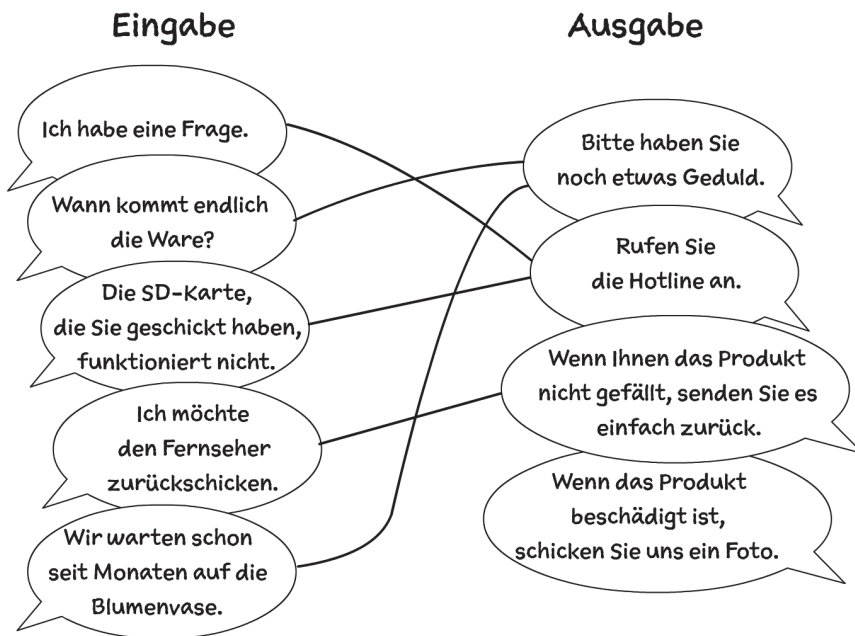


Lösungen der Aufgaben

Kapitel 1

Aufgabe 1: Digitaler Kummerkasten



Aufgabe 2: Tricks mit Listen

a)

Ausgabe	Erklärung
('rot' , 38)	Element der Liste <code>schuhe</code> mit Index 1
<code>schwarz</code>	Erstes Element des ersten Tupels in der Liste <code>schuhe</code>
<code>braun</code>	Die Liste wurde um das Tupel ('braun' , 44) verlängert. Die Variable <code>farbe</code> ist das erste Element des Tupels mit Index 2. Der Inhalt dieser Variablen wird ausgegeben.
3	Die erweiterte Liste <code>schuhe</code> enthält 3 Elemente.

b) Beispiel:

```
paare = [(1, 2)]
```

Aufgabe 3: Machine Learning

1. Die Analyse der Bankgeschäfte ist ein Beispiel für unüberwachtes Lernen, denn es gibt keine etikettierten Daten und die KI findet die Muster ganz allein.
2. Das Lernen vom Meister ist ein Beispiel für überwachtes Lernen. Nach vielen Beispielen bekommt der Auszubildende selbst ein Gefühl dafür, wie ein Motor mit einem bestimmten Fehler klingt.
3. Der Versand von E-Mails ist ein Beispiel für Verstärkungslernen. Positive und negative Reaktionen der Empfänger wirken als Belohnung und Strafe.
4. Wie beim ersten Beispiel werden die ungewöhnlichen Autos ohne voriges Training mit etikettierten Daten gefunden, es handelt sich also um unüberwachtes Lernen. Sie befinden sich übrigens an folgenden Positionen: 6. Reihe, 3. Spalte und 3. Reihe, 8. Spalte.

Kapitel 2

Aufgabe 1: Ein verbesserter Währungsrechner

Programm:

```
print('Währungsrechner')
eingabe = input('Betrag in Dollar: ')
while eingabe != '':
    try:
        dollars = float(eingabe)
        euros = 0.92 * dollars
        print('Wert in Euro: ', round(euros, 2))
    except:
        print('Bitte eine Zahl eingeben!')
        eingabe = input('Betrag in Dollar: ')
print('Auf Wiedersehen!')
```

So funktioniert es:

- #1: Solange der Benutzer etwas eingegeben und nicht einfach nur gedrückt hat, wird der eingerückte Anweisungsblock ausgeführt.
- #2: Hier beginnt eine try-except-Anweisung. Der folgende eingerückte Block wird nur versuchsweise ausgeführt. Wenn ein Laufzeitfehler auftritt, wird die Ausführung abgebrochen und die except-Klausel ausgeführt.

