

Neuronale Netze programmieren (II)

TALIT-Kurs, Semesterschlusswoche 2022

Andreas Schärer

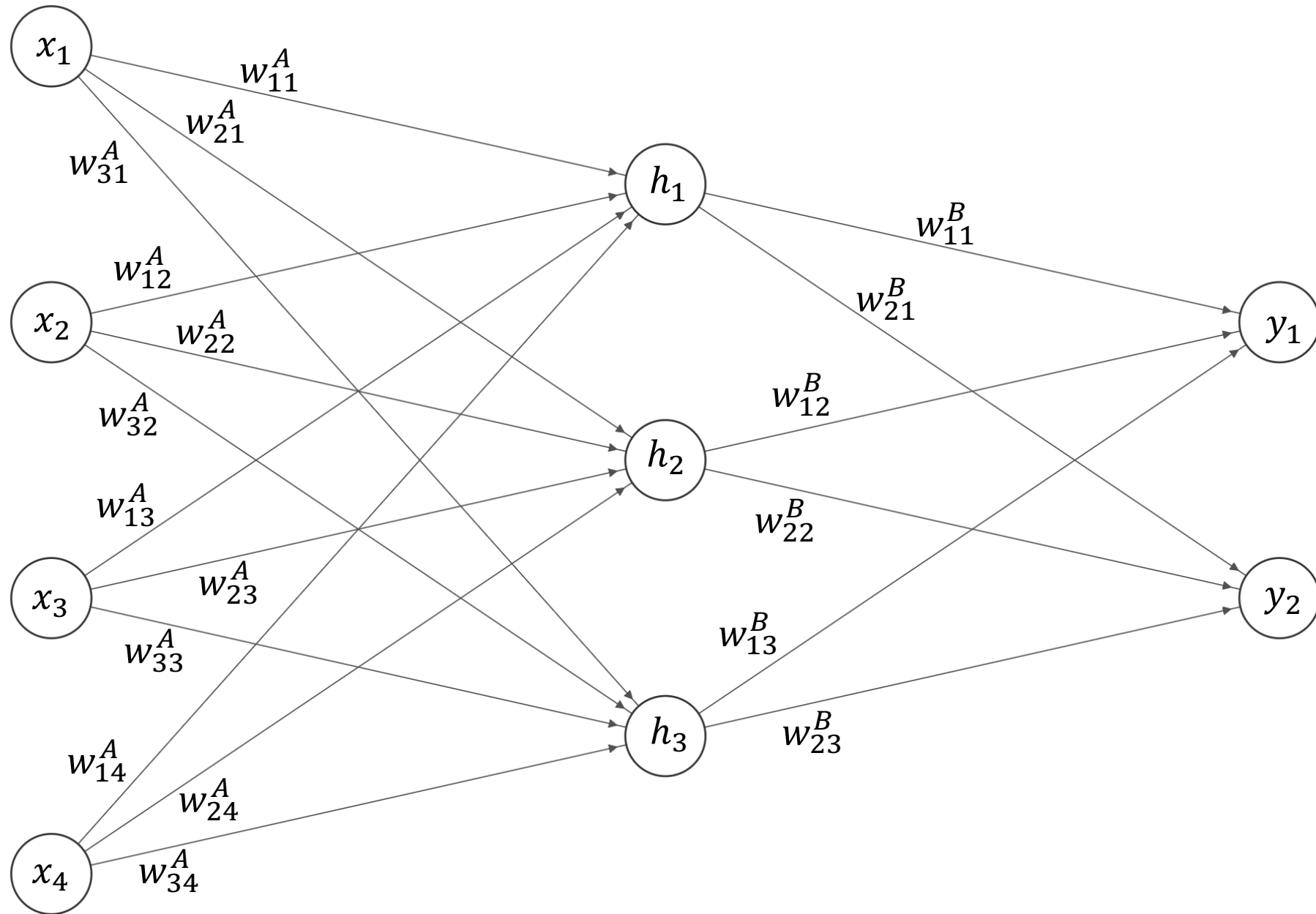
Tom Hofmann

Simon Graf

input layer (n=4)

Hidden Layer (n=3)

Output Layer (n=2)



Ziel

- Optimale Werte für Gewichte finden!
- Anders ausgedrückt: **Gewichtsmatrizen optimieren**

$$w^A = \begin{pmatrix} w_{11}^A & w_{12}^A & w_{13}^A & w_{14}^A \\ w_{21}^A & w_{22}^A & w_{23}^A & w_{24}^A \\ w_{31}^A & w_{32}^A & w_{33}^A & w_{34}^A \end{pmatrix}, \quad w^B = \begin{pmatrix} w_{11}^B & w_{12}^B & w_{13}^B \\ w_{21}^B & w_{22}^B & w_{23}^B \end{pmatrix}$$

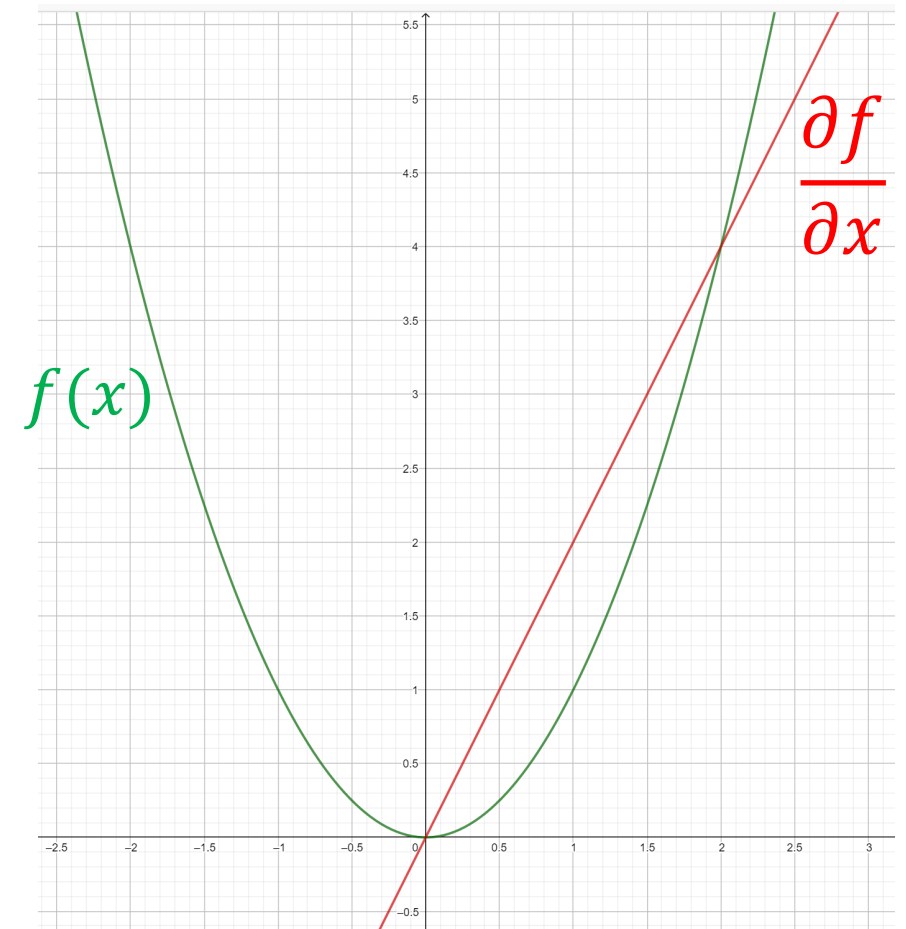
- Durch Backpropagation:
 - **Feedforward**: Input in NN → Output berechnen
 - **Backpropagation**: Output betrachten → Layer um Layer «backwards» durchs NN um zu optimieren
- Benötigen Wissen in Mathematik:
 - Matrizen
 - Analysis: Ableitung, Gradient

