Neuronale Netze programmieren (I)

TALIT-Kurs, Semesterschlusswoche 2020

Andreas Schärer

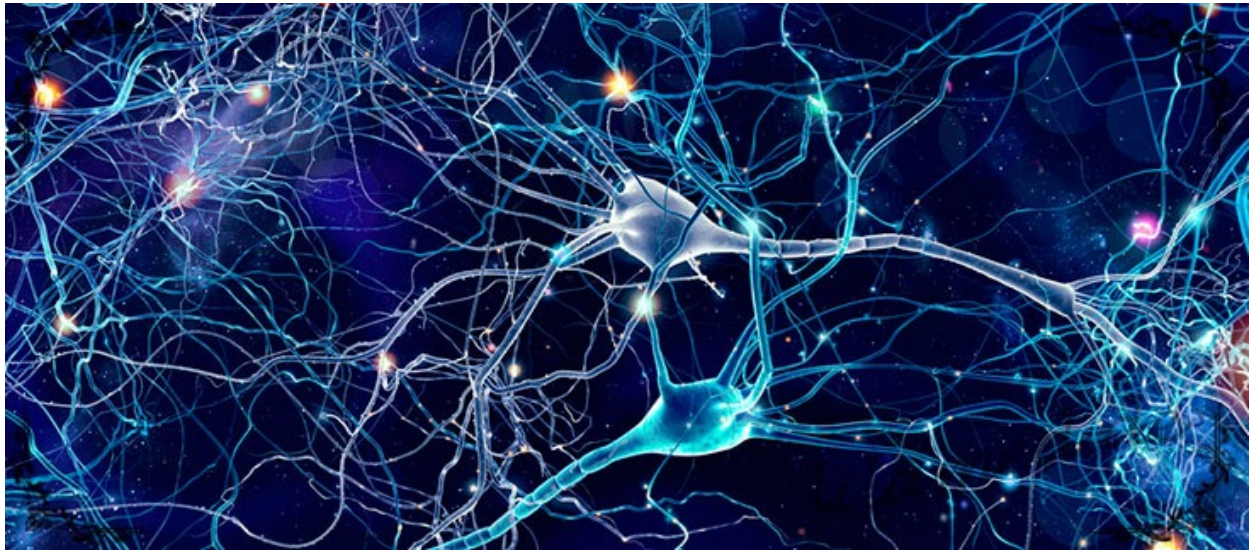
Part 0
Einführung & Übersicht

Was ist künstliche Intelligenz?

- ‘normales’ Programmieren:
  - Programmierer programmiert einen Algorithmus …
  - … er sagt Computer also genau, was dieser in welcher Situation tun soll
  - Programmierer muss Vorgehen also selbst genau verstanden haben!
- Künstliche Intelligenz:
  - Programmierer programmiert ein neuronales Netz (NN) …
  - … aber keine Logik.
  - Diese baut das Netz selbst auf, in dem man es mit Daten trainiert.
  - Programmierer versteht nicht (oder nur ansatzweise), wie das trainierte neuronale Netz genau vorgeht
  - Grundidee: Funktionsweise des Hirns imitieren

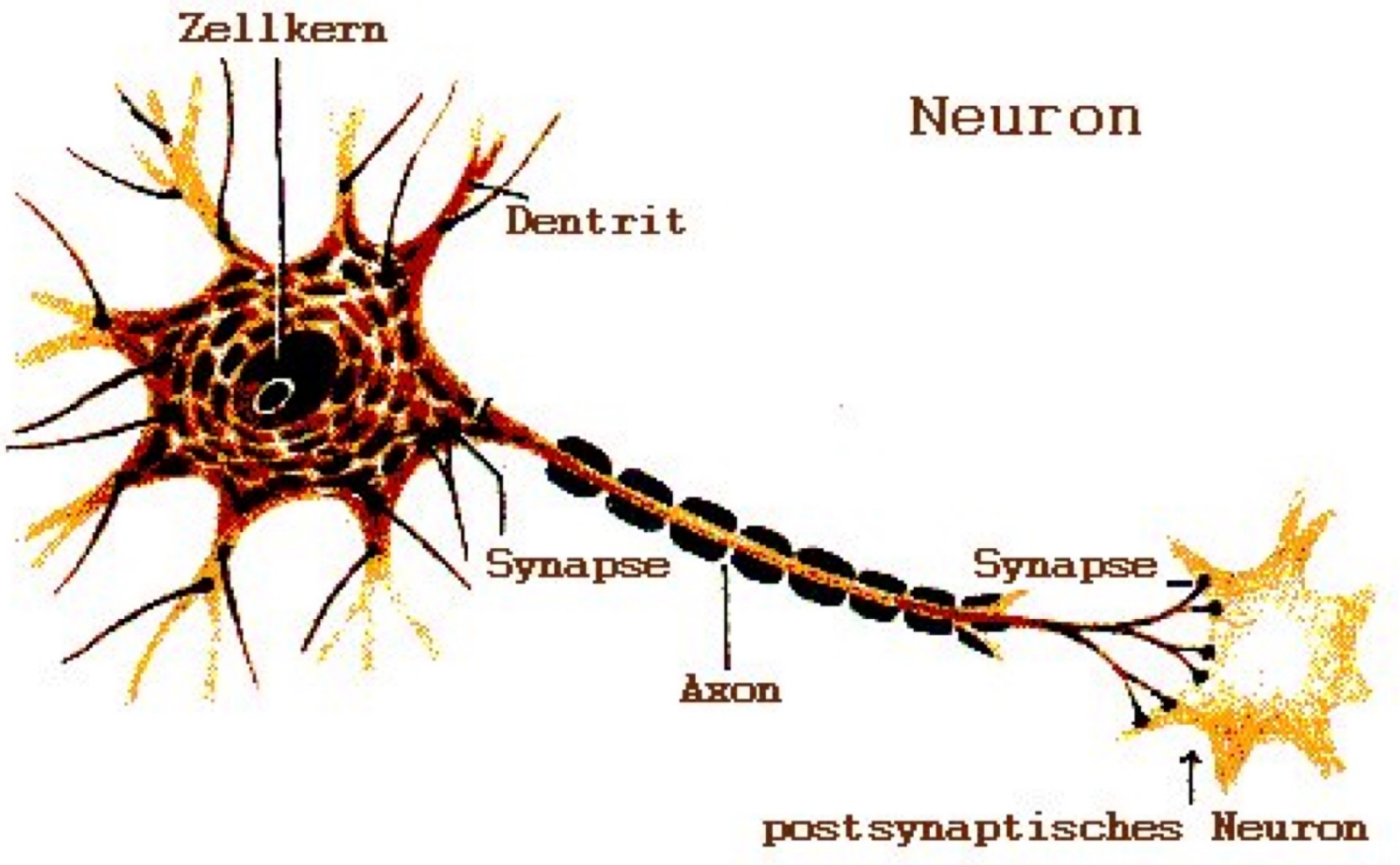
Das Hirn

- Hirn ist Netzwerk von Neuronen
- ca. 85 Milliarden Neuronen (Nervenzelle)

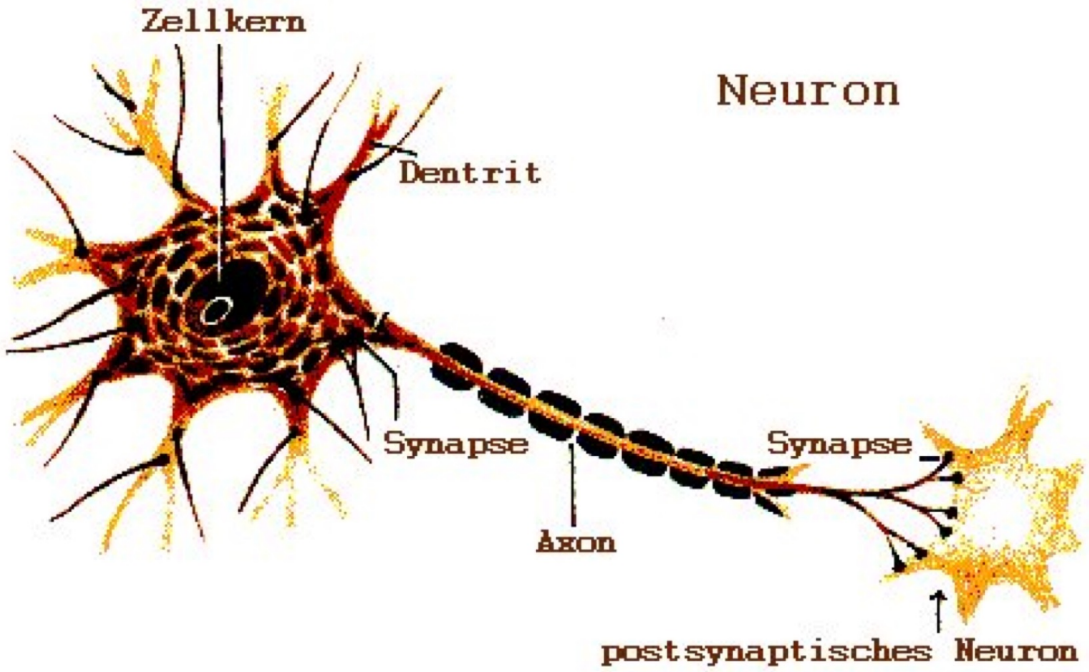


Aufbau Neuron

1. **Dendriten**:
   - Informationsaufnahme
   - Verbindung zu anderen Neuronen (Input)
2. **Zellkörper**:
   - Informationsverarbeitung
3. **Axon**:
   - Informationsweiterleitung
4. **Synapse**:
   - Informationsübertragung
   - Verbindung zu anderen Neuronen (Output)