- #3: Wenn bei der letzten `input()`-Anweisung keine Zahl eingegeben worden ist, tritt hier ein Fehler auf und die `except`-Klausel wird ausgeführt.
- #4: Wenn die `try`-Klausel abgebrochen werden musste, wird diese Anweisung ausgeführt. Auf dem Bildschirm erscheint ein Hinweis.
- #5: Diese Anweisung ist nicht mehr eingerückt. Sie gehört also nicht mehr zum `while`-Block und wird erst ausgeführt, wenn die `while`-Schleife verlassen worden ist.

Beispieldialog:

```
Währungsrechner
Betrag in Dollar: zehn
Bitte eine Zahl eingeben!
Betrag in Dollar: 10
Wert in Euro: 9.2
Betrag in Dollar:
Auf Wiedersehen!
```

Aufgabe 2: Ein universeller Währungsrechner

Programm:

```
# universell.py
a = 1
währung = input('Gib die Fremdwährung an: ') #1
eingabeEuros = input('Betrag in Euro :') #2
while eingabeEuros != '':
    euros = float(eingabeEuros)
    vorhersageFremdwährung = a * euros #3
    print('Vorhersage: ',
          round(vorhersageFremdwährung, 2), währung) #4
    eingabeFremd = input('Tats. Betrag in der Fremdwährung: ')
    fremdwährung = float(eingabeFremd)
    fehler = fremdwährung - vorhersageFremdwährung #5
    print('Fehler:', fehler)
    a += 0.5 * fehler / euros #6
    print('Neuer Wechselkurs a:', round(a, 4))
    eingabeEuros = input('Betrag in Euro :')

print('Auf Wiedersehen!')
input()
```

So funktioniert es:

- #1: Der Benutzer kann einen String eingeben, der in der Variablen `währung` gespeichert wird.
- #2: Hier wird ein String eingegeben und in der Variablen `eingabeEuros` gespeichert. Später wird aus dem String eine Zahl gewonnen.
- #3: Hier wird mithilfe des Umrechnungsfaktors `a` der Betrag in der Fremdwährung berechnet, der dem eingegebenen Euro-Betrag entspricht.
- #4: Der Ausgabertext besteht aus drei Teilen:
 - dem String `'Vorhersage: '`,
 - dahinter dem berechneten Betrag in der Fremdwährung, der auf zwei Stellen nach dem Punkt gerundet wurde,
 - und schließlich dem Inhalt der Variablen `währung`, also der Bezeichnung der Währung.
- #5: Der Unterschied zwischen dem berechneten und dem tatsächlichen Euro-Betrag wird berechnet und anschließend ausgegeben.
- #6: Der Wert des Umrechnungsfaktors `a` wird aktualisiert.

Aufgabe 3: Zahlenfolgen und Funktionsgraphen

a) `zahlen = range(1, 7)`

b)

Programm:

```
# diagramm_aufgabe_3.py
from matplotlib import pyplot
xWerte = range(-10, 11)
yWerte = []
for x in xWerte:
    yWerte += [x**3 - 20*x]
pyplot.plot(xWerte, yWerte)
pyplot.show()
```

Aufgabe 4: Raffinierte Listen

a)

```
gerade = [2*x for x in range(1, 11)]
```

b)

```
paare = [(1, 3), (12, 7), (4, 5)]
summen = [a + b for a, b in paare]
```

c)

```
tage = ['Montag', 'Dienstag', 'Mittwoch']
längen = [len(tag) for tag in tage]
```

Kapitel 3

Aufgabe 1: Moderation im Alltag

Man kann es so sehen: Die Bildhauerin arbeitet mit kleinen Tonportionen. Sie gibt ein wenig Ton an eine Stelle, die dicker werden soll. Dann prüft sie das Ergebnis. Eventuell trägt sie noch etwas mehr Ton an die gleiche Stelle auf oder sie schabt etwas Ton ab, wenn es zu dick geworden ist. In kleinen, *moderaten* Schritten nähert sie sich allmählich der gewünschten Form.