Part II

Mathematik

Ableitung

- Die **Ableitung** einer Funktion $f(x)$ *nach der Variablen x* ist die Funktion $\frac{\partial f}{\partial x}$ oder $f'(x)$
- Sie gibt die **Steigung der Tangente** im Punkt x an.
- Ableitung von Polynom:
 $f(x) = k x^n$
 $\frac{\partial f}{\partial x} = n k x^{n-1}$
- Beispiel:
 - $f(x) = x^2$
 - $\frac{\partial f}{\partial x} = 2x$

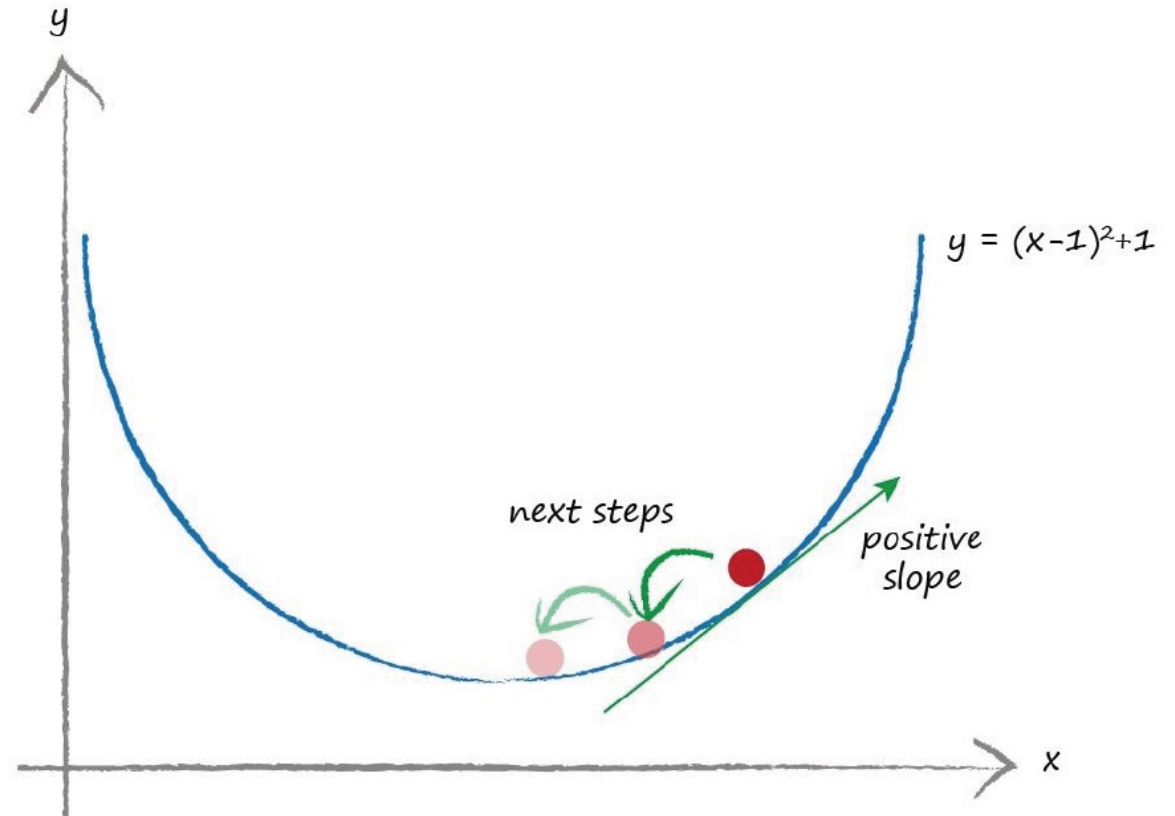


Minimum von Funktion finden: Vorgehen

Ziel: Möchten **Minimum** einer Funktion $y(x)$ finden.

Vorgehen:

1. Starte an beliebigem Punkt
2. Bestimme Steigung (Ableitung) in diesem Punkt
3. Bewege dich eine vordefinierte **Schrittweite** in Richtung *negativer Steigung*
4. Wiederhole Schritte 1-3 mehrfach



Minimum von Funktion finden: Probleme

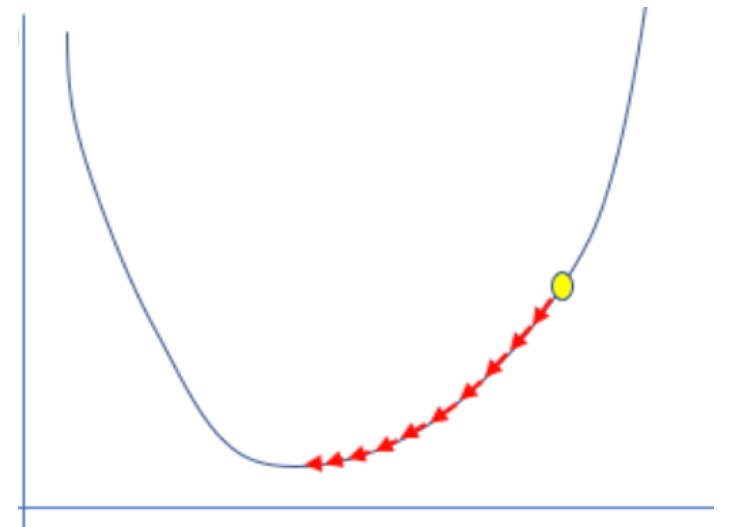
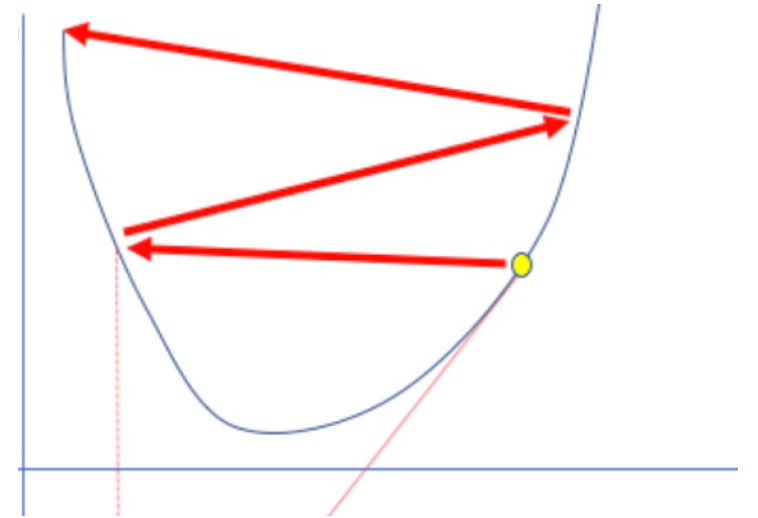
Frage: Welche Probleme können auftreten?

