Künstliche Intelligenz kapieren und programmieren

Teil 7: Ziffern erkennen



Michael Weigend Universität Münster



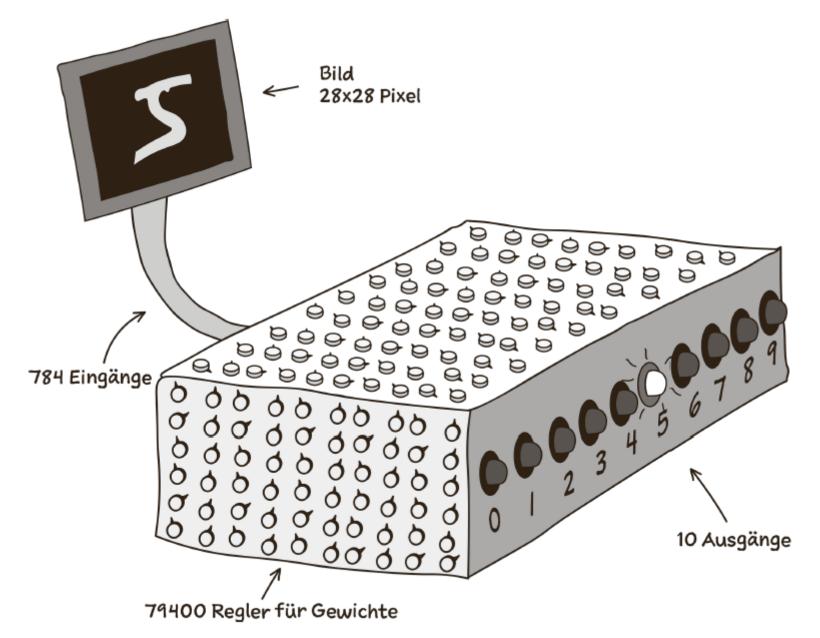
mw@creative-informatics.de www.creative-informatics.de 2024

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



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
wilh1 = uniform(-W, W)
wi2h1 = uniform(-W, W)
wi1h2 = uniform(-W, W)
wi2h2 = uniform(-W, W)
wilh3 = 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

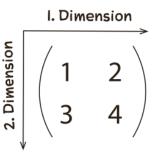
$$wih = \begin{pmatrix} wih_{11} & wih_{21} \\ wih_{12} & wih_{22} \\ wih_{13} & wih_{23} \end{pmatrix}$$

```
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
wilh1 = uniform(-W, W)
wi2h1 = uniform(-W, W)
wilh2 = uniform(-W, W)
wi2h2 = uniform(-W, W)
wilh3 = 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

rechteckige Anordnung von Werten

Unterschied zwischen Matrix und Array?

Station 1: Arrays erzeugen

Keine Kommas

```
>>> import numpy as np
>>> a = np.array([1, 2, 3])
>>> a
array([1, 2, 3])
>>> print(a)
[1 2 3]
                                                     Nur ein Datentyp
>>> np.array([1, 1.5, 2])
array([1., 1.5, 2.])
                                                    Typ explizit festlegen
>>> array([1. , 1.5, 2. ])
b = np.array(['1', '2'], dtype=int)
>>> print(b)
[1 2]
>>> b = np.array([[1, 2], [3, 4]])
                                                    Zweidimensionale
>>> print(b)
                                                   Matrix (Mathematik)
[[1 \ 2]
 [3 4]]
```

>>> np.ndim(b)

Vektoren

Zeilenvektor: (123)

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

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

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

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

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

Vektoren werden durch zweidimensionale Arrays dargestellt.

Station 2: Operationen mit Arrays und Zahlen

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 $\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} + \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} = \begin{pmatrix} 1+5 & 2+6 \\ 3+7 & 4+8 \end{pmatrix} = \begin{pmatrix} 6 & 8 \\ 10 & 12 \end{pmatrix}$

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

 $\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} + \begin{pmatrix} 1 \\ 2 \end{pmatrix} = \begin{pmatrix} 1+1 & 2+1 \\ 3+2 & 4+2 \end{pmatrix} = \begin{pmatrix} 2 & 3 \\ 5 & 6 \end{pmatrix}$

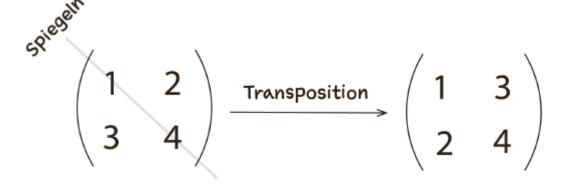
passende Formem

Michael Weigend: Künstliche Intelligenz kapieren und prog

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]
                                                 Anzahl Spalten
                                  Anzahl Zeilen
```