Aufgabe 2: Die Trennlinie sichtbar machen

Zu Beginn des Programms fügst du eine Importanweisung ein:

```
# klassifizierer_plot.py
from matplotlib import pyplot as plt
```

Damit wird das Modul `matplotlib.pyplot` importiert. Es erhält in unserem Programm den kürzeren Namen `plt`.

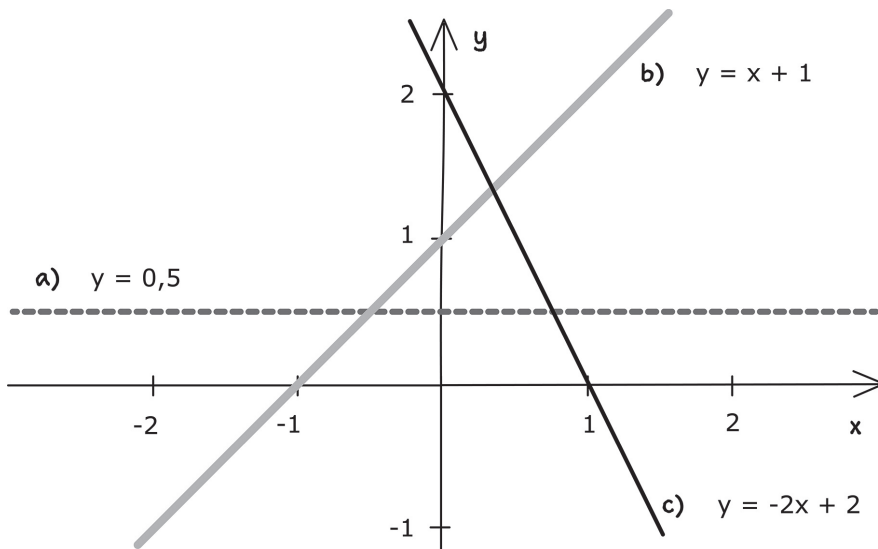
Hinter den zweiten Programmteil (Training) fügst du folgende Anweisungen ein:

```
# Diagramm mit Trennlinie und Datenpunkten
xDaten = [x for x, y, label in DATEN]          #1
yDaten = [y for x, y, label in DATEN]          #2
xTrennlinie = [0, 150]                        #3
yTrennlinie = [0, a * 150]                    #4
plt.plot(xDaten, yDaten, '.')                  #5
plt.plot(xTrennlinie, yTrennlinie)             #6
plt.xlabel('Breite')                           #7
plt.ylabel('Höhe')                             #8
plt.show()
```

So funktioniert es:

- #1: Hier wird eine Liste von Zahlen erzeugt und dem Namen `xDaten` zugewiesen. Es sind die x-Werte der Trainingsdaten, also die Angaben für die Breite eines Bildausschnitts. In diesem Fall ist `xDaten` die Liste `[77, 88, 70, 50, 45, 79, 50, 45, 67]`.
- Die List Comprehension (der Ausdruck in den eckigen Klammern) kann man so lesen: Durchlaufe alle Tupel der Liste `DATEN`, nimm jeweils das erste Element eines Tupels und bilde aus diesen Zahlen eine Liste.
- #2: Entsprechend wird die Liste der y-Werte der Trainingsdaten gebildet, also `[135, 55, 115, 85, 27, 132, 91, 80, 45]`.
- #3: Weil die Trennlinie eine gerade Linie ist, brauchen wir nur zwei Punkte, um die Linie zeichnen zu können. Die x-Werte dieser Punkte können wir frei wählen. Wir nehmen die x-Werte 0 und 150.
- #4: Die y-Werte der beiden Punkte können mithilfe der Funktionsgleichung der Trennlinie berechnet werden: $y = a \cdot x$
- #5: Hier werden die Trainingsdaten in das Diagramm eingetragen. Das Argument `'.'` bewirkt, dass einzelne Datenpunkte gezeichnet werden.
- #6: Jetzt wird zusätzlich die Trennlinie (zwischen 0 und 150) in das Diagramm eingetragen. Voreingestellt ist, dass eine Linie gezeichnet wird.
- #7: Die waagrechte x-Achse wird beschriftet.
- #8: Auf dem Bildschirm erscheint ein Fenster mit dem Diagramm (wie im Bild zur Aufgabe). Erst wenn dieses Fenster geschlossen wird, läuft das Programm weiter.

