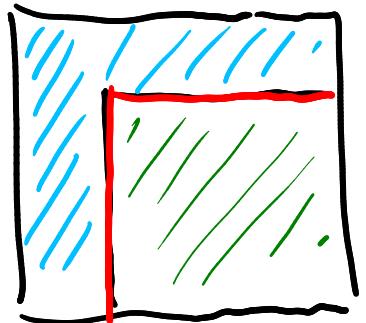


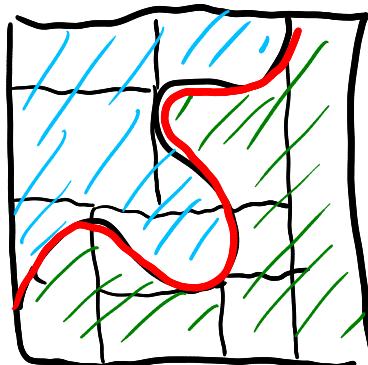
Beim letzten Mal haben wir Varianten von Decision Trees gesehen, die die Performance eines einzelnen Baumes verbessern

→ Bagging, Random Forest, Boosting ...

Die Verbesserung hat aber logischerweise ihre Grenzen



aber



← Decision-Boundary
hochgradig nicht-
linear =>

⇒ Wir brauchen eine Modellklasse, die besser mit Nicht-Linearity umgehen kann !!

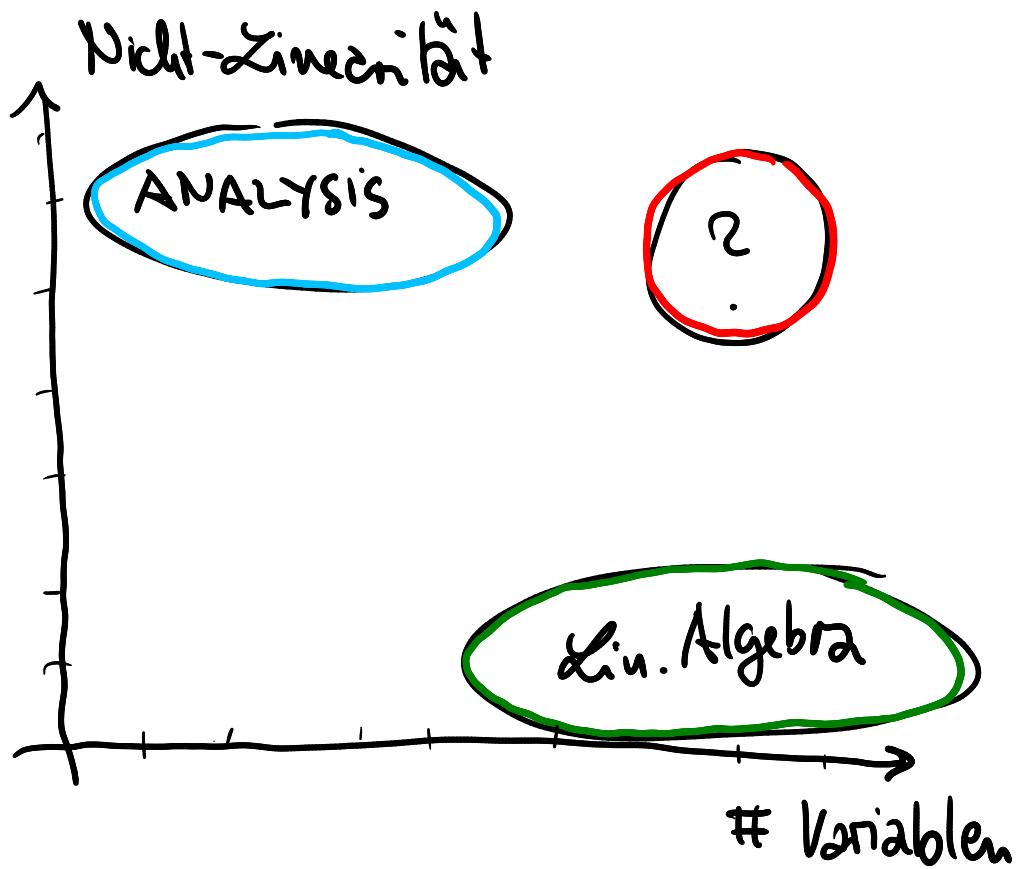
5.) Neuronale Netze

Überlegung: Für die Modellierung komplexer Systeme (sowohl Regression als auch Klassifikation) braucht man eine Modellklasse, die auch Nicht-Linearität in der Zusammensetzung-Struktur zufriedenstellend abbilden kann.

→ Klassische Verfahren kommen aus dem Bereich der Analysis

Hier gibt es allerdings Probleme, wenn zu viele erklärende Variablen (Features X_1, \dots, X_p , großes p) beteiligt sind.

Dann kommen oft Methoden aus der linearen Algebra zum Einsatz \Rightarrow nicht-lineares Verhalten kann nicht beschrieben werden...

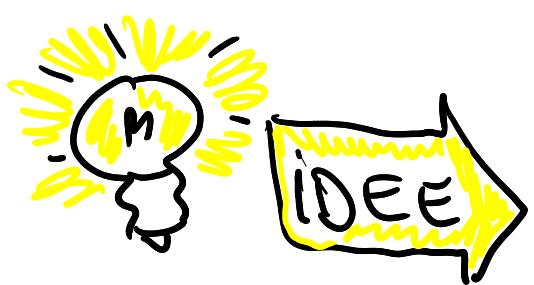


→ Dazwischen gibt es
eine "Lücke"

↓
diese "Lücke" können
Neuronale Netze schließen!

5.1) Grundideen Neuronaler Netze

Die Entwicklung Neuronaler Netze begann schon
sehr früh ...



Im menschlichen Gehirn sind (unglaublich) viele Nervenzellen miteinander verschaltet. Lernen findet als Interaktion dieser Zellen statt.



