Künstliche Intelligenz selbst gemacht

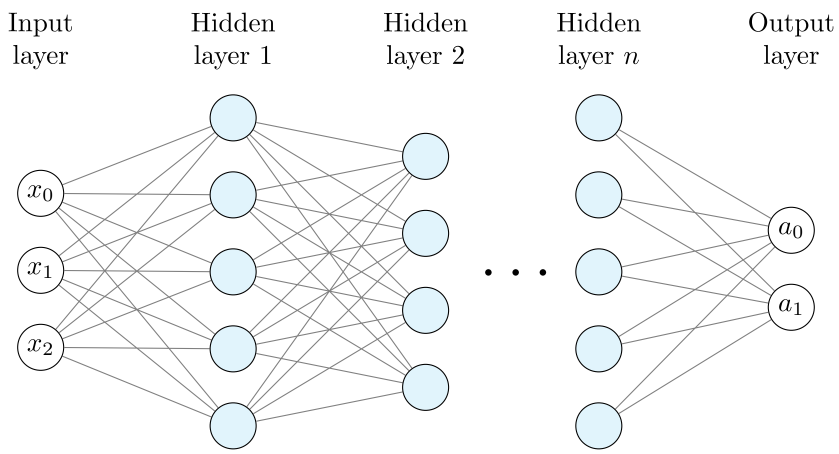
Ein neuronales Netz mit Python entwickeln

Ein neuronales Netz besteht aus **Neuronen** und **Gewichten**.

Die Neuronen sind in Schichten („**Layer**") organisiert.

Jedes Neuron eines Lasers ist mit jedem Neuron des nächsten Lasers verbunden.

Supervised Learning: Das Netz wird mit vielen Beispielen „trainiert", es findet dabei möglichst gute Werte für die Gewichte.



Python- und numpy-Grundlagen

Einer Variable einen Wert zuweisen:

```
i = 5
name = “ChatGPT”
```

Kommentar in den Quellcode einfügen:

```
# Dieser Text wird von Python ignoriert
```

Werte auf der Konsole ausgeben:

```
print(“Hello world”)       # Einen String (Zeichenkette) ausgeben
print(i)        # Wert einer Variable ausgeben
print(“Wert von i = ”, i)       # Kombination aus String und Variable
```

Funktion definieren und aufrufen:

```
def add_numbers(a, b):
  return a + b

sum = add_numbers(3, 7)
print(sum)      # 10
```

Bedingte Ausführung von Anweisungen:

```
if age < 18:
  print(“Du bist zu jung!”)
else:
  print(“Bitte treten Sie ein.”)
```

Wiederholte Ausführung von Anweisungen:

```
for i in range(10):
  print(i)      # 0 1 2 3 4 5 6 7 8 9
```

Anlegen einer Liste und Zugriff auf einzelne Elemente:

```
l = [“a”, “b”, “c”, “d”, “e”, “f”]
print(l[2])     # “c”
l[0] = “z”
print(l)        # [“z”, “b”, “c”, “d”, “e”, “f”]
```

Ausgabe aller Elemente einer Liste:

```
for e in len(list):
  print(e)
```

Importieren von Paketen:

```
import numpy
import long_package_name as lpn
from package import function
```

Klasse mit Konstruktor definieren und instanziieren:

```
class Rectangle():
  def __init__(self, width, height):
     self.width = width
     self.height = height

  def area(self):
     return self.width * self.height

rect1 = Rectangle(4, 3)
print(rect1.area())       # 12
```

Matrix in numpy anlegen:

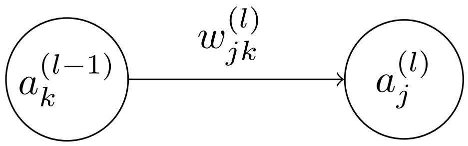
```
mat = numpy.array([
  [1, 2, 3],
  [4, 5, 6]
])

z = numpy.zeros((2, 3))  # 2×3-Matrix mit 0en
r = numpy.random.rand(2, 3)  # 2×3-Matrix mit Zufallszahlen zw. 0 und 1
```

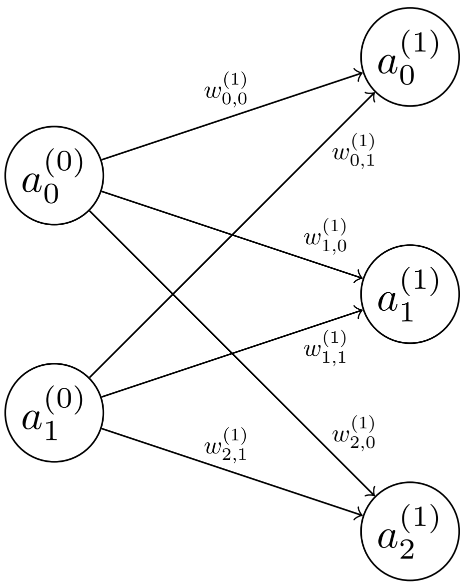
Matrixmultiplikation in numpy:

```
result = numpy.matmul(mat1, mat2)
```