Aufgabe 3: Lineare Funktionen



Aufgabe 4: Lineare Separierbarkeit

Am besten passt das erste Diagramm. Pflaumen sind kleiner und leichter als Birnen. Die Datenpunkte (Durchmesser, Gewicht) von Pflaumen liegen deshalb links unten im Diagramm, die Datenpunkte der Birnen eher rechts oben. Die Daten sind also linear separierbar.

Kapitel 4

Aufgabe 1: Auf der Suche nach dem Alles-oder-nichts-Prinzip

A: Damit die Kiste geschoben werden kann, muss erst die Haftreibung überwunden werden. Solange man nicht genügend Kraft anwendet, rührt sich die Kiste nicht von der Stelle. Erst ab einer gewissen Kraftanstrengung (Schwellenwert) setzt sie sich in Bewegung.

B: Sobald eine gewisse Helligkeit unterschritten ist (Schwellenwert), wird die Straßenbeleuchtung eingeschaltet.

C: Ab einer bestimmten Kraftanstrengung (Schwellenwert) bricht der Stock.

Aufgabe 2: Knifflige logische Ausdrücke

```
>>> a = 0
>>> not not a
False
```

Die Variable `a` erhält den Wahrheitswert `False`. Wenn wir für `a` `False` einsetzen, erhalten wir den Ausdruck `not not False`. Der Operator `not` kehrt den Wahrheitswert um. Der Ausdruck `not False` hat den Wahrheitswert `True`. Somit hat `not not False` den Wahrheitswert `False`.

```
>>> (a and not a) or a
False
```

Die Variable `a` hat den Wahrheitswert `False`. Wenn wir in den Ausdruck für `a` den Wert `False` einsetzen, erhalten wir:

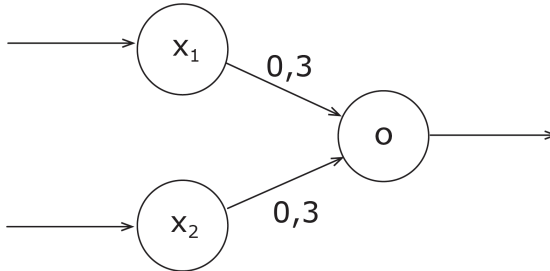
`(False and not False) or False`

Nun hat `not False` den Wahrheitswert `True`. Damit vereinfachen wir den Klammerausdruck zu `False and True`. Das hat den Wahrheitswert `False`. Somit können wir für den gesamten Ausdruck `False or False` schreiben. Und das ergibt `False`.

```
>>> True or not True
True
```

Der Ausdruck `not True` hat den Wahrheitswert `False`. Somit können wir für den gesamten Ausdruck `True or False` schreiben. Das ergibt den Wahrheitswert `True`.

Aufgabe 3: Andere Gewichte



Dieses Perzeptron erkennt UND-verknüpfte Werte. Das heißt: Nur wenn beide Eingaben 1 sind, wird eine 1 ausgegeben, sonst eine 0. Du siehst leicht, dass die Aktivierungsfunktion

$$o = \begin{cases} 1, & \text{falls } 0,3 \cdot x_1 + 0,3 \cdot x_2 > 0,5 \\ 0, & \text{sonst} \end{cases}$$

tatsächlich nur dann den Wert 1 liefert, wenn *beide* Eingaben, also x_1 und x_2 den Wert 1 haben, denn 0,6 ist größer als 0,5.

Statt 0,3 kannst du für die Gewichte auch andere Zahlen nehmen. Sie müssen nur jeweils kleiner als 0,5 sein und ihre Summe muss größer als 0,5 sein, damit die Aktivierungsfunktion funktioniert.