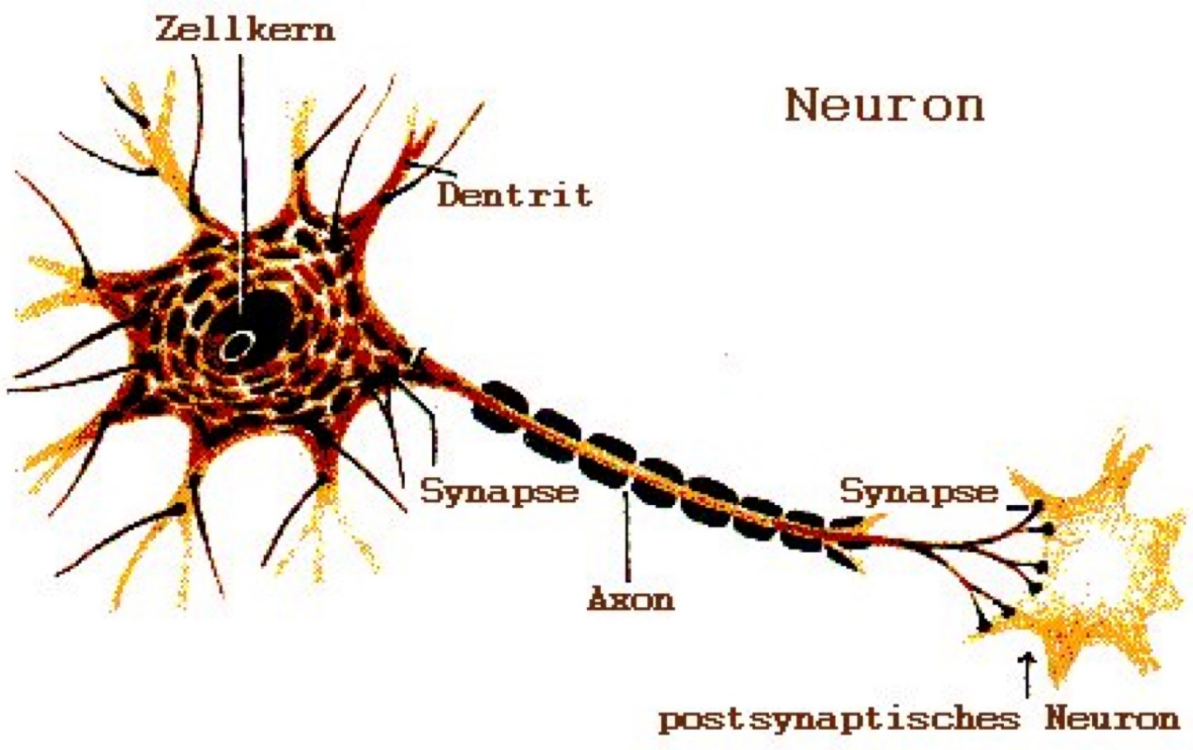
Funktionsweise des Hirns



- Hirn ist Netzwerk von Neuronen
- Besonders wichtig:
  Verbindungen (Dendriten & Axone) zwischen Neuronen
- Vorgang in einem Neuron:
  1. **Dendriten** erhalten Signale von Input-Neuronen und leiten diese zum …
  2. **Zellkörper**: entscheidet, ob Summe aller Inputsignale stark genug sind. Falls ja, dann sendet Ausgangssignal über …
  3. **Axone** weiter an …
  4. **Synapsen**. Dies sind Verbindungsstellen mit Dendriten von weiteren Neuronen. Je stärker Axon aktiviert ist, desto mehr von Überträgerstoff (**Neurotransmitter**) wandert zu nächstem Neuron.

Funktionsweise des Hirns

- **Zusammengefasst:**
  Ist die Summe der Eingangssignale grösser als ein gewisser Schwellenwert, so sendet das Neuron ein Ausgangssignal an die mit ihm verbundenen Neuronen
- Quelle: https://www.lernwelt.at/downloads/solerntdasgehirn.pdf

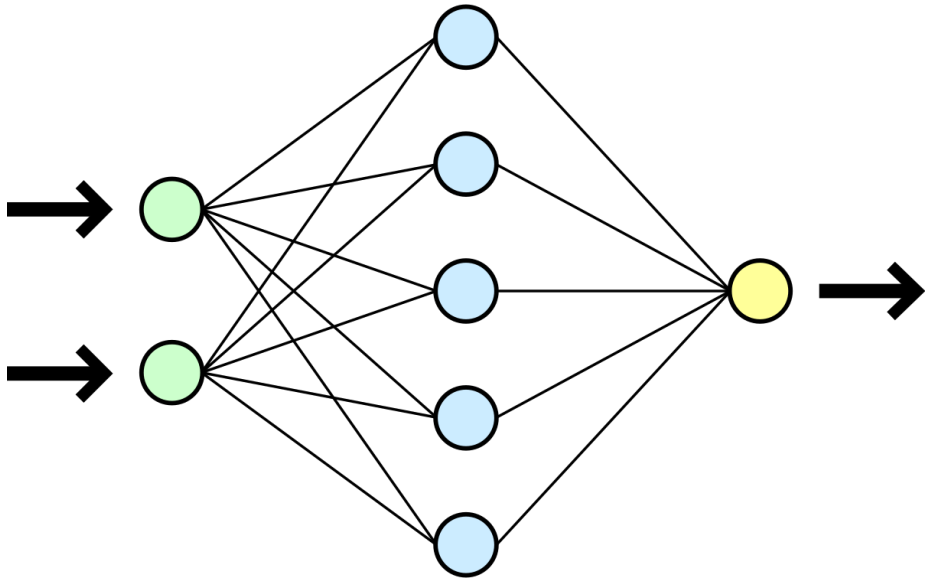


Trainieren des Hirns (Lernen)

- Neugeborenes: ca. 50 Billionen Verbindung zwischen Neuronen
- **Lernen**:
  - *Aufbauen* neuere Verbindungen zwischen Neuronen
  - *Stärken* bestehender Verbindungen zwischen Neuronen

Neuronale Netze in künstlicher Intelligenz

- Imitiere Aufbau und Funktionsweise von unserem Hirn mit Computer
- Vorgehen:
  1. Neuronales Netz programmieren
     - Gewisse Anzahl Neuronen vorgeben

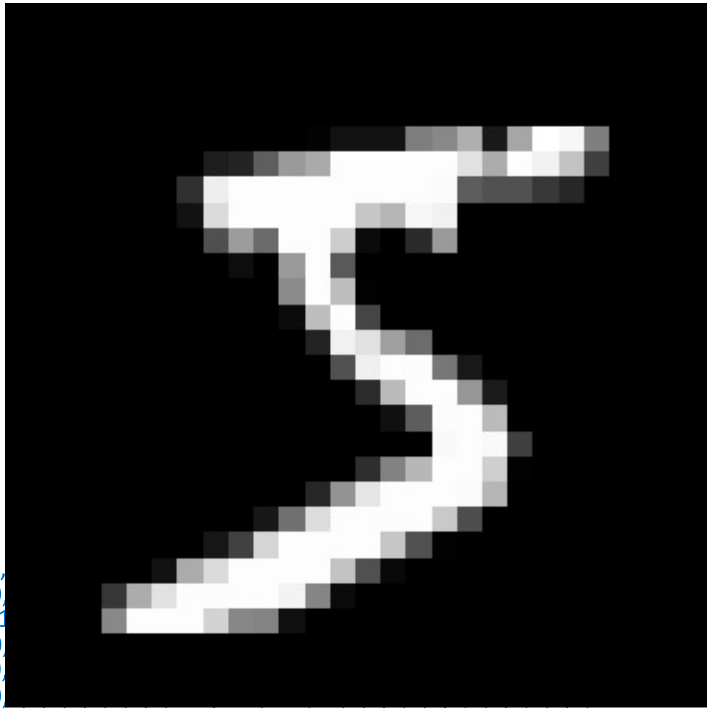


  2. NN mit Daten trainieren:
     - Lernen: Verbindungen zwischen Neuronen stärken
  3. Trainiertes NN testen: NN auf unbekannte Daten anwenden

Ziel dieser Woche

- “From scratch” **neuronales Netz (NN) selbst programmieren**
- Also ohne Verwendung von spezialisierten Libraries wie Keras, TensorFlow, …
- Nutze dein NN als **Handschrifterkenner von Zahlen**
- Vorgehen um NN zu Programmieren**:**
  1. Struktur des NN programmieren
  2. NN trainieren (60’000 Datensamples)
  3. NN testen (10’000 neue Datensamples):
     Wie genau/erfolgreich ist mein NN?