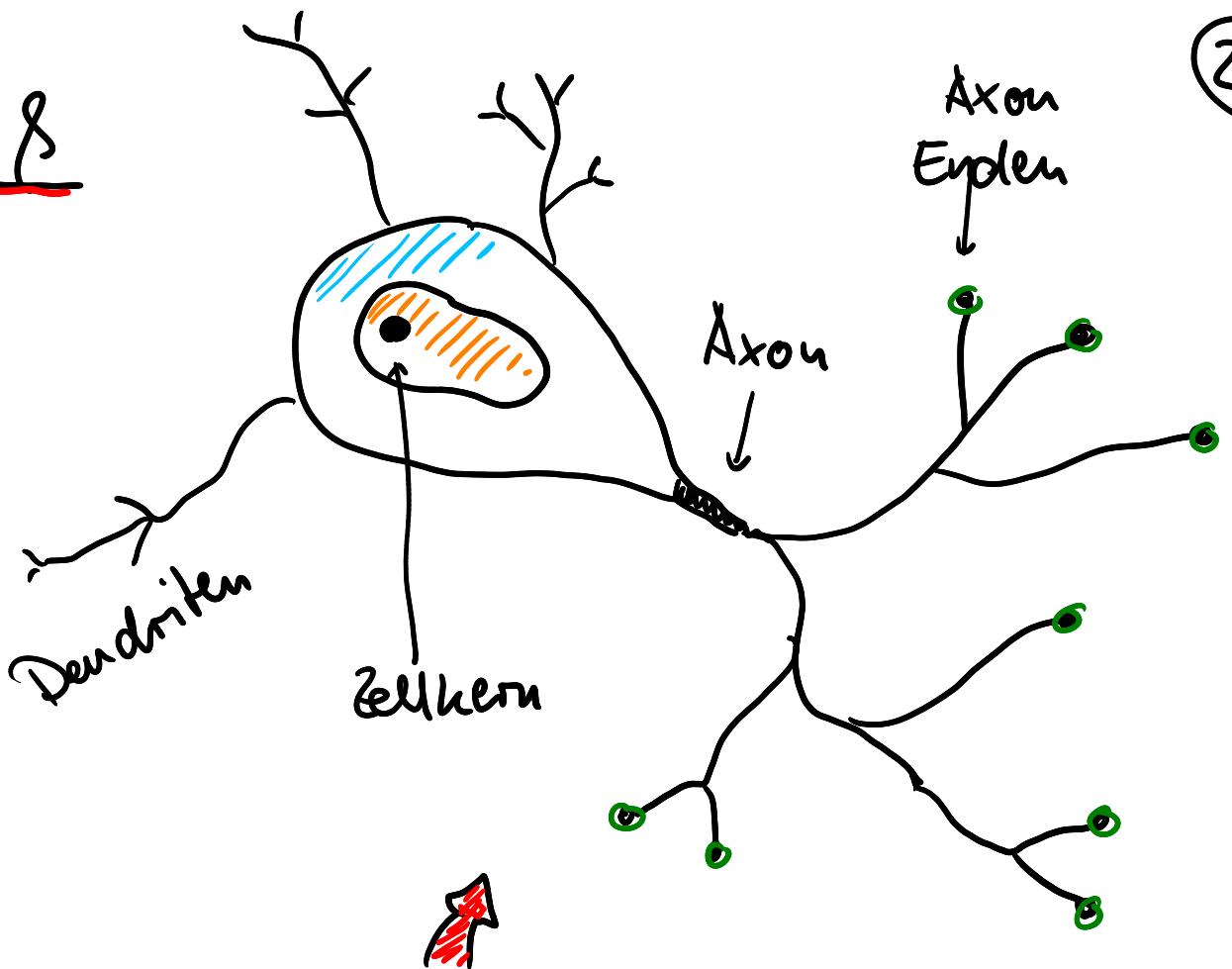
Bilde zunächst eine einzelne Nervenzelle durch ein mathematisches Modell ab und verschalte dann mehrere dieser mathematischen Neuronen.

1943

Warren McCulloch &

Walter Pitts

285



Biologisches NEURON

- Reize werden von den Dendriten aufgenommen
- gerichtete Weiterleitung zum Zellkern
- Bündelung im Zellkern
- Weiterleitung zum Axon, wenn eine gewisse Reizschwelle überschritten ist
- bei Überschreitung: neuer Reiz geht an die Axon-Enden und wird von dort (als neuer Eingangssignal) zu anderen Neuronen weitergeleitet.

Def.: Mathematisches Neuron

Betrachte eingehende Reize x_1, \dots, x_m mit Gewichten w_1, \dots, w_m ($w_i \in \mathbb{R}$), ein Schwellenpotential Θ und eine Threshold-Funktion (Aktivierungsfunktion) f .

Ein mathematisches Neuron ist dann wie folgt aufgebaut:

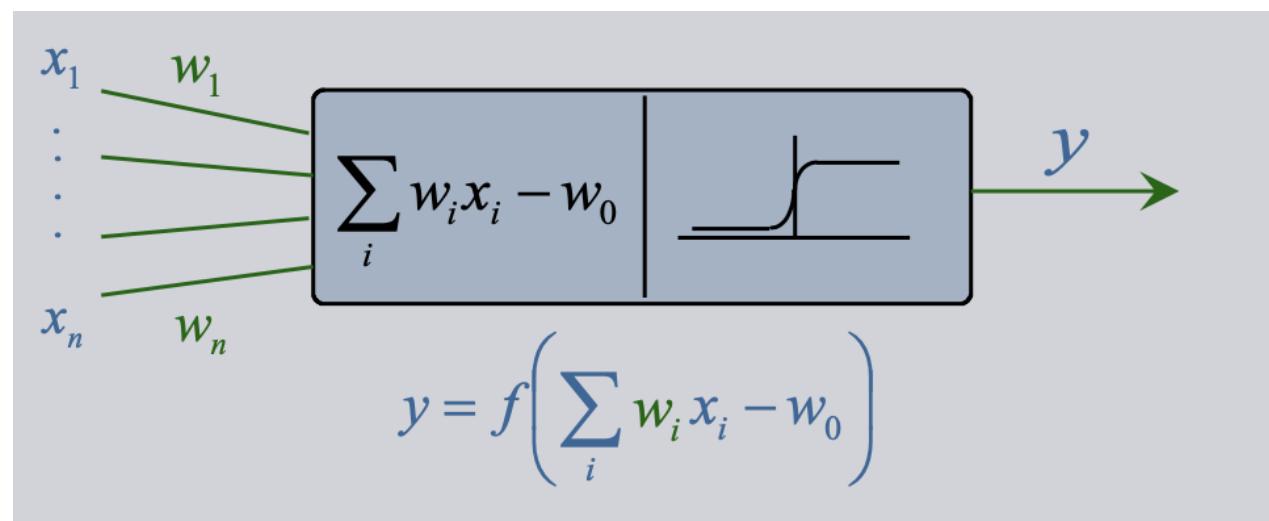
x_1, \dots, x_m : Inputs

$w_1, \dots, w_m \in \mathbb{R}$: Gewichte

w_0 : threshold

f : Aktivierungsfunktion

y : Output



Zum einfachsten Fall ist f eine Sprungfunktion mit Schwellenwert $\Theta \in \mathbb{R}$

$$y = f_{\Theta}(\tilde{x}) = \begin{cases} 1 & \text{für } \tilde{x} > \Theta \\ 0 & \text{für } \tilde{x} \leq \Theta \end{cases}$$

d.h. das Neuron "feiert"!

Den Schwellenwert Θ nennt man auch Bias.

