

Neuronale Netze programmieren (I)

TALIT-Kurs, Semesterschlusswoche 2022

Andreas Schärer

Tom Hofmann

Simon Graf

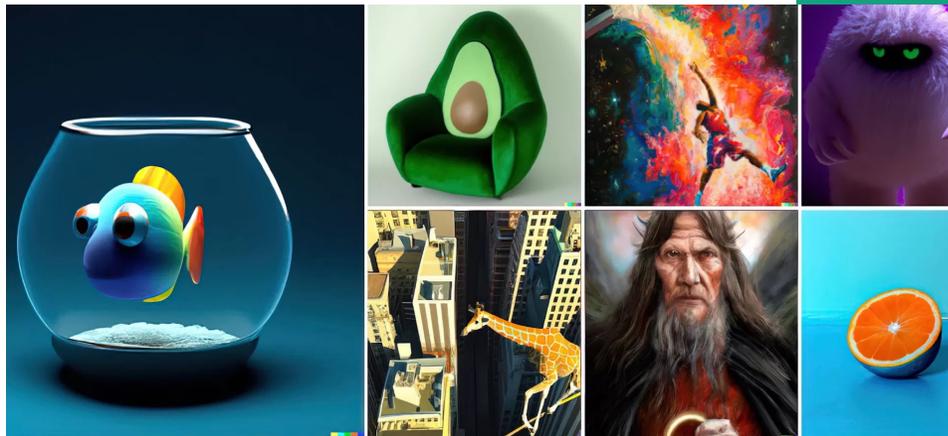
Pacifico Rodriguez

Part 0

Einführung & Übersicht

KI überall

- KI: Künstliche Intelligenz
- AI: Artificial Intelligence
- Neuronale Netze
-> benötigt für KI / AI



Bild

- Was kannst du über Bild sagen?
 - Foto, von ...
 - kleinem, jungen Hund (Welpen)
 - <Aussehen beschreiben>
- Dein Hund?
- Kennst du diesen Hund?
- Hast du diesen schon einmal gesehen?
- Nein!
- Hund existiert nicht, 'Foto' von Dall-E erzeugt.
- Warum erkennen wir sofort, dass es sich um einen kleinen, jungen Hund handelt, obwohl niemand je diesen Hund gesehen hat?



Hirn trainieren

- Baby:
 - keine Ahnung, was Hund ist
- Kleinkind:
 - Eltern zeigen Kind viele versch. Bilder von Hunden mit Info 'das ist ein Hund' ...
 - ... sowie echte Hunde
 - Wichtig: benötigt immer Bild plus Info, 'das ist ein Hund'.
 - Dadurch wird das neuronale Netz im Hirn von Kind trainiert



«Das ist ein Hund.»

Sie toben und spielen gerne! Hunden macht es großen Spaß, hinter geworfenen Bällen oder Stöcken herzurrennen und sie wiederzubringen. Das nennt man Apportieren. Hast du das gewusst? Und wie macht der kleine Welpe?

«Das ist ein anderer Hund.»



Hirn trainieren

- Durch diese Training ...
- ... baut Kind *Konzept* von Hund auf:
Abstraktes Modell von Merkmalen,
die einen Hund ausmachen
- Sieht jetzt Hund, den noch nie vorher
gesehen ...



Wau-Wau!



Hund erkennen

- Warum also erkennst du sofort, dass Hund ist?
- Neuronale Netz in unserem Hirn wurde seit Kleinkindalter entsprechend trainiert:
 - Haben viele Hunde gesehen mit Information 'das ist ein Hund'
 - Unser Hirn hat abstraktes Modell von Konzept 'Hund' entwickelt
- Benötigt **Intelligenz**.



Ziel diese Woche

- Gleiches Konzept wie mit Kind und Hund ...
- ... einfach für neuronales Netz (simuliertes Hirn) im Python-Code ..
- ... und handgeschriebene Zahlen



- Programmieren neuronales Netz:
 - Am Anfang ist 'dumm' (Baby, weiss nicht, wie eine 7 aussieht).
 - ... und trainieren es mit vielen Daten (Bild von 5 plus Info 'Das ist eine 5!')
 - ... bis es Bilder von Zahlen, die es noch nie gesehen hat, richtig interpretieren kann

Was ist künstliche Intelligenz?

- **‘normales’ Programmieren:**

- Programmierer programmiert einen Algorithmus ...
- ... er sagt Computer also genau, was dieser in welcher Situation tun soll
- Beispiele:
 - `is_prime(x)`
 - `decimal_to_binary(x)`
- Programmierer muss Vorgehen also selbst genau verstanden haben!