Datensatz

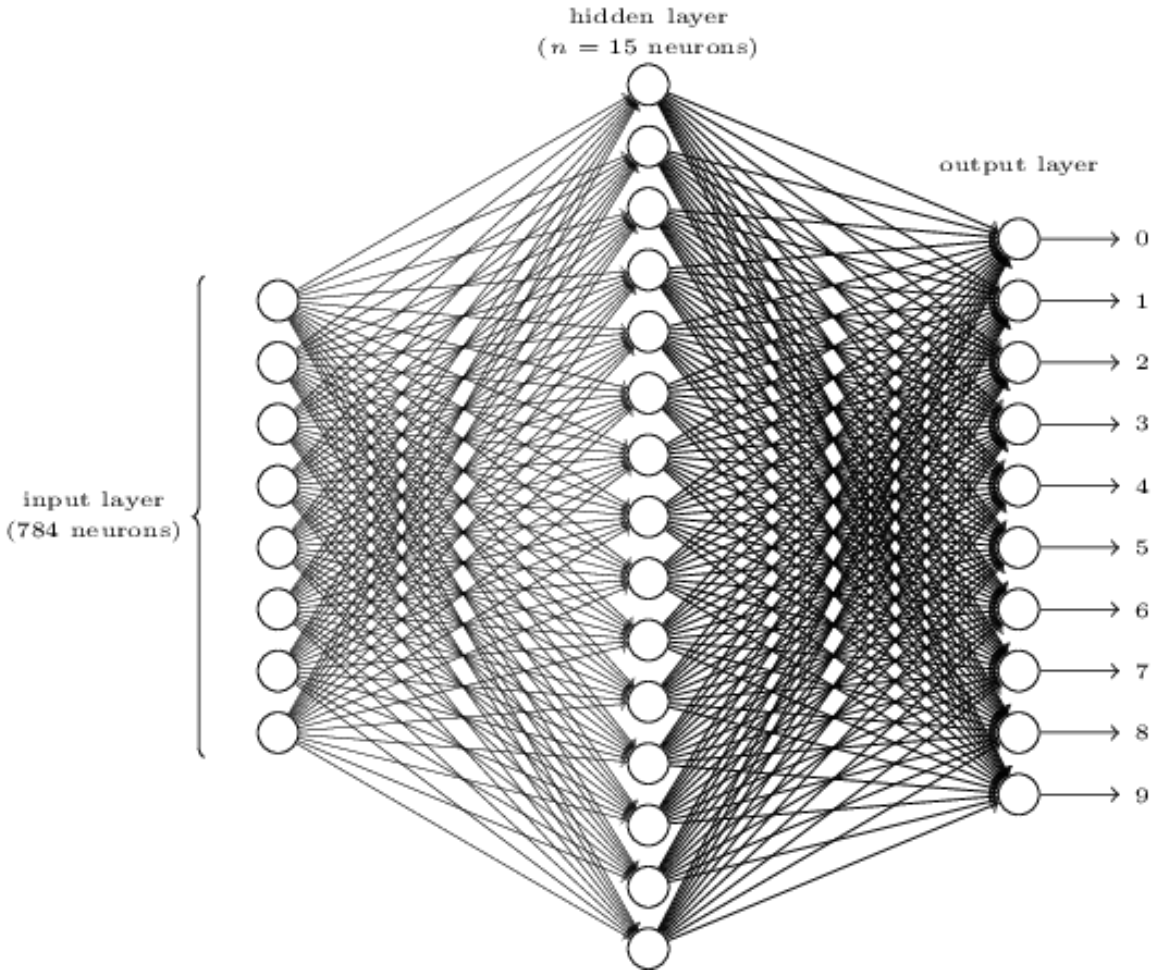


- MNIST Datensatz:
  - 60’000 + 10’000 Datensamples
  - (verwenden 60’000 für Training & 10’000 für Test)
- 1 Datensample, Beispiel:

5,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,18,18,18,126,136,175,26,166,255,247,127,0,0,
253,225,172,253,242,195,64,0,0,0,0,0,0,0,0,0,0,0,0,0,0,49,238,253,253,253,253,253,253,253,251,93,82,82,56,39,0,0,0,0,0,0,0,0,0,0,0,0,0,0,18,21
0,0,0,0,0,0,0,0,0,0,0,0,80,156,107,253,253,205,11,0,43,154,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,14,1,154,253,90,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,11,190,253,70,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,35,241,225,160,108,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,45,186,253,253,150,27,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,16,93,252,253,187,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,46,130,183,253,253,207,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,39,148,229,253,253,253,250,182,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,24,114,221,253,253,253,253,201,78,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,23,66,213,253,253,253,253,198,81,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,18,171,219,253,253,253,253,195,80,9,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,55,172,226,253,253,253,253,244,133,
11,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,136,253,253,253,212,135,132,16,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

- Blaue Zahlen:
  - Anzahl: 784 = 28x28
  - Pixel Information des Bildes: Werte 0 bis 255 entsprechen Helligkeit des Pixels
  - 0: schwarz, 255: weiss (RGB, $256 = 2^8$)
  - Im Code: dividiere Werte durch 255 → Werte von 0 bis 1
- Rote Zahl:
  - Wert der angezeigten Zahl

Struktur des NN



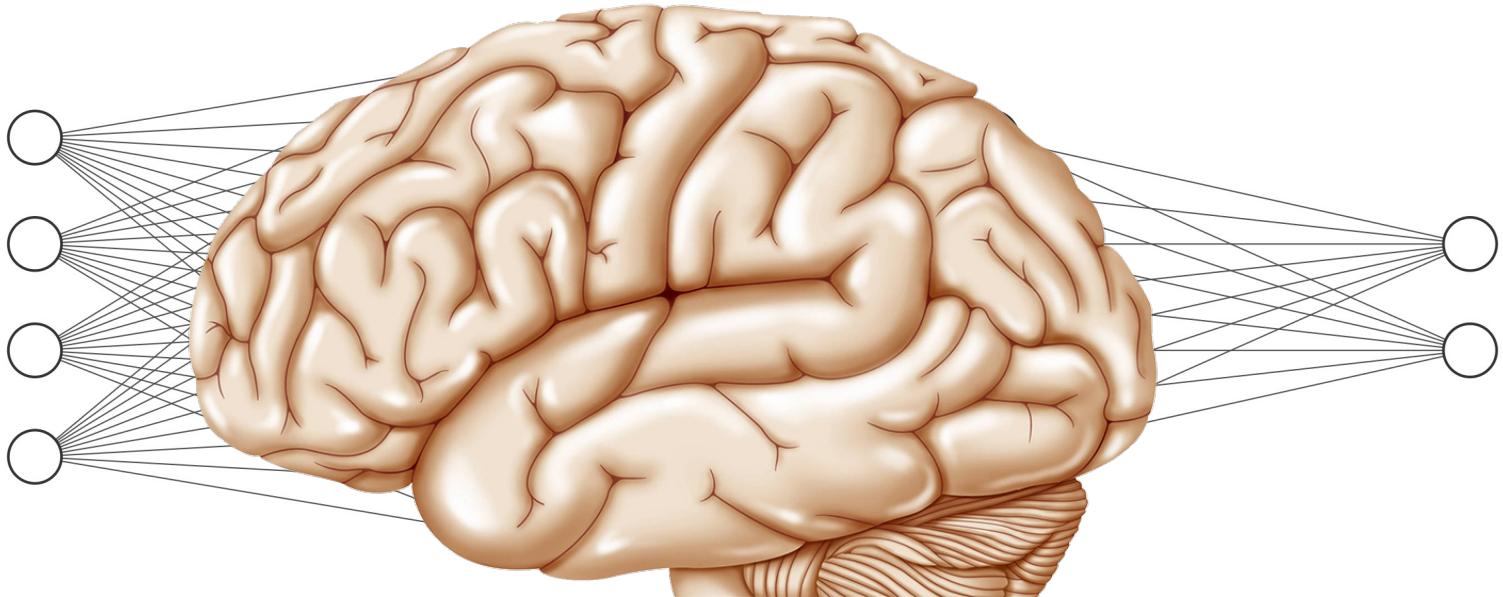
- Input Layer:
  - Pixelwerte
  - 784 = 28x28 Neuronen
- Hidden Layer(s):
  - 1 oder mehrere Hidden Layers
  - Jedes Layer: gewisse Anzahl Neuronen
- Output Layer:
  - 10 Neuronen
  - Ein Neuron für jede Zahl

Ablauf

- Betrachten zuerst Toy-Problem:
  - Gleiches Prinzip wie für Handschrifterkenner
  - Aber viel kleinere Datensamples → kleine Gewichtsmatrizen (z.B. 3x4 statt 200x784)
  - Code kann nachher ganz einfach erweitert werden
- Programmieren:
  1. Grundstruktur des NN (Toy-Problem)
  2. Forward-Propagation:
     - Daten in NN hineingeben
     - Output berechnen
     - **Ziel für heute**
  3. NN trainieren durch Backpropagation
  4. NN testen

Toy-Problem

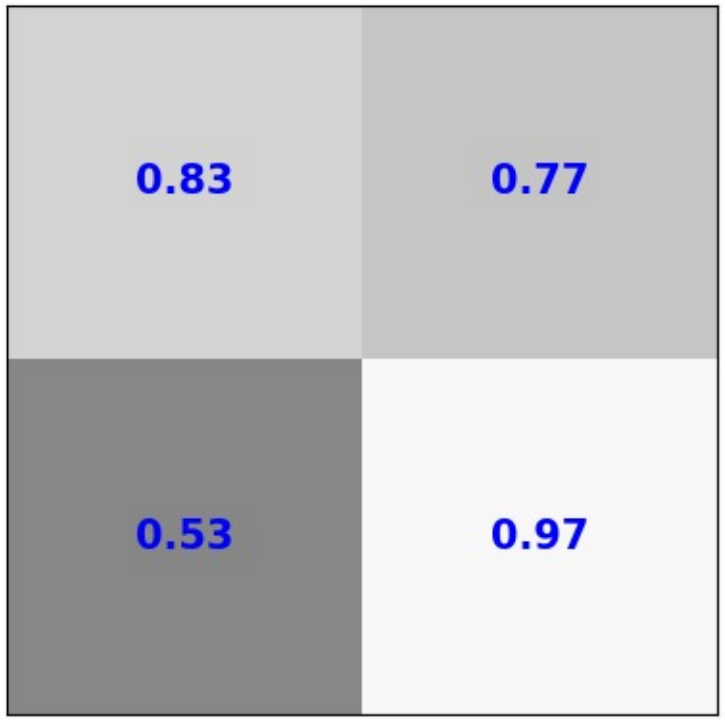
- 4 Input Neuronen:
  - Jedes Neuron: Zahl zw. 0 (Schwarz) und 255 (Weiss)
- 2 Output Neuronen:
  - ???
- **Trainiere nun dein neuronales Netz:**