Er kann bereits beim Zusammenfassen berücksichtigt werden:

$$y = f\left(\sum_{i=1}^n w_i x_i - \Theta\right)$$

oder in Vektornotation

$$y = f(\vec{w} \cdot \vec{x}^t - \Theta)$$

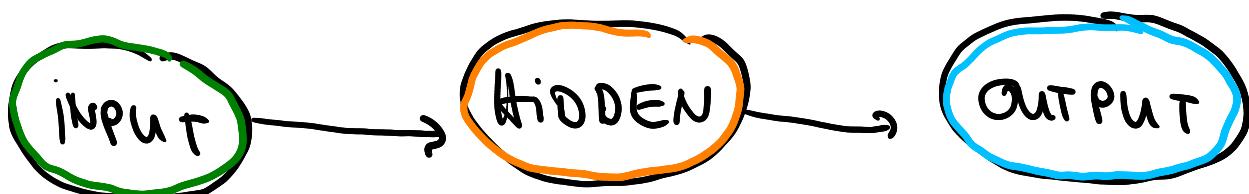
VORTEIL: die Aktivierungsfunktion f ist identisch für alle möglichen Bias-Werte $\Theta \in \mathbb{R}$.

Die so formulierten mathematischen Neuronen bilden die Grundbausteine Neuronaler Netze!

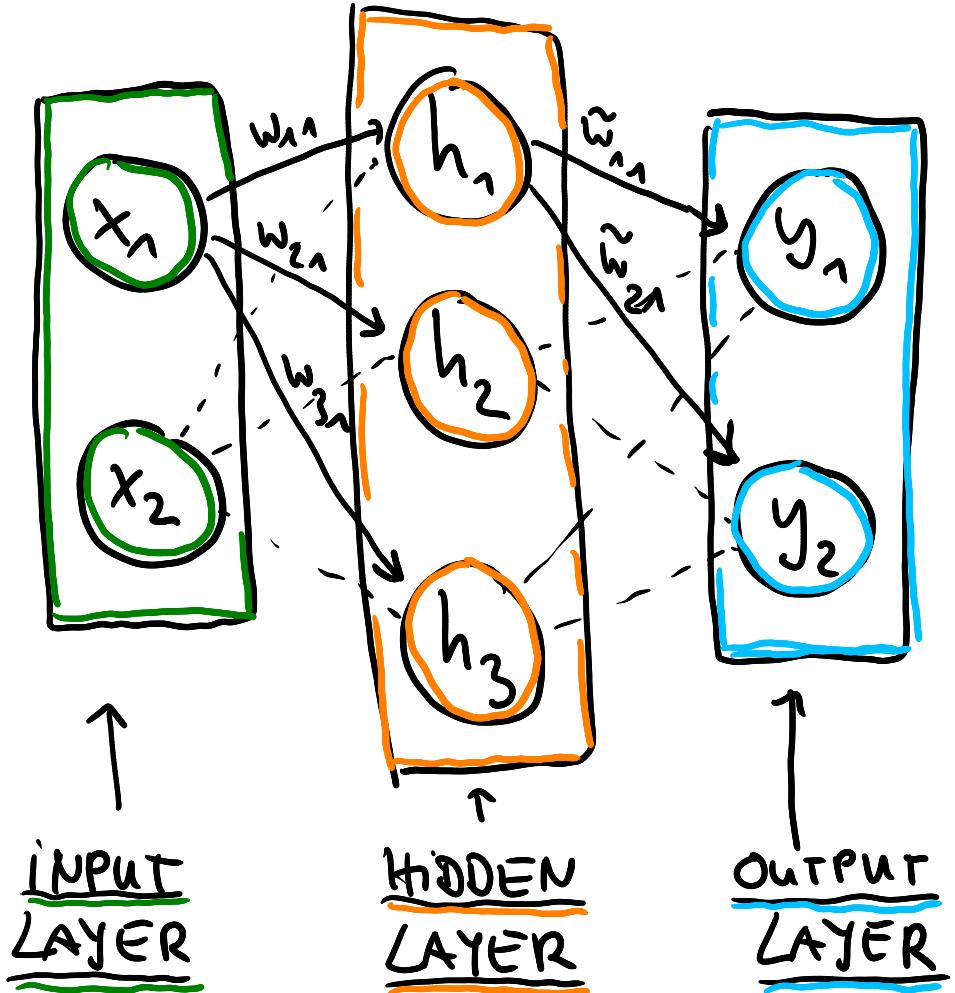
Wir müssen sie "nur" noch sinnvoll verschalten...

5.2) Das 3-layer Perzeptron

Die einfachste Form der Verschaltung mehrerer aktiver Neuronen zu einem Neuronalen Netz ist im sogenannten 3-Layer-Perzeptron realisiert:



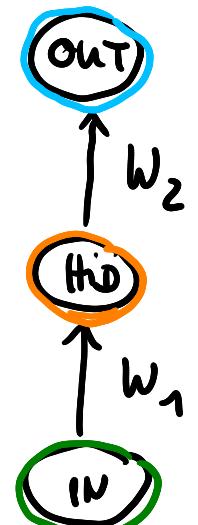
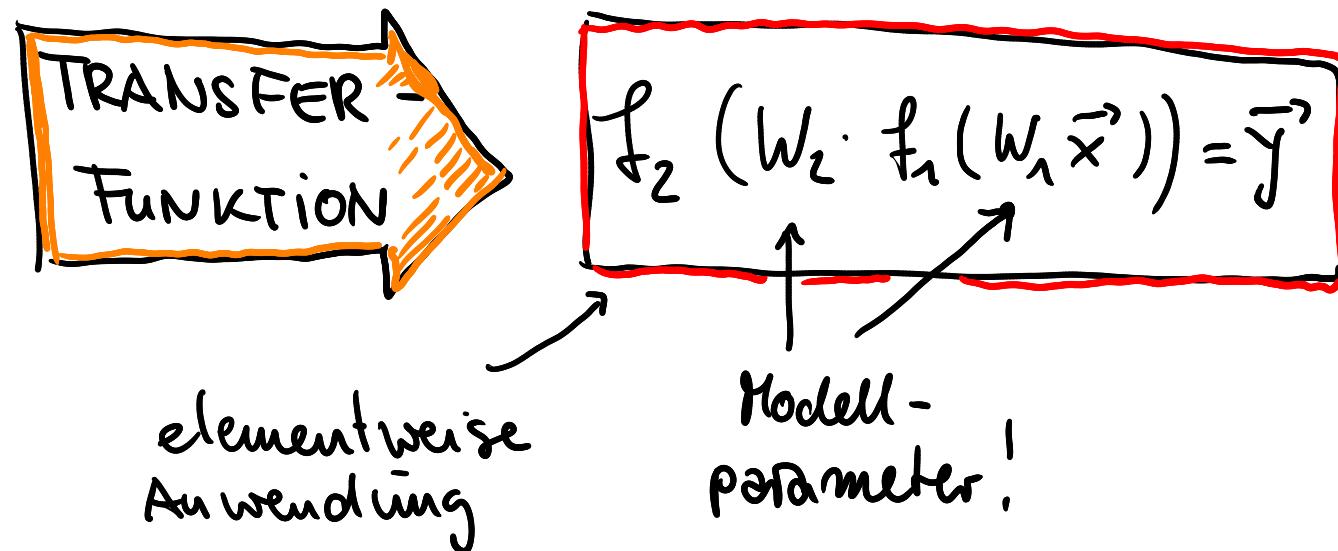
3-layer Perzeptron



