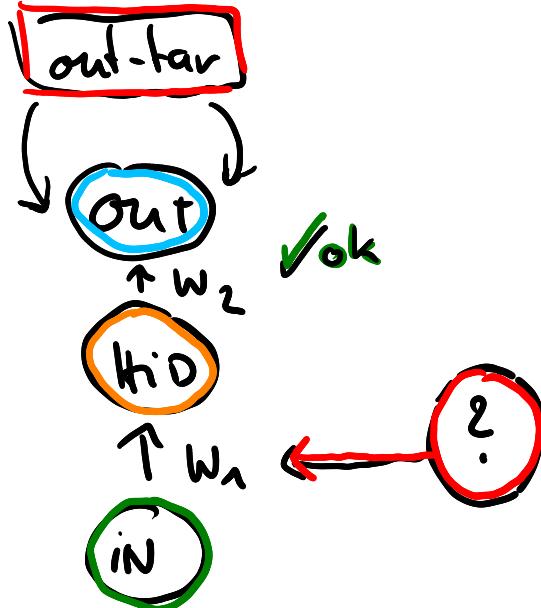
mit  $\delta_j^{hid} := f'_1(in_j^{hid}) \sum_{k=1}^n w_{kj} \delta_k^{out}$

# Interpretation des Fehlergradienten – die Vorstellung von der Backpropagation

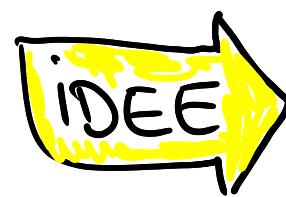


UPDATE  
 $w_2$

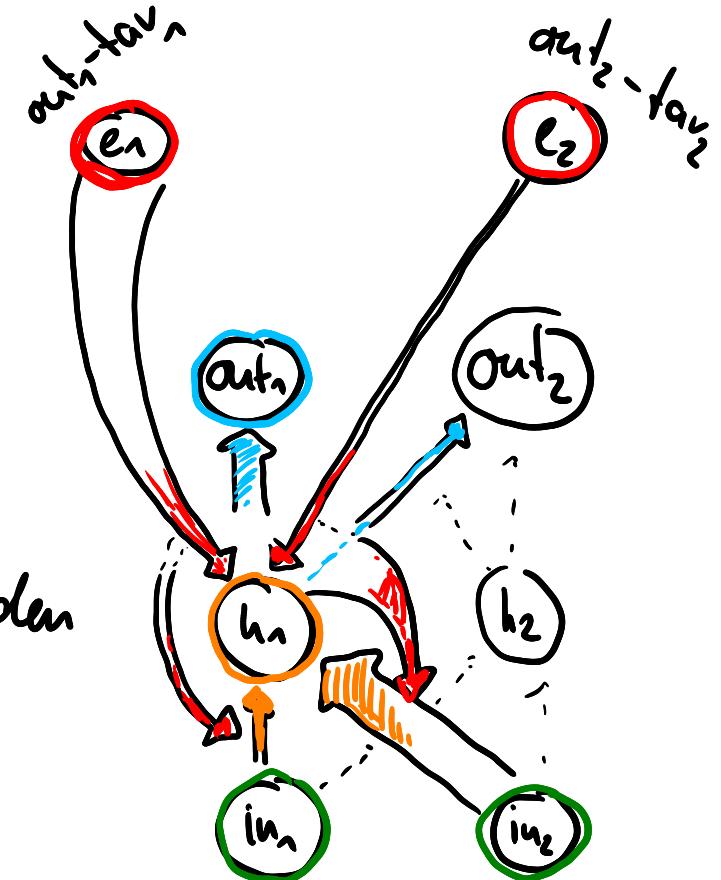
- ein Output wird durch die gewichtete Summe der Beiträge der Hidden-Neuronen erzeugt
- der Fehler  $out_i - tar_i$  stammt also je nach Größe der Beiträge (anteilig  $w_{ij}$ ) von den einzelnen Neuronen
- das Gewicht, das für den Hauptteil des Fehlers verantwortlich ist muss am meisten korrigiert werden