Kapitel 5

Aufgabe 1: Die Sigmoid-Funktion und ihre Ableitung

Die eingefügten Passagen sind fett gedruckt.

Programm:

```
# sigmoid_plot.py
from matplotlib.pyplot import *
from math import e
```



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

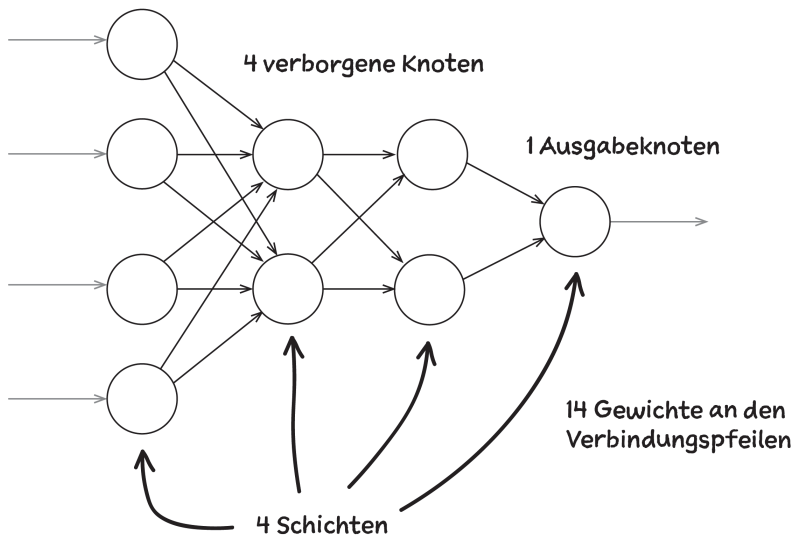
def sigmoid_ableitung(x):
    return sig(x) * (1 - sig(x)) #1
x_ = [x/10 for x in range(-100, 100)]
y_1 = [sig(x) for x in x_]
y_2 = [sigmoid_ableitung(x) for x in x_] #2
plot(x_, y_1)
plot(x_, y_2) #3
xlabel('x')
grid()
show()
```

So funktioniert es:

- #1: Die Funktion für die Ableitung der Sigmoid-Funktion. Hier wird einfach die mathematische Formel für die Ableitung verwendet.
- #2: Liste mit den Werten der Ableitung der Sigmoid-Funktion.
- #3 In das Diagramm wird der Graph der Ableitung der Sigmoid-Funktion als zweite Linie mit einer automatisch gewählten Farbe eingetragen.

Aufgabe 2: Struktur eines neuronalen Netzes

4 Eingabeknoten



Aufgabe 3: Backpropagation von Hand

Mit den Zahlenwerten aus der Tabelle und aus der Abbildung werden die Fehler folgendermaßen berechnet:

Fehler an den Ausgabeknoten:

$$\begin{aligned} eo1 &= t1 - o1 = 1 - 0,9 = 0,1 \\ eo2 &= t2 - o2 = 0 - 0,2 = -0,2 \end{aligned}$$

Fehler an den verborgenen Knoten:

$$\begin{aligned} eh1 &= wh1o1 \cdot eo1 + wh1o2 \cdot eo2 \\ &= 0,8 \cdot 0,1 + 0,5 \cdot (-0,2) \\ &= 0,08 - 0,1 = -0,02 \\ eh2 &= wh2o1 \cdot eo1 + wh2o2 \cdot eo2 \\ &= 0,2 \cdot 1 + 0,1 \cdot (-0,2) \\ &= 0,02 - 0,02 = 0 \end{aligned}$$

Kapitel 6

Aufgabe 1: Arrays erzeugen

a) Du verwendest eine Liste, die zwei Listen enthält:

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

Du erzeugst ein Array mit vier Zeilen und zwei Spalten:

```
>>> b = np.zeros(shape=(4, 2))
>>> print(b)
[[0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]]
```

Du verwendest die Funktion `range()`, um die Zahlenfolge zu erstellen:

```
>>> c = np.arange(5)
>>> print(c)
[0 1 2 3 4]
```

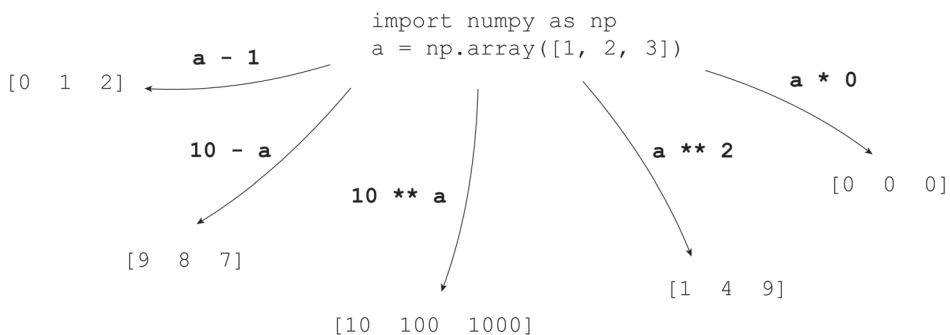
b) Mit dem Schlüsselwortargument `ndmin` kannst du die gewünschte Dimension einstellen:

```
>>> import numpy as np
>>> a = np.array([1], ndmin=5)
>>> print(a)
[[[[[1]]]]]
>>> a.ndim
5
```

Alternativ kannst du auch eine geschachtelte Liste mit fünf öffnenden und fünf schließenden Klammern verwenden:

```
>>> a = np.array([[[[[1]]]]])
```

Aufgabe 2: Rechnen mit Arrays und Skalaren



Aufgabe 3: Kopfrechnen mit Arrays

Deine Lösungen kannst du mit Python-Anweisungen überprüfen:

```
>>> import numpy as np
>>> a = np.array([1, 2])
>>> b = np.array([0, 1])
>>> print(a + b)
[1 3]
>>> print(a * b)
[0 2]
```

```
[0 2]
>>> print(a - b + 1)
[2 2]
```

Aufgabe 4: Arrays in Form bringen

```
>>> a = 2** np.arange(16)
>>> print(a.reshape(4, 4))
```

Aufgabe 5: Matrizenmultiplikation

a) Achte darauf: In allen Fällen stimmen die Spaltenzahl der ersten Matrix und die Zeilenzahl der zweiten Matrix überein.

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

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

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

b) Hier ist ein kleines Programm, das alle drei Matrizenmultiplikationen umsetzt und jeweils die Produktmatrix ausgibt:

```
# aufgabe_5.py
import numpy as np
a = np.zeros(shape=(2, 2))
b = np.zeros(shape=(2, 3))
c = np.dot(a, b)
print(c)
print()
a = np.zeros(shape=(3, 1))
b = np.zeros(shape=(1, 3))
c = np.dot(a, b)
print(c)
print()
a = np.zeros(shape=(1, 3))
b = np.zeros(shape=(3, 1))
c = np.dot(a, b)
print(c)
```

Ausgabe:

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

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

[[0.]]
```

Aufgabe 6: Arrays mit Zufallszahlen

a) 2-mal-2-Array aus gleichverteilten Zufallszahlen zwischen -10 und +10:

```
>>> import numpy as np
>>> a = np.random.rand(2, 2) * 20 - 10
>>> print(a)

[[ 6.420923  -3.34030364]
 [ 3.9498281  1.45843262]]
```

b) Der IQ ist normalverteilt bei einem Mittelwert von 100 und einer Standardabweichung von 15. Erzeuge eine Folge von 100 ganzen Zufallszahlen mit dieser Verteilung:

```
>>> a = np.random.randn(100)*15 + 100
>>> intelligenz = np.round(a)
>>> print(intelligenz)

[ 98. 117. 106.  97. 141. 115.  89. 125.  89. 104.  ... ]
```

Aufgabe 7: Elemente eines Arrays verarbeiten

```
>>> import numpy as np
>>> a = np.arange(10)
>>> np.min(a)
0
>>> np.sum(a)
45
>>> np.average(a)
```

```
4.5
>>> np.sum(a**2)
285
```

Aufgabe 8: Auf Elemente eines Arrays zugreifen

Wir nehmen ein konkretes Array als Beispiel:

```
>>> a = np.array([2, 8, 13, 7])
```