- Inputs sind in der Input-Layer zusammengefasst
- jeder Input interagiert mit jedem Neuron der Hidden-Layer
- jedes Neuron gibt einen gewichteten Output weiter
- alle Outputs werden in der Output-Layer aggregiert

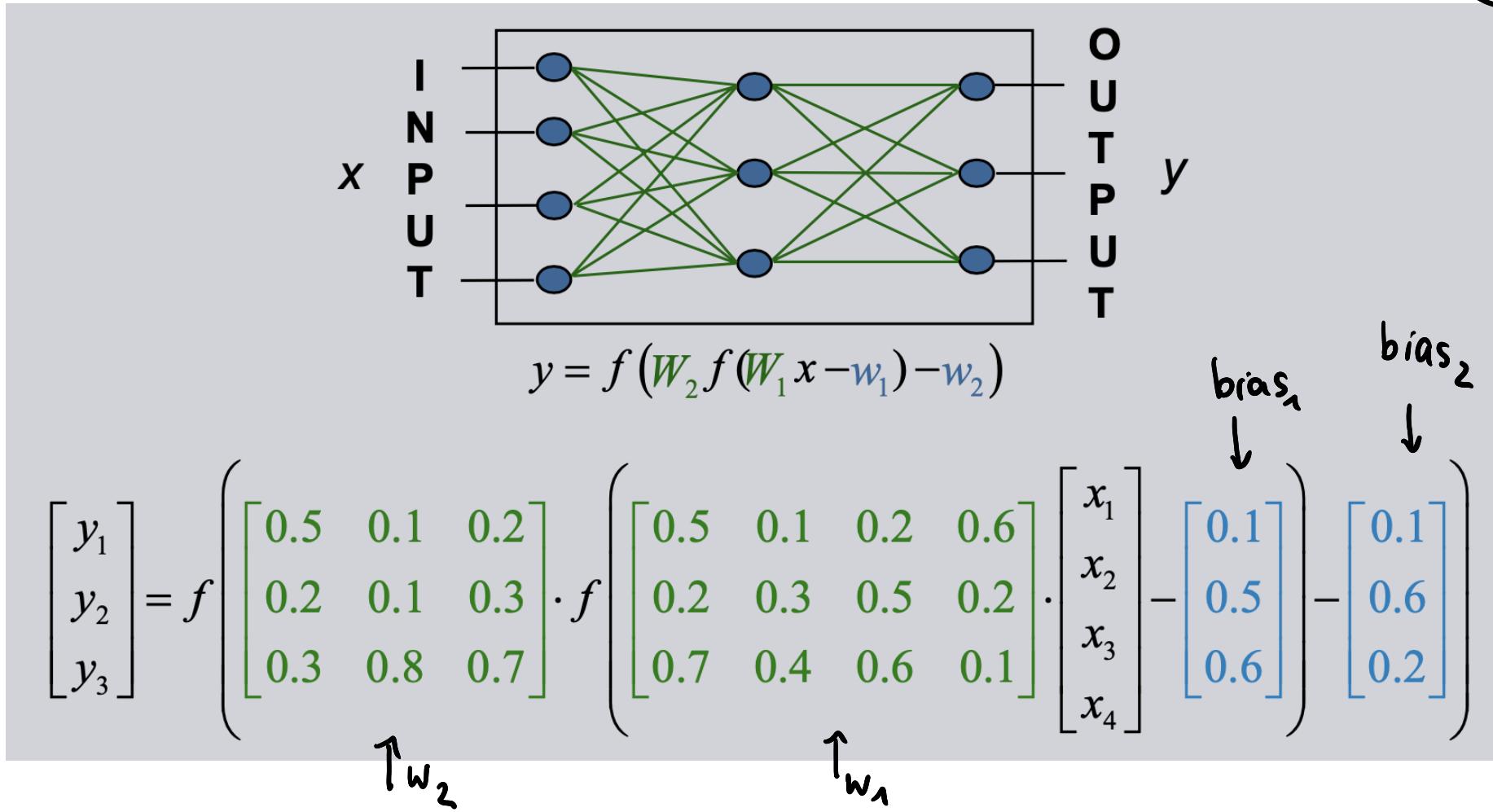
→ Sinnvolle Darstellung im Matrix Schreibweise

- einzelne Inputs lassen sich als Vektor darstellen
- Gewichte in den Konnektoren werden in Matrizen zusammengefasst
- Aktivierungsfunktionen f_1 (Hidden Layer), f_2 (Output-Layer)



Topologie des Netzes

Bsp.:



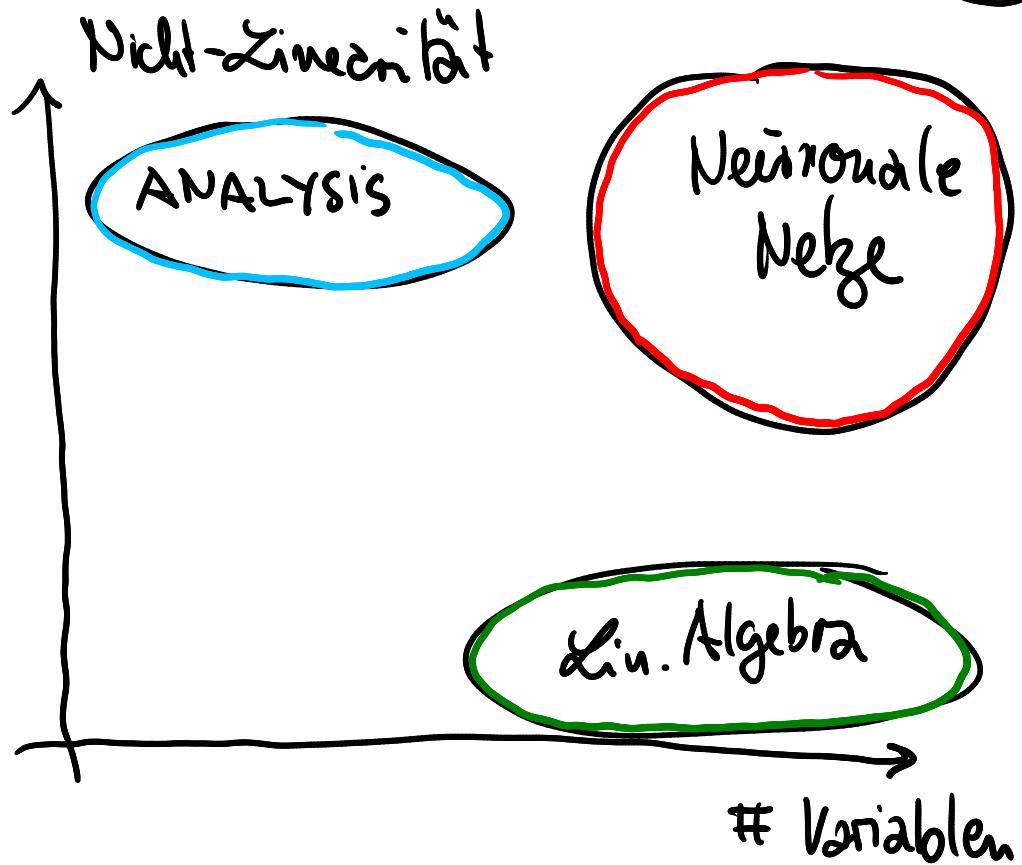
$$f\left(\begin{pmatrix} z_1 \\ z_2 \\ z_3 \end{pmatrix}\right) = \begin{pmatrix} f(z_1) \\ f(z_2) \\ f(z_3) \end{pmatrix} \quad \text{mit z.B.}$$

$$f(z) = \text{logistic}(z) = \frac{1}{1+e^{-z}}$$

$$f(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

← Aktivierungsfunktionen

- Neuronale Netze sind eine Verknüpfung von linearen und nicht-linearen Funktionen
- Sie schließen die Lücke!