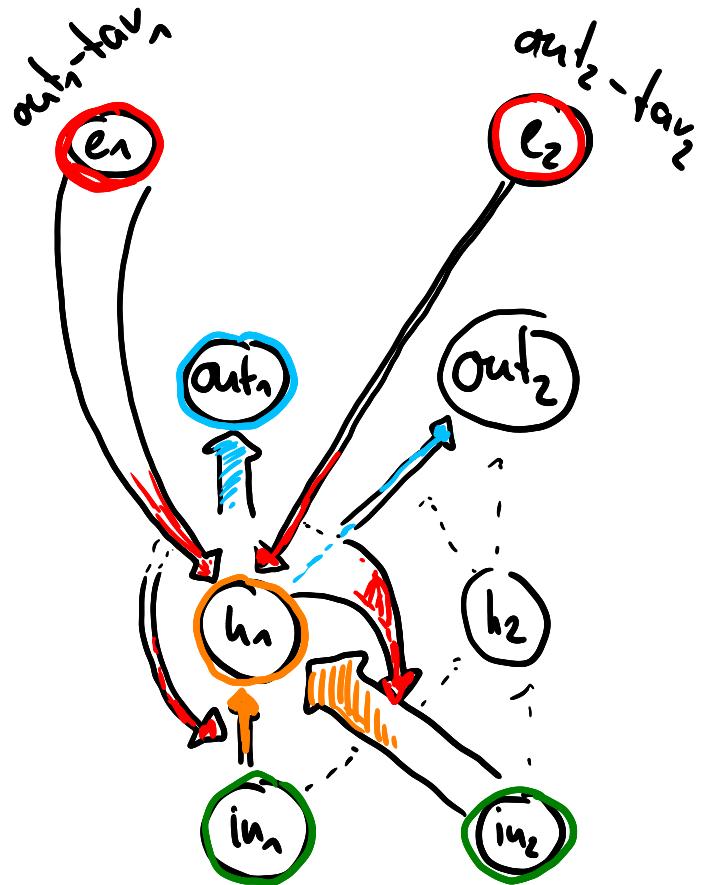


An der Hidden-Schicht hat man eigentlich keinen Fehler, den man analog zum Update von  $w_1$  anpassen könnte!



- sammle den aut.ig "zurück-fließenden" Fehler an den Hidden-Neuronen auf und verteile ihn wieder entsprechend der Beiträge von den Input-Verbindungen ...





Der berechnete Fehler "fliegt" von oben nach unten durch das Netz und wird jeweils lokal für das Gewichtsupdate genutzt

⇒ BACK PROPAGATION



Bemerkung: der Algorithmus zur Berechnung des Gradienten ist "lokal"

⇒ Berechnung ist effizient machbar!

