

# Herzlich Willkommen

Masterarbeit Schraubenerkennung

## Einleitung

Wer kennt es nicht, es ist gerade wieder Lockdown und man möchte ein Heimwerker Projekt umsetzen. Nun stellt man fest, dass man von einigen Schrauben nicht genügend vorrätig hat und Nachschub besorgen muss. Da alle Baumärkte geschlossen haben hat man nur noch die Möglichkeit online die gewünschten Exemplare zu ordern. Der Händler des Vertrauens benötigt lediglich die genaue Bezeichnung der benötigten Schrauben und da liegt das Problem, woher soll man die Bezeichnung der Schraube wissen ohne Seitenweise Kataloge zu studieren?

Die Lösung wäre eine automatische Schraubenerkennung. Ein Foto von der Schraube als Eingabe und die Bezeichnung der Schraube als Output.

Eine Suche nach Objekterkennung bei Google ergab:

- Object recognition is a computer vision technique for identifying objects in images or videos. Object recognition is a key output of deep learning and machine learning algorithms. When humans look at a photograph or watch a video, we can readily spot people, objects, scenes, and visual details.

Wir wollen also Objekte, in unserem Fall Schrauben in Bildern erkennen und dafür machine learning und deep learning Techniken verwenden.



The Turing Way project illustration by Scriberia. Zenodo. <http://doi.org/10.5281/zenodo.3332807>

## Über Mich

bla bla

The Turing Way is:

- a book
- a community
- a global collaboration

## How to Interact With This Book

This book was created with Jupyter Book, an open-source project for designing online interactive books that contain code and other computational material.

Below the interactive features of this Jupyter Book are explained in further detail. Most interactive features can be found in the toolbar at the top of the page.

### Open Jupyter Notebook in the Cloud

You can open most pages from this book in the cloud and run the code live. Hover over the rocket icon xxx at the top of the page and click “Binder” to open a version of the same page in the cloud.

Binder is a service that allows you to run Jupyter notebooks without any prior configuration or installation. It may take a few minutes for the Jupyter notebook to load, so be patient.

### Download Jupyter Notebook

You can download any Jupyter notebook page from this book as a Jupyter notebook file (.ipynb). Hover over the download icon and click “.ipynb”

#### Attention

To work with this .ipynb file, you will need to have Jupyter installed and running on your own computer.

### Download PDF

You can download any Jupyter notebook page from this book as a PDF file. Hover over the download icon and click “.pdf”

### Make Full Screen

To make any page from this book full screen, click the full screen icon at the top of the page.

### Open Issue on GitHub

If you run into any issues or would like to make a suggestion, you can open an issue on GitHub by hovering over the GitHub icon and clicking “open issue.”

You can also email me with issues or suggestions at [#weileppmartin@gmail.com](mailto:#weileppmartin@gmail.com)

### Click to Show Output/Code

Many of the pages in this book include hidden content. You can reveal the hidden content by clicking “Click to Show” beneath the cell on the right.

Revealing content is meant to replicate the experience of running code live.

```
print("Here's an example of hidden output! Click 'Click to Show' to see the output!")
```

```
42 * 42
```

## Arbeiten mit Bildern in Jupyter Notebook

Bibliotheken:

```
import os
os.getcwd() # get working directory
#change working directory
os.chdir('C://Users//Martin//Desktop//jupyter_Book_data//2020-jupyterbook-with-turing-way-master')
os.getcwd() # get working directory
```

```
'C:\\Users\\Martin\\Desktop\\jupyter_Book_data\\2020-jupyterbook-with-turing-way-master'
```

```
import numpy as np
from PIL import Image

img = Image.open("my_jupyter-book/figures/philips_28x28_gray.jpg") # hier rgb bild einlesen um FarbkanaL zu haben
np_img = np.array(img)
np_img.shape
```

```
(28, 28)
```

Wie setzt sich ein Bild in einem Array zusammen? Hier den Aufbau eines Bildes veranschaulichen:

Bei Quadratischen Bildern:

Einlesen als Graustufenbild:

Shape:

Resize:

In Graustufen umwandeln:

IMG to Numpy Array:

Speichern:

Mehrere Bilder in ein Array laden:

## Theoretische Grundlagen Neuronale Netze

In den folgenden Seiten werden die Grundlagen erläutert.

### Mathematik

#### Lineare Regression

...bildet den Grundstein neuronaler Netze.

#### Lineare Regression (mit einem Punkt)

```