- **Künstliche Intelligenz:**

- Programmierer programmiert ein **neuronales Netz (NN)** ...
- Welches den Aufbau des Hirns imitiert.
- ... aber **keine Logik** (also was wann zu tun ist).
- Diese baut das Netz **selbst** auf, in dem man es mit Daten **trainiert**.
- Programmierer:in 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:

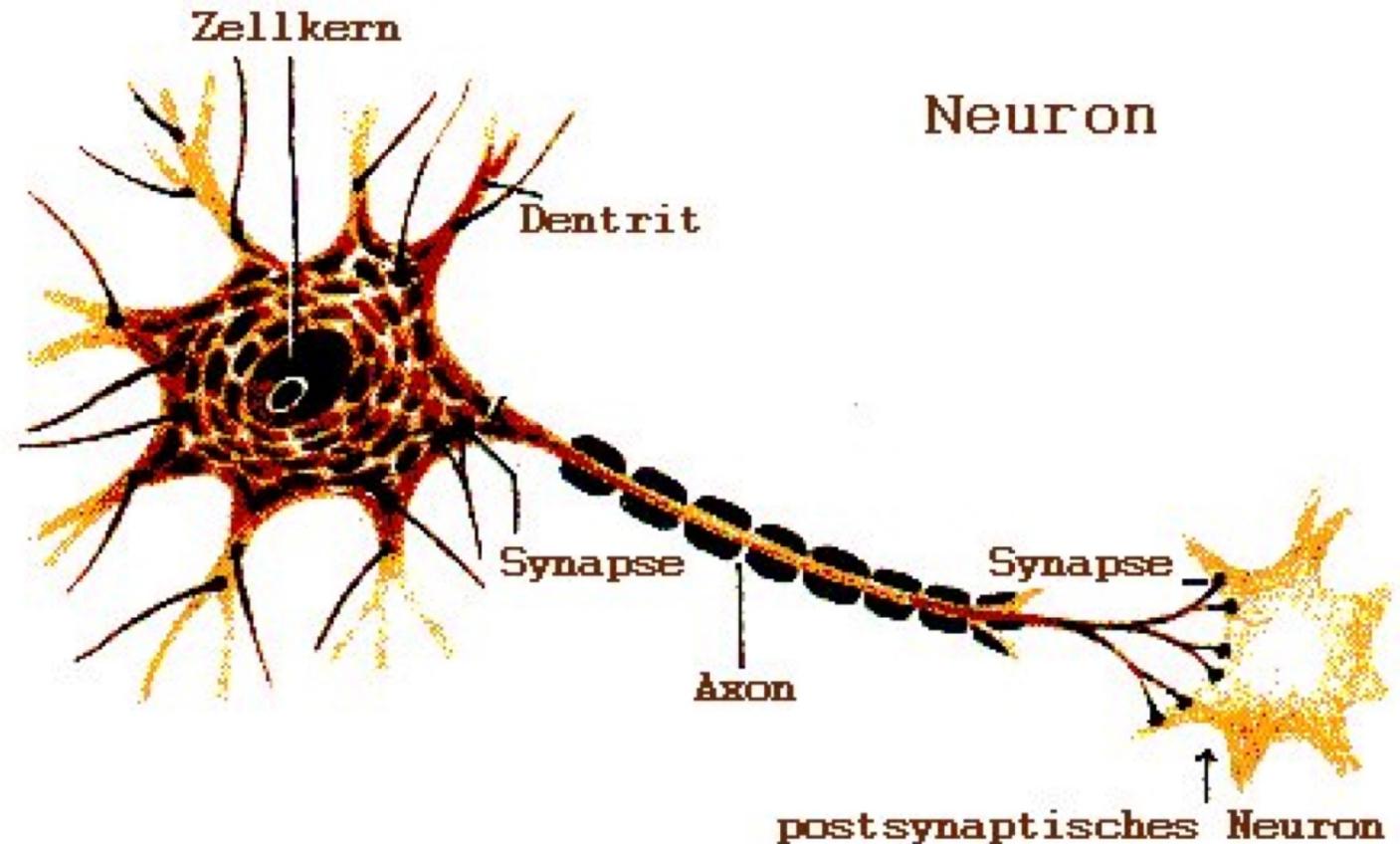
- Informationsverarbeitung

3. Axon:

- Informationsweiterleitung

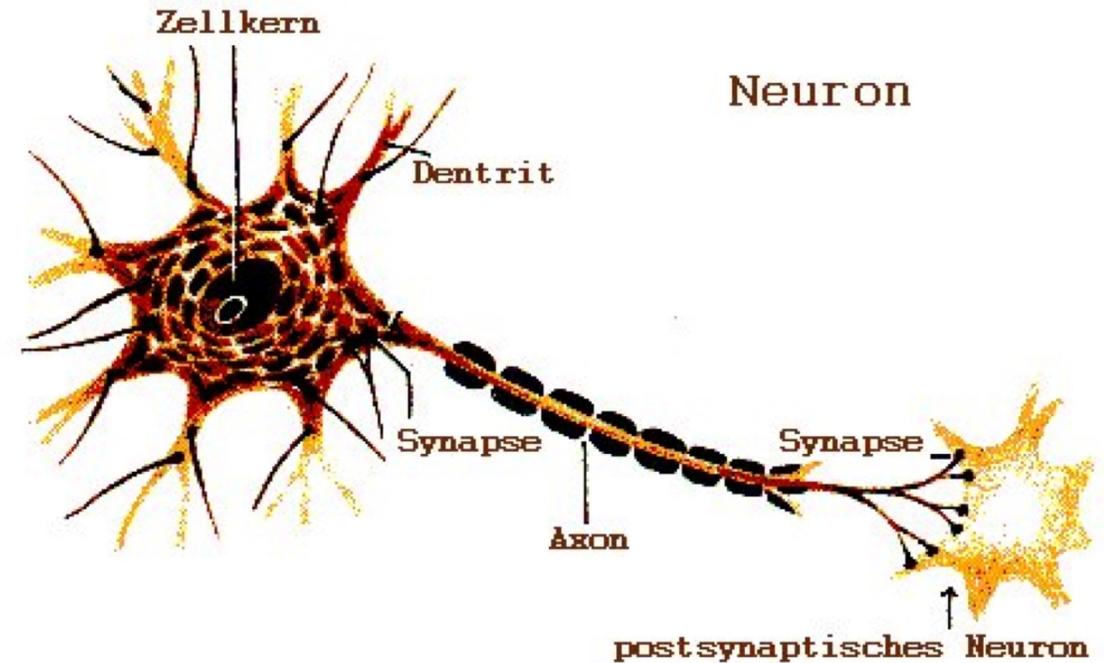
4. Synapse:

- Informationsübertragung
- Verbindung zu anderen Neuronen (Output)



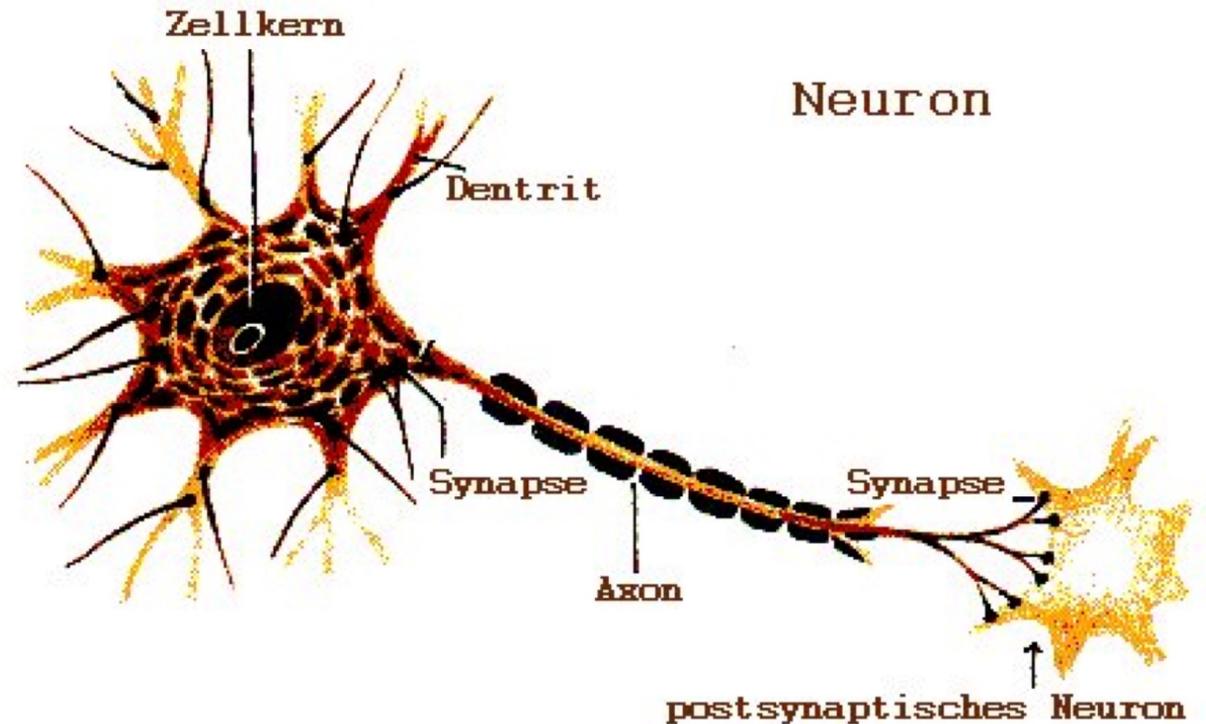
Funktionsweise des Hirns

- 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

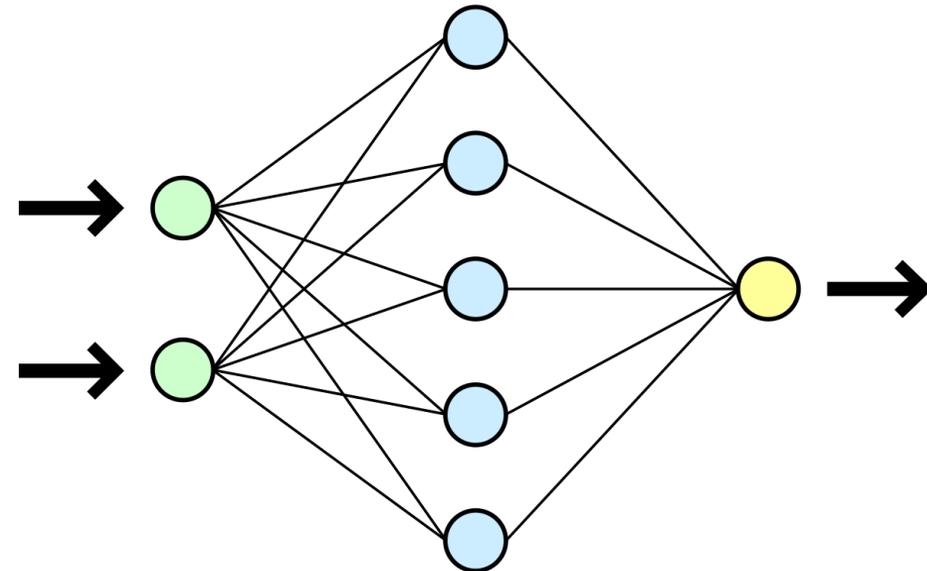
Neuronale Netze in künstlicher Intelligenz