$$E = \sum_{t=1}^T (\text{out}_t - \text{tar}_t)^2 \rightarrow \min_{w_1, w_2}$$

$$y = w_2 f(w_1 x)$$



KORRESPONDENZ-PRINZIP

output  $y$

$$w_2 \uparrow$$

$$\Delta w_2 = -\eta \frac{\partial E}{\partial w_2} = \uparrow * \downarrow$$

$$f(z) = \tanh(z)$$

$$w_1 \uparrow$$

input  $x$

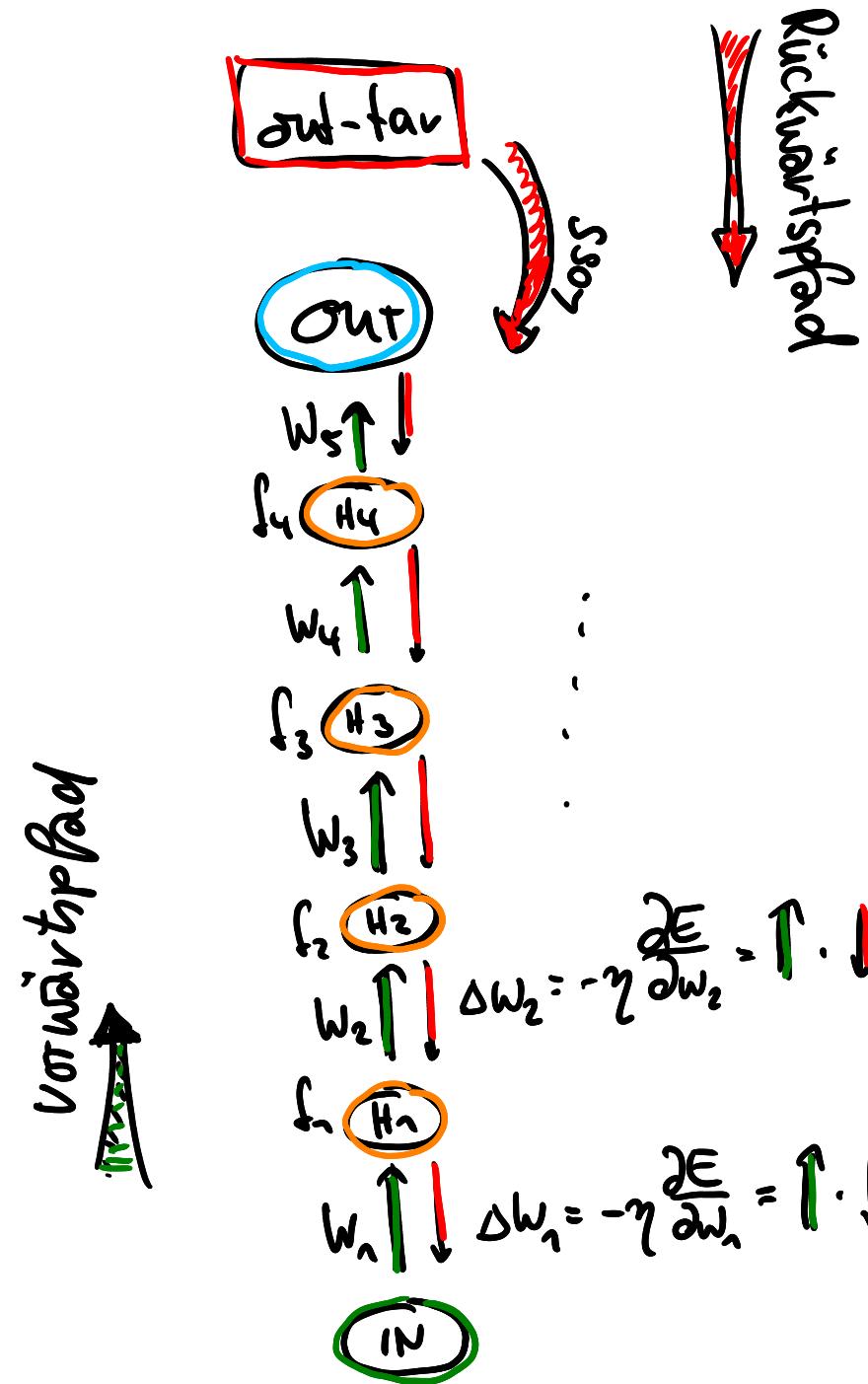
$$\Delta w_1 = -\eta \frac{\partial E}{\partial w_1} = \uparrow * \downarrow$$

Gleichungen

Architekturen

lokale Algorithmen

Die Topologie lässt sich beliebig erweitern!



- für jede Berechnung werden wir die Infos aus dem Vorwärtspfad benötigt, die lokal relevant sind

➡ Vorwärtspfad  $\hat{=}$  Informationsfluss vorwärts durch das Netz

➡ Rückwärtspfad  $\hat{=}$  Fehlerfluss rückwärts durch das Netz

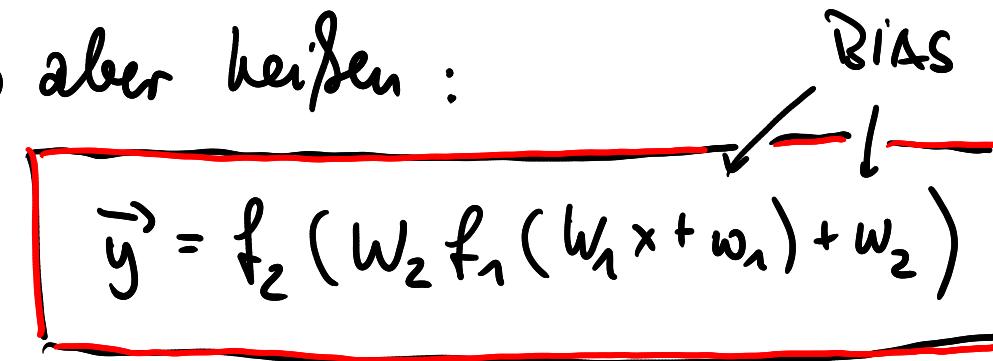
... können beide durch Anpassung der Topologie gezielt gesteuert werden ...

Bemerkung:

Für die Herleitung des Gradientenabstiegs halten wir die Transferfunktion

$$\vec{y} = f_2(w_2 f_1(w_1 \vec{x})) \quad \text{betrachtet}$$

eigentlich müsste es aber heißen:



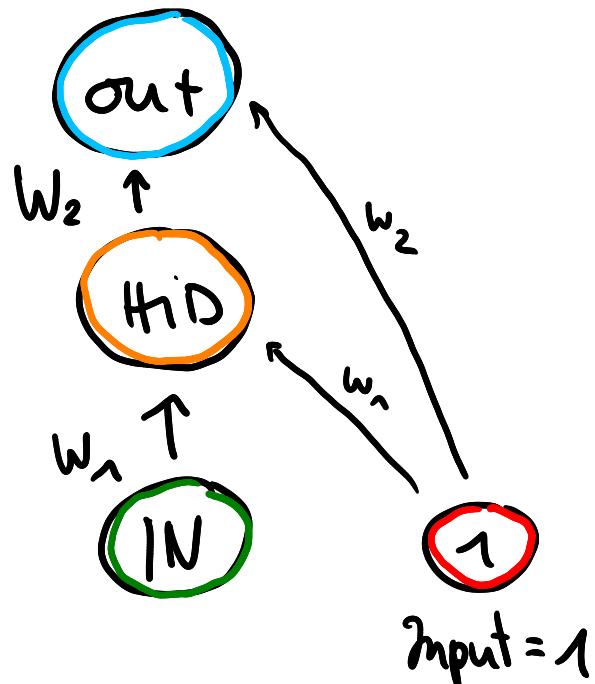
$$\vec{y} = f_2(w_2 f_1(w_1 x + w_0) + w_2)$$

Und jetzt ???

Müssen wir alles nochmal rechnen ???

NEIN, denn:

Der Bias äußert sich im Netz (Topologie) ganz einfach durch einen zusätzlichen Input = 1 realisieren  $\Rightarrow$  alles bestens 😊



- so wird das meistens auch in der Software implementiert
- ob man den Bias wirklich braucht, hängt von der jeweiligen Aufgabe ab.

... bisher hatten wir uns den Gradientenabstieg so vorgestellt, dass für jedes einzelne Trainingsbeispiel ein Gewichtsupdate erfolgt.

Dieses Verfahren bezeichnet man als "pattern by pattern" Lernen.

→ "Pattern by Pattern"-Lernen ist von großer Stochastizität geprägt.  
jedes einzelne Sample beeinflusst (gleichwertig) die Entwicklung der Gewichte.



Gibt es noch andere Möglichkeiten?

### 5.3.2) Lernregeln

314

Der Task besteht darin das Neuronale Netz so anzupassen, dass der Trainingsfehler (insgesamt) minimal wird:

$$E = \frac{1}{T} \sum_{t=1}^T E_t = \frac{1}{T} \sum_{t=1}^T (\text{out}_t - \text{tar}_t)^2 \rightarrow \min_w$$

Sei:  $g_t = \frac{\partial E_t}{\partial w}$  sowie

↑  
Gradient für  
ein Trainings-  
beispiel

$$g = \frac{1}{T} \sum_{t=1}^T g_t$$

← mittlerer Gradient für alle  
Trainingsbeispiele

## ① "Pattern-by-Pattern"

heißt dann :

$$\Delta w_t = -\eta g_t = -\eta g - \eta(g_t - g)$$

↑                                      ↑  
 mittlerer                            stochastische  
 Gradient                            Säiche



- das Verfahren ist leicht zu rechnen
- die Stochastizität bei der Säiche hilft gegen das Problem im lokalen Minimum stecken zu bleiben
- bei viel Redundanz in den Daten konvergiert das Verfahren schnell

## PROBLEM

- Einzelne Ausreißer oder Fehler in den Trainingsdaten können den Lernfortschritt komplett einziehen!

$$\Delta w_t = -\eta g_t = -\eta g - \eta(g_t - g)$$

↑                      ↑  
 mittlerer            stochastische  
 Gradient            Suche

einzelnes Sample bestimmt über das Update!

⇒ Bei vielen (vermischten) Fehlern in den Daten könnte es ja eine gute Idee sein, den stochastischen Teil  $-\eta(g_t - g)$  aufzugeben...

## ② "Steepest Descent"

... jetzt nur noch

$$\Delta w = -\eta g$$

= "Schrittweite · im Mittel steiler Abstieg"



kaum man zeigen, dass damit der Fehler tatsächlich reduziert wird?

→ Betrachte die Taylor-Entwicklung (2. Ordnung) des Fehlers in  $w$ :

$$E(w + \Delta w) = E(w) + g^T \Delta w + \frac{1}{2} \Delta w^T G \Delta w$$

$$G = \frac{1}{T} \sum_{t=1}^T \frac{\partial^2 E_t}{\partial w_i \partial w_j}$$

2. Ableitung

$$\Delta w = -\eta g$$

einsetzen

$$= E(w) - \eta g^T g + \underbrace{\frac{\eta^2}{2} g^T G g}_{\text{klein für kleine } \eta} \approx E(w) - \eta g^T g < E(w)$$



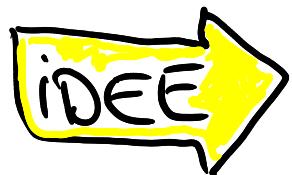
Folgerung: Steepest Descent ist also auch keine schlechte Idee!