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

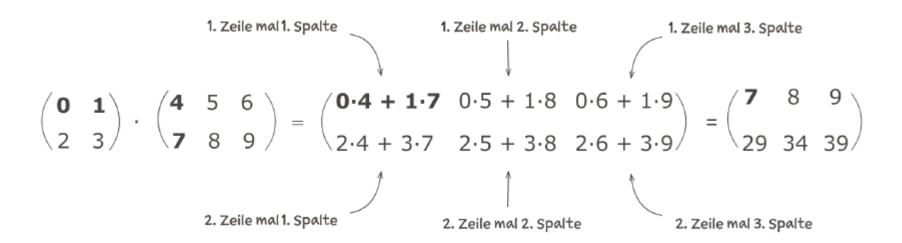
$$(1 \ 2) \xrightarrow{\text{Transposition}} \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

Zeilenvektor

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

Zweidimensional (minimum number of dimensions)

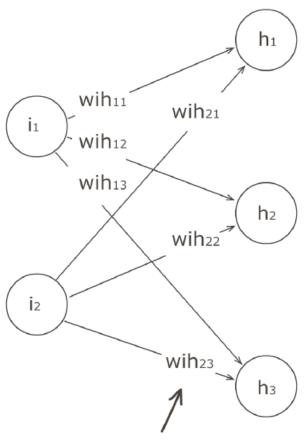
Station 5: Matrizenmultiplikation mit dot()



"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!

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

```
>>> 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.]]
```

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

Station 7: Elemente eines Arrays verarbeiten

Funktion	Erklärung
<pre>average(a)</pre>	Liefert den Mittelwert aller Zahlen im Array a.
max(a)	Liefert die größte Zahl im Array a.
mean(<i>a</i>)	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])
                                         Element mit Index 0
>>> a[0]
                                Index des größten
                                   Elements
>>> np.argmax(a)
>>> b = np.array([[1, 2], [3, 4]])
>>> print(b)
[[1 2]
                                  Element in Zeile 1 und Spalte 0
[3 4]]
>>> b[1,0]
```

Übung 7.1

Aufgabe 1

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

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

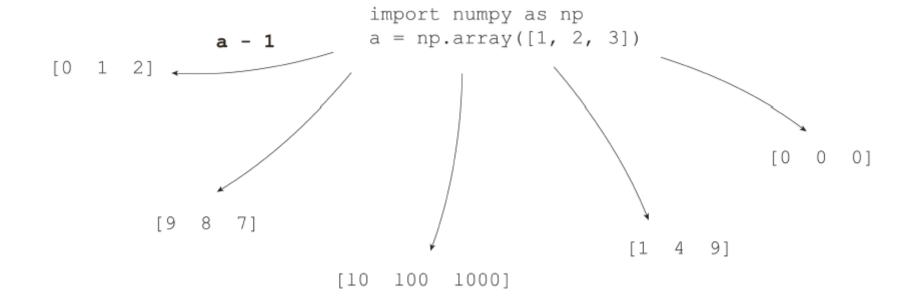
Aufgabe 2

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.



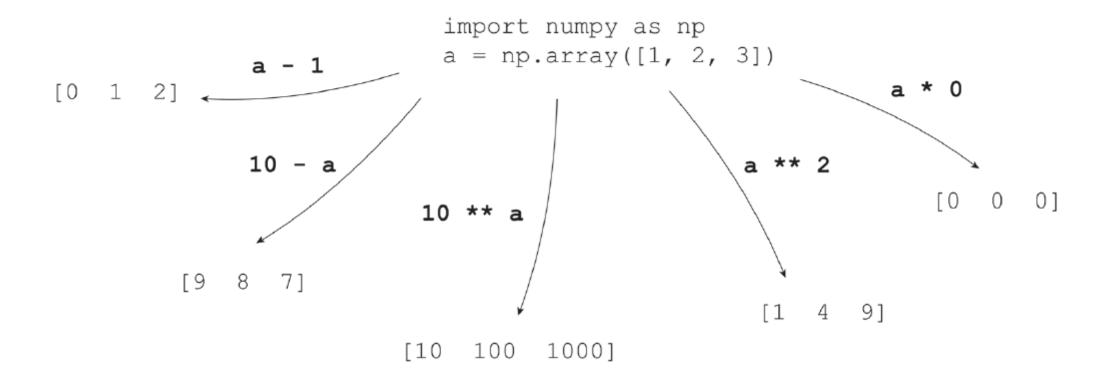
Aufgabe 4 (Bleistift und Papier)

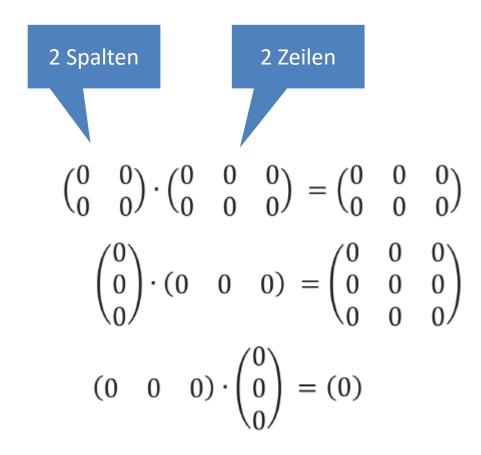
Welche Ergebnisse liefern die folgenden Matrizenmultiplikationen? Das Rechnen ist hier besonders einfach. Es kommt hier vor allem auf die Form der Produktmatrix an.

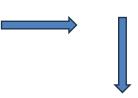
$$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \cdot \begin{pmatrix} 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \cdot \begin{pmatrix} 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix}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