Tipp: Stichwort *Schrittweite*

Probleme:

1. Schrittweite zu gross: Kommt nicht nahe genug an Minimum (Overshooting)
2. Schrittweite zu klein: Kommt Minimum nur sehr langsam näher (Undershooting)

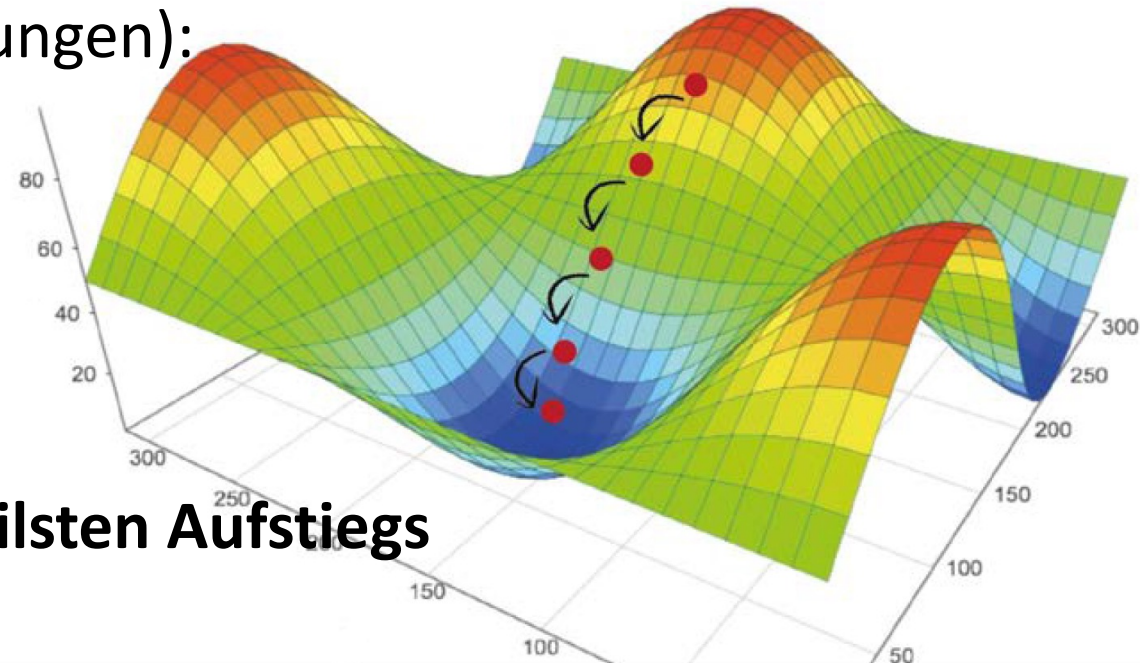
Ziel: Optimale Schrittweite finden, z.B. durch trial and error



Gradientenabstieg

- Betrachte Funktion, die von 2 Variablen abhängt: $f(a, b)$
- Wollen nun auch hier **Minimum** finden
- Bestimme **Gradienten** (Vektor mit Ableitungen):

$$\nabla f := \begin{pmatrix} \frac{\partial f}{\partial a} \\ \frac{\partial f}{\partial b} \end{pmatrix}$$



- Gradient-Vektor zeigt in **Richtung des steilsten Aufstiegs**
- **Frage:** Wie findet man Minimum?
- **Antwort:** Bewege dich in Richtung $-\nabla f$
- Dieses Prinzip heisst **Gradientenabstieg**

Ableitung: Kettenregel

- Betrachte eine Funktion, die von einer Funktion abhängt:

$$f(g(x))$$

- f hängt also indirekt (über g) von x ab:

$$x \rightarrow g(x) \rightarrow f(g)$$

- Ableitung dieser Funktion nach x :

$$\frac{\partial f}{\partial x} = \frac{\partial g}{\partial x} \frac{\partial f}{\partial g}$$

- Heisst **Kettenregel**

Ableitung: Kettenregel

- Kettenregel:

$$f(g(x)) \rightarrow \frac{\partial f}{\partial x} = \frac{\partial g}{\partial x} \frac{\partial f}{\partial g}$$

- Beispiel: $f(x) = 2(x^3 + 3)^2$
 - $f(g) = 2g^2$
 - $g(x) = x^3 + 3$
- Wende Kettenregel an:
 - $\frac{\partial f}{\partial g} = 4g$
 - $\frac{\partial g}{\partial x} = 3x^2$
 - $\rightarrow \frac{\partial f}{\partial x} = \frac{\partial f}{\partial g} \frac{\partial g}{\partial x} = 4g \cdot 3x^2 = 4(x^3 + 3)3x^2 = 12(x^5 + 3x^2)$

Part III

NN Programmieren:
Training durch Backpropagation

Cost Function

- Erinnerung: Gewichte werden zu Beginn rein *zufällig* gewählt
- **Fehler** auf Output:

$$E_{\text{out}} := t - y = \begin{pmatrix} t_1 - y_1 \\ t_2 - y_2 \end{pmatrix}$$

- Definiere **Cost Function**:

$$C := \frac{1}{2} \left((t_1 - y_1)^2 + (t_2 - y_2)^2 \right) = \frac{1}{2} \left(E_{\text{out},1}^2 + E_{\text{out},2}^2 \right)$$

- Cost Function vergleicht den *tatsächlichen Output* (y_1, y_2) mit dem *gewünschten (target) Output* (t_1, t_2)
- Je grösser C , desto schlechter funktioniert NN
- Ziel muss also sein?
- **Minimierung der Cost Function durch Optimierung der Gewichte**

Cost Function minimieren

- **Ziel:** Wollen die Cost Function C minimieren.
- **Vorgehen:**
 1. Füttere einen Datenpunkt durch das NN (feedforward)
 2. Verändere Gewichte ein wenig, so dass Cost Function kleiner wird
 3. Wiederhole diese Schritte für alle Datenpunkte (also 10'000e Male!)
- Fasse Cost Function auf als Funktion der w^B Gewichte:

$$C(w_{11}^B, w_{21}^B, w_{12}^B, w_{22}^B, w_{13}^B, w_{23}^B)$$

- Was ist mit w^A ? Betrachten im Moment nur Hidden Layer → Output Layer
- Frage: Wie können wir diese Funktion minimieren?
- Antwort: **Gradientenabstieg!**