Bemerkungen:

- ① Im Unterschied zu z.B. Taylor - Approximation wird wachsende Komplexität nicht additiv modelliert sondern durch steigende Zahl von Verknüpfungen
- ② Die Nicht-Linearität (Aktivierungsfunktion) ist essenziell! Sonst: lineare Abbildung \Leftrightarrow

Tatsächlich stellt eine höhere Komplexität keine Erweiterung der Funktionenklasse dar!

(heißt nicht, dass eine "flache" Topologie immer auch effizient sein muss)

DENN:

➡ UNIVERSAL APPROXIMATION

THEOREM (Hornik, Stinchcombe, White 1989)

3-layer NNs

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

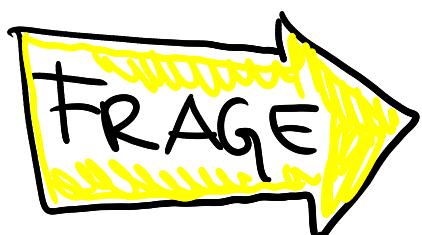
können jede stetige Funktion auf einem kompakten Träger beliebig genau approximieren.

Die Transferfunktion des 3-Layer NNs beschreibt erstmal nur den Informationsfluss vorwärts durch das Netz



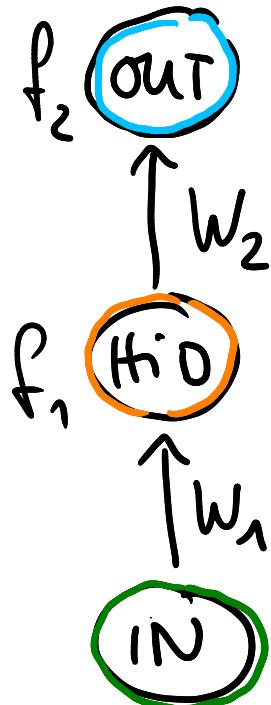
VORWÄRTSPFAD

Mit "Lernen" hat das erstmal nichts zu tun ...



Wie sieht bei einem Neuronalen Netz die Lernaufgabe aus?

Überlegung:



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

output

- für eine beliebige Belegung der Gewichtsmatrizen ergibt sich ein großes Residuum

$$\vec{\tau} = \hat{\vec{y}} - \vec{y}$$

→ "Lernen" heißt:

finde optimale Gewichte für w_1 und w_2 , so dass der Fehler $out - \tau$ minimal wird!

5.3) Lernen!

Formalisierung der Lernaufgabe 

- Gewichte im W_1 :
Inputs . # Neuronen
- Gewichte im W_2 :
Neuronen . # Outputs



Anderer als z.B. bei der linearen Regression gibt es keine analytische Lösung für die Lernaufgabe!

bestimme die Gewichte in W_1 und W_2 so, daß für $t=1, \dots, T$ Trainingsbeispiele eine Fehlerfunktion z.B.

$$\frac{1}{T} \sum_{t=1}^T (\hat{y}_t - y_t)^2$$

minimal wird.

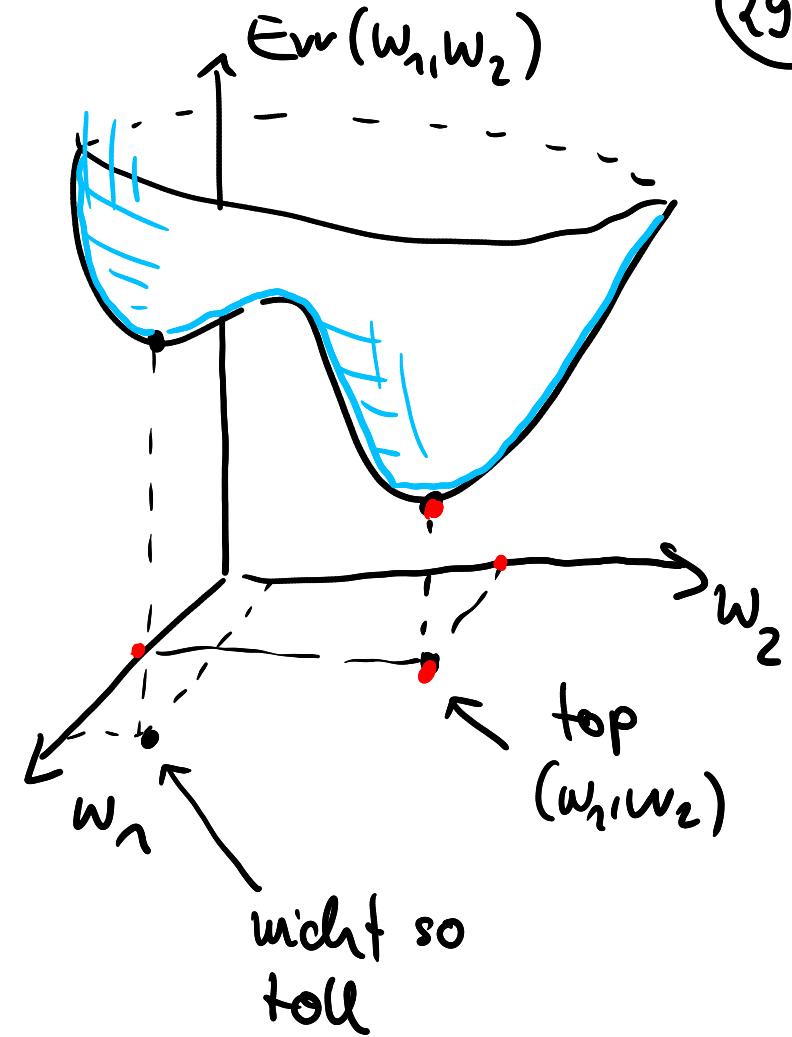
AUSGANGSLAGE :