Toy-Problem: Training

- Input:

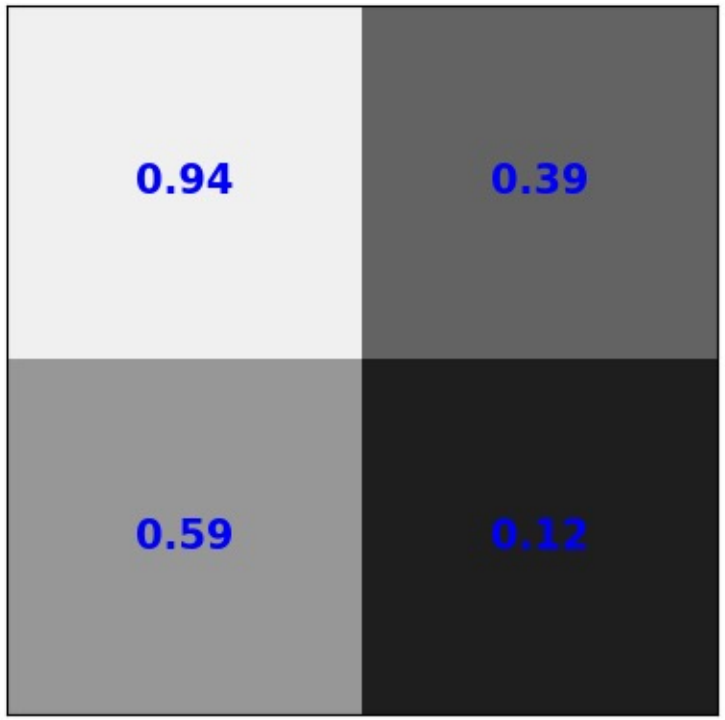
Target Output: **1**



Toy-Problem: Training

- Input:

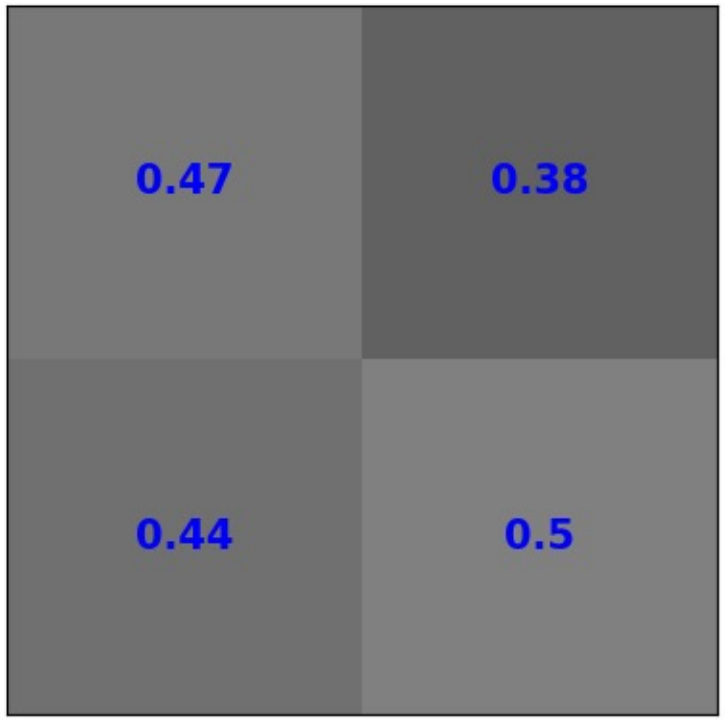
Target Output: **1**



Toy-Problem: Training

- Input:

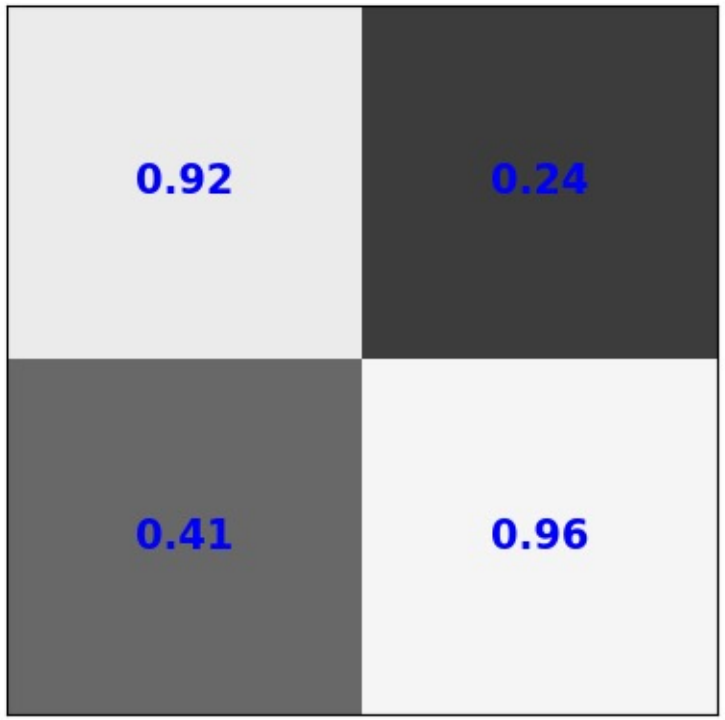
Target Output: **0**



Toy-Problem: Training

- Input:

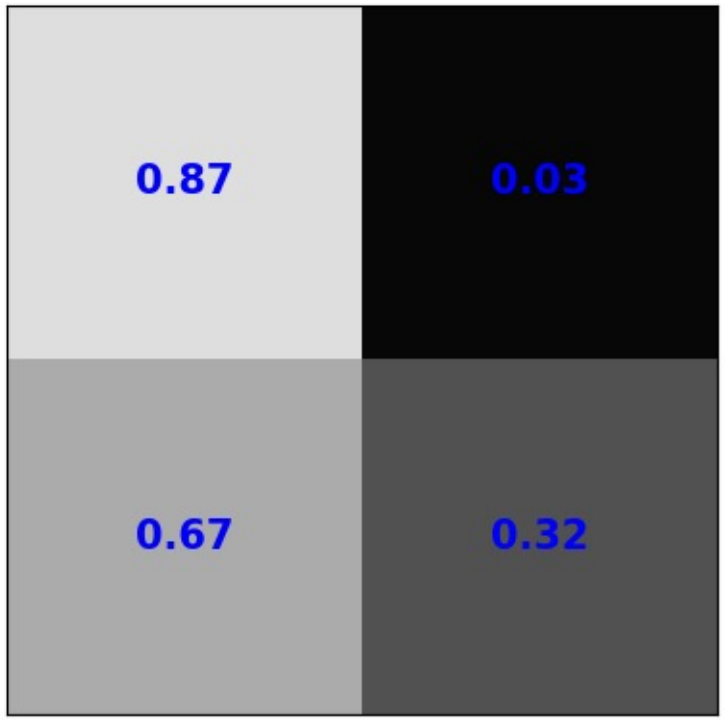
Target Output: **1**



Toy-Problem: Training

- Input:

Target Output: **0**

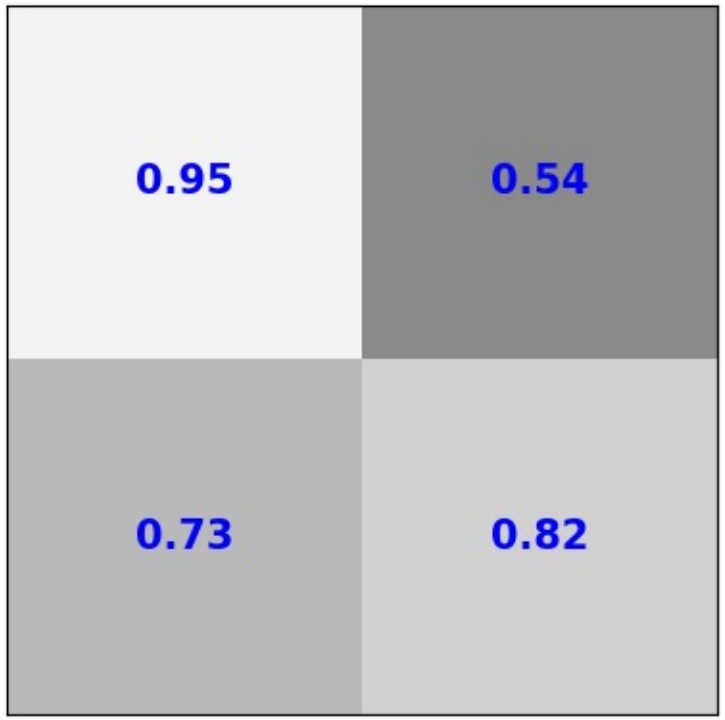


Toy-Problem

- Training abgeschlossen
- Nun: Teste NN

Toy-Problem: Test

- Input:

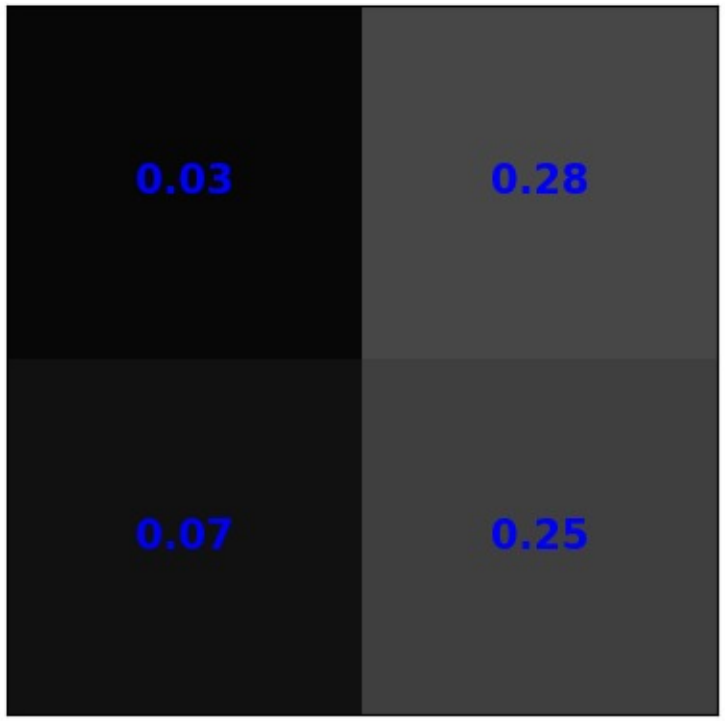


Target Output: **?**

Antwort: **1**

Toy-Problem: Test

- Input:

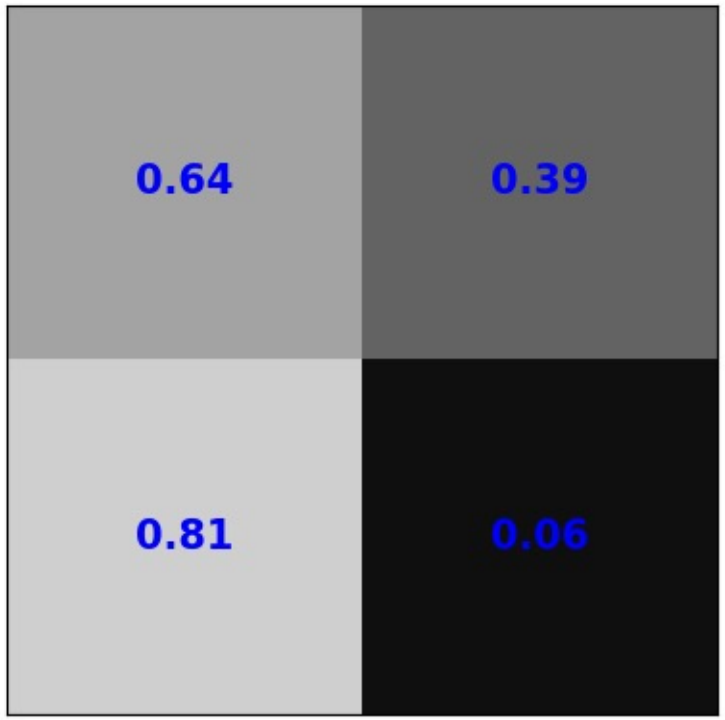


Target Output: **?**

Antwort: **0**

Toy-Problem: Test

- Input:

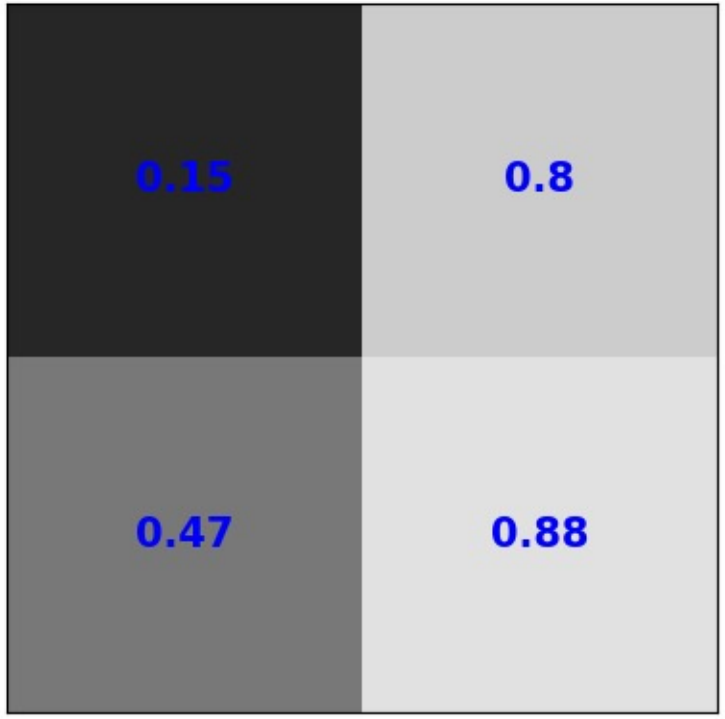


Target Output: **?**

Antwort: **0**

Toy-Problem: Test

- Input:

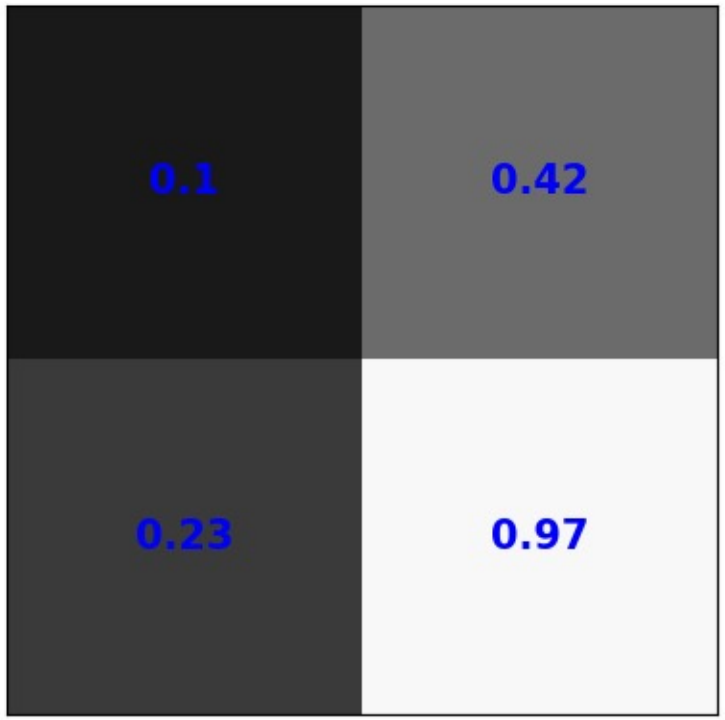


Target Output: **?**

Antwort: **1**

Toy-Problem: Test

- Input:



Target Output: **?**

Antwort: **0**

Toy-Problem: Regel?

- Wie lautet die Regel?
- Antwort:
  - Falls Durchschnitt der 4 Werte ≥ 0.5 $\rightarrow$ Output: 1 (Bild ist hell)
  - Falls Durchschnitt der 4 Werte < 0.5 $\rightarrow$ Output: 0 (Bild ist dunkel)
- Erkennt also, ob Bild im **Gesamten eher dunkel oder hell** ist
- Wollen nun ein NN für dieses Problem programmieren

NN programmieren: Ablauf

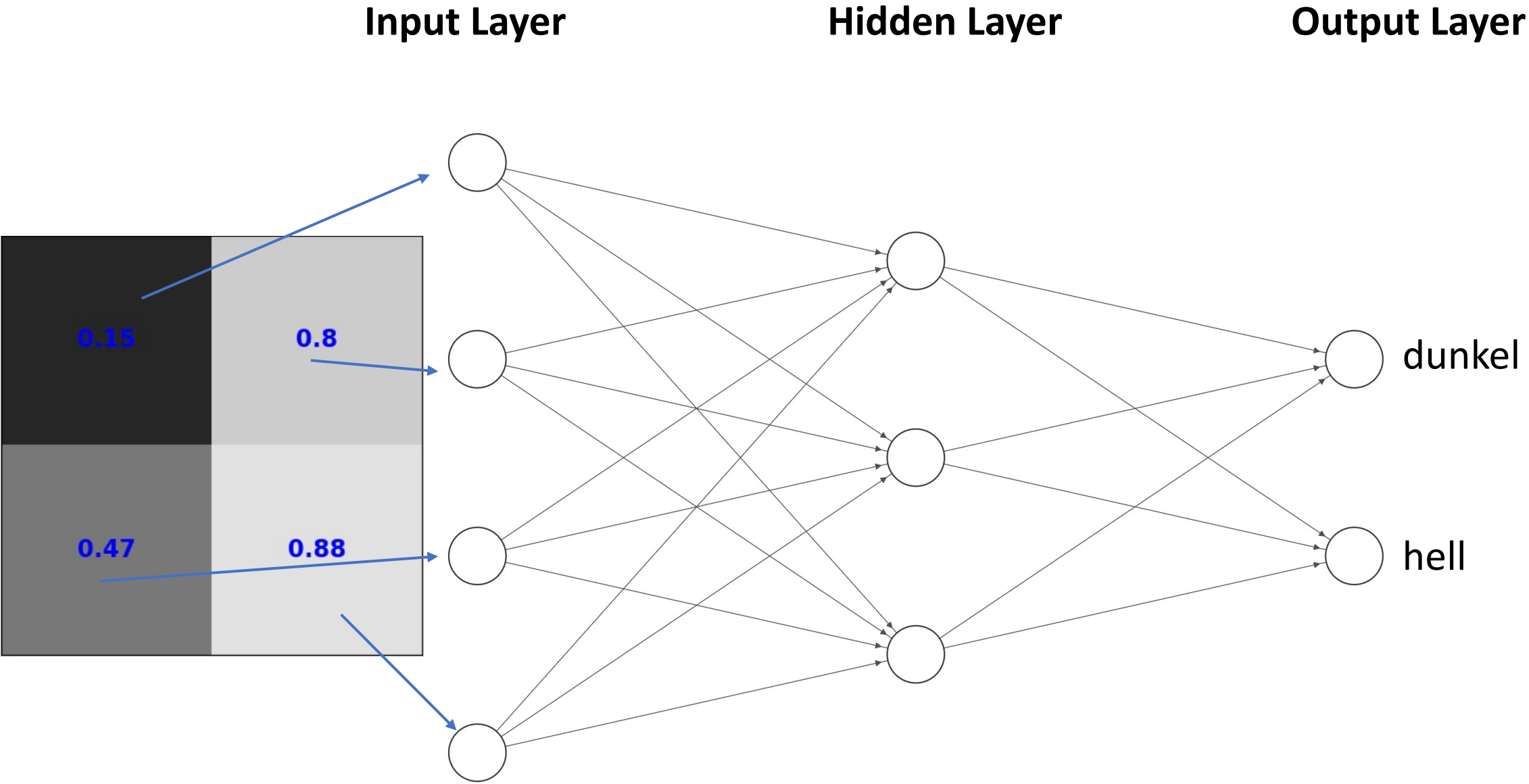
1. Grundstruktur des NN (Toy-Problem)
2. Feedforward:
   - Input in NN hineingeben
   - Output berechnen
   - Berechneten Output mit dem Target Output vergleichen
   - **Ziel für heute**
3. NN trainieren durch Backpropagation
4. NN testen