Allerdings:

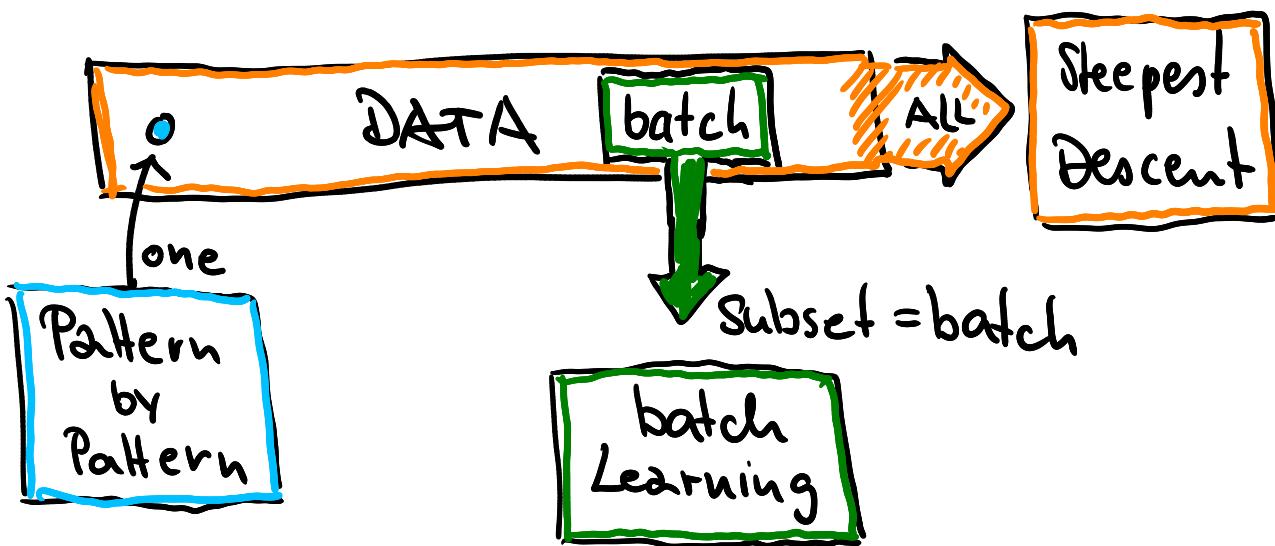
- mehr Rechenaufwand
- langsamere Konvergenz
- Gefahr in einem lokalen Minimum stecken zu bleiben



Gibt es vielleicht einen goldenen Mittelweg zwischen den beiden Verfahren?



Verwende nicht alle \ ein Trainingspattern  
zum Berechnen von  $g = \frac{1}{T} \sum_{t=1}^T g_t$  sondern  
nur eine Teilmenge  
→ BATCH



Meta-Parameter : BATCH-SIZE !

- geringere Stochastizität als "Pattern by Pattern"
- Stabilität gegen Ausreißer
- Parallelisierbarkeit

Bemerkung: Andere Lernverfahren adhären die Probleme, die sich aus einer festgelegten Lernrate  $\eta$  ergeben

- Überspringen guter Minima
- langsame Konvergenz bei "flacher" Fehlerfläche
- ...

Beispiel:

④ "Vario Eta"



Variiere  $\eta$  im Abhängigkeit von der Struktur des Fehlergradienten ...

~

$$\Delta w_t = \sqrt{\frac{1}{T} \sum (g_t - g)^2} \quad g_t$$

- viel Zickzack in  $w_2$ -Richtung  $\Rightarrow g_t$  sehr verschieden von  $g$

$\Rightarrow$  großer Nenner

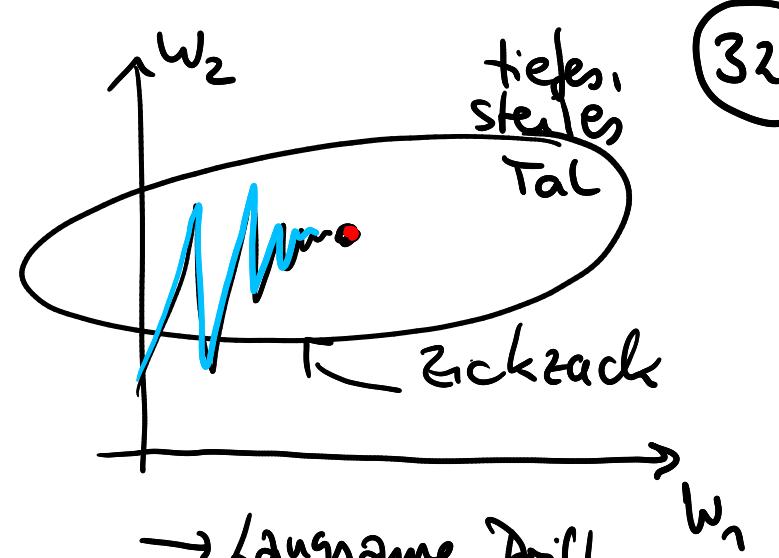
$\Rightarrow$  Bewegungsrichtung in  $w_2$  gedämpft

