Arbeitsweise von neuronalen Netzwerken

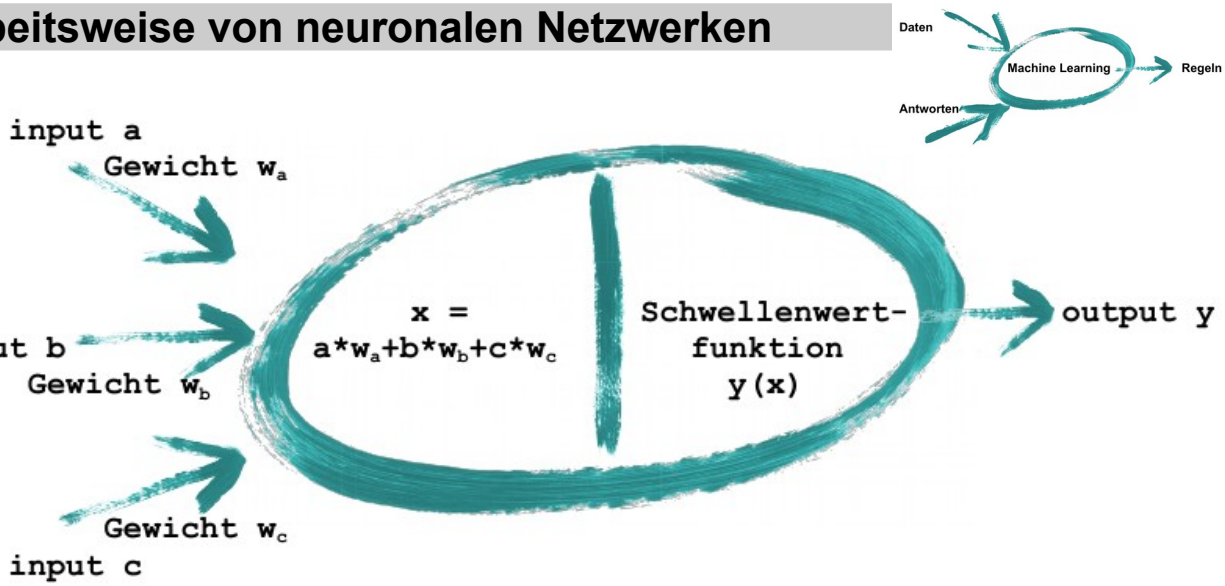


Abb. 1: Lernen in einem Neuron mit Gewichten, der *output* wird mit einer Schwellenwert-funktion berechnet (z. B. Sigmoid, ReLu, tanh usw.)

Die einzelnen Neuronen werden dann über verschiedene Ebenen (*layer*) hinweg miteinander verknüft, jede Verknüpfung besitzt ein eigenes Gewicht.

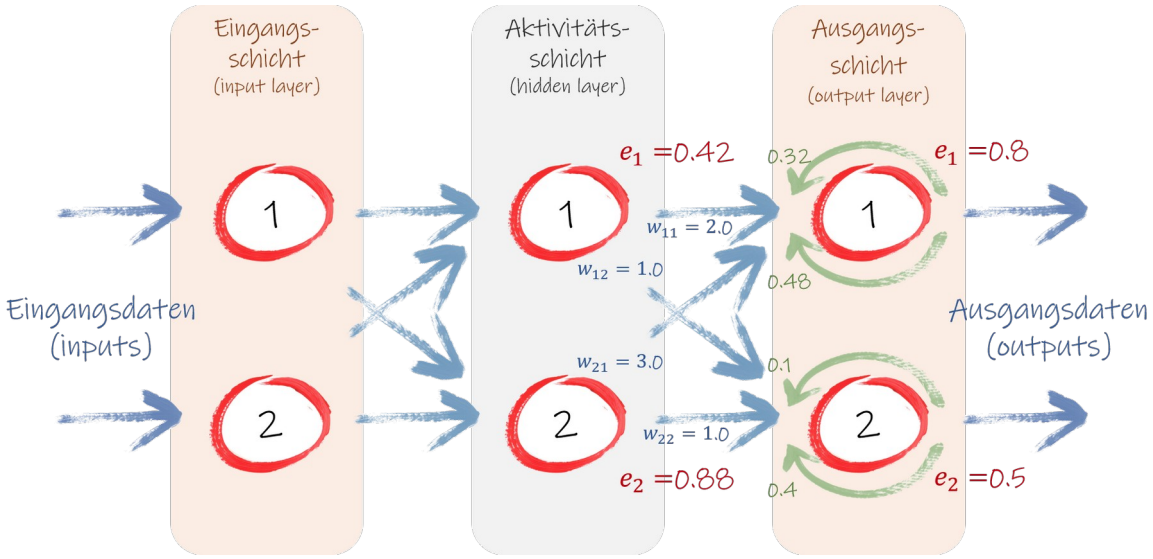


Abb. 2. Lernen mit vernetzten Neuronen durch Gewichtsänderung. Blaue Pfeile: Richtung des Datenflusses. Grüne Pfeile *backpropagation* (Fehlerrückführung, Anpassung der Gewichte)

Ein einfaches NN besteht aus *input layer*, *hidden layer* und *output layer*. Wie man in Abbildung 3 sieht, können die Neuronen aber auch in mehreren Ebenen miteinander kombiniert werden.

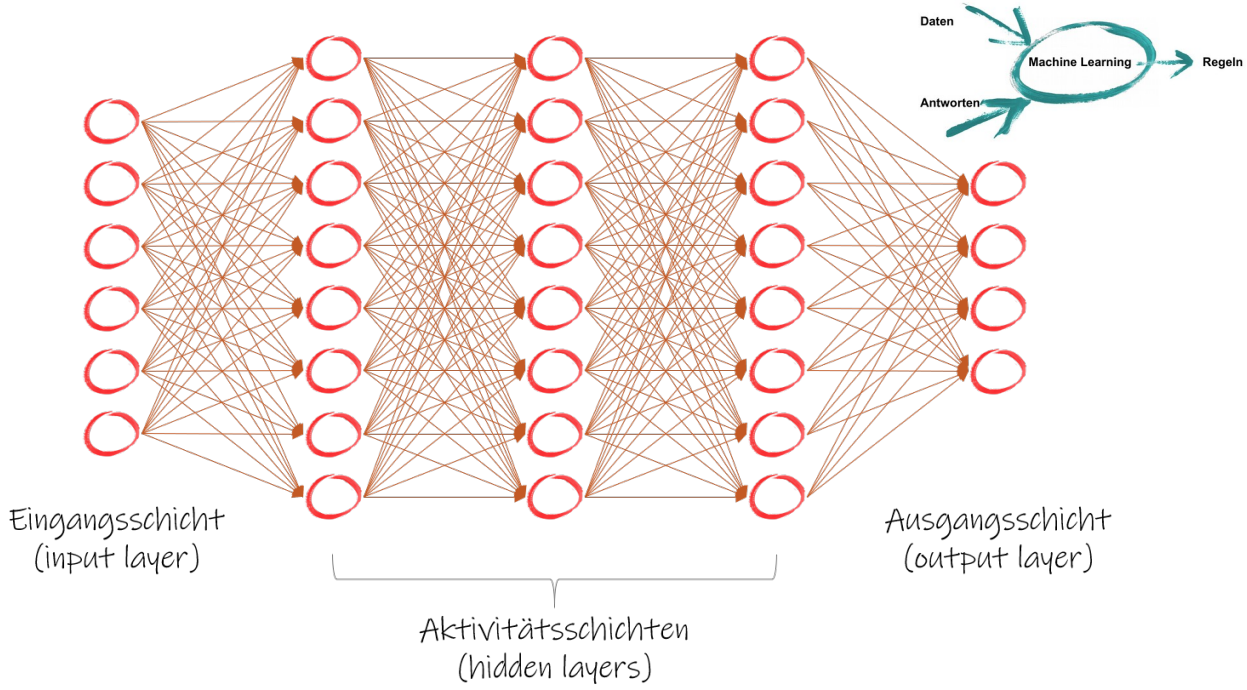
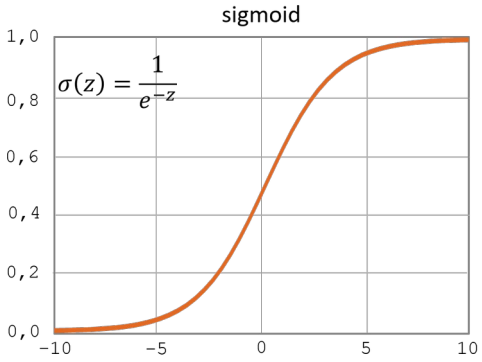
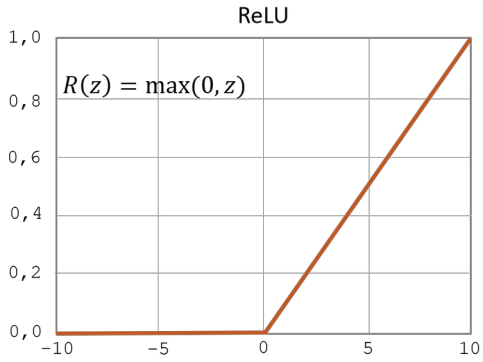


Abb. 3.: Neuronales Netzwerk mit mehreren Ebenen und vielen Neuronen.

Glossar

- *accuracy*: Anteil der richtigen Vorhersagen
- *activation function* (Aktivierungsfunktion auch Schwellenwertfunktion z. B. Sigmoid, ReLu, tanh): Art und Weise wie ein Neuron aufgrund bestimmter Eingaben feuert.
- *epoch* (Epoche): Wiederholtes trainieren derselben Trainingsmuster.
- *weights* (Gewichte): Stärken oder schwächen Verbindungen zwischen Knoten.
- *learning rate* (Lernrate): Höhe der Anpassung der Gewichte.
- *loss value*: Abstand zum globalen Minimum.
- *overfitting*: Netzwerk lernt alle Trainingsmuster auswendig.
- *overshooting*: Bei zu hoher Lernrate kann das Fehlerminimum nicht erreicht werden.
- ReLu (Rectified Linear Unit): Funktion kommt dem Feuern eines natürlichen Neurons am nächsten.






```
1 import tensorflow as tf
2 import numpy as np
3
4 # Eingabemuster
5 inputMuster = np.array([[0,0],[0,1],[1,0],[1,1]])
6 print ("Daten: ")
7 print (inputMuster)
8
9 # Ausgabemuster: AND
10outputMuster = np.array([[0],[0],[0],[1]])
11print ("Antwort: ")
12print (outputMuster)
13
14# Model erstellen
15model = tf.keras.models.Sequential()
16model.add(tf.keras.layers.Dense(32, input_dim=2,
  activation=tf.nn.relu))
17model.add(tf.keras.layers.Dense(1, activation=tf.nn.sigmoid))
18
19# Model compilieren
20model.compile(loss='mean_squared_error', optimizer='Adam')
21
22# Model für n Epochen trainieren
23epochenAnzahl = 500
24model.fit( x = inputMuster, y = outputMuster, epochs =
  epochenAnzahl, verbose = 0)
25
26# Model testen
27print("Durch tensorflow erreichte Mustererkennung:")
28print(model.predict(inputMuster))
```

Aufgaben

1. Starte Spyder und verschaffe Dir einen Überblick.
2. Beschreibe das Ergebnis welches das neuronale Netzwerk liefert.
3. Erkläre warum das Ergebnis nicht gleich dem des `outputMuster` (Zeile 10) ist.
4. Verändere das `outputMuster`, so dass andere Operatoren trainiert werden. z.B. OR, Addition, usw.
5. Verändere die Anzahl der Epochen. Beschreibe / erkläre die Auswirkungen. Ermitlle dazu sinnvolle Ober-/ Untergrenzen.
6. Verändere die Anzahl der Knoten in Zeile 16. Beschreibe die Auswirkungen auf das Ergebnis und erkläre die Veränderungen.
7. Ermittle wo eine sinnvolle Unter- / Obergrenzefür die Knotenanzahl liegt.
8. Überlege welche Auswirkungen eine zu große Knotenanzahl bei großen Datensätzen haben kann.
9. Das Ergebnis soll so aufbereitet werden, dass nur noch 0 und 1 darin vorkommen.
10. Bewerte die Bedeutung der Daten für Machine Learning. Vergleiche mit der klassischen algorithmischen Prorammierung.
11. Vergleiche wie bei der klassischen algorithmischen Programmierung und Machine Learning das Expertenwissen in die Problemlösung einfließt.