Notationen im neuronalen Netz

x_i	i-ter Eintrag im Inputvektor des Trainingsdatensatzes
$a_j^{(l)}$	Aktivierung des j-ten Neurons im Layer l
$b^{(l)}$	Bias im Layer l zum Links-Rechts-Shift der Aktivierungsfunktion
$w_{jk}^{(l)}$	Gewicht vom k-ten Neuron im Layer $l-1$ zum j-ten Neuron im Layer l
$z_j^{(l)}$	Gewichtete Summe aller eingehenden Neuronenaktivierungen im j-ten Neuron im Layer l
$\sigma^{(l)}$	Aktivierungsfunktion im Layer l
y_i	i-ter Eintrag im erwarteten Outputvektor des Trainingsdatensatzes





$$A^{(1)} = \sigma^{(1)}\left(Z^{(1)}\right)$$

$$= \sigma^{(1)}\left(W^{(1)}A^{(0)}\right)$$

$$= \sigma^{(1)}\left[\begin{pmatrix} w_{0,0}^{(1)} & w_{0,1}^{(1)} \\ w_{1,0}^{(1)} & w_{1,1}^{(1)} \\ w_{2,0}^{(1)} & w_{2,1}^{(1)} \end{pmatrix} \begin{pmatrix} a_0^{(0)} \\ a_1^{(0)} \end{pmatrix}\right]$$

Aktivierungsfunktionen

Name	$f(x)$	$f'(x)$ **(1. Ableitung)**
Identität	$id(x) = x$	$id'(x) = 1$
Rectified Linear Unit	$ReLU(x) = \begin{cases} x & \text{wenn } x > 0 \\ 0 & \text{sonst} \end{cases}$	$ReLU'(x) = \begin{cases} 1 & \text{wenn } x > 0 \\ 0 & \text{sonst} \end{cases}$
Sigmoid	$\sigma(x) = \dfrac{1}{1 + e^{-x}}$	$\sigma'(x) = \sigma(x) \cdot (1 - \sigma(x))$

Die Kostenfunktion und ihre Ableitung

Squared Error Loss: $C = (A^{(l)} - Y)^2$

Ableitung:

$$\frac{\partial C}{\partial w_{jk}^{(l)}} = \frac{\partial C}{\partial a_j^{(l)}} \frac{\partial a_j^{(l)}}{\partial z_j^{(l)}} \frac{\partial z_j^{(l)}}{\partial w_{jk}^{(l)}} \tag{1}$$

$$\frac{\partial z_j^{(l)}}{\partial w_{jk}^{(l)}} = \frac{\partial w_{jk}^{(l)} a_k^{(l-1)}}{\partial w_{jk}^{(l)}} = a_k^{(l-1)} \tag{2}$$

$$\frac{\partial a_j^{(l)}}{\partial z_j^{(l)}} = \sigma^{(l)'}(z_j^{(l)}) \tag{3}$$

$$\frac{\partial C}{\partial a_j^{(l)}} = \frac{1}{2}\left(a_j^{(l)} - y_j\right) \tag{4}$$

$$\frac{\partial C}{\partial a_j^{(l)}} = \sum_{i=0}^{n^{(l+1)}-1} \left(\frac{\partial C}{\partial a_i^{(l+1)}} \frac{\partial a_i^{(l+1)}}{\partial z_i^{(l+1)}} w_{ij}^{(l+1)} \right) \tag{5}$$

$$\frac{\partial C}{\partial w_{jk}^{(l)}} = \frac{\partial C}{\partial a_j^{(l)}} \frac{\partial a_j^{(l)}}{\partial z_j^{(l)}} \cdot a_k^{(l-1)} = \delta_j^{(l)} \cdot a_k^{(l-1)} \tag{6}$$

$$\delta_j^{(l)} = \begin{cases} \frac{1}{2}\left(a_j^{(l)} - y_j\right) \cdot \sigma^{(l)'}(z_j^{(l)}) & \text{für den Output-Layer} \\ \left(\sum_{i=0}^{n^{(l+1)}-1} \delta_i^{(l+1)} w_{ij}^{(l+1)}\right) \cdot \sigma^{(l)'}(z_j^{(l)}) & \text{für innere Layer} \end{cases} \tag{7}$$

Anpassung der Gewichte

$$\Delta w_{jk}^{(l)} = -\eta \cdot \frac{\partial C}{\partial w_{jk}^{(l)}} = -\eta \cdot \delta_j^{(l)} \cdot a_k^{(l-1)} \tag{8}$$