- Imitiere Aufbau und Funktionsweise von unserem Hirn mit Computer

- Vorgehen:

1. Neuronales Netz programmieren

- Gewisse Anzahl Neuronen vorgeben



2. NN mit Daten trainieren:

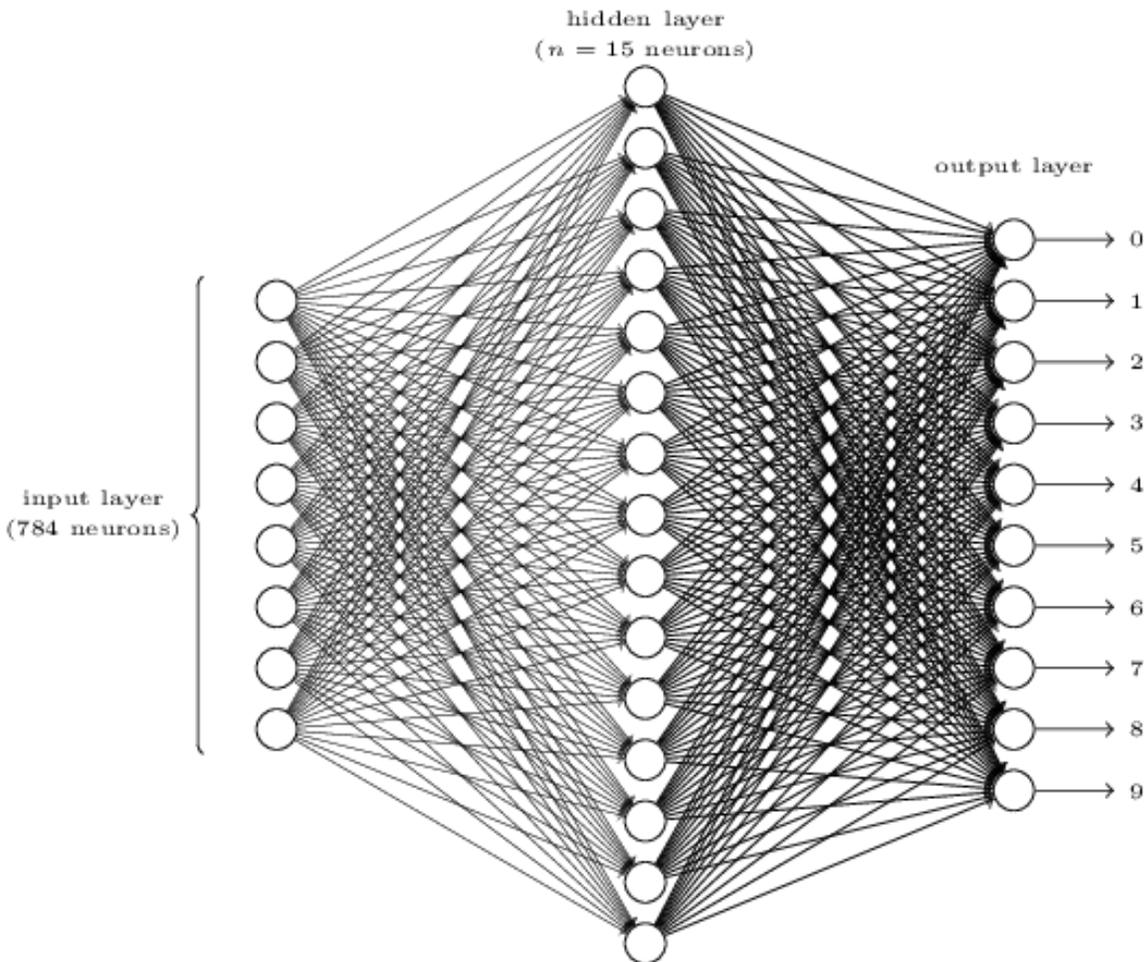
- Lernen: Verbindungen zwischen Neuronen stärken

3. Trainiertes NN testen: NN auf unbekannte Daten anwenden

Vorgehen dieser Woche

- “From scratch” **neuronales Netz (NN) selbst programmieren**
- Also ohne Verwendung von spezialisierten Libraries wie Keras, TensorFlow, ...
- Nutze dein NN als **Handschrifterkennung 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?

