Künstliche Intelligenz kapieren und programmieren

Teil 7: Ziffern erkennen

Michael Weigend
Universität Münster

mw@creative-informatics.de
www.creative-informatics.de
2024

Materialien bei GitHub:
https://github.com/mweigend/ki-workshop

Tag 2

Zeit	Thema	Inhalte
9.00	Perzeptron	Neuron, Aktivierungsfunktion, Daten visualisieren mit Matplotlib, Rosenblatt-Perzeptron für logische Operationen
11.00	Aus Fehlern lernen	Error-Backpropagation, einfaches künstliches neuronales Netz (KNN) mit verborgenen Knoten
12.30	*Mittagspause*	
13.30	**Ziffern erkennen**	**NumPy, KNN mit Array-Operationen, das Ziffern erkennen kann**
15.00	Anwendung von KI	Verkehrsschilder erkennen, Gesichter erfassen, Experimente mit OpenCV
16.00	*Ende*	

Das Ziel

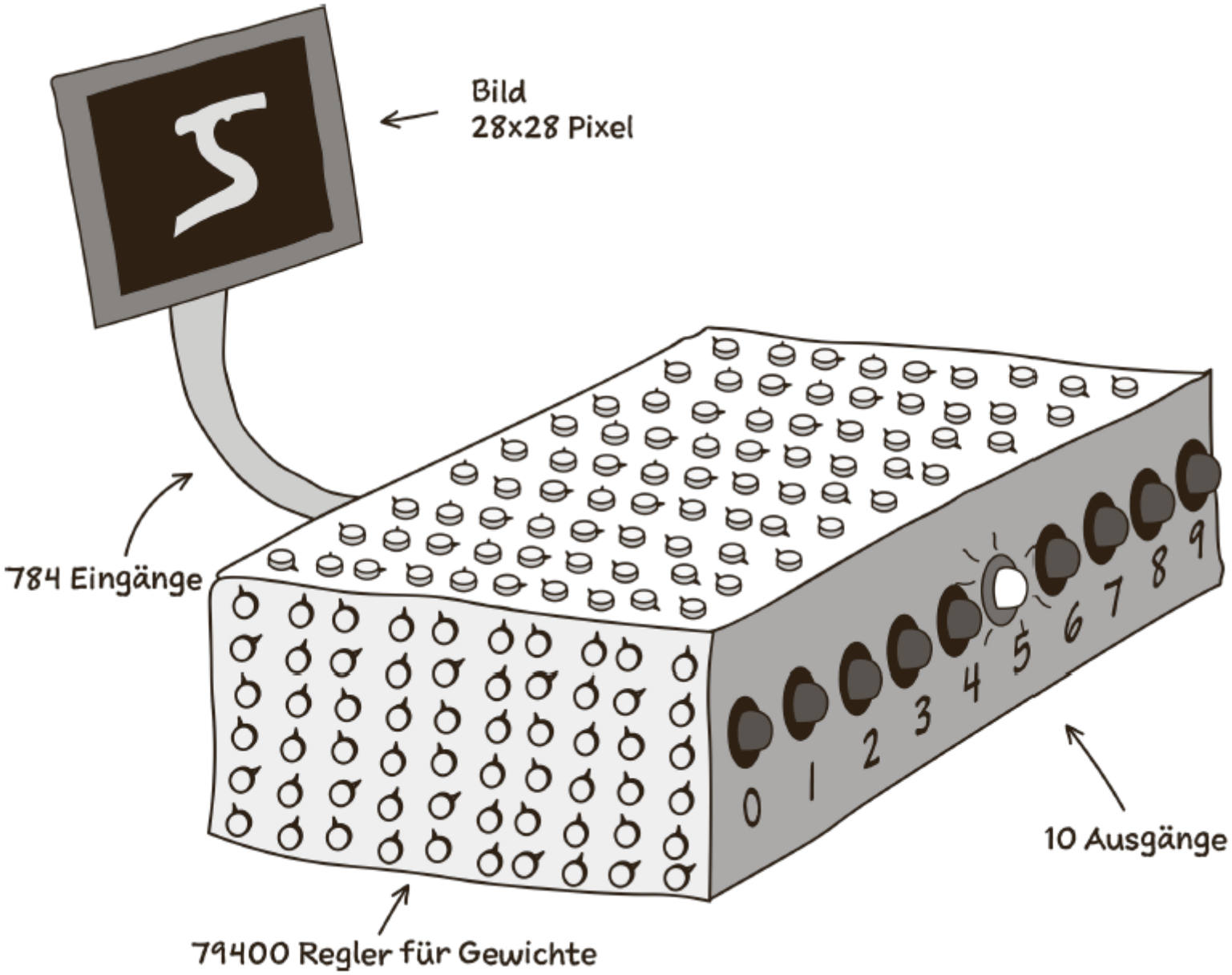


Bild
28x28 Pixel
784 Eingänge
79400 Regler für Gewichte
0
1
2
3
4
5
6
7
8
9
10 Ausgänge

Blick zurück: XOR-Detektor

Viele Variablen
aber nur wenige Typen von Variablen

Gewichte zwischen Eingabeschicht und verborgener Schicht

Gewichte zwischen verborgener Schicht und Ausgabeschicht

```
from random import uniform, shuffle
from math import e
W = 0.5
LR = 0.2

# Initialisierung der Gewichte
wi1h1 = uniform(-W, W)
wi2h1 = uniform(-W, W)
wi1h2 = uniform(-W, W)
wi2h2 = uniform(-W, W)
wi1h3 = uniform(-W, W)
wi2h3 = uniform(-W, W)
wh1o1 = uniform(-W, W)
wh2o1 = uniform(-W, W)
wh3o1 = uniform(-W, W)
wh1o2 = uniform(-W, W)
wh2o2 = uniform(-W, W)
wh3o2 = uniform(-W, W)
```

Die Rettung: Arrays

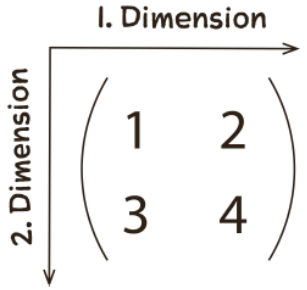
$$wih = \begin{pmatrix} wih_{11} & wih_{21} \\ wih_{12} & wih_{22} \\ wih_{13} & wih_{23} \end{pmatrix}$$

```
wih = np.random.rand(3, 2) - 0.5
who = np.random.rand(2, 3) - 0.5
```

```
from random import uniform, shuffle
from math import e
W = 0.5
LR = 0.2

# Initialisierung der Gewichte
wi1h1 = uniform(-W, W)
wi2h1 = uniform(-W, W)
wi1h2 = uniform(-W, W)
wi2h2 = uniform(-W, W)
wi1h3 = uniform(-W, W)
wi2h3 = uniform(-W, W)
wh1o1 = uniform(-W, W)
wh2o1 = uniform(-W, W)
wh3o1 = uniform(-W, W)
wh1o2 = uniform(-W, W)
wh2o2 = uniform(-W, W)
wh3o2 = uniform(-W, W)
```

7.1 Trainingscamp: Mit Numpy Arrays verarbeiten



Matrix

```
[[1 2]      [['a' 'b']
 [3 4]]      ['a' 'b']]
```

rechteckige Anordnung von Werten

Unterschied zwischen Matrix und Array?