- leise Drift in  $w_1$ -Richtung

$\Rightarrow g_t$  sehr ähnlich zu  $g$

$\Rightarrow$  kleiner Nenner

$\Rightarrow$  Speed-Up in  $w_1$ -Richtung (Richtung Minimum)



$\rightarrow$  Langsame Drift  
im Richtung Min

321

⇒ Methoden mit variablen  $\eta$  sorgen für  
Beschleunigung beim Lernen!

Weitere Beispiele: → Momentum Backprop:

$$\begin{aligned} v_0 &= 0 \quad \beta = 0,9 \\ v_{t+1} &= \beta v_t + g_t \\ \Delta w_t &= -\eta v_{t+1} \end{aligned}$$

→ ADAM  
(adaptive Gradient + Momentum)

Kingsma & Ba (2015)

adaptive Lernrate aus Gradienten  
1. und 2. Ordnung

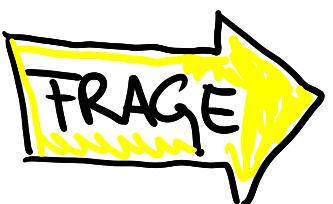
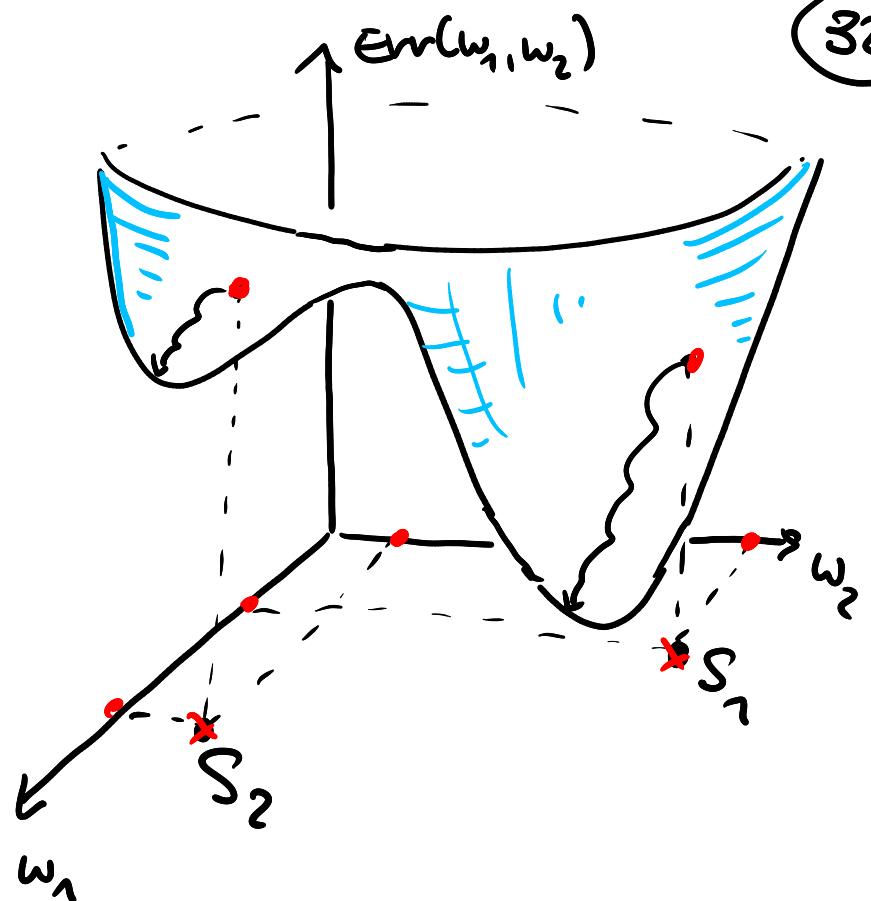
⊕ viele weitere Varianten ...