Struktur des NN



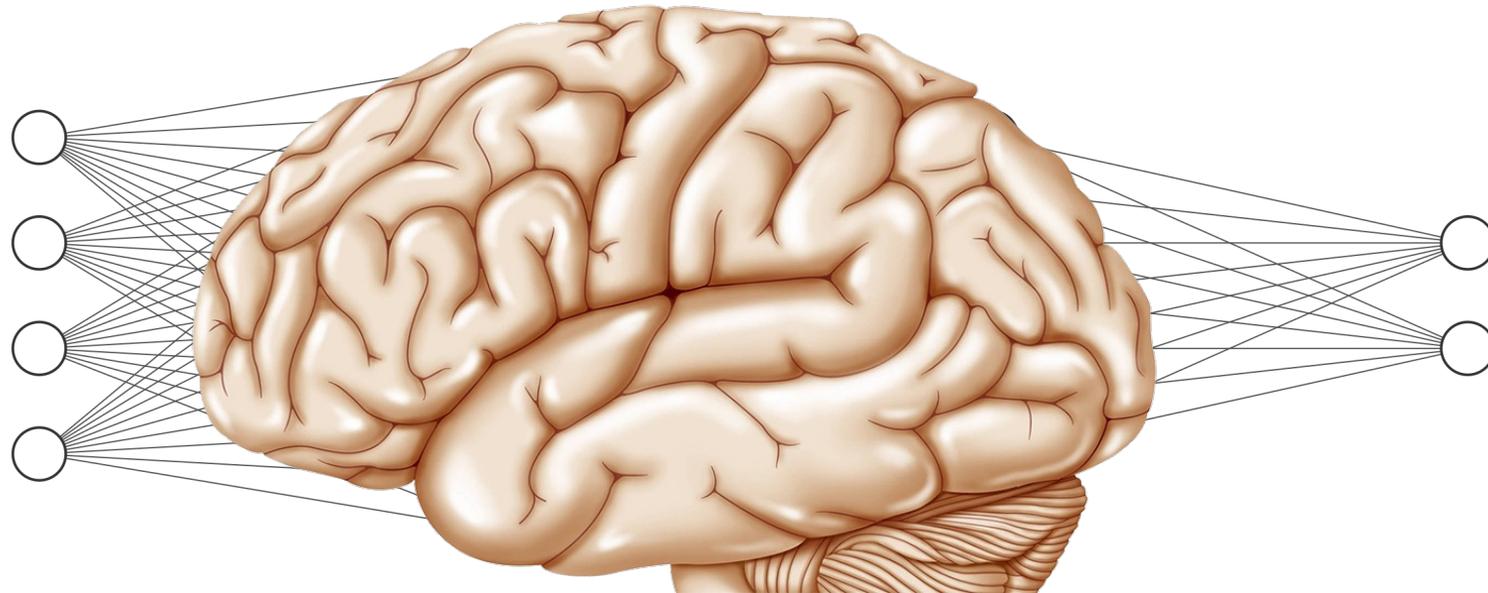
- Input Layer:
 - Pixelwerte
 - 784 = 28x28 Neuronen
- Hidden Layer(s):
 - 1 oder mehrere Hidden Layers
 - Jedes Layer: gewisse Anzahl Neuronen
- Output Layer:
 - 10 Neuronen
 - Ein Neuron für jede Zahl

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**

0.83	0.77
0.53	0.97

Toy-Problem: Training

- Input:

Target Output: **1**

0.94	0.39
0.59	0.12

Toy-Problem: Training

- Input:

Target Output: **0**

0.47	0.38
0.44	0.5

Toy-Problem: Training

- Input:

Target Output: **1**

0.92	0.24
0.41	0.96

Toy-Problem: Training

- Input:

Target Output: **0**

0.87	0.03
0.67	0.32

Toy-Problem

- Training abgeschlossen
- Nun: Teste NN

Toy-Problem: Test

- Input:

0.95	0.54
0.73	0.82

Target Output: ?

Antwort: **1**

Toy-Problem: Test

- Input:

0.03	0.28
0.07	0.25

Target Output: ?

Antwort: 0

Toy-Problem: Test

- Input:

0.64	0.39
0.81	0.06

Target Output: ?

Antwort: **0**

Toy-Problem: Test

- Input:

0.15	0.8
0.47	0.88

Target Output: ?

Antwort: **1**

Toy-Problem: Test

- Input:

0.1	0.42
0.23	0.97

Target Output: ?

Antwort: **0**

Toy-Problem: Regel?

- Wie lautet die Regel?
- Antwort:
 - Falls Durchschnitt der 4 Werte ≥ 0.5 → Output: 1 (Bild ist hell)
 - Falls Durchschnitt der 4 Werte < 0.5 → Output: 0 (Bild ist dunkel)
- Erkennt also, ob Bild im **Gesamten eher dunkel oder hell** ist
- Wollen nun ein NN für dieses Problem programmieren
- Dazu brauchen wir aber mehr Wissen über NN