Station 1: Arrays erzeugen

```
>>> import numpy as np
>>> a = np.array([1, 2, 3])
>>> a
array([1, 2, 3])
>>> print(a)
[1 2 3]

>>> np.array([1, 1.5, 2])
array([1. , 1.5, 2. ])
>>> array([1. , 1.5, 2. ])

b = np.array(['1', '2'], dtype=int)
>>> print(b)
[1 2]

>>> b = np.array([[1, 2],[3, 4]])
>>> print(b)
[[1 2]
 [3 4]]
>>> np.ndim(b)
2
```

Keine Kommas

Nur ein Datentyp

Typ explizit festlegen

Zweidimensionale Matrix (Mathematik)

Vektoren

Zeilenvektor: (1 2 3)

```
import numpy as np
zeilenvektor = np.array([[1, 2, 3]])
print(zeilenvektor)
```

```
[[1 2 3]]
```

Spaltenvektor: $\begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$

```
spaltenvektor = np.array([[1], [2], [3]])
print(spaltenvektor)
```

```
[[1]
 [2]
 [3]]
```

Vektoren werden durch zweidimensionale Arrays dargestellt.

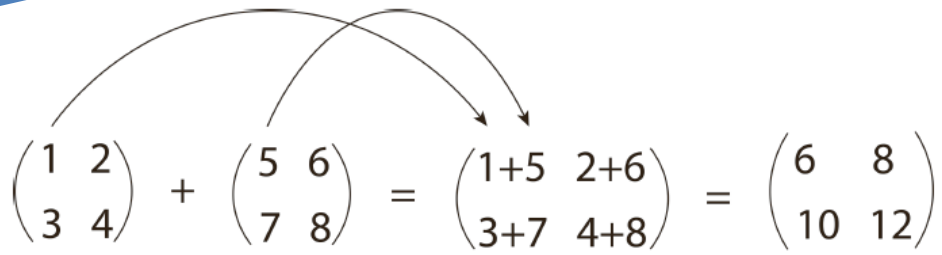
Station 2: Operationen mit Arrays und Zahlen

```
>>> import numpy as np
>>> a = np.array([1, 2, 3])
>>> a + 1
array([2, 3, 4])
>>> print(a * 2)
[2 4 6]
>>> print(a**2)
[1 4 9]
print(1/a)
[1.         0.5        0.33333333]
```

Station 3: Operationen mit zwei Arrays

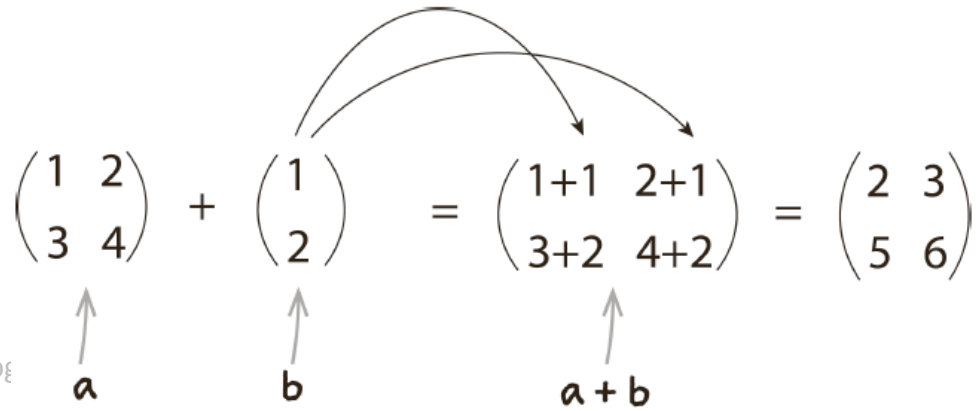
```
>>> import numpy as np
>>> a = np.array([[1, 2], [3, 4]])
>>> b = np.array([[5, 6],[7, 8]])
>>> print(a + b)
[[ 6 8]
[10 12]]
```

gleiche Formem



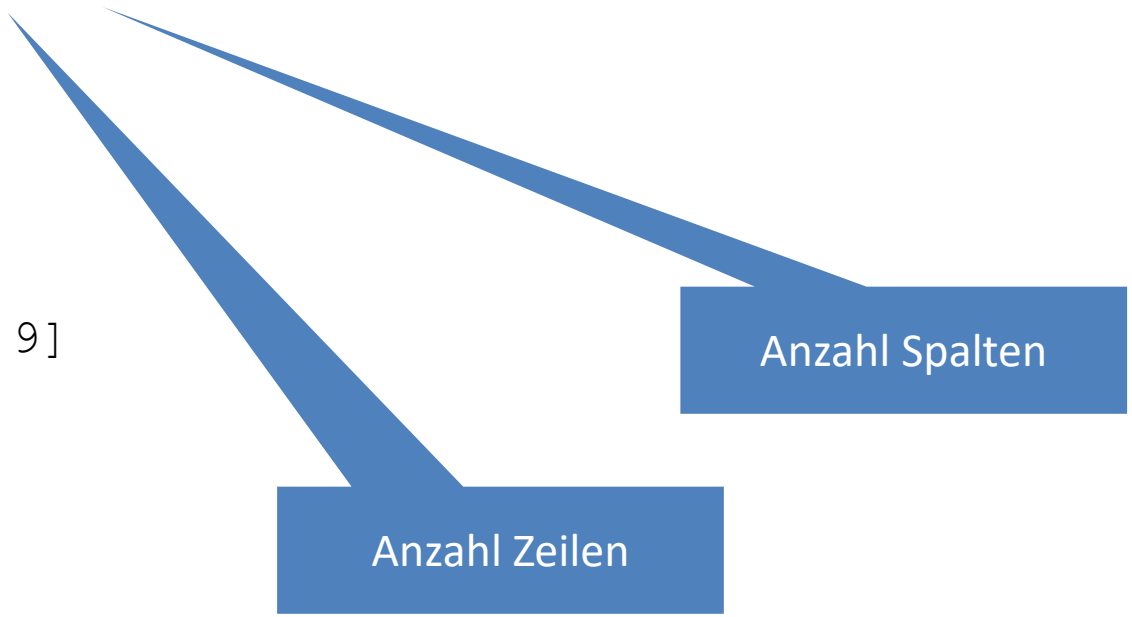

```
>>> a = np.array([[1, 2], [12, 2]])
>>> b = np.array([[1], [2]])
>>> print(a + b)
[[2 3]
[5 6]]
```