1. Grundstruktur des NN

- Input Layer:
  - 4 Neuronen
  - Wert: zw. 0 und 1
- Hidden Layer
  - 1 Hidden Layer mit 3 Neuronen
- Output Layer
  - 2 Neuronen (hell oder dunkel)

2. Feedforward

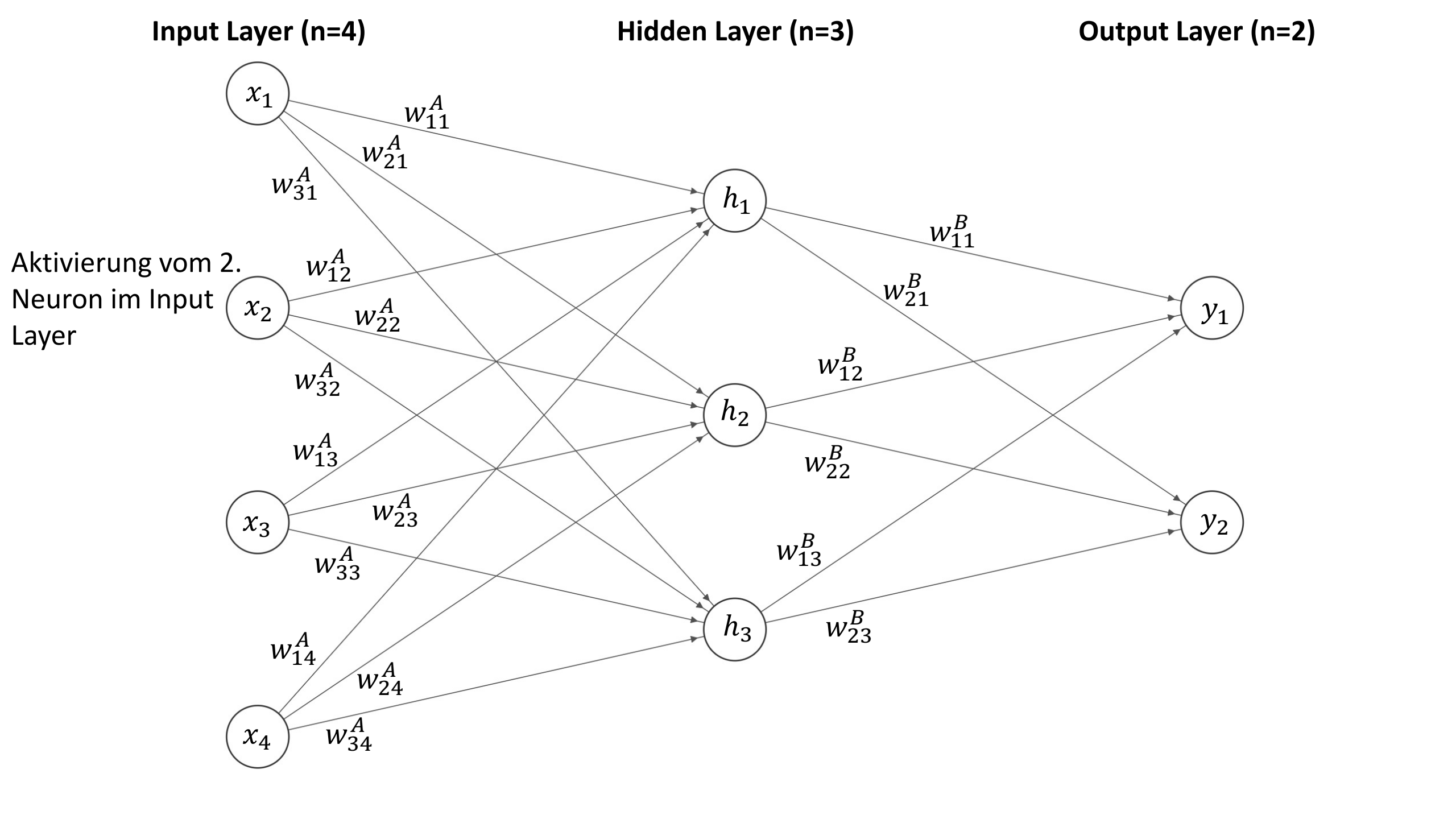
- Feedforward = Forward-Propagation
- Füttere Daten von Datensample in Input Layer
- Lasse diese durch das NN hindurch laufen
- Bestimme Output


Input Layer
Hidden Layer
Output Layer
0.15
0.8
0.47
0.88
dunkel
hell

Part I
NN Programmieren:
Struktur & Feedforward

Struktur des NN

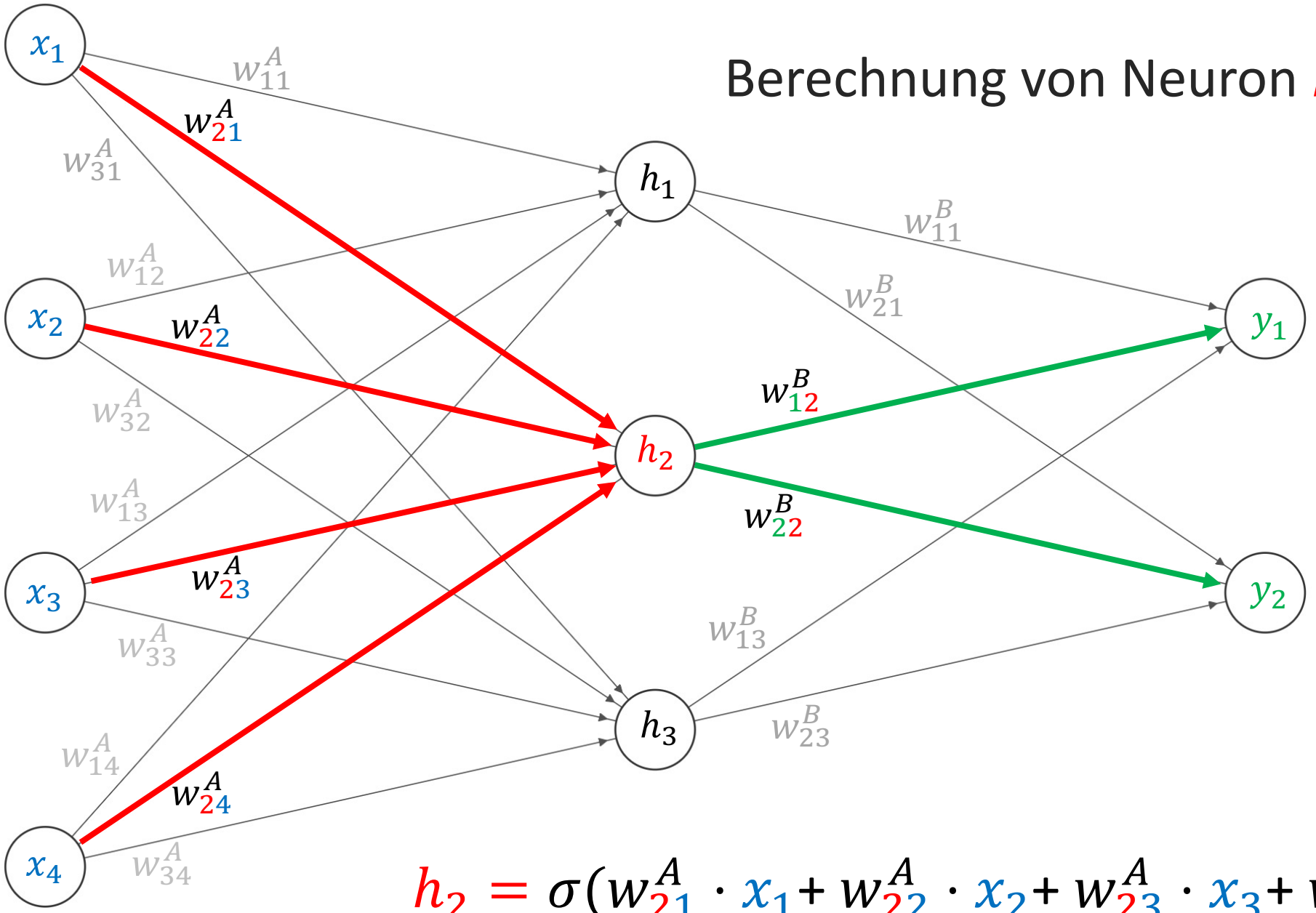
- Neuronales Netz wird durch **Gewichte** beschrieben.
- Ein Gewicht beschreibt, wie 2 Neuronen miteinander verknüpft sind.
- Trainieren eines NN heisst: Möglichst optimale Werte für Gewichte finden.