Part I

NN Programmieren: Struktur & Feedforward

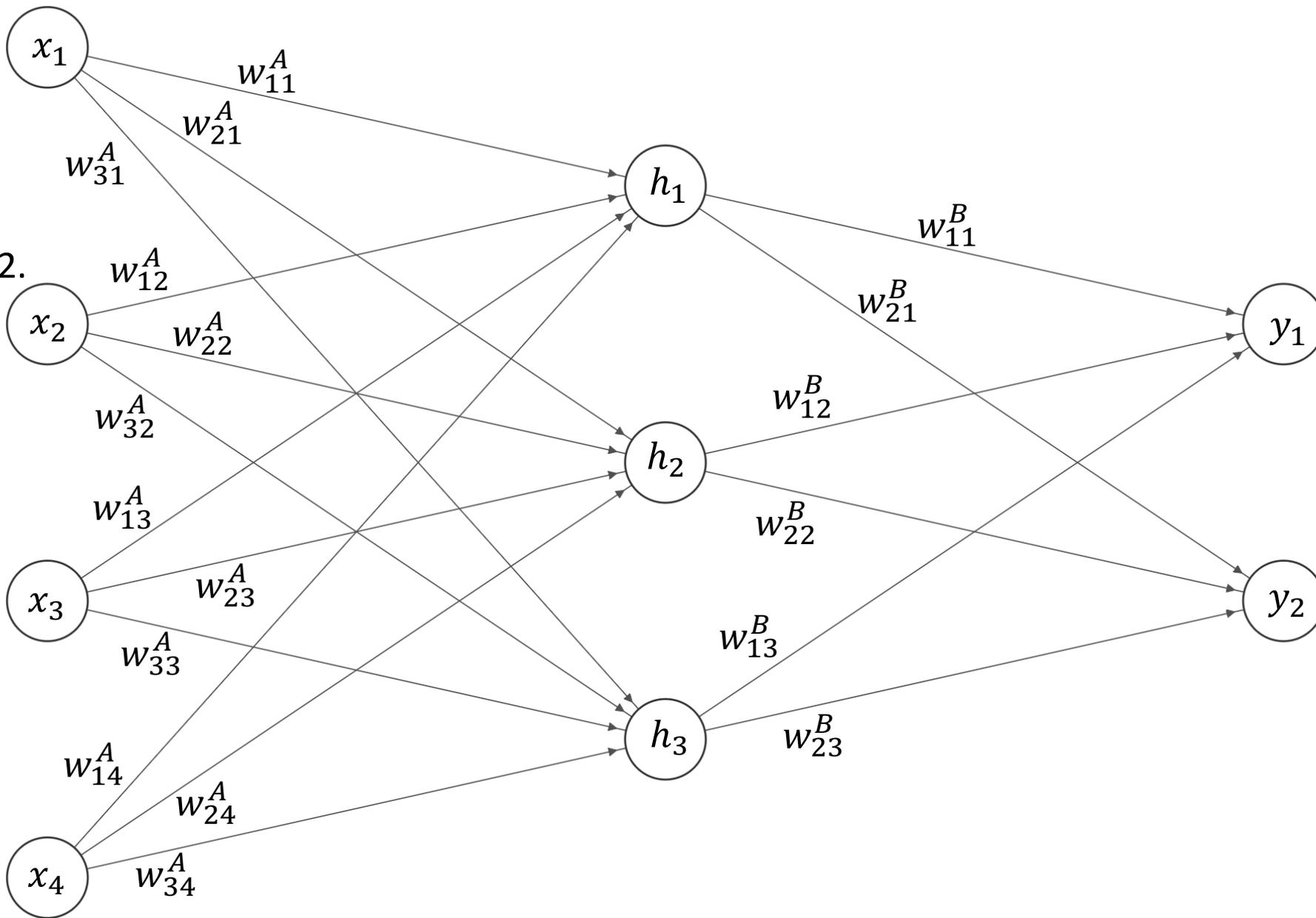
Struktur des NN

- 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

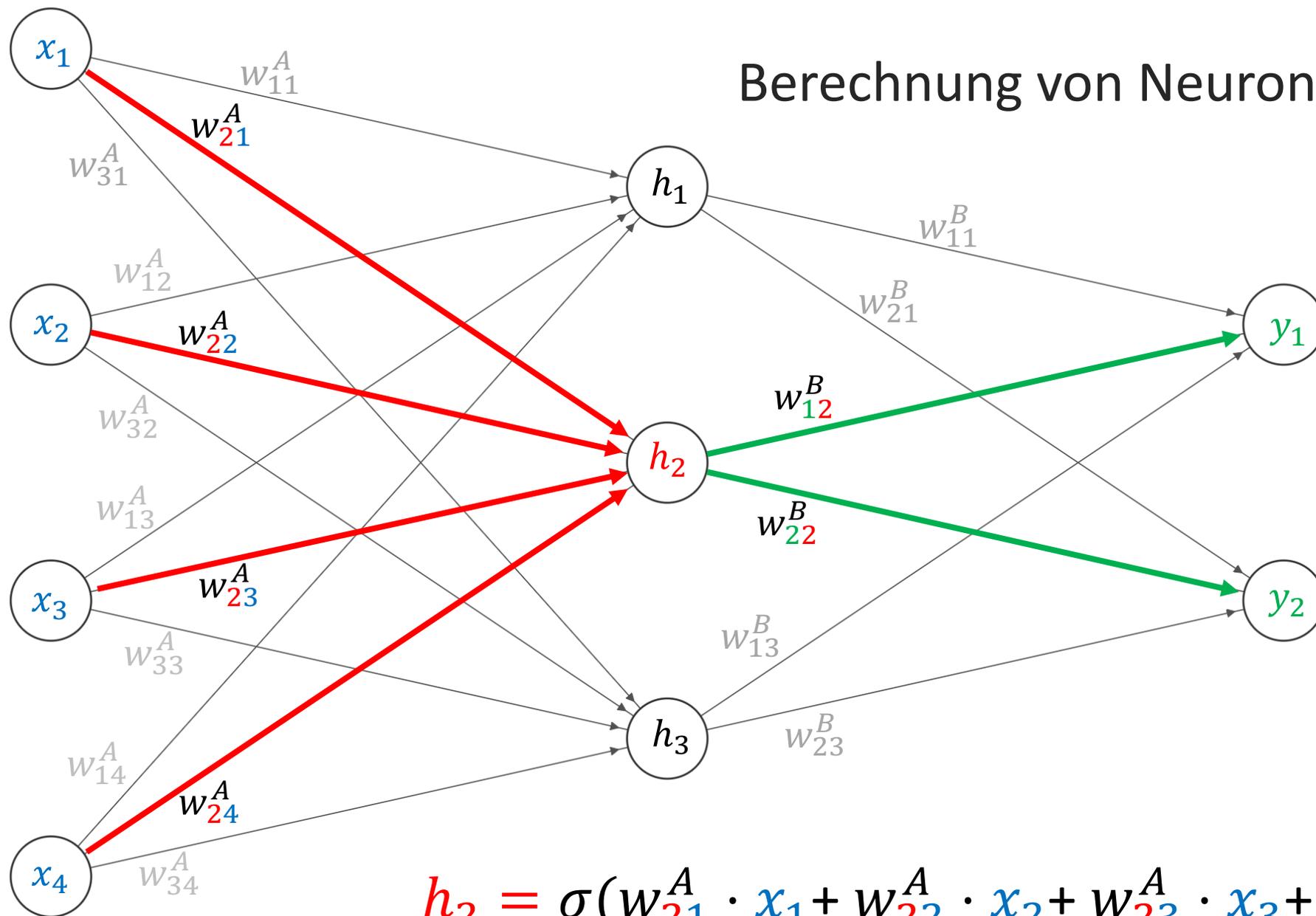
Berechnung Hidden Layer

- 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:

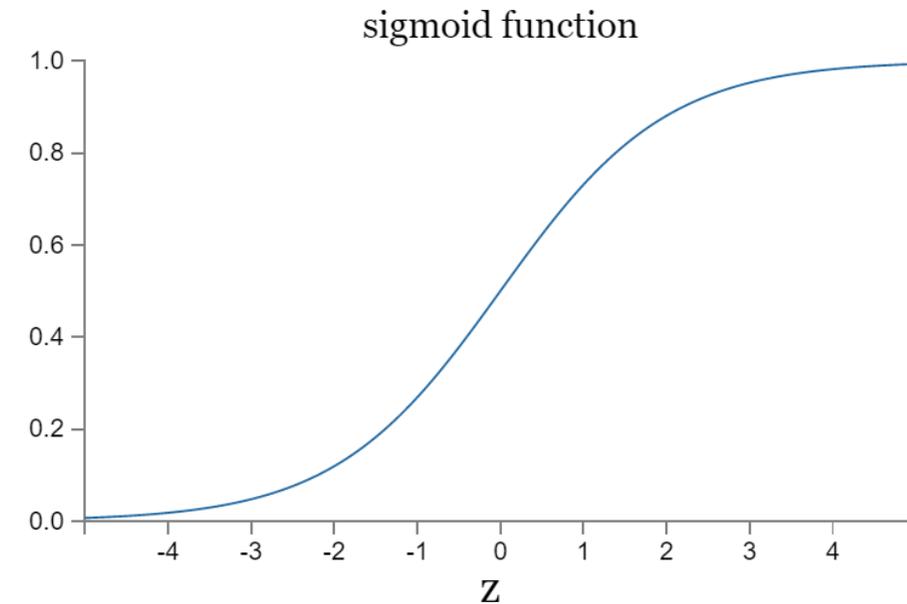
- $h_1 = \sigma(w_{11}^A \cdot x_1 + w_{12}^A \cdot x_2 + w_{13}^A \cdot x_3 + w_{14}^A \cdot x_4)$

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

- $h_3 = \sigma(w_{31}^A \cdot x_1 + w_{32}^A \cdot x_2 + w_{33}^A \cdot x_3 + w_{34}^A \cdot x_4)$

- Wie programmieren?

- Uncool: 2 for-Schleifen
- Cool: Schreibe kompakt in Matrizen-Schreibweise!