passende Formem

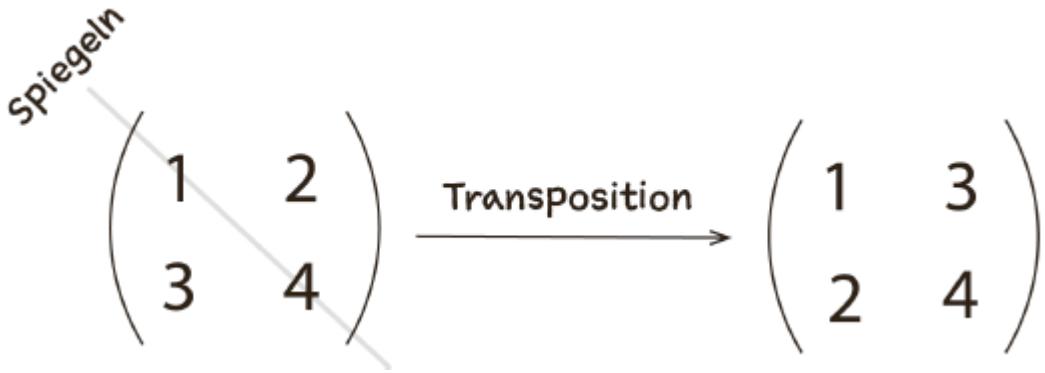


Station 4: Die Form eines Arrays verändern

```
>>> import numpy as np
>>> a = np.arange(10)
>>> print(a)
[0 1 2 3 4 5 6 7 8 9]
>>> b = a.reshape(2, 5)
>>> print(b)
[[0 1 2 3 4]
[5 6 7 8 9]]
>>> c = b.ravel()
>>> print(c)
[0 1 2 3 4 5 6 7 8 9]
```



Station 4: Die Form eines Arrays verändern




```
>>> a = np.array([[1, 2], [3, 4]])
>>> print(a)
[[1 2]
 [3 4]]
>>> print(a.T)
[[1 3]
 [2 4]]
```

Transponiertes Array

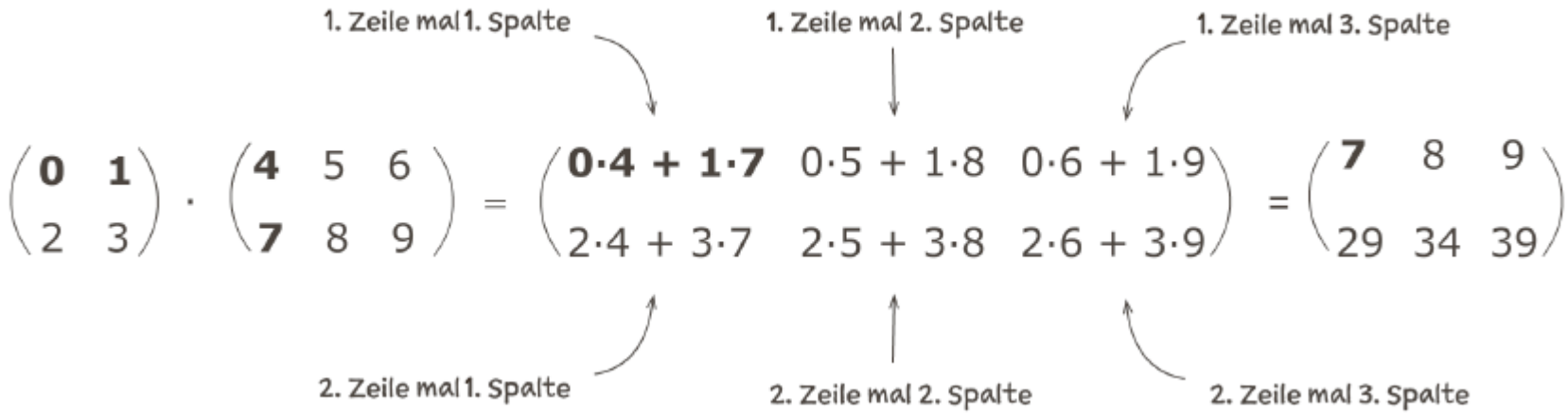
Station 4: Die Form eines Arrays verändern

Zeilenvektor

```
>>> a = np.array([1, 2], ndmin=2)
>>> print(a)
[[1 2]]
>>> print(a.T)
[[1]
[2]]
```

Zweidimensional (minimum number of dimensions)

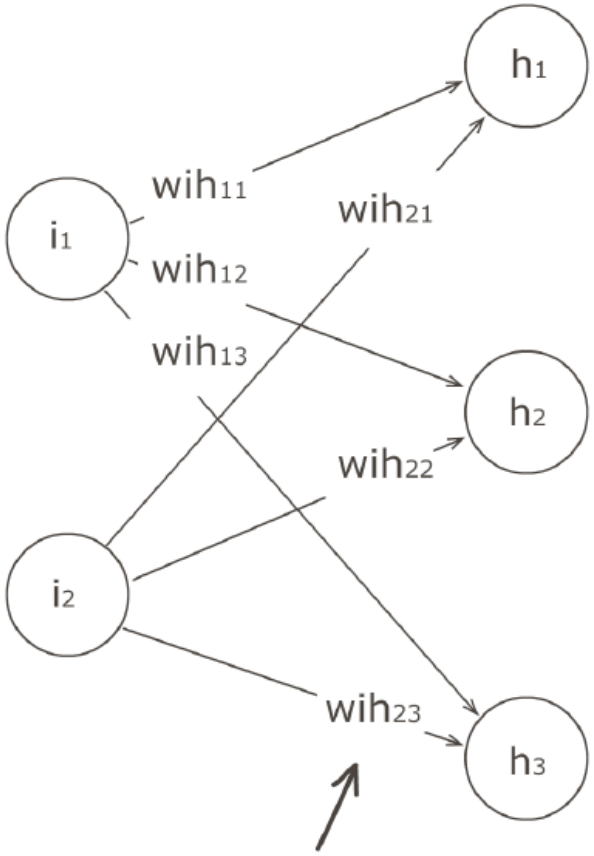
Station 5: Matrizenmultiplikation mit dot()