Netzt suchen wir den Index des größten Elements:

```
>>> index = np.argmax(a)
```

Wir erhöhen den Wert des größten Elements um 1:

```
>>> a[index] +=1
>>> print(a)
[ 2  8 14  7]
```

Aufgabe 9: Eine Ziffer anzeigen

Die geänderten Stellen im Programmtext sind fett gedruckt:

```
...
datensatz = datenliste[1].split(',') #1
pixel = datensatz[1:]
bildArray = np.array(255 - pixel,
                      dtype=int).reshape((28, 28)) #2
...
```

So funktioniert es:

- #1: Der Index des zweiten Elements der Liste ist 1.
- #2: Für jedes Pixel des Bilds wird anstelle des Originalgrauwerts *n* der Wert 255 - *n* genommen. So werden die Grautöne umgekehrt. Aus Schwarz wird Weiß und aus Weiß wird Schwarz.

Aufgabe 10: Experimentieren

In den ersten Programmzeilen nimmst du folgende Änderungen vor:

```
EPOCHEN = 2
...
H_KNOTEN = 200
```

Damit kann die Trefferquote auf etwa 97 % gesteigert werden.

Kapitel 7

Aufgabe 1: Bildstatistik

Programm:

```
# bildstatistik.py
from PIL import Image
PFAD = 'bilder/4.png'
bild = Image.open(PFAD)
kleinesBild = bild.resize(size=(28, 28))
sequenz = kleinesBild.getdata()
hell = 0 #1
for pixel in sequenz: #2
    if sum(pixel) > 400: #3
        hell += 1
prozent = round(hell/len(sequenz) * 100) #4
print('Anteil heller Pixel:', prozent, '%') #5
```

Ausgabe (Beispiel):

```
Anteil heller Pixel: 89 %
```

So funktioniert es:

- #1: Anzahl der hellen Pixel. Die Variable `hell` ist ein Zähler für helle Pixel.
- #2: Die Sequenz der Pixel des Bilds wird durchlaufen. Jedes Pixel wird durch ein 3-Tupel der Form `(r, g, b)` dargestellt. Dabei sind `r`, `g` und `b` jeweils Zahlen zwischen 0 und 255.
- #3: Wenn die Summe der RGB-Werte des Pixels größer als 400 ist, wird der Inhalt der Variablen `hell` um 1 erhöht.
- #4: Hier wird der Anteil der hellen Pixel in Prozent berechnet und in der Variablen `prozent` gespeichert. Zuerst wird der Inhalt der Variablen `hell` durch die Anzahl der Pixel dividiert (Länge von `sequenz`) und das Ergebnis mit 100 multipliziert.

- #5: Der Computer gibt das Ergebnis aus. Der ausgegebene Text besteht aus drei Teilen: Zuerst kommt der String 'Anteil heller Pixel:', danach der Inhalt der Variablen `prozent` und dann der String '%'.

Aufgabe 2: Rotfilter

Programm:

```
# rotfilter.py
from PIL import Image
PFAD = 'bilder/4.png'
bild = Image.open(PFAD)
kleinesBild = bild.resize(size=(100, 100))
sequenz = kleinesBild.getdata()
rot = [(r, 0, 0) for r, g, b in sequenz]           #1
neuesBild = Image.new(mode='RGB', size=(100, 100)) #2
neuesBild.putdata(rot)                             #3
neuesBild.show()
```

So funktioniert es:

- #1: Das Sequenz-Objekt `sequenz` enthält die Pixel des Bilds als 3-Tupel. Die List Comprehension berechnet nach folgendem Verfahren aus den 3-Tupeln des Sequenz-Objekts eine neue Liste von 3-Tupeln: Der Wert für den Rotanteil wird übernommen, für Grün und Blau wird 0 eingesetzt. Der Name der neuen Liste ist `rot`.
- #2: Hier wird ein neues schwarzes Farbbild der Größe 100 mal 100 Pixel erzeugt. Es ist ein Image-Objekt mit dem Namen `neuesBild`. Es hat die gleiche Größe wie das Image-Objekt `kleinesBild`.
- #3: Die Werte der Liste `rot` werden in das neue Bild eingetragen.