Berechnung Hidden Layer

- 1. Gewichtsmatrix (Input zu Hidden Layer):

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

- Ist 3×4 –Matrix

- Input-Vektor:

$$\bullet x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix}$$

- Ist 4er-Vektor (= 4×1 –Matrix)

Matrixmultiplikation: Hidden Layer

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

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

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

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:

- $y = \sigma(w^B \cdot h) = \dots$
 - 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):

- $w^A = \begin{pmatrix} -0.3 & -0.7 & -0.9 & -0.9 \\ -1 & -0.6 & -0.6 & -0.6 \\ 0.8 & 0.5 & 0.7 & 0.8 \end{pmatrix}$

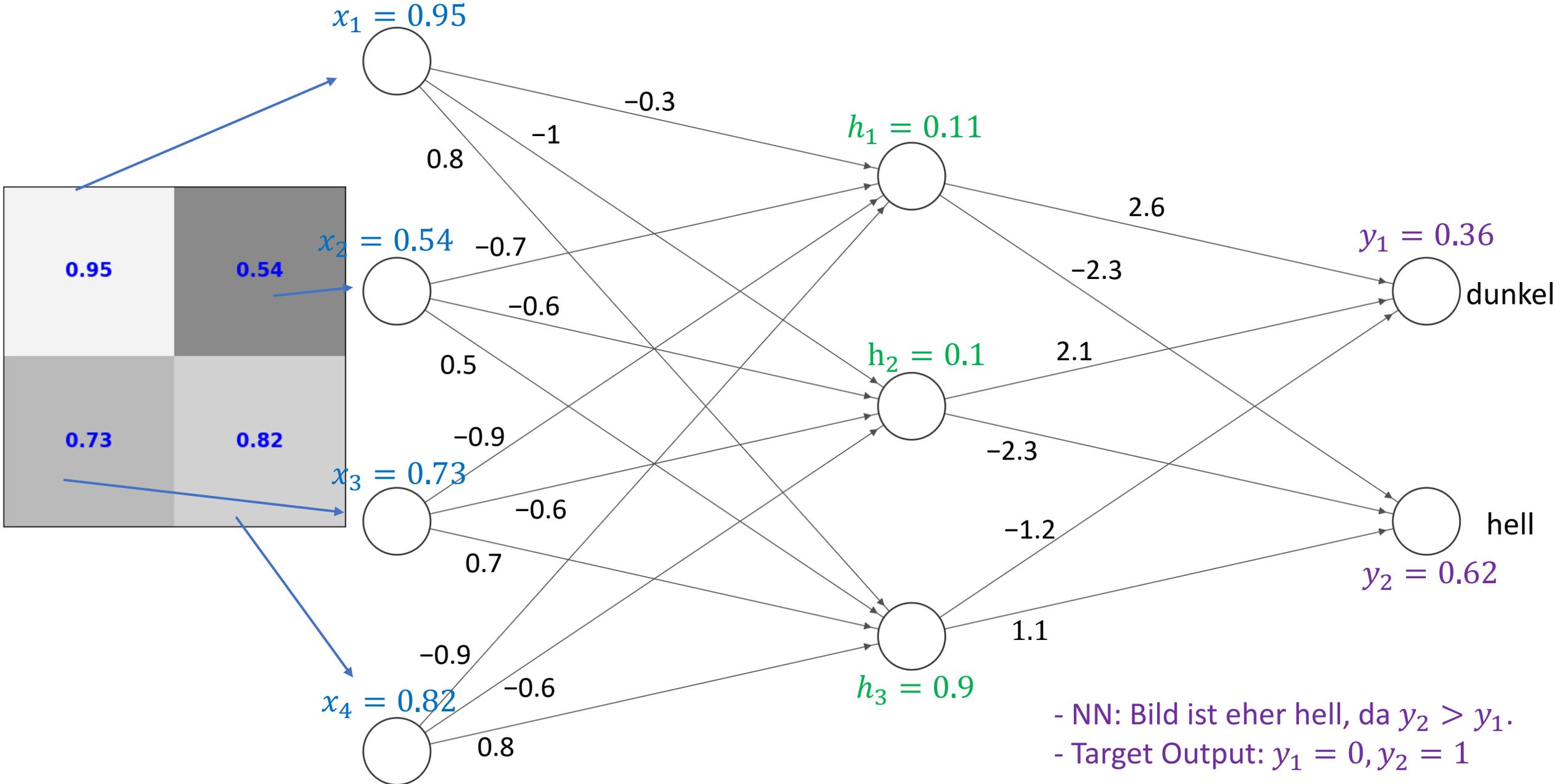
- $w^B = \begin{pmatrix} 2.6 & 2.1 & -1.2 \\ -2.3 & -2.3 & 1.1 \end{pmatrix}$

- Das Neuronale Netz sieht also wie folgt aus ...

Input Layer (n=4)

Hidden Layer (n=3)

Output Layer (n=2)



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

→ `m.shape` → Output: (2, 3)

Matrizen in Python: Dimension von Matrizen

- Dimension von Matrix: *m.shape*
- Beispiel für $m = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$, 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: (1 2 3), 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 auf Wiki.
- Zusatzaufgaben findest du ebenfalls auf Wiki.
- Tipp: Arbeite mit Jupyter-Notebook (.ipynb)
- Sämtliche Aufgaben **müssen abgegeben** werden.
 - Erstelle dazu ein GitHub-Repository und gebe es deinem Lehrer frei.
 - Beschrifte die Files genau so wie vorgegeben.

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.