„Zeile mal Spalte“

```
>>> a = np.array([[0, 1], [2, 3]])
>>> b = np.array([[4, 5, 6], [7, 8, 9]])
>>> c = np.dot(a, b)
>>> print(c)
[[ 7  8  9]
[29 34 39]]
```

Beispiel: Berechnung der gewichteten Eingabe x



Gewichte zwischen Eingabeknoten und verborgenen Knoten

$$\begin{pmatrix} wih_{11} & wih_{21} \\ wih_{12} & wih_{22} \\ wih_{13} & wih_{23} \end{pmatrix} \cdot \begin{pmatrix} i_1 \\ i_2 \end{pmatrix} = \begin{pmatrix} wih_{11} \cdot i_1 + wih_{21} \cdot i_2 \\ wih_{12} \cdot i_1 + wih_{22} \cdot i_2 \\ wih_{13} \cdot i_1 + wih_{23} \cdot i_2 \end{pmatrix} = \begin{pmatrix} xh_1 \\ xh_2 \\ xh_3 \end{pmatrix}$$

Verwechslungsgefahr!

```
wih = np.array([[wih11, wih21],
                [wih12, wih22],
                [wih13, wih23]])
i = np.array([[i1], [i2]])
xh = np.dot(wih, i)
```

Achtung! Verwechslungsgefahr!

$$\begin{pmatrix} wih_{11} & wih_{21} \\ wih_{12} & wih_{22} \\ wih_{13} & wih_{23} \end{pmatrix} \cdot \begin{pmatrix} i_1 \\ i_2 \end{pmatrix} = \begin{pmatrix} wih_{11} \cdot i_1 + wih_{21} \cdot i_2 \\ wih_{12} \cdot i_1 + wih_{22} \cdot i_2 \\ wih_{13} \cdot i_1 + wih_{23} \cdot i_2 \end{pmatrix} = \begin{pmatrix} xh_1 \\ xh_2 \\ xh_3 \end{pmatrix}$$

Nicht Index des Array-Elements!

Station 6: Zufallsarrays

`random.rand()` liefert Zufallszahlen zwischen 0 und 1 in Gleichverteilung

```
>>> import numpy as np
>>> a = np.random.rand(3, 4)
>>> print(a)
[[0.41965252 0.29836361 0.57745735 0.86615516]
[0.53214169 0.58787476 0.10119548 0.44031523]
[0.35768981 0.0679965 0.85630156 0.49519094]]
```

Array mit Zufallszahlen zwischen 0 und 1000:

```
>>> a = np.round(np.random.rand(2, 2)*1000)
>>> print(a)
[[615. 629.]
[769. 143.]]
```

Station 7: Elemente eines Arrays verarbeiten

Funktion	Erklärung
`average(a)`	Liefert den Mittelwert aller Zahlen im Array *a*.
`max(a)`	Liefert die größte Zahl im Array *a*.
`mean(a)`	Liefert den Median aller Zahlen im Array *a*.
`min(a)`	Liefert die kleinste Zahl im Array *a*.
`sum(a)`	Liefert die Summe aller Zahlen im Array *a*.

Station 8: Auf Elemente eines Arrays zugreifen

```
>>> import numpy as np
>>> a = np.array([2, 56, 7])
>>> a[0]
2

>>> np.argmax(a)
1

>>> b = np.array([[1, 2], [3, 4]])
>>> print(b)
[[1 2]
[3 4]]
>>> b[1,0]
3
```

Element mit Index 0

Index des größten Elements

Element in Zeile 1 und Spalte 0

Übung 7.1

Aufgabe 1

Schreiben Sie Anweisungen, die die beiden Vektoren und die Matrix als Arrays darstellen.

$$\begin{pmatrix} 1 \\ 2 \end{pmatrix} \quad \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 & 2 & 3 & 4 \end{pmatrix}$$

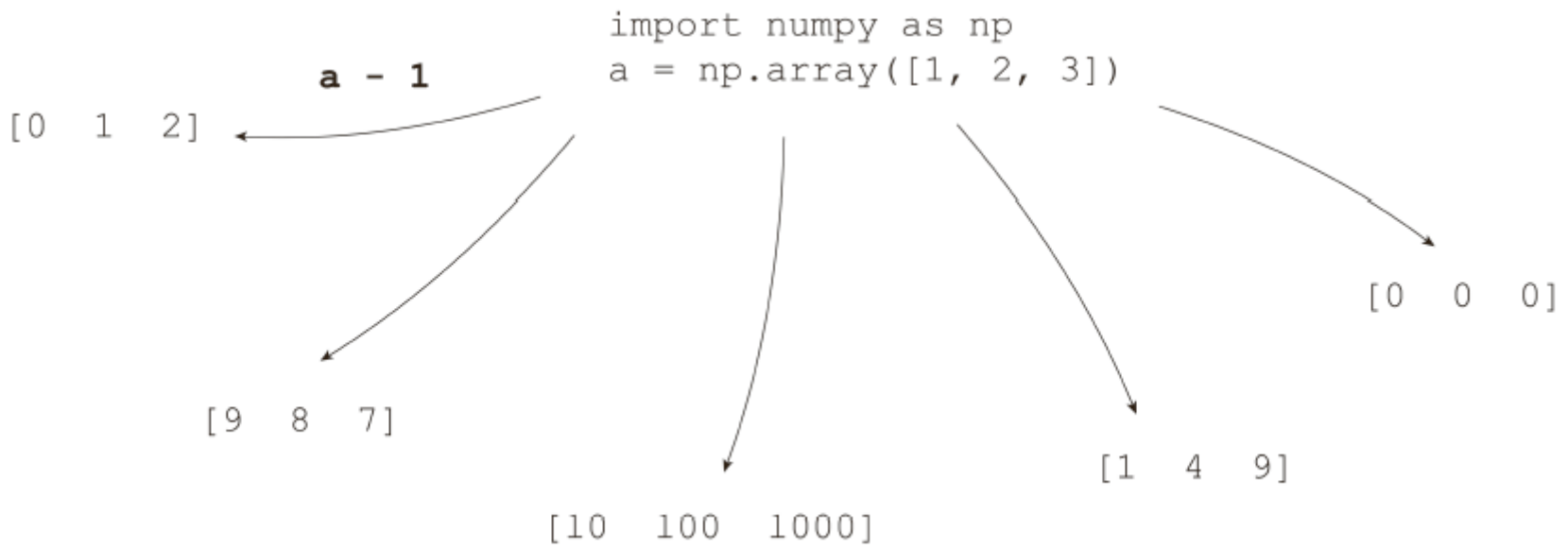
Aufgabe 2

Erzeugen Sie dieses Array mit 16 Zweierpotenzen
auf möglichst einfache Weise:

```
[[ 1    2    4     8    ]
 [ 16   32   64    128  ]
 [ 256  512  1024  2048 ]
 [ 4096 8192 16384 32768]]
```

Aufgabe3

Aus dem Array a sollen neue Arrays berechnet werden. Schreiben Sie passende Anweisungen auf.