```
(0 1 2 3 4)
>>> np.array(range(5), ndmin=2)
array([[0, 1, 2, 3, 4]])
Oder:
>>> np.arange(5).reshape(1, 5)
array([[0, 1, 2, 3, 4]])
```

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





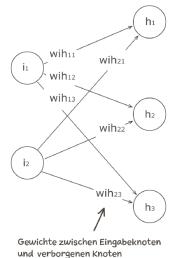


7.2 Mit Arrays die Programmierung vereinfachen: XOR-Detektor (1)

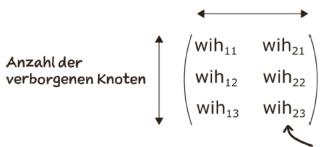
```
# nn_xor.py
import numpy as np
from math import e
from random import shuffle
LR = 0.2

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
```



Anzahl der Eingabeknoten



Gewicht der Verbindung vom 2. Eingabeknoten zum 3. verborgenen Knoten

XOR-Detektor (2)

$$yh \qquad xh \qquad \downarrow \qquad \qquad$$

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

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)

```
# 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)
```

```
    i1
    i2
    t1
    t2

    0
    0
    0
    1

    0
    1
    1
    0

    1
    0
    1
    0

    1
    1
    0
    1
```

```
# Training
```

Spaltenvektoren

XOR-Detektor(4)

Die 4 Zeilen der Trainingstabelle werden einmal durchlaufen

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

- Modifizierte Datenbank des National Institute of Standards and Technology (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

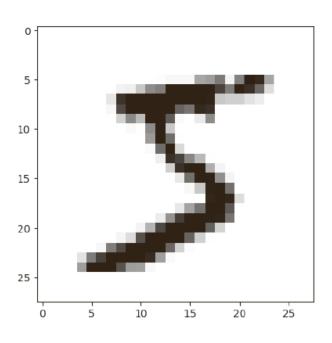
Target



Ordentlich ausgedruckt und ohne Target (5)

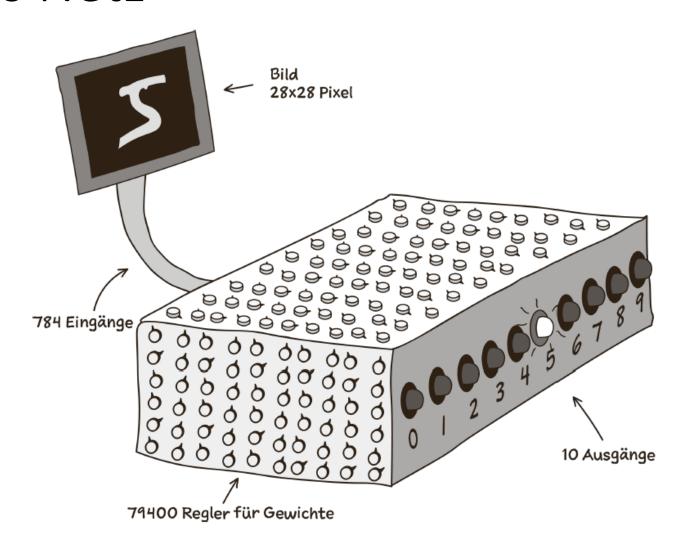


Eine Ziffer auf dem Bildschirm darstellen

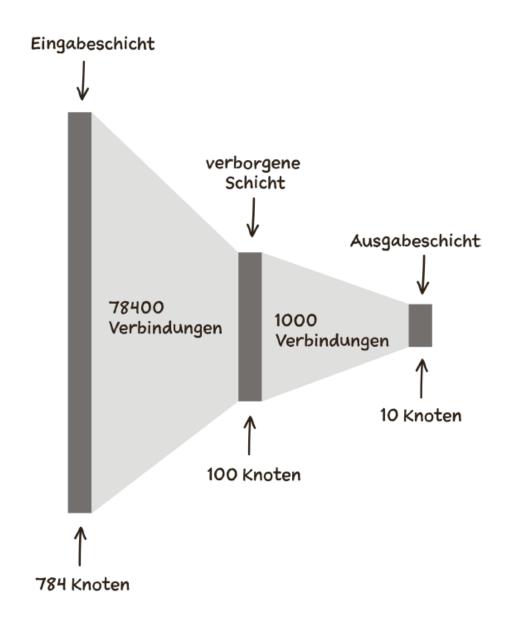


```
# zeige_ziffer.py
import numpy as np
import matplotlib.pyplot as plt
stream = open('daten/mnist10_train.csv', 'r')
datenliste = stream.readlines()
stream.close()
datensatz = datenliste[0].split(',')
pixel = datensatz[1:]
bildArray = np.array(pixel, dtype=int).reshape((28, 28))
plt.imshow(bildArray,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

Neuronales Netz



Neuronales Netz



Was soll das Programm leisten?

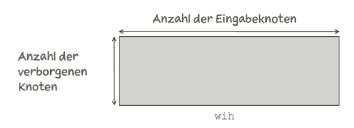
- Training des neuronalen Netzes mit 60000 MNIST-Datensätzen mit Ziffern
- Test des trainierten neuronalen Netzes mit 10000 MNIST-Test-Datensätzen
- Ausgabe: Trefferquote bei 10 000 Tests

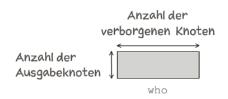
Trefferquote: 94.895 %

Programm (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 siq(x):
   return 1 / (1 + math.e **-x)
```

Viele Konstanten Warum?





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. 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()
                                                      Tracing
    daten = []
                                                  (nur für Testläufe)
    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))