## Beobachtung:

Zu Beginn des Lernens müssen die Gewichte in den Matrizen  $W_1$  und  $W_2$  zufällig belegt werden, um einen Startpunkt für das Lernen zu haben.

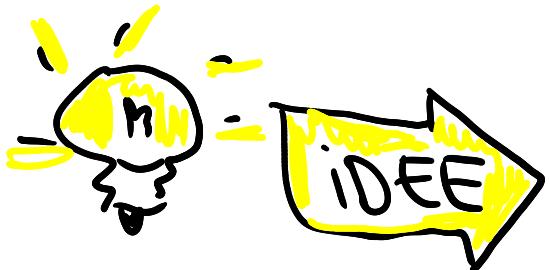


323

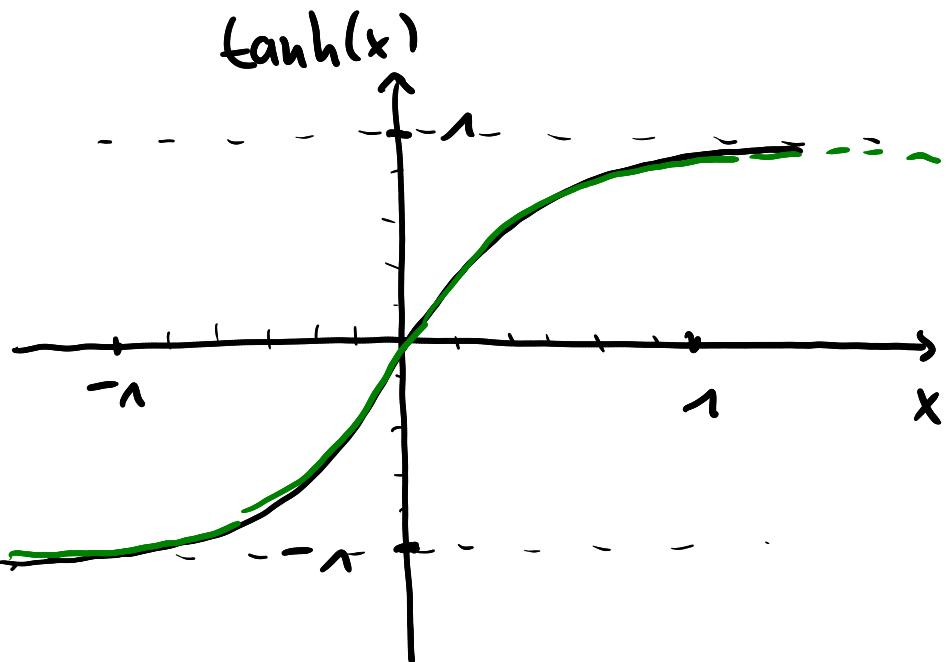


Wie macht man das am besten?

← Welche Größenordnung?



Die Aktivierungsfunktion der ersten Hidden-Schicht ist der Schlüssel!

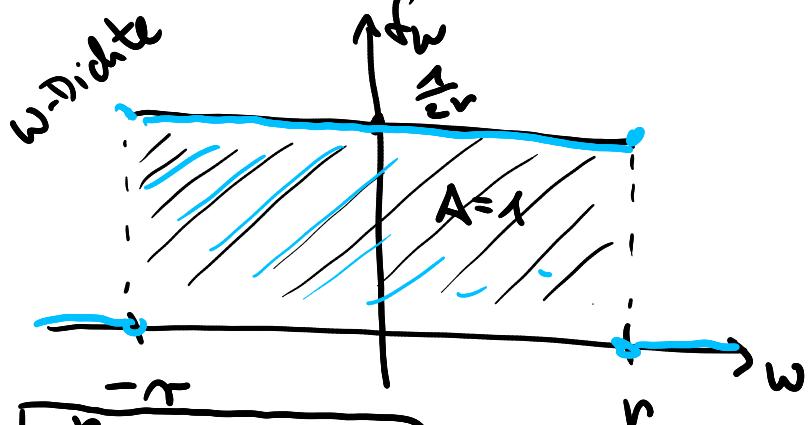


- für standardisierte Inputs mit  $E(X_i) = 0 \quad \sigma^2(X_i) = 1$  sollten die Gewichte so gewählt werden, dass die Eingangssignale in die Hidden-Layer  $W_h \vec{x}$  (ungefähr) im linearen Bereich des  $\tanh(\dots)$  liegen.