Aufgabe 4 (Bleistift und Papier)

Welche Ergebnisse liefern die folgenden Matrizenmultiplikationen? Das Rechnen ist hier besonders einfach. Es kommt hier vor allem auf die Form der Produktmatrix an.

$$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} =$$

$$\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \cdot \begin{pmatrix} 0 & 0 & 0 \end{pmatrix} =$$

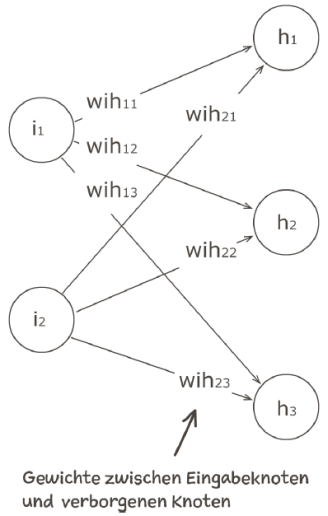
$$\begin{pmatrix} 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} =$$

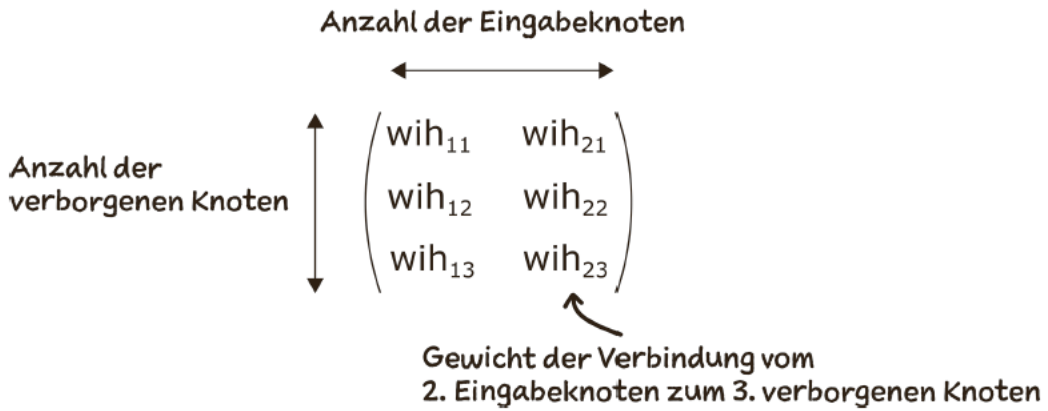
7.2 Mit Arrays die Programmierung vereinfachen: XOR-Detektor (1)

```
# nn_xor.py
import numpy as np
from math import e
from random import shuffle
LR = 0.2

def sig(x):
    return 1 / (1 + e**-x)

# Initialisierung der Gewichte
wih = np.random.rand(3, 2) - 0.5
who = np.random.rand(2, 3) - 0.5
```





XOR-Detektor (2)

$$\begin{pmatrix} xh_1 \\ xh_2 \\ xh_3 \end{pmatrix} = \begin{pmatrix} wih_{11} & wih_{21} \\ wih_{12} & wih_{22} \\ wih_{13} & wih_{23} \end{pmatrix} \cdot \begin{pmatrix} i_1 \\ i_2 \end{pmatrix} = \begin{pmatrix} wih_{11} \cdot i_1 + wih_{21} \cdot i_2 \\ wih_{12} \cdot i_1 + wih_{22} \cdot i_2 \\ wih_{13} \cdot i_1 + wih_{23} \cdot i_2 \end{pmatrix}$$

```
def vorhersagen(i):
    xh = np.dot(wih, i)
    yh = sig(xh)
    xo = np.dot(who, yh)
    o = sig(xo)
    return o
```

$$\begin{pmatrix} yh_1 \\ yh_2 \\ yh_3 \end{pmatrix} = sig\left(\begin{pmatrix} xh_1 \\ xh_2 \\ xh_3 \end{pmatrix}\right) = \begin{pmatrix} sig(xh_1) \\ sig(xh_2) \\ sig(xh_3) \end{pmatrix}$$

Ausgabe der verborgenen Knoten

Eingabe der verborgenen Knoten

$$\begin{pmatrix} xo_1 \\ xo_2 \end{pmatrix} = \begin{pmatrix} who_{11} & who_{21} & who_{31} \\ who_{12} & who_{22} & who_{32} \end{pmatrix} \cdot \begin{pmatrix} yh_1 \\ yh_2 \\ yh_3 \end{pmatrix} = \begin{pmatrix} who_{11} \cdot yh_1 + who_{21} \cdot yh_2 + who_{31} \cdot yh_3 \\ who_{12} \cdot yh_1 + who_{22} \cdot yh_2 + who_{32} \cdot yh_3 \end{pmatrix}$$

XOR-Detektor(3)

Arrays (Spaltenvektoren)

```
def trainieren(i, t):
    global wih, who
    # Berechnung der Ausgabe (Vorhersage)
    xh = np.dot(wih, i)
    yh = sig(xh)
    xo = np.dot(who, yh)
    o = sig(xo)
    # Aktualisierung der Gewichte
    eo = t - o
    who += LR * np.dot((eo * o * (1.0 - o)), yh.T)
    eh = np.dot(who.T, eo)
    wih += LR * np.dot((eh * yh * (1.0 - yh)), i.T)
    return eo
```

Fehler der Vorhersage wird für Testzwecke zurückgegeben

XOR-Detektor(4)

i1	i2	t1	t2
0	0	0	1
0	1	1	0
1	0	1	0
1	1	0	1

```
# Zufällige Trainingsdaten erzeugen
d =[(0, 0, 0, 1), (0, 1, 1, 0), (1, 0, 1, 0), (1, 1, 0, 1)]
daten = 2000 * d
shuffle(daten)

# Training

for ep in range(10):
    summeFehlerQuadrate = 0
    for i1, i2, t1, t2 in daten:
        i = np.array([i1, i2],ndmin=2).T
        t = np.array([t1, t2], ndmin=2).T
        eo = trainieren(i, t)
        summeFehlerQuadrate += np.sum(eo**2)
    mFehlerQuadrate = summeFehlerQuadrate / len(daten)
    print('Epoche:',ep, 'mittlere Fehlerquadratsumme:',
          mFehlerQuadrate)
```

Spaltenvektoren

XOR-Detektor(4)

Die 4 Zeilen der Trainingstabelle werden einmal durchlaufen

```
# Testen

for i1, i2, t1, t2 in d:
    i = np.array([i1, i2],ndmin=2).T
    t = np.array([t1, t2], ndmin=2).T
    o = vorhersagen(i)
    o1 = round(o[0,0], 4)
    o2 = round(o[1,0], 4)

    print('Eingabe:', i1, i2, 'Vorhersage: ', round(o1, 4),
round(o2, 4),
          'Target:', t1, t2)
```

Übung 7.2 (5 min)

1. Testen Sie das Programm. Vergleichen Sie die Rechenzeit dieses Programms mit der Rechenzeit der Version ohne Arrays.
2. Wie viele Lernschritte macht das Programm beim Training?