                                           Liste von Tupeln
    return daten
                                      (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:
                                                                                         [[0.]]
                                                                [[1.27980197e-02]
    o = vorhersagen(i)
                                                                 [9.45761590e-04]
                                                                                          [0.],
    ziffer = np.argmax(o)
                                                                 [4.87761731e-03]
                                                                                          [0.],
    erwarteteZiffer = np.argmax(t)
                                                                [9.66018536e-04]
                                                                                          [0.],
                                                                [3.20945303e-05]
                                                                                          [0.1,
    if (ziffer == erwarteteZiffer):
                                                                 [5.90642663e-04]
                                                                                          [0.],
        testbericht.append(1)
                                                                [3.44425547e-04]
                                                                                          [0.],
    else:
                                                                 [6.96053362e-05]
                                                                                          [0.],
        testbericht.append(0)
                                                                 [9.97581101e-01]
                                                                                          [1.],
                                                                 [6.22060274e-02]]
                                                                                          [0.1]
trefferquote = sum(testbericht) / len(testbericht)
print ('Trefferquote:', trefferquote * 100, '%')
input()
```

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

Lösung 3

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?

Normalerweise mehr als 96%

- 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.

```
print('Erstes Item der Testdaten: ')
print(testdaten[0])
print('Ausgabevektor o':)
print(o)
```

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

Die Arrays mit den Gewichten werden gespeichert.

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.