Input Layer (n=4)
Hidden Layer (n=3)
Output Layer (n=2)
Aktivierung vom 2. Neuron im Input Layer
x_1
x_2
x_3
x_4
h_1
h_2
h_3
y_1
y_2
w_{11}^A
w_{21}^A
w_{31}^A
w_{12}^A
w_{22}^A
w_{32}^A
w_{13}^A
w_{23}^A
w_{33}^A
w_{14}^A
w_{24}^A
w_{34}^A
w_{11}^B
w_{21}^B
w_{12}^B
w_{22}^B
w_{13}^B
w_{23}^B

Berechnung Hidden Layer

- Wie berechnet man die Werte der Neuronen im Hidden Layer für gegebene Gewichte und Input?

Input Layer (n=4) **Hidden Layer (n=3)** **Output Layer (n=2)**

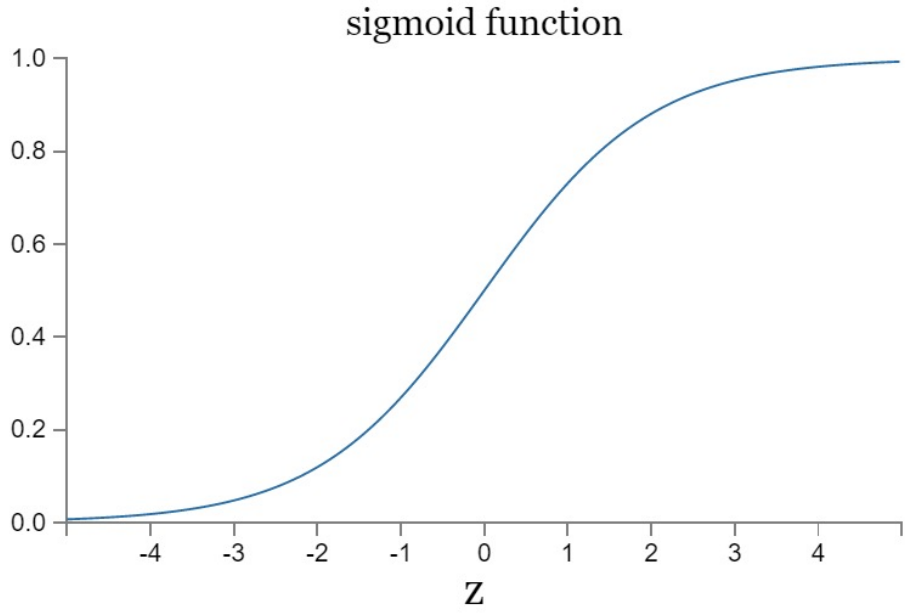


Berechnung von Neuron h_2

$$h_2 = \sigma(w_{21}^A \cdot x_1 + w_{22}^A \cdot x_2 + w_{23}^A \cdot x_3 + w_{24}^A \cdot x_4)$$

Berechnung Hidden Layer

- Aktivierungsfunktion:
  - Sigmoid-Funktion: σ [Sigma]
  - $\sigma(z) = \frac{1}{1+e^{-z}}$
  - Schränkt Wert des Neurons in Bereich [0,1] ein
- Berechnung:
  - $h_1 = \sigma(w_{11}^A \cdot x_1 + w_{12}^A \cdot x_2 + w_{13}^A \cdot x_3 + w_{14}^A \cdot x_4)$
  - $h_2 = \sigma(w_{21}^A \cdot x_1 + w_{22}^A \cdot x_2 + w_{23}^A \cdot x_3 + w_{24}^A \cdot x_4)$
  - $h_3 = \sigma(w_{31}^A \cdot x_1 + w_{32}^A \cdot x_2 + w_{33}^A \cdot x_3 + w_{34}^A \cdot x_4)$
- Wie programmieren?
  - Uncool: 2 for-Schleifen
  - Cool: Schreibe kompakt in Matrizen-Schreibweise!



Berechnung Hidden Layer

- 1. Gewichtsmatrix (Input zu Hidden Layer):
  - $w^A = \begin{pmatrix} w_{11}^A & w_{12}^A & w_{13}^A & w_{14}^A \\ w_{21}^A & w_{22}^A & w_{23}^A & w_{24}^A \\ w_{31}^A & w_{32}^A & w_{33}^A & w_{34}^A \end{pmatrix}$
  - Ist 3×4 −Matrix

- Input-Vektor:
  - $x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix}$
  - Ist 4er-Vektor ($= 4\times1$ −Matrix)

Matrixmultiplikation: Hidden Layer

- Betrachten Vereinfachung: “Matrix mal Vektor”
- $(3\times 4$ −Matrix) × $(4\times 1$ −Matrix) = 3×1 −Matrix = 3er-Vektor
- Hidden Layer: berechne durch Matrixmultiplikation

- $$w^A \cdot x = \begin{pmatrix} w_{11}^A & w_{12}^A & w_{13}^A & w_{14}^A \\ w_{21}^A & w_{22}^A & w_{23}^A & w_{24}^A \\ w_{31}^A & w_{32}^A & w_{33}^A & w_{34}^A \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} w_{11}^A \cdot x_1 + w_{12}^A \cdot x_2 + w_{13}^A \cdot x_3 + w_{14}^A \cdot x_4 \\ w_{21}^A \cdot x_1 + w_{22}^A \cdot x_2 + w_{23}^A \cdot x_3 + w_{24}^A \cdot x_4 \\ w_{31}^A \cdot x_1 + w_{32}^A \cdot x_2 + w_{33}^A \cdot x_3 + w_{34}^A \cdot x_4 \end{pmatrix}$$

- $$h = \sigma(w^A \cdot x) = \begin{pmatrix} \sigma(w_{11}^A \cdot x_1 + w_{12}^A \cdot x_2 + w_{13}^A \cdot x_3 + w_{14}^A \cdot x_4) \\ \sigma(w_{21}^A \cdot x_1 + w_{22}^A \cdot x_2 + w_{23}^A \cdot x_3 + w_{24}^A \cdot x_4) \\ \sigma(w_{31}^A \cdot x_1 + w_{32}^A \cdot x_2 + w_{33}^A \cdot x_3 + w_{34}^A \cdot x_4) \end{pmatrix}$$

Matrixmultiplikation: Output Layer

- 2. Gewichtsmatrix (hidden zu Output Layer)
  - $w^B = \begin{pmatrix} w_{11}^B & w_{12}^B & w_{13}^B \\ w_{21}^B & w_{22}^B & w_{23}^B \end{pmatrix}$
- Hidden Layer (berechnet wie vorher)
  - $h = \begin{pmatrix} h_1 \\ h_2 \\ h_3 \end{pmatrix}$
- Output Layer:
  - $y = \sigma(w^B \cdot h) = \cdots$
  - ist 2er-Vektor (also 2×1 −Matrix)