3 Projekt „Ziffern erkennen“

MNIST-Datenbank

- **M**odifizierte Datenbank des **N**ational **I**nstitute of **S**tandards and **T**echnology (NIST)
- Handgeschriebene Ziffern
- 28 mal 28 Pixel
- 60.000 Datensätze zum Training
- 10.000 Datensätze zum Testen.

Die modifizierte DB sind csv-Dateien von Joseph Redmon.
Wir verwenden (zunächst) kleinere Datensätze zum Testen!
mnist10_train.csv und mnist10_train.csv (im GitHub)

Mehr als 100 MB!

Trainingsdaten http://www.pjreddie.com/media/files/mnist_train.csv
Testdaten: http://www.pjreddie.com/media/files/mnist_test.csv

Andere Speicherorte der MNIST-Daten

https://github.com/phoebetronic/mnist/blob/main/mnist_train.csv.zip
https://github.com/phoebetronic/mnist/blob/main/mnist_test.csv.zip

CSV-Datei

Nur eine Zeile = ein Trainingsdatensatz

Target

mnist10_train.csv - Editor

Datei Bearbeiten Format Ansicht Hilfe

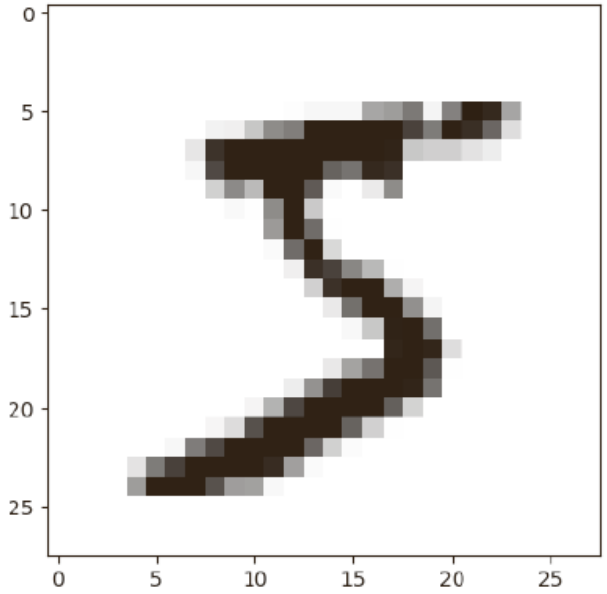
```
5,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,18,18,18,126,136,175,26,166,255,247,127,0,0,0,0,0,0,0,0,
0,0,0,0,30,36,94,154,170,253,253,253,253,253,225,172,253,242,195,64,0,0,0,0,0,0,0,0,0,0,0,4
9,238,253,253,253,253,253,253,253,253,251,93,82,82,56,39,0,0,0,0,0,0,0,0,0,0,0,0,18,219,253
,253,253,253,253,198,182,247,241,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,80,156,107,253,253,205
,11,0,43,154,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,14,1,154,253,90,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,139,253,190,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,11,
190,253,70,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,35,241,225,160,108,1,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,81,240,253,253,119,25,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,45,186,253,253,150,27,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,16,93,252
,253,187,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,249,253,249,64,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,46,130,183,253,253,207,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,39,
148,229,253,253,253,250,182,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,24,114,221,253,253,253,253,
201,78,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,23,66,213,253,253,253,253,198,81,2,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,18,171,219,253,253,253,253,195,80,9,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,55,172
,226,253,253,253,253,244,133,11,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,136,253,253,253,212,135
,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
```

Zeile 1, Spalte 1846 100% Windows (CRLF) UTF-8

Ordentlich ausgedruckt und ohne Target (5)

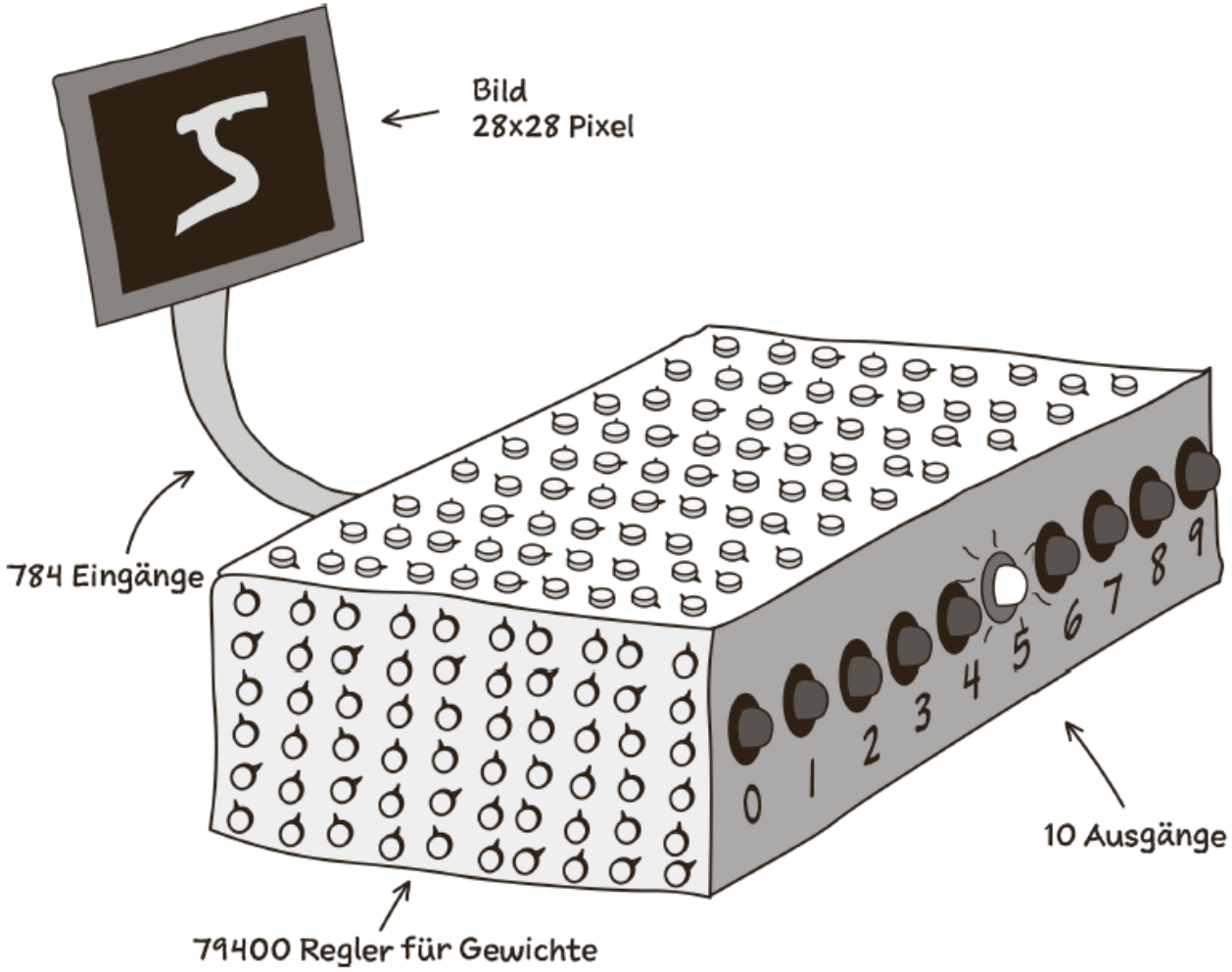
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	3	18	18	18	126	136	175	26	166	255	247	127	0	0	0	0
0	0	0	0	0	0	0	0	30	36	94	154	170	253	253	253	253	253	225	172	253	242	195	64	0	0	0	0
0	0	0	0	0	0	0	49	238	253	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	18	219	253	253	253	253	253	198	182	247	241	0	0	0	0	0	0	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	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	139	253	190	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	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	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	81	240	253	253	119	25	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	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	249	253	249	64	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	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	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	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	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	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	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