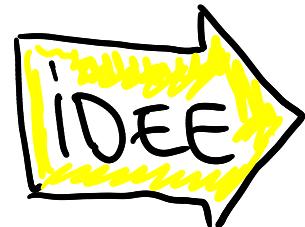
- T Trainingsbeispiele (\vec{x}_t, \vec{y}_t)
 $(t=1, \dots, T)$
- Zielvektor \vec{y}_t mit K Komponenten
- Fehlerfunktion für das t-te Sample

$$E_t = \sum_{k=1}^K \frac{1}{2} (\text{out}_k - \text{tar}_k)^2$$

Funktion der Gewichte

↑ globales Minimum
wäre perfekt
→ finden wir aber nicht

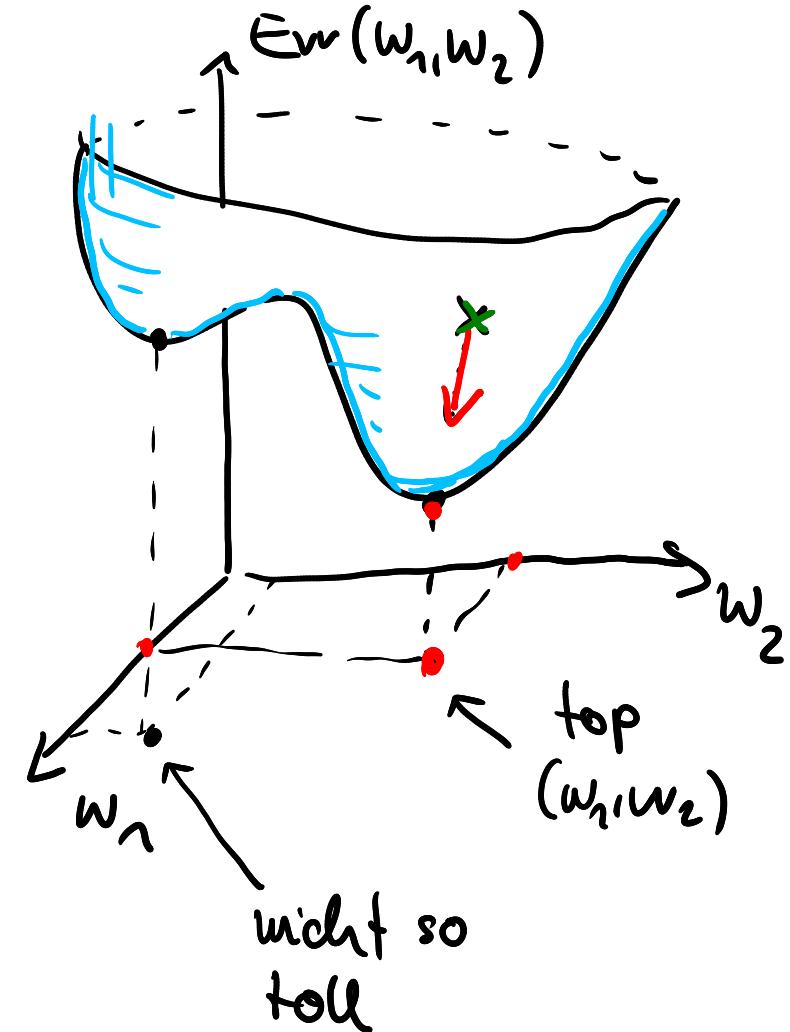




Gradientenabstieg

Benütze den Gradienten der Fehlerfläche um von einer beliebigen Stelle aus die Richtung zu bestimmen, in der es am steilsten "bergab" geht

⇒ Richtung in der die Gewichte angepasst werden müssen um maximal besser zu werden ...

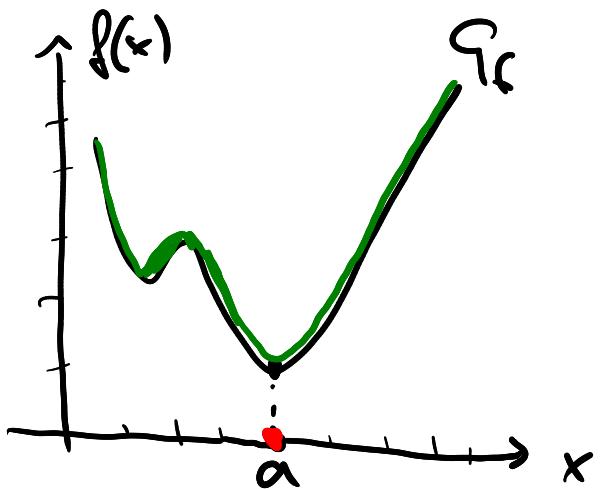


EXKURS in die Analysis :

2gg

$$f: \mathbb{R} \rightarrow \mathbb{R}$$

$$x \mapsto f(x)$$



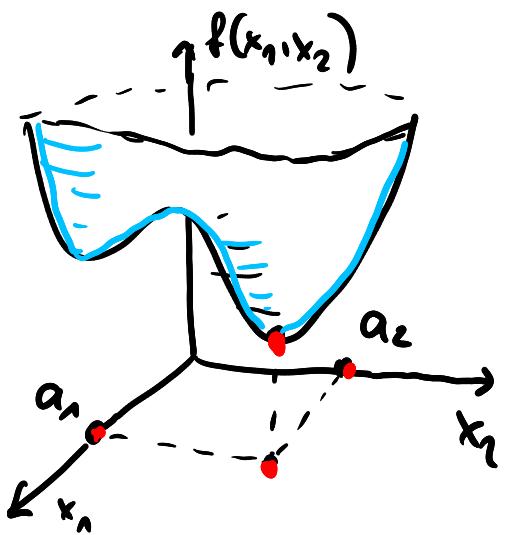
← Bedingung zu einem (lokalen) Minimum in $(a, f(a))$

$f'(a) = 0$

← Ableitung verschwindet

$$f: \mathbb{R}^n \rightarrow \mathbb{R}$$

$$(x_1, \dots, x_n) \mapsto f(x_1, \dots, x_n)$$



← Bedingung zu einem (lokalen) Minimum

$\text{grad } f = \left(\frac{\partial f}{\partial x_1}(a_1, \dots, a_n), \dots, \frac{\partial f}{\partial x_n}(a_1, \dots, a_n) \right) = \vec{0}$

$$\text{grad } f = \left(\frac{\partial f}{\partial x_1}(a_1, \dots, a_n), \dots, \frac{\partial f}{\partial x_n}(a_1, \dots, a_n) \right) = \vec{0}$$

↑ gradient verschwindet

⇒ für mehrdimensionale Funktionen verschwindet der Gradient im einem (lokalen) Minimum.

Das hatten wir bei der KQ-Methode zur Schätzung linearer Modelle ausgenutzt um analytische Lösungen zu kriegen.