$$\Rightarrow \tanh\left(\sigma\left(\sum_{i=1}^h w_i x_i\right)\right) = 1$$

Seien die  $w_i$  uniform verteilt mit

$$w_i \in (-r; r)$$



$$\int_{-r}^r \frac{1}{2r} dw = 1$$

$$\Rightarrow \sigma^2(w) = \int_{-r}^r \frac{1}{2r} w^2 dw = \frac{1}{2r} \left[ \frac{1}{3} r^3 + \frac{1}{3} (-r)^3 \right] = \\ = \frac{1}{2r} \cdot \frac{2}{3} r^3 = \frac{1}{3} r^2$$

$$\Rightarrow \boxed{\sigma(w) = \frac{r}{\sqrt{3}}}$$

für  $x_i$  uniform verteilt  $x_i \in (-1; 1) \xrightarrow{\text{analog}}$

$$\boxed{\sigma(x_i) = \frac{1}{\sqrt{3}}}$$

für unabhängige  $w_i, x_i$  folgt dann mit  $f_1(w_i) = f_1(x_i) = 0$

$$\sigma(w_i x_i) = \sigma(w_i) \sigma(x_i) = \frac{r}{\sqrt{3}} \cdot \frac{1}{\sqrt{3}} = \frac{r}{3} \quad \rightarrow$$

$$\boxed{\sigma(w_i x_i) = \frac{r}{3}}$$

Müsse jetzt das Gesetz der Großen Zahlen:

$$\sigma \left( \sum_{i=1}^n (w_i x_i) \right) = \sqrt{n} \sigma(w_i x_i) = \sqrt{n} \cdot \frac{r}{3}$$

Also muß gelten:

$$\boxed{\sqrt{n} \cdot \frac{r}{3} = 1}$$

$$\Rightarrow \boxed{r = \frac{3}{\sqrt{n}}}$$

keine  
Empirie !!

Für die Initialisierung der Gewichte  
w<sub>i</sub> folgt also



$$\boxed{w_i \in \left( -\frac{3}{\sqrt{n}} ; \frac{3}{\sqrt{n}} \right)}$$

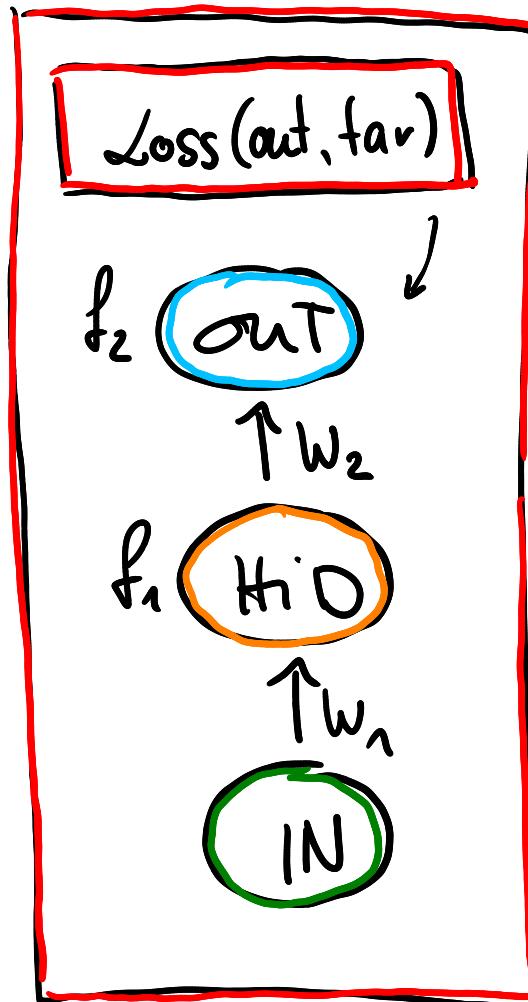
Damit läßt sich ein Argument finden (Statistik), das die Größenordnung der Gewichte bei der Initialisierung motiviert.

... das "Standardprogramm"  $w_i \in (-0,3 ; +0,3)$  funktioniert aber  
weiterhin auch ganz gut 😊

# METAPARAMETER

## TOPOLOGIE

- Anzahl der Schichten
- Anzahl der Neuronen pro Schicht
- Typ der Aktivierungsfunktionen  $f_1$  und  $f_2$
- Bias?
- wo gibt es Konektoren?



## LERNEN

- Loss-function
- Initialisierung der Gewichte
- $\eta = ?$  bei Verfahren mit festem  $\eta$
- welches Lernverfahren?
- Batch-size
- Anzahl der Epochen?

→iemlich viele Optionen ...

### Gute Nachricht:

An vielen Stellen gibt es mathematisch begründete Argumente für eine "gute" Wahl!

Mehr dazu beim nächsten Mal ...