Eine Ziffer auf dem Bildschirm darstellen

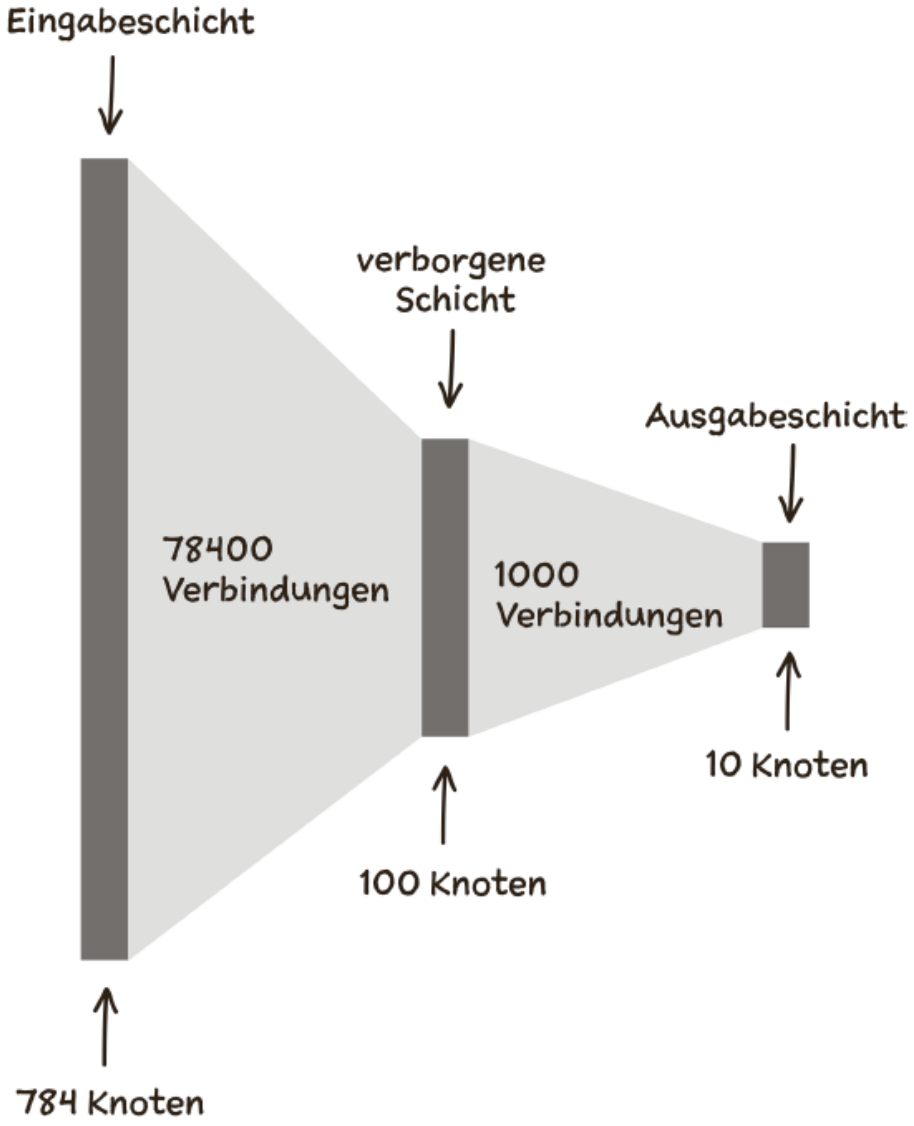



```
# zeige_ziffer.py
import numpy as np
import matplotlib.pyplot as plt
stream = open('daten/mnist10_train.csv', 'r’)
datenliste = stream.readlines()
stream.close()
datensatz = datenliste[0].split(',')
pixel = datensatz[1:]
bildArray = np.array(pixel, dtype=int).reshape((28, 28))
plt.imshow(bildArray, cmap='Greys')
plt.show()
```

Neuronales Netz



Neuronales Netz



Was soll das Programm leisten?

- Training des neuronalen Netzes mit 60000 MNIST-Datensätzen mit Ziffern
- Test des trainierten neuronalen Netzes mit 10000 MNIST-Test-Datensätzen
- Ausgabe: Trefferquote bei 10 000 Tests

```
Trefferquote: 94.895 %
```

Programmierung (1)

```
import numpy as np
import math

EPOCHEN = 1        # Anzahl Trainingsdurchläufe
LR = 0.1           # Lernrate
I_KNOTEN = 784     # Anzahl Eingabeknoten
H_KNOTEN = 100     # Anzahl verborgene Knoten
O_KNOTEN = 10      # Anzahl Ausgabeknoten
PFAD_TRAINING = 'daten/mnist_train.csv'  # Pfad zu Trainingsdaten
PFAD_TEST = 'daten/mnist_test.csv'       # Pfad zu Testdaten

# Initialisierung des neuronalen Netzes
wih = np.random.rand(H_KNOTEN, I_KNOTEN) - 0.5
who = np.random.rand(O_KNOTEN, H_KNOTEN) - 0.5

def sig(x):
    return 1 / (1 + math.e **-x)
```

Programm (1)

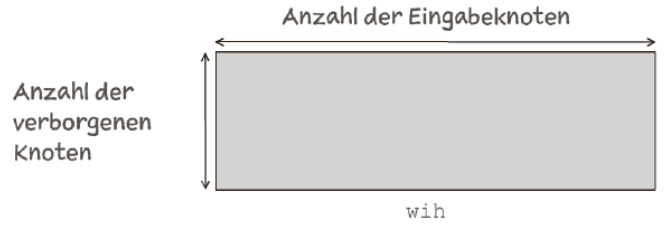
Viele Konstanten
Warum?