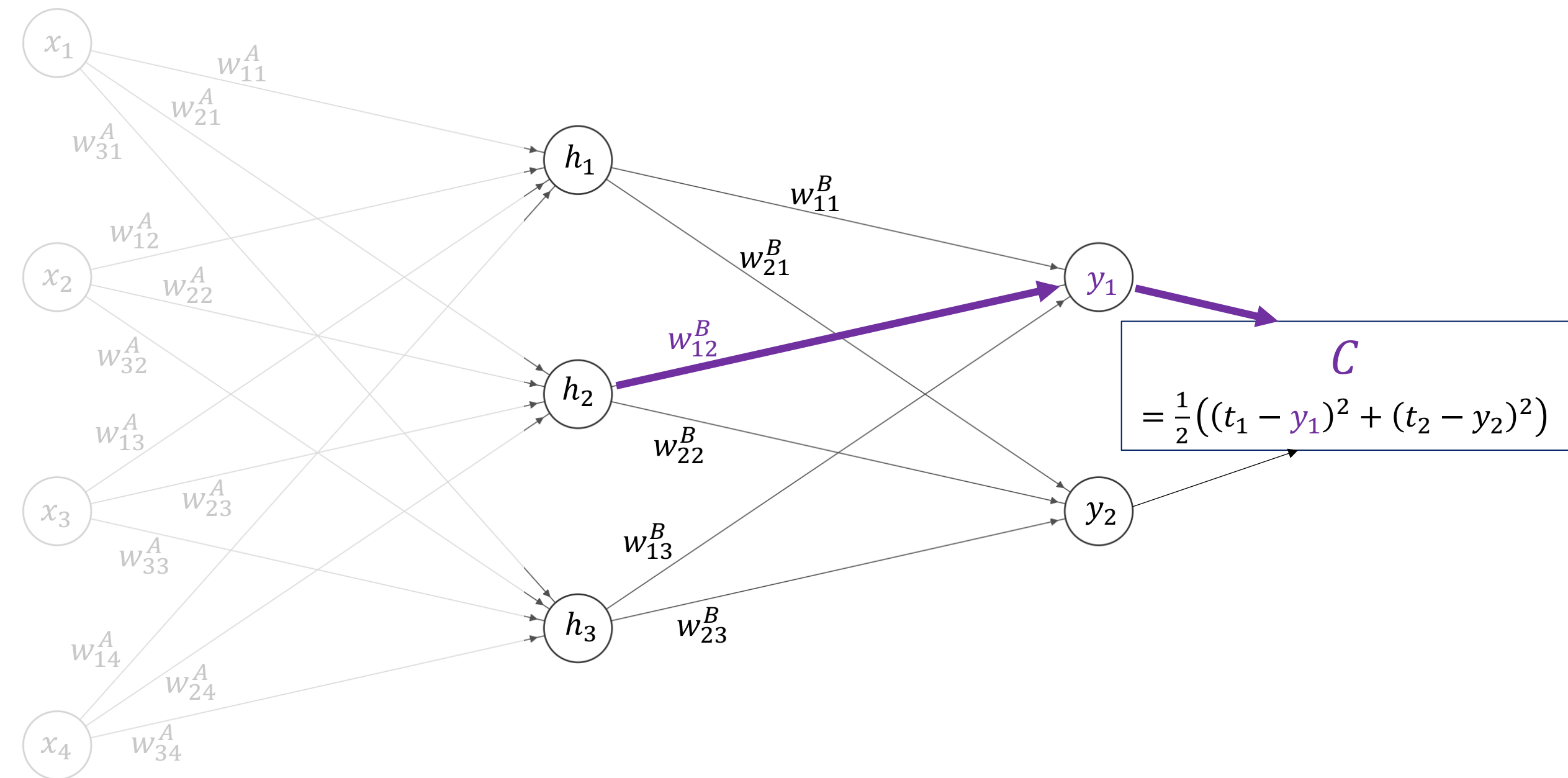
Ableitung der Cost Function

- Welchen Einfluss hat Gewicht w_{12}^B auf C ?

input layer (n=4)

Hidden Layer (n=3)

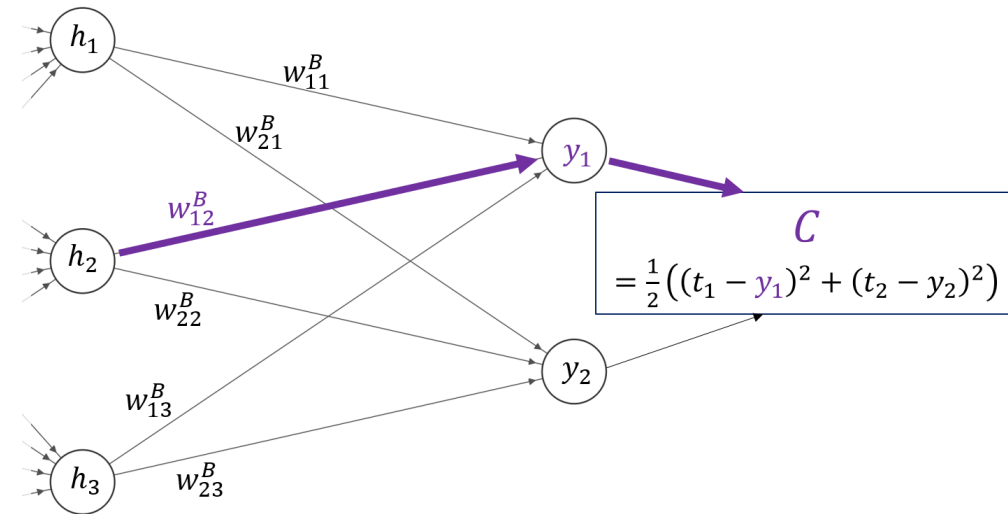
Output Layer (n=2)



Ableitung der Cost Function

- Welchen Einfluss hat Gewicht w_{12}^B auf C ?
- Bestimme dazu die Ableitung $\frac{\partial C}{\partial w_{12}^B}$
- Achtung: w_{12}^B beeinflusst C indirekt:

$$w_{12}^B \rightarrow y_1 \rightarrow C$$



- Wende deshalb Kettenregel an:

$$\frac{\partial C}{\partial w_{12}^B} = \frac{\partial y_1}{\partial w_{12}^B} \frac{\partial C}{\partial y_1}$$

Ableitung der Cost Function

- Erinnerung: Berechnung Output:

$$y_1 = \sigma(w^B \cdot h)_1 = \sigma(w_{11}^B h_1 + w_{12}^B h_2 + w_{13}^B h_3)$$

- Verwende Ableitung der Sigmoid Funktion:

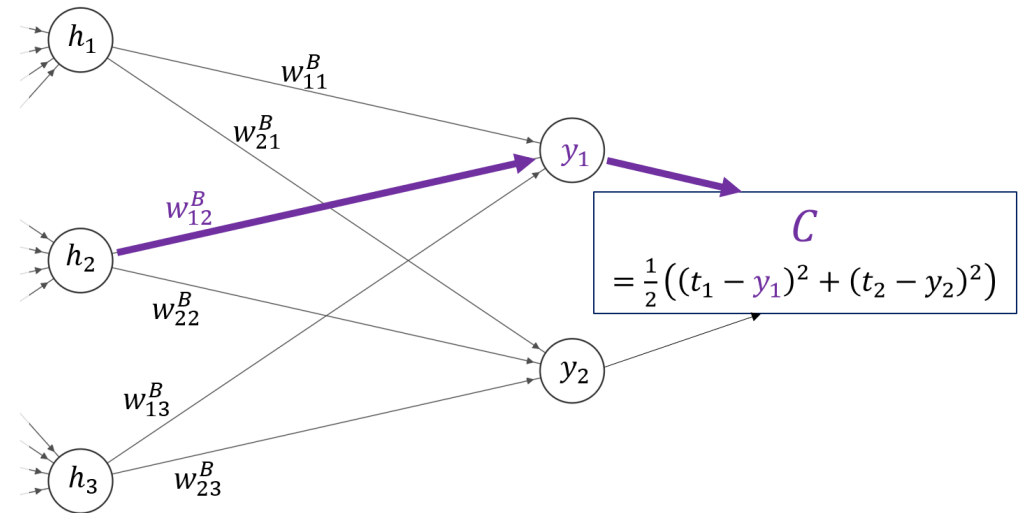
- $\sigma(z) = \frac{1}{1+e^{-z}}$

- $\frac{\partial \sigma}{\partial z} = \sigma(z)(1 - \sigma(z))$

- Einzelne Ableitungen sind dann:

- $\frac{\partial C}{\partial y_1} = -(t_1 - y_1) = -E_{\text{out},1}$

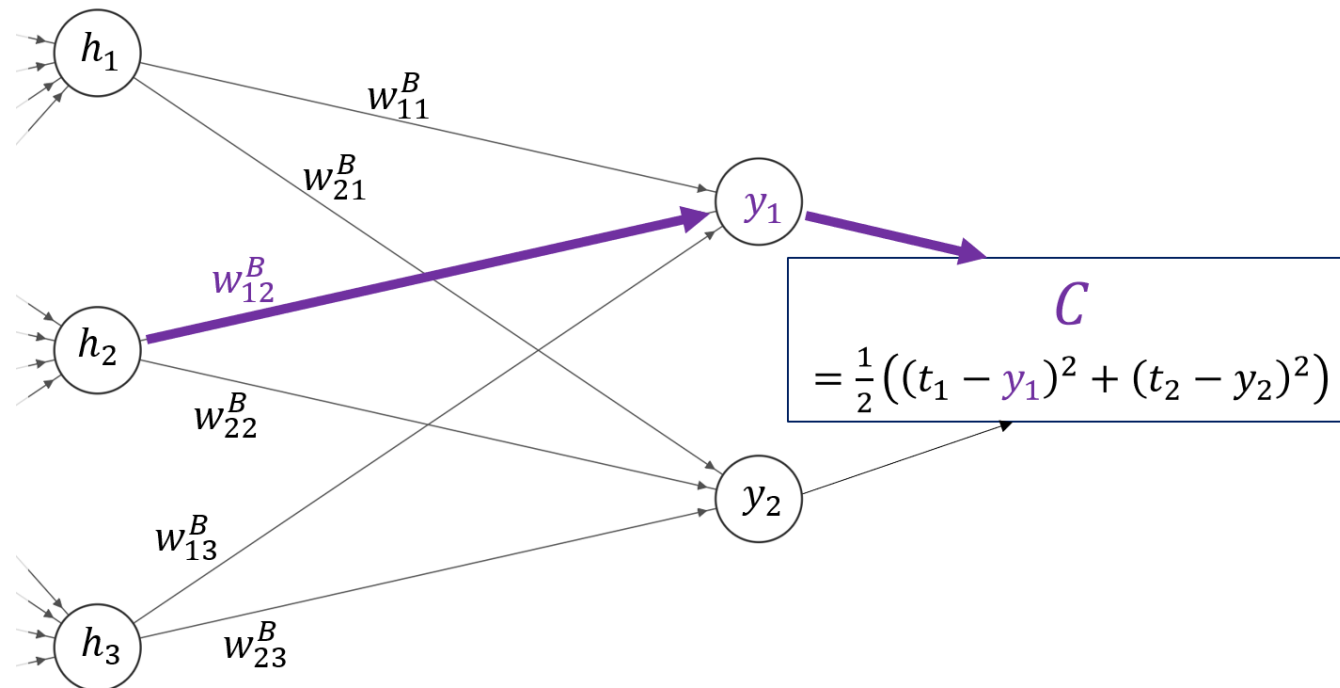
- $\frac{\partial y_1}{\partial w_{12}^B} = \frac{\partial}{\partial w_{12}^B} (\sigma(w^B \cdot h))$
 $= \sigma(w^B \cdot h)(1 - \sigma(w^B \cdot h)) \cdot h_2 = y_1(1 - y_1)h_2$



Ableitung der Cost Function

- Fügen zusammen:

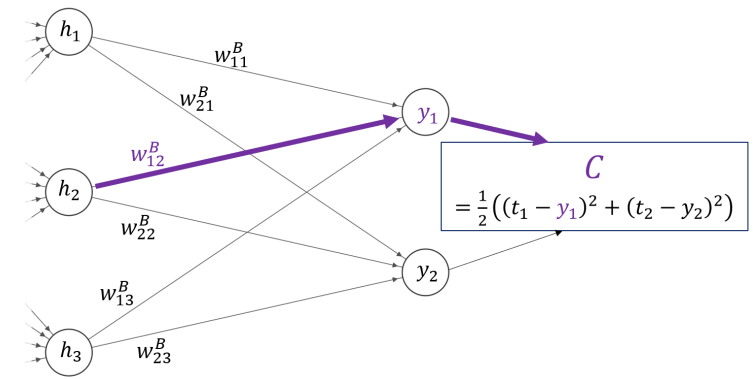
$$\frac{\partial C}{\partial w_{12}^B} = \frac{\partial y_1}{\partial w_{12}^B} \frac{\partial C}{\partial y_1} = -E_{\text{out},1} \cdot y_1(1 - y_1)h_2$$



Optimierung der Gewichtsmatrix w^B

- Berechne Gradient:

$$\nabla C = \begin{pmatrix} \frac{\partial C}{\partial w_{11}^B} & \frac{\partial C}{\partial w_{12}^B} & \frac{\partial C}{\partial w_{13}^B} \\ \frac{\partial C}{\partial w_{21}^B} & \frac{\partial C}{\partial w_{22}^B} & \frac{\partial C}{\partial w_{23}^B} \end{pmatrix}$$



- In kompakter Schreibweise: $\nabla C = -(E_{\text{out}} \otimes y \otimes (1 - y)) \cdot h^T$
- Wobei:
 - $\otimes$ ist komponentenweise Multiplikation, für numpy-Arrays einfach `*`
 - Punkt `·` ist Vektormultiplikation, in numpy einfach `np.dot()`
 - $E_{\text{out}} \otimes y \otimes (1 - y)$ hat Dimension 2×1 , h^T hat Dimension 1×3
 - ∇C hat deshalb Dimension 2×3