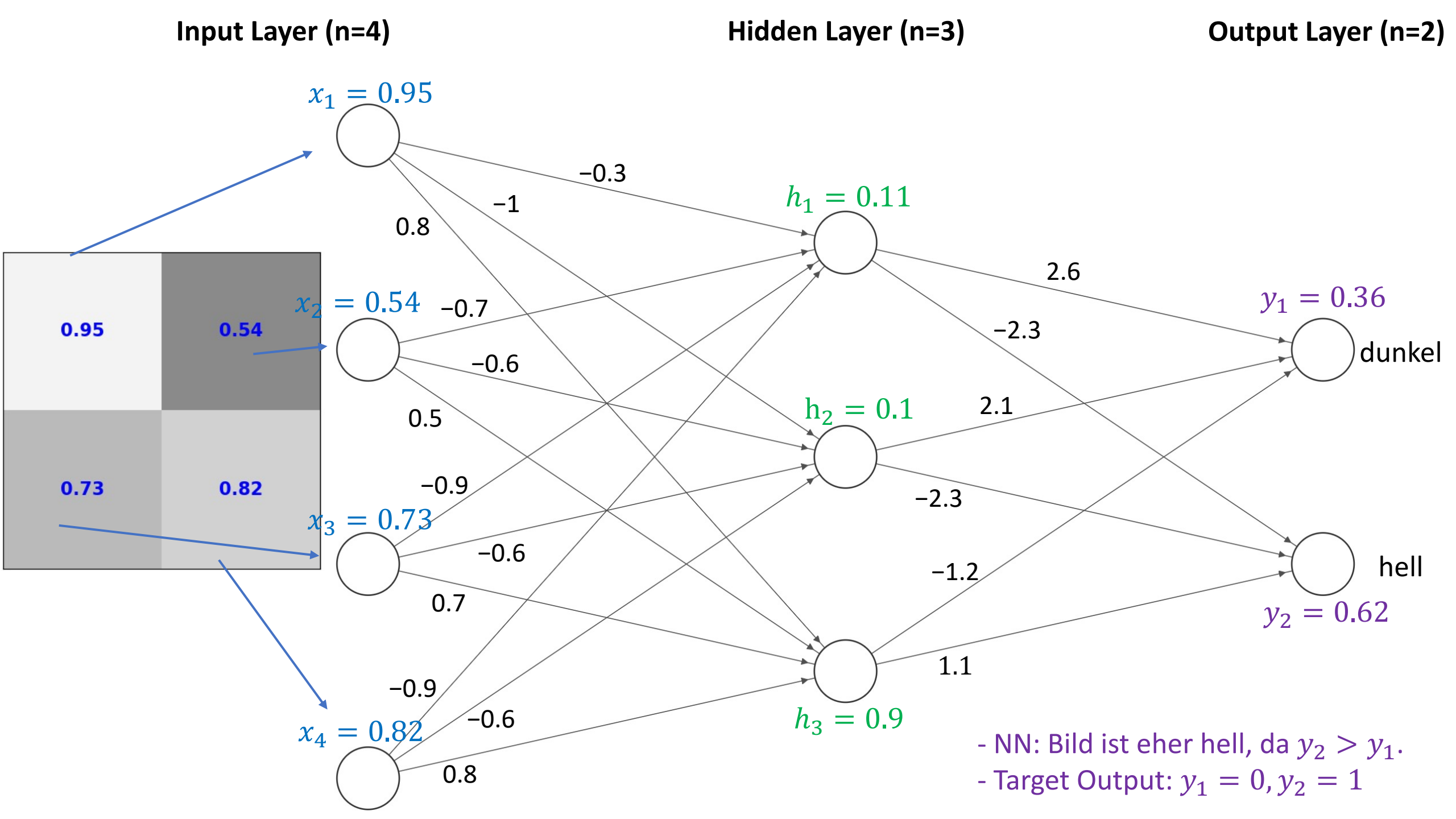
Zusammenfassung: Feedforward

- Gegeben:
  - Input Layer: x (4er-Vektor)
  - Gewichtsmatrizen:
    - w^A (input zu hidden, 3×4 −Matrix)
    - w^B (hidden zu Output, 2×3 −Matrix)

- 1. Schritt: berechne Hidden Layer: $h = \sigma(w^A \cdot x)$ (3er-Vektor)
- 2. Schritt: berechne Output Layer: $y = \sigma(w^B \cdot h)$ (2er-Vektor)

Beispiel: forward-propagation im Toy-Problem

- Zurück zum Toy-Problem (dunkel/hell-Erkenner)
- Verwende Gewichtsmatrizen (bereits trainiert):
  - $w^A = \begin{pmatrix} -0.3 & -0.7 & -0.9 & -0.9 \\ -1 & -0.6 & -0.6 & -0.6 \\ 0.8 & 0.5 & 0.7 & 0.8 \end{pmatrix}$
  - $w^B = \begin{pmatrix} 2.6 & 2.1 & -1.2 \\ -2.3 & -2.3 & 1.1 \end{pmatrix}$
- Das Neuronale Netz sieht also wie folgt aus …


Input Layer (n=4)
Hidden Layer (n=3)
Output Layer (n=2)
$x_1 = 0.95$
$x_2 = 0.54$
$x_3 = 0.73$
$x_4 = 0.82$
0.95
0.54
0.73
0.82
−0.3
−1
0.8
−0.7
−0.6
0.5
−0.9
−0.6
0.7
−0.9
−0.6
0.8
$h_1 = 0.11$
$h_2 = 0.1$
$h_3 = 0.9$
2.6
−2.3
2.1
−2.3
−1.2
1.1
$y_1 = 0.36$
dunkel
hell
$y_2 = 0.62$
- NN: Bild ist eher hell, da $y_2 > y_1$.
- Target Output: $y_1 = 0, y_2 = 1$

Matrizen in Python

- Beispiel: $\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$
- Verwende **numpy** package → Matrizen
- Importiere numpy: *import numpy as np*
- Erstelle Matrix (genauer: numpy-array):
  m=np.array([[1,2,3],[4,5,6]])
- Dimension von numpy-array: *m.shape*
  Resultat: (2, 3)
- Matrixmultiplikation:
  m = np.array([[1,2,3],[4,5,6]])
  v = np.array([7,8,9])
  np.dot(m,v)

Matrizen in Python: Spezielle Matrizen

- Nullmatrix:
  - $\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$
  - *np.zeros((2,3))*
  - Beachte die doppelten Klammern!
- Matrix mit Zufallswerten
  - Werte im Intervall [0,1)
  - *np.random.rand(2,3)*
- Matrix transponieren:
  - $\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \rightarrow \begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix}$
  - *m.T*
- Dimension (Grösse, Form) von Matrix:

  Beispiel für $\mathrm{m} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$, also *m = np.zeros((2,3))*

  → *m.shape* → Output: (2, 3)

Matrizen in Python: Dimension von Matrizen

- Dimension von Matrix: *m.shape*
- Beispiel für $\mathrm{m} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$, also *m = np.zeros((2,3))*
  → *m.shape* → Output: (2, 3)
- Betrachte *v = np.array([1,2,3])*
- *v* ist 1D numpy array, also: *v.shape* → Output: (3,)
- Oft ist aber wichtig, ob es sich um 3×1 oder 1×3 Matrix handelt:
  - 3×1 Matrix: $\begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$, erzeuge durch: *v = v.reshape((-1,1))*
  - 1×3 Matrix: $\begin{pmatrix} 1 & 2 & 3 \end{pmatrix}$, erzeuge durch: *v = v.reshape((1,-1))*

Datenfile (csv) in Python einlesen

- CSV: Comma Separated Values
- Beispiel:
  0,13,44,14,27
  0,90,153,58,31
  1,153,188,255,75
- *with open('my_file.csv', 'r') as f:*
  data_list = f.readlines()

Aufgaben

- Die Aufgaben findest du jeweils am Schluss der Slides.
- Zusatzaufgaben findest du auf einer separaten Slide.
- Sämtliche Aufgaben **müssen abgegeben** werden.
  - Erstelle dazu ein GitHub-Repository und gebe es deinem Lehrer frei.
  - Beschrifte die Files genau so wie vorgegeben.
  - Schicke nach erfolgreichem absolvieren den Link per Teams an den Lehrer.

Auftrag 1

- **Ziel**: **Feedforward** programmieren, **Erfolgsquote** berechnen
- Erstelle File **«01_toy_problem_feedforward.py»**
- Lese den Datensatz (CSV-Format) «data_dark_bright_test_4000.csv» ein.
- Berechne für jeden Datenpunkt (1 Zeile im File) den Output durch Feedforward. Verwende dazu die beiden Gewichtsmatrizen:

$$w^A = \begin{pmatrix} -0.3 & -0.7 & -0.9 & -0.9 \\ -1 & -0.6 & -0.6 & -0.6 \\ 0.8 & 0.5 & 0.7 & 0.8 \end{pmatrix}, \quad w^B = \begin{pmatrix} 2.6 & 2.1 & -1.2 \\ -2.3 & -2.3 & 1.1 \end{pmatrix}$$

- Vergleiche jeweils den berechneten Output mit dem Target Output
- Berechne in %, für wie viele Datenpunkte das NN den richtigen Output liefert (Erfolgsquote). Diese Zahl gibt an, wie gut das NN funktioniert.
- Besprich deine Lösung mit dem Lehrer.

Auftrag 2

- **Ziel: Grundgerüst** für ein neuronales Netzwerk **objektorientiert** programmieren
- Erstelle File **«02_feedforward_oop.py»**
- Der Code soll eine **Klasse «*Network*»** enthalten:
  - Dieser soll als Argumente übergeben werden:
    - Anzahl Input-Neuronen
    - Anzahl Neuronen im Hidden Layer
    - Anzahl Output-Neuronen
  - Diese soll die Gewichtsmatrizen als Attribute enthalten:
    - in der *init*-Methode sollen die beiden Gewichtsmatrizen erzeugt werden:
      self.wA = …
      self.wB = …
    - diese sollen die richtigen Dimensionen haben
    - die Werte sollen Zufallszahlen zw. -0.5 und 0.5 sein (*np.random.rand*)
- Die Klasse hat eine Methode *feedforward*. Dieser wird als Argument ein Input-Array übergeben. Diesen ‘feeded’ sie dann ‘forward’ und berechnet den zugehörigen Output.
- Weiter auf nächster Folie

Auftrag 2 [weiter]

- Die Klasse hat eine Methode *test*. Dieser kann man als Argument einen Datensatz übergeben, der dann durch das NN gefüttert wird. Es wird die Erfolgsquote zurückgegeben: Für wie viele (in Prozent) der Datenpunkte produziert das NN einen korrekten Output?.
- Lade sowohl die Daten des Toy-Problems wie auch die MNIST Daten ein. Erstelle *je* ein Neuronales Netz (Anzahl Neuronen pro Layer richtig wählen) und teste mit der *test* Methode, wie gut dein NN funktioniert.
  Tipp: Da die Gewichtsmatrizen zufällig erzeugt werden, sollte also bei unserem Toy-Problem eine Erfolgsquote von etwa 50% resultieren.
- Besprich deine Lösung mit dem Lehrer.