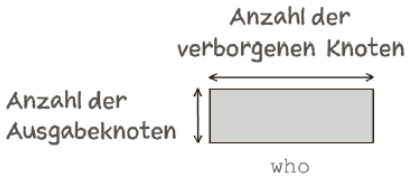
```
import numpy as np
import math

EPOCHEN = 1         # Anzahl Trainingsdurchläufe
LR = 0.1            # Lernrate
I_KNOTEN = 784      # Anzahl Eingabeknoten
H_KNOTEN = 100      # Anzahl verborgene Knoten
O_KNOTEN = 10       # Anzahl Ausgabeknoten
PFAD_TRAINING = 'daten/mnist_train.csv'  # Pfad zu Trainingsdaten
PFAD_TEST = 'daten/mnist_test.csv'       # Pfad zu Testdaten

# Initialisierung des neuronalen Netzes
wih = np.random.rand(H_KNOTEN, I_KNOTEN) - 0.5
who = np.random.rand(O_KNOTEN, H_KNOTEN) - 0.5

def sig(x):
    return 1 / (1 + math.e **-x)
```





Programm (2)

```
['4,0,0, ... 0,0,0,0\n',
 '2,0,0, ... 0,0,0,0\n',
  ...
 '0,0,0, ... 0,0,0,0\n']
```

```
['4', '0', '0', ... '0', '0', '0', '0']
```

```
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

```
[0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
```

```
def datenLesen(pfad):
    """ Etikettierte Daten werden aus der csv-Datei mit dem
        Pfad pfad gelesen. Daraus wird eine Liste mit Tupeln aus
        Numpy-Arrays (i, t) erzeugt und zurückgegeben."""
    stream = open(pfad, 'r')
    datenliste = stream.readlines()
    stream.close()
    daten = []
    for zeile in datenliste:
        zeileListe = zeile.split(',')
        # print(zeileliste)
        # Eingaben aus dem Datensatz extrahieren und skalieren
        eingaben = np.array(zeileListe[1:], dtype=float)
        eingabenSkaliert = (eingaben / 255 * 0.99) + 0.01
        targets = np.zeros(O_KNOTEN)
        targets[int(zeileListe[0])] = 1
        i = np.array(eingabenSkaliert, ndmin=2).T
        t = np.array(targets, ndmin=2).T
        daten.append((i, t))
    return daten
```

Tracing
(nur für Testläufe)

Liste von Tupeln
(Eingabevektor, Targetvektor)

Programm (3)

```
def trainieren(i, t):
    global wih, who
    xh = np.dot(wih, i)
    yh = sig(xh)
    xo = np.dot(who, yh)
    o = sig(xo)
    eo = t - o
    eh = np.dot(who.T, eo)
    who += LR * np.dot((eo * o * (1.0 - o)), yh.T)
    wih += LR * np.dot((eh * yh * (1.0 - yh)), i.T)

def vorhersagen(i):
    xh = np.dot(wih, i)
    yh = sig(xh)
    xo = np.dot(who, yh)
    o = sig(xo)
    return o
```

Programm (4)

```
# Neuronales Netz trainieren
for ep in range(EPOCHEN):
    daten = datenLesen(PFAD_TRAINING)
    for i, t in daten:
        trainieren(i, t)

# Neuronales Netz testen
testbericht = []    # liste aus 0 (falsches Ergebnis) und 1 (richtiges Ergebnis)
testdaten = datenLesen(PFAD_TEST)
for i, t in testdaten:
    o = vorhersagen(i)
    ziffer = np.argmax(o)
    erwarteteZiffer = np.argmax(t)
    if (ziffer == erwarteteZiffer):
        testbericht.append(1)
    else:
        testbericht.append(0)

trefferquote = sum(testbericht) / len(testbericht)
print ('Trefferquote:', trefferquote * 100, '%')
input()
```

```
[[1.27980197e-02]          [[0.],
 [9.45761590e-04]           [0.],
 [4.87761731e-03]           [0.],
 [9.66018536e-04]           [0.],
 [3.20945303e-05]           [0.],
 [5.90642663e-04]           [0.],
 [3.44425547e-04]           [0.],
 [6.96053362e-05]           [0.],
 [9.97581101e-01]           [1.],
 [6.22060274e-02]]          [0.]]
```

0

t

Übung 7.3 Ziffern erkennen

1. Verändern Sie das neuronale Netz und verwenden Sie nun 200 verborgene Knoten. Intensivieren Sie das Training, indem Sie die Anzahl der Epochen auf 2 setzen. Welche Trefferquote (Prozent) erzielt das neuronale Netz nun?

2. Erweitern Sie das Programm um einige Anweisungen und sorgen Sie dafür, dass es seine Arbeitsweise durch weitere Bildschirmausgaben dokumentiert:

- Ausgabe des neuronalen Netzes beim letzten Testlauf (Spaltenvektor `o`) .
- Das erste Item der Liste `testdaten`.

3. Beschreiben Sie, wie man das trainierte neuronale Netz speichern könnte.

Weitere Ideen?

https://docs.google.com/document/d/140lNslEWA_AwUwtpVMCZqtH7_eOkU8rmvfrgWqE5M3E/edit?usp=sharing

Rückblick

- Arrays sind Anordnungen von Zahlen.
- Mit NumPy kann man Arrays verarbeiten.
- Wir haben ein neuronales Netzwerk mit Array-Operationen implementiert. Für die Eingabewerte, die Ausgabewerte, die erwarteten Werte und die Fehler werden Vektoren verwendet. Für die Gewichte werden Matrizen definiert.
- Ein solches Programm kann für das Erkennen von Ziffern auf Graustufenbildern verwendet werden.
- Wir haben die MNIST-Daten mit 70.000 etikettierten Datensätzen zum Trainieren und 10.000 etikettierten Datensätzen zum Testen verwendet.
- Unser neuronales Netz mit 100 verborgenen Knoten hat eine Trefferquote von 95 % erreicht.