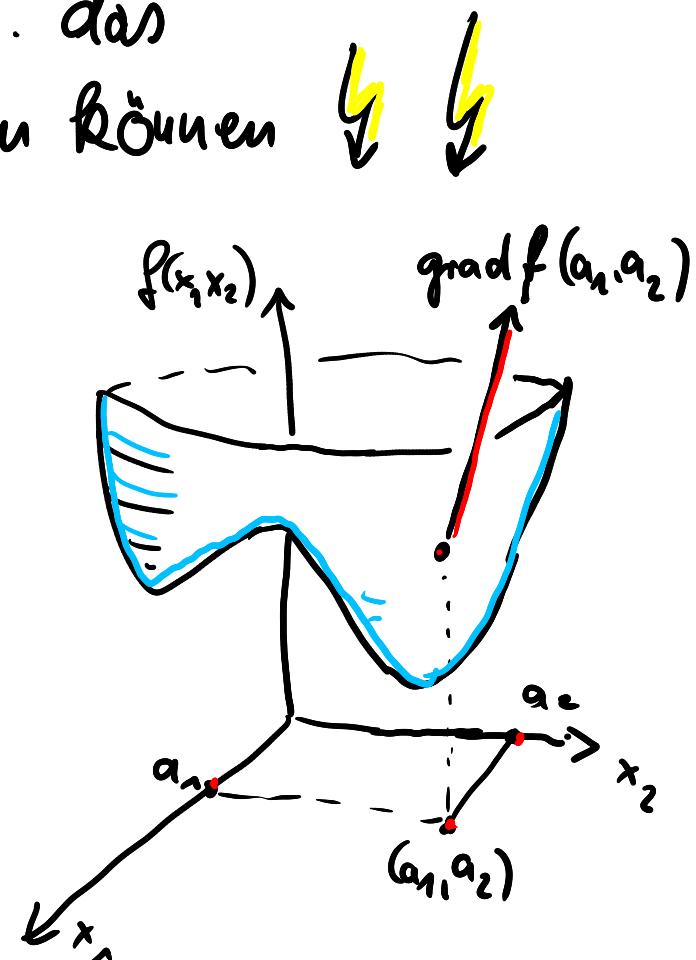
→ hilft hier leider nicht weiter, weil wir i.A. das resultierende Gleichungssystem nicht lösen können ⚡ ⚡

ABER

Der Gradient ist ein Vektor, der immer in Richtung des stärksten Anstieges zeigt.

$\|\nabla f\| \doteq$ stärkste Änderungsrate von (a_1, \dots, a_n) aus gesehen

⇒ Idee für den GRADIENTENABSTIEG



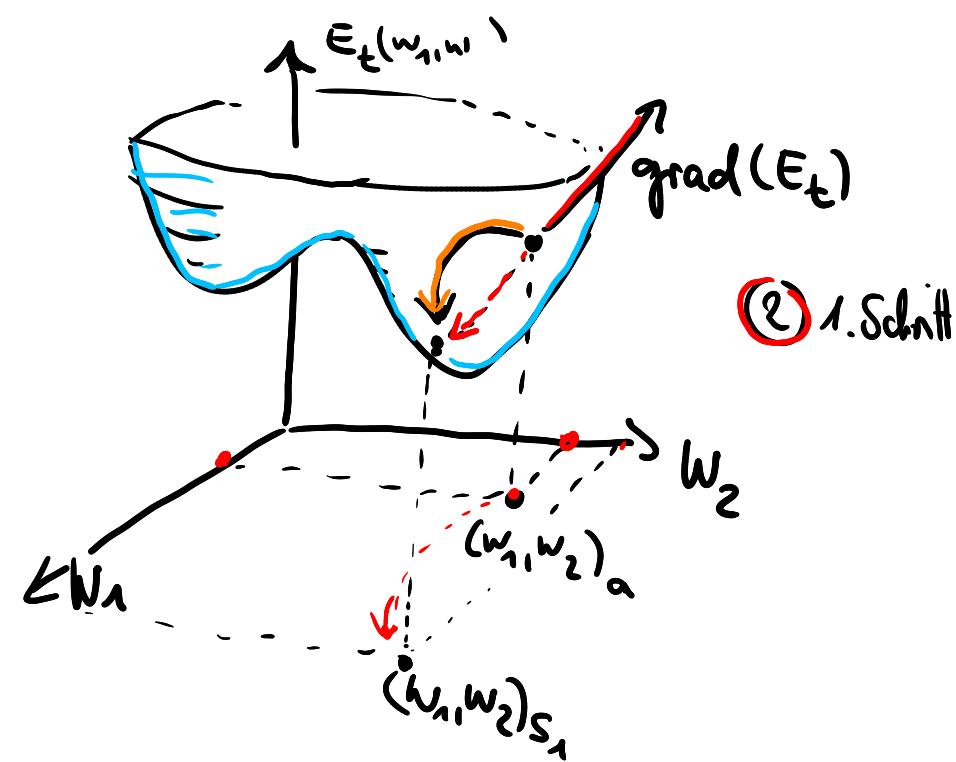
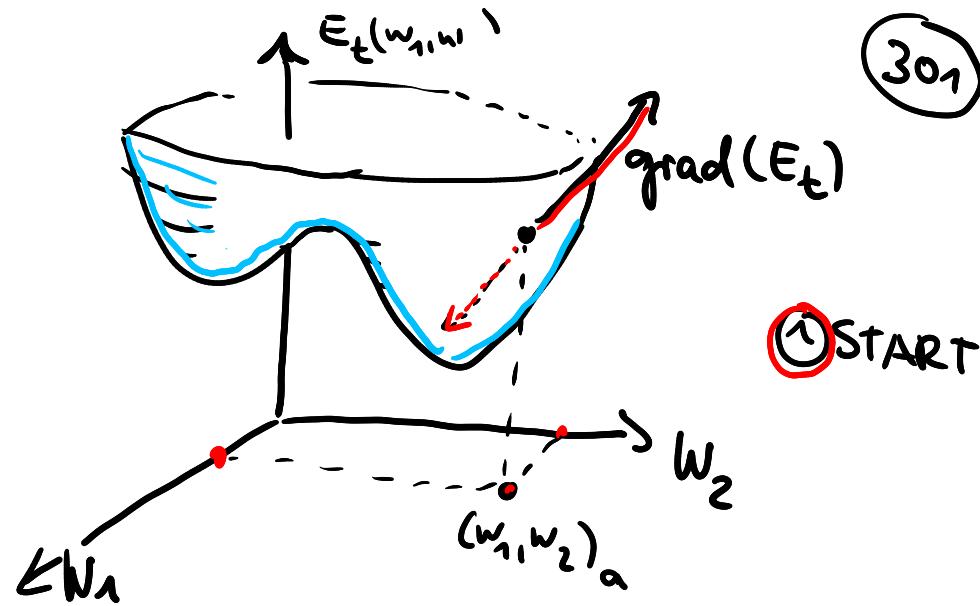
5.3.1) Der Gradientenabstieg

301

Fehlerfunktion $E_t = \sum_{k=1}^K \frac{1}{2} (\text{out}_k - \text{tar}_k)^2$
 $(t=1, \dots, T)$

- zufällige Belegung von w_1, w_2 : wir starten an einem beliebigen Ort auf der Fehlerfläche
- $\text{grad } E$ sagt uns, wo es am steilsten "bergauf" geht $\Rightarrow -\text{grad } E_t$ zeigt wo es am steilsten "bergab" geht
- gehe einen Schritt der Länge η in die Richtung des steilsten Abstieges $-\text{grad } E_t$