Aufgabe 3: Mehrere Bilder auswerten

Programm:

```
...
stream.close()
eingabe = 'j'           #1
while eingabe == 'j':   #2
    dateiname = input('Dateiname: ')
    i = bildLesen('bilder/' + dateiname)
    o = vorhersagen(i)
```



```

ziffer = np.argmax(o)
print('Ich erkenne die Ziffer', ziffer)
eingabe = input('Noch ein weiteres Bild? (j/n) ') #3

```

So funktioniert es:

- #1: Die neue Variable `eingabe` wird auf den Wert `'j'` gesetzt. So ist garantiert, dass die folgende `while`-Schleife wenigstens einmal durchlaufen wird.
- #2: Solange die Variable `eingabe` den Wert `'j'` hat, wird der eingerückte Anweisungsblock ausgeführt.
- #3: Am Ende des `while`-Anweisungsblocks kann der Benutzer entscheiden, ob er noch ein weiteres Bild prüfen will.

Aufgabe 4: Erweiterung

Programm:

Im folgenden Listing sind die geänderten Passagen fett gedruckt:

```

...
def bildLesen():
    get, bild = kamera.read()
    cv2.imshow('Kontrollbild', bild) #1
    cv2.waitKey(0) #2
    ...
    ersteGeste = input('Name der ersten Geste: ')
    zweiteGeste = input('Name der zweiten Geste: ')
    anzahl = int(input('Wie viele Bilder von jeder Geste? '))
    breite = int(input('Bildbreite in Pixel (z.B. 30): ')) #3
    höhe = int(input('Bildhöhe in Pixel (z.B. 24): '))
    I_KNOTEN = breite * höhe #4
    wih = np.random.rand(H_KNOTEN, I_KNOTEN) - 0.5
    who = np.random.rand(O_KNOTEN, H_KNOTEN) - 0.5
    ...
    while eingabe == 'j':
        input('Mache eine Geste und drücke ENTER!')
        i = bildLesen()
        o = vorhersagen(i)
        print('Ausgabe des neuronalen Netzes:',
            o[0,0], o[1,0]) #5
    ...

```

So funktioniert es:

- #1: Es öffnet sich ein Viewer-Fenster mit dem Titel »Kontrollbild«. Es zeigt das Bild.
- #2: Das Programm wartet, bis das Viewer-Fenster geschlossen wird.
- #3: Über die Tastatur wird die Breite des Bilds eingegeben. Auch wenn Zahlen eingetippt werden, gibt die Funktion `input()` einen String zurück. Mit `int()` wird aus dem String eine ganze Zahl.
- #4: Breite mal Höhe ergibt die Anzahl der Pixel des Bilds. Das neuronale Netz braucht für jedes Pixel einen Eingabeknoten.
- #5: Hier werden die Zahlen, die das neuronale Netz ausgibt, auf dem Bildschirm ausgegeben. Die Variable `o` bezeichnet einen Spaltenvektor mit zwei Elementen, d.h. ein zweidimensionales Array mit dieser Form:

```
[[o1]
 [o2]]
```

Das erste Element wird mit `o[0,0]` angesprochen, denn es ist in der Zeile mit Index 0 und in der Spalte mit Index 0. Das zweite Element ist in Zeile 1 und Spalte 0.

Kapitel 8

Damit alle Gesichter unkenntlich gemacht werden und der Titel des Ausgabefensters angepasst wird, musst du nur drei Zeilen am Ende des Programmtexts abwandeln (fett gedruckt):

```
for x,y,w,h in rechtecke:
    cv2.rectangle(img, (x, y), (x+w, y+h), (155, 155, 155), -1)
    cv2.imshow('Foto mit unkenntlich gemachten Gesichtern', img)
```

In der `for`-Anweisung wird die gesamte Liste `rechtecke` durchlaufen und jedes Tupel sofort ausgepackt. In der eingerückten Anweisung wird zum aktuellen Tupel der Liste in das Bild ein gefülltes Rechteck gezeichnet. Die Farbe des Rechtecks wird durch das Tupel `(155, 155, 155)` bestimmt (ein Grauton). Als Liniendicke wird `-1` angegeben (letztes Argument).