Optimierung der Gewichtsmatrix w^B

- Gradientenabstieg
- Wähle **Schrittweite**, genannt **learning rate** λ [Lambda]
- Gradient:

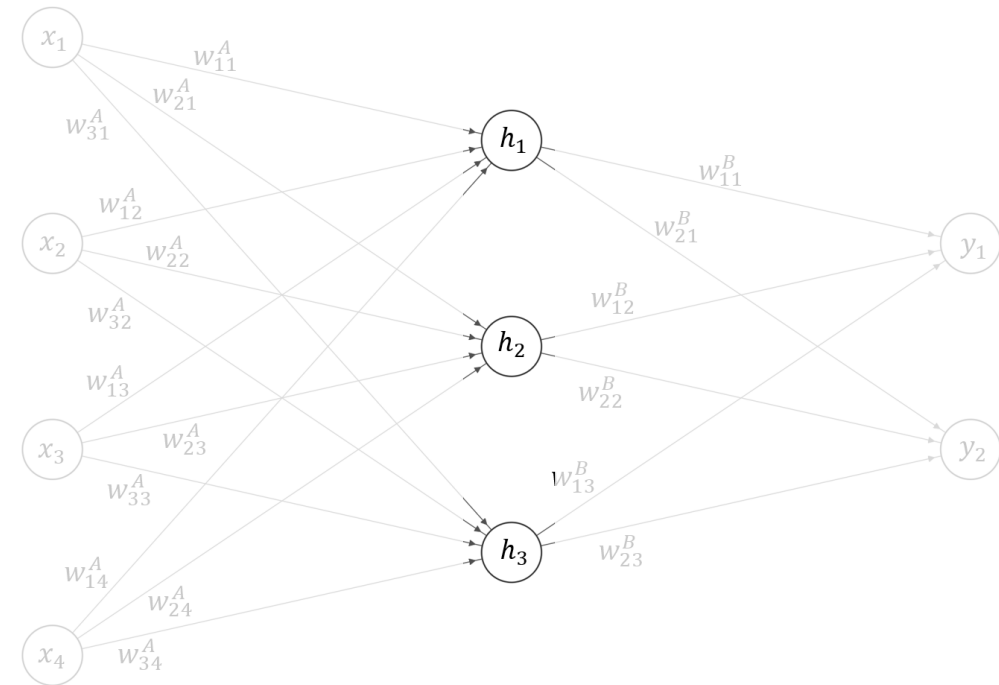
$$\nabla C = -(E_{\text{out}} \otimes y \otimes (1 - y)) \cdot h^T$$

- Update Gewichte w^B durch Gradientenabstieg:

$$w^B \rightarrow w^B - \lambda \cdot \nabla C = w^B + \lambda (E_{\text{out}} \otimes y \otimes (1 - y)) \cdot h^T$$

Optimierung der Gewichtsmatrix w^A

- Für Optimierung von w^B haben nur letzte beiden Layers betrachtet
- Optimierung von w^A :
 - Analog zu w^B
 - betrachten nur ersten beiden Layers
- $w^A \rightarrow w^A + \lambda(E_{\text{hidden}} \otimes h \otimes (1 - h)) \cdot x^T$
- Anstelle von:
 - w^B betrachte w^A
 - E_{out} betrachte E_{hidden}
 - y betrachte h
 - h betrachte x



Beispiel: Toy-Problem

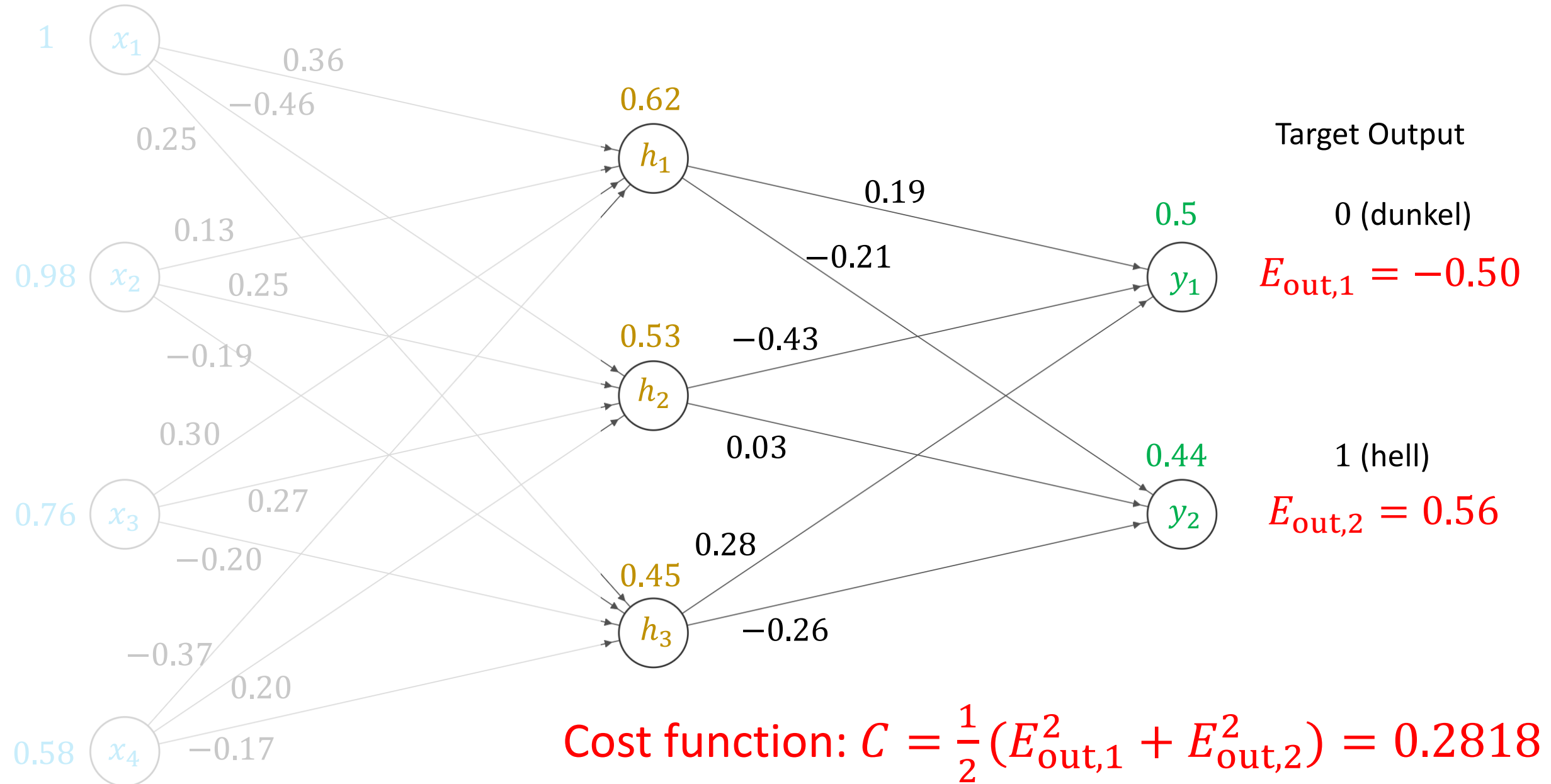
1. Bestimme Gewichte zufällig (angegeben auf nächster Slide)
2. Nimm einen Datenpunkt:
 - $x = [1.00, 0.98, 0.76, 0.58]$
 - Korrektes Resultat: 1 (also 'hell')
 - Target Output: $t = [0, 1]$
3. Feed forward durch NN:
 - Hidden Layer: $h = [0.62, 0.53, 0.45]$
 - Output Layer: $y = [0.5, 0.44]$
4. Berechne Cost Function:

$$\begin{aligned} C &= \frac{1}{2} (E_{\text{out},1}^2 + E_{\text{out},2}^2) = \frac{1}{2} ((y_1 - t_1)^2 + (y_2 - t_2)^2) \\ &= \frac{1}{2} ((0.5 - 0)^2 + (0.44 - 1)^2) = 0.2818 \end{aligned}$$

input layer (n=4)

Hidden Layer (n=3)

Output Layer (n=2)



Toy-Problem: Optimierung von w^B

- Output: $y = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} 0.5 \\ 0.44 \end{pmatrix}$
- Target Output: $t = \begin{pmatrix} t_1 \\ t_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$
- Fehler Output: $E_{\text{out}} = \begin{pmatrix} E_{\text{out},1} \\ E_{\text{out},2} \end{pmatrix} = \begin{pmatrix} -0.50 \\ 0.56 \end{pmatrix}$
- Cost Function: $C = 0.2818$

Toy-Problem: Optimierung von w^B

- $E_{\text{out}} \otimes y \otimes (1 - y) = \begin{pmatrix} E_{\text{out},1} \cdot y_1 (1 - y_1) \\ E_{\text{out},2} \cdot y_2 (1 - y_2) \end{pmatrix} = \begin{pmatrix} -0.13 \\ 0.14 \end{pmatrix}$ [2×1 -Matrix]
- $h^T = (0.62 \quad 0.53 \quad 0.45)$ [1×3 -Matrix]
- $\nabla C = -\left(E_{\text{out}} \otimes y \otimes (1 - y)\right) \cdot h^T = -\begin{pmatrix} 0.62 \cdot (-0.13) & 0.53 \cdot (-0.13) & 0.45 \cdot (-0.13) \\ 0.62 \cdot 0.14 & 0.53 \cdot 0.14 & 0.45 \cdot 0.14 \end{pmatrix}$
 $= \begin{pmatrix} 0.08 & 0.07 & 0.06 \\ -0.09 & -0.07 & -0.06 \end{pmatrix}$ [2×3 -Matrix]
- Altes Gewicht: $w^B = \begin{pmatrix} 0.19 & -0.43 & 0.28 \\ -0.21 & 0.03 & -0.26 \end{pmatrix}$
- Wähle hier learning rate $\lambda = 1$
- **Update Gewichte:** $w^B \rightarrow w^B - \lambda \cdot \nabla C = \begin{pmatrix} 0.11 & -0.5 & 0.22 \\ -0.12 & 0.1 & -0.2 \end{pmatrix}$

Toy-Problem: Optimierung von w^A

- Ist analog zu Optimierung von w^B :

$$w^A \rightarrow w^A + \lambda (E_{\text{hidden}} \otimes h \otimes (1 - h)) \cdot x^T$$

- Benötigen dazu aber Fehler auf Neuronen im Hidden Layer
- **Idee:**
 - Betrachte *Fehler auf Output* und ...
 - *verteile diesen rückwärts* auf Neuronen im Hidden Layer ...
 - *proportional zu Gewicht*

Toy-Problem: Optimierung von w^A

1. Fehler auf Output:

$$E_{\text{out}} = t - y = \begin{pmatrix} -0.50 \\ 0.56 \end{pmatrix}$$

2. **Rückverteilung des Fehlers** *proportional zu Gewicht, geschrieben in Matrixnotation:*

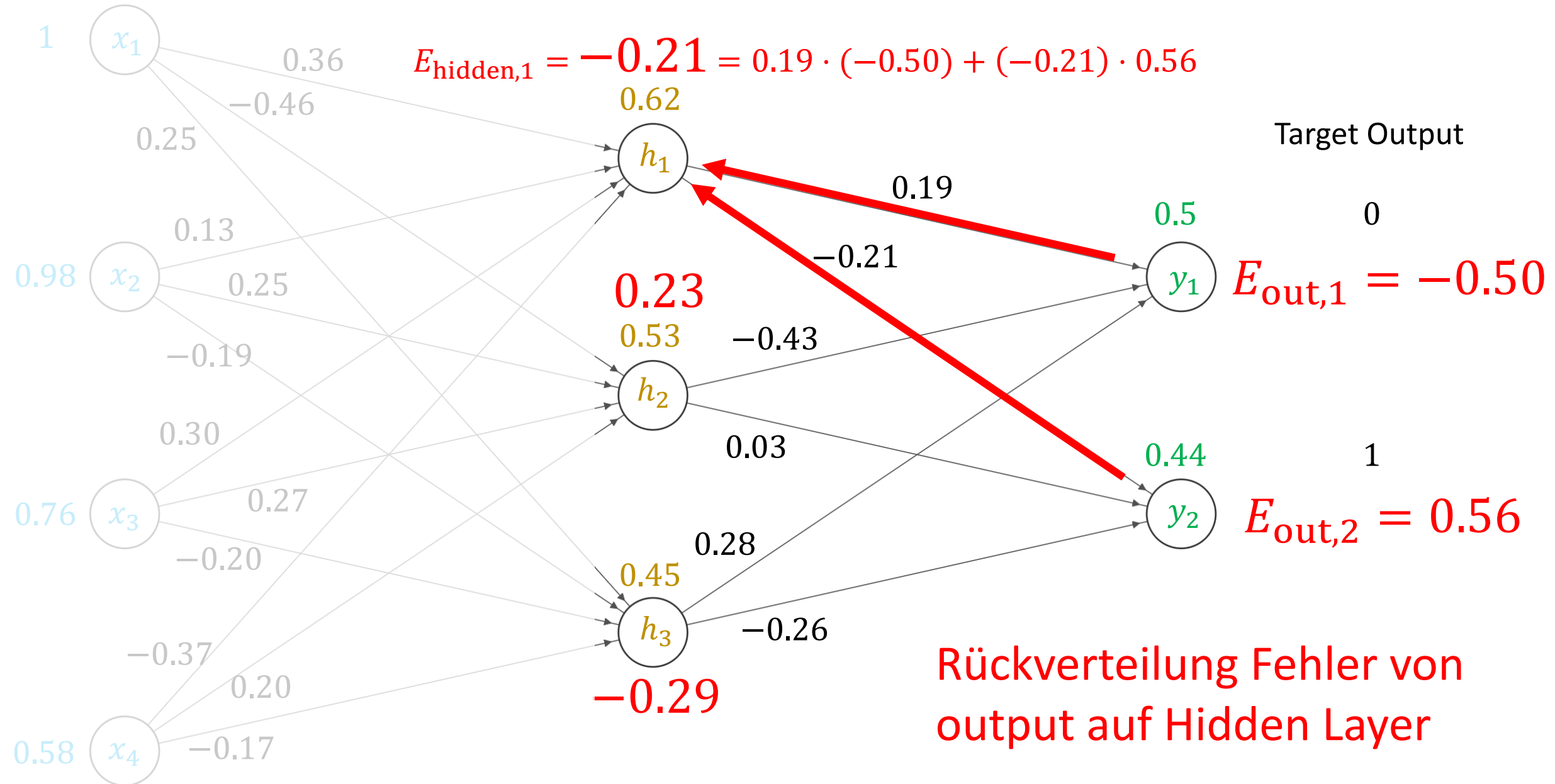
$$E_{\text{hidden}} = w_B^T \cdot E_{\text{out}}$$

$$E_{\text{hidden}} = \begin{pmatrix} E_{\text{hidden},1} \\ E_{\text{hidden},2} \\ E_{\text{hidden},3} \end{pmatrix} = \begin{pmatrix} w_{11}^B & w_{21}^B \\ w_{12}^B & w_{22}^B \\ w_{13}^B & w_{23}^B \end{pmatrix} \begin{pmatrix} E_{\text{out},1} \\ E_{\text{out},2} \end{pmatrix} = \begin{pmatrix} w_{11}^B E_{\text{out},1} + w_{21}^B E_{\text{out},2} \\ w_{12}^B E_{\text{out},1} + w_{22}^B E_{\text{out},2} \\ w_{13}^B E_{\text{out},1} + w_{23}^B E_{\text{out},2} \end{pmatrix}$$

input layer (n=4)

Hidden Layer (n=3)

Output Layer (n=2)



Toy-Problem: Optimierung von w^A

1. Fehler auf Output:

$$E_{\text{out}} = t - y = \begin{pmatrix} -0.50 \\ 0.56 \end{pmatrix}$$

2. **Rückverteilung des Fehlers** *proportional zu Gewicht, geschrieben in Matrixnotation:*

$$E_{\text{hidden}} = w_B^T \cdot E_{\text{out}}$$

$$E_{\text{hidden}} = \begin{pmatrix} E_{\text{hidden},1} \\ E_{\text{hidden},2} \\ E_{\text{hidden},3} \end{pmatrix} = \begin{pmatrix} w_{11}^B & w_{21}^B \\ w_{12}^B & w_{22}^B \\ w_{13}^B & w_{23}^B \end{pmatrix} \begin{pmatrix} E_{\text{out},1} \\ E_{\text{out},2} \end{pmatrix} = \begin{pmatrix} w_{11}^B E_{\text{out},1} + w_{21}^B E_{\text{out},2} \\ w_{12}^B E_{\text{out},1} + w_{22}^B E_{\text{out},2} \\ w_{13}^B E_{\text{out},1} + w_{23}^B E_{\text{out},2} \end{pmatrix}$$

3. Update Gewichtsmatrix w^A :

$$w^A \rightarrow w^A + \lambda (E_{\text{hidden}} \otimes h \otimes (1 - h)) \cdot x^T$$

Zusammenfassung: Training NN

- Bestimme **Output** y mit Feedforward:

$$y = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}$$

- **Fehler** auf Output:

$$E_{\text{out}} := t - y = \begin{pmatrix} t_1 - y_1 \\ t_2 - y_2 \end{pmatrix}$$

- **Rückverteilung** Fehler:

$$E_{\text{hidden}} = w_B^T \cdot E_{\text{out}}$$

- **Update Gewichtsmatrix** w^B :

$$w^B \rightarrow w^B + \lambda (E_{\text{out}} \otimes y \otimes (1 - y)) \cdot h^T$$

- Update Gewichtsmatrix w^A :

$$w^A \rightarrow w^A + \lambda (E_{\text{hidden}} \otimes h \otimes (1 - h)) \cdot x^T$$

- numpy:

- `np.dot()` für $\cdot$
- `*` für $\otimes$

Typischer Fehler in Python: Tipps

- Verwendung von Listen statt numpy array
 - Wandle Liste in numpy array um: `m = np.array(L)`
- Falsche Dimension (Form der Matrizen)
 - Finde heraus, welche Matrizen falsche Form haben, dazu ...
 - Debugger verwenden (Breakpoints setzen, Form der Matrizen überprüfen)
 - Form der Matrizen printen: `print(m.shape)`
 - Matrix Form ändern (siehe “Matrizen in Python”), z.B. `m = m.reshape((1,-1))`

Auftrag 3

- **Ziel: NN für Toy-Problem trainieren**
- Mache eine Kopie des Files «02_feedforward_oop.py» und speichere diese unter dem Namen «03_toyproblem_training.py»
- Kommentiere alles, was mit dem MNIST-Datensatz zu tun hat, aus. Für den Moment wollen wir uns nur um das Toy-Problem kümmern.
- Erweitere deine *Network*-Klasse um das Attribut *learning_rate*. Ein Wert für diese soll als Argument für die *__init__*-Methode übergeben werden, wenn ein *Network*-Objekt erstellt wird.
- Erweitere deine *Network*-Klasse um eine Methode *train*. Dieser soll ein Datensatz übergeben werden, mit dem das Netzwerk trainiert werden soll.
- Alles Wichtige für diese Methode findest du auf der Slide «Zusammenfassung: Training NN»
- Bestimme vor und nach dem Trainieren die Erfolgsquote deines Netzwerks.
- Gratuliere: Nun hast du dein erstes komplettes NN programmiert!
- Contest: Wer erzielt das NN mit der höchsten Erfolgsquote?
Spielregeln: Verwende ausschliesslich die Trainingsdaten fürs Training und die Testdaten für das Bestimmen der Erfolgsquote.
- Besprich deine Lösung mit dem Lehrer.

Auftrag 4

- **Ziel: NN für beliebigen Datensatz**
- Mache eine Kopie des Files «03_toyproblem_training.py» und speichere diese unter dem Namen «**04_neural_network.py**»
- Passe nun deinen Code so an, dass du einen beliebigen Datensatz einlesen kannst.
- Wenn du die letzte Aufgabe gut und sauber programmiert hast, solltest du diese Aufgabe sehr schnell erledigt haben.
- Lade sowohl die Daten des Toy-Problems wie auch die MNIST Daten ein. Erstelle *je* ein Neuronales Netz und trainiere dieses mit der *train* Methode. Bestimme danach die Erfolgsquote des NN.
- Contest: Wer erzielt das beste Resultat für den Handschrifterkenner? Es sollen wieder nur die Trainingsdaten fürs Training und die Testdaten für die Erfolgsberechnung verwendet werden.
- Besprich deine Lösung mit dem Lehrer.