\Rightarrow neue, "bessere" Gewichte



$$w_{ij \text{ neu}} = w_{ij \text{ alt}} - \eta \cdot \text{grad}(E_t)$$

GEWICHTSUPDATE

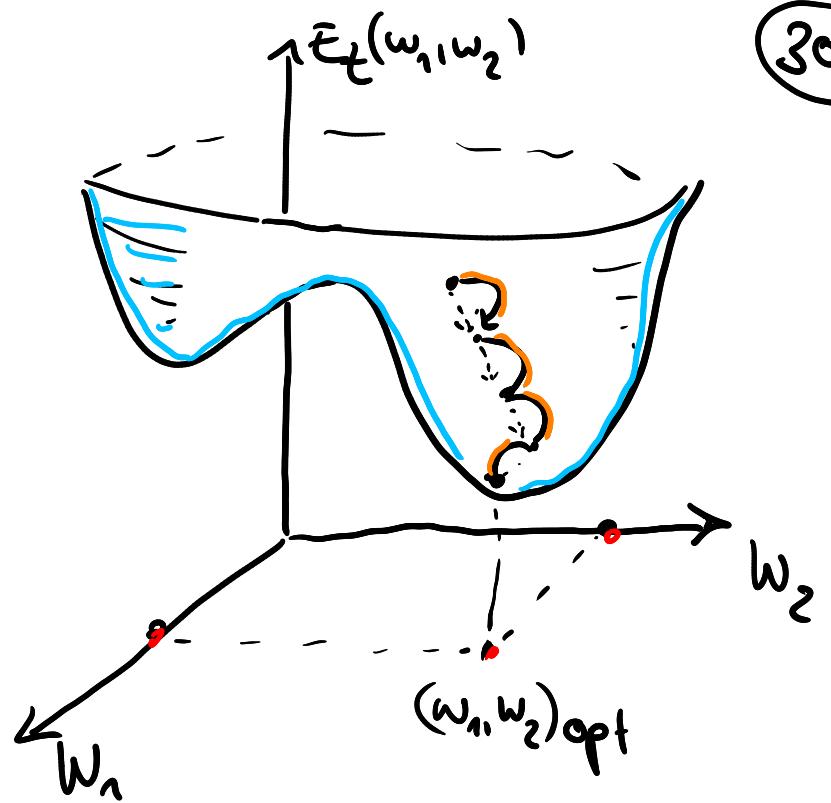
- wiederhole den Abstieg so lange bis sich keine Verbesserung mehr ergibt

→ Der Gradientenabstieg besteht aus (sehr) vielen Gewichtsupdates der Form

η ist ein Meta-parameter

$$w_{ij \text{ neu}} = w_{ij \text{ alt}} - \eta \cdot \text{grad}(E_t)$$

Lernrate η



$\text{grad}(E_t)$ muss berechnet werden!
hängt von der Fehlerfunktion δ der Transferfkt. f ab

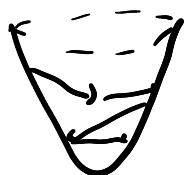
Bemerkung:

- Im einfachsten Fall wird für jedes einzelne Sample der Trainingsmenge ein Update aller Gewichte gemacht
- Einen kompletten Durchlauf aller Trainingsbeispiele nennt man eine Trainingsepoch
- Die Trainingsdaten dürfen natürlich auch mehrfach durchlaufen werden
⇒ mehrere Epochen (üblicherweise immer mehr als eine)

PROBLEME

Bei einem festgelegten Startpunkt (zufällig) und einer konstanten Schrittweite η (\rightarrow Metaparameter des Lernens) ...

→ ... landet man möglicherweise\ sehr wahrscheinlich im einem lokalen Minimum (nicht das Optimum!)



→ ... verpasst man überhaupt das (lokale) Minimum

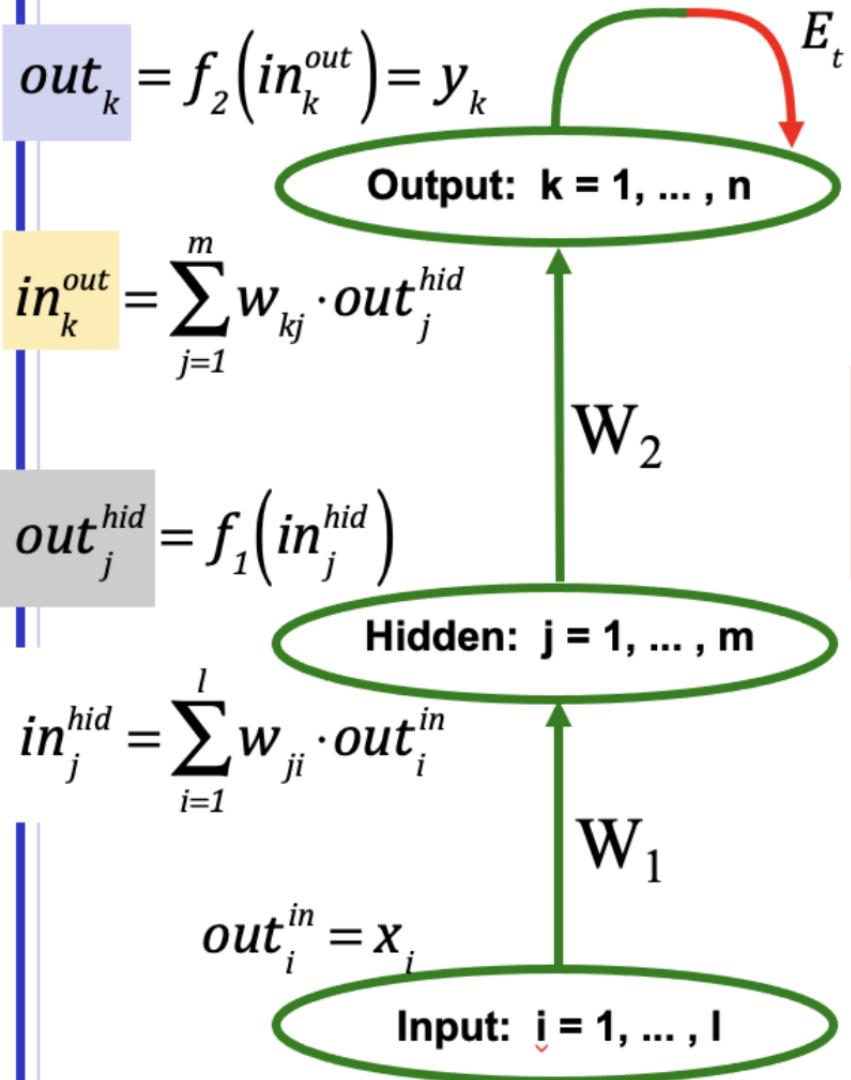


→ ... braucht man ewig um besser zu werden, wenn die Fehlerfläche zu "flach" ist

... prinzipiell aber eine gute Idee 😊

$$\text{grad}(E_t) = ?^2$$

5.3.2) Die Berechnung des Fehlergradienten



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

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

$$out_j^{hid}$$

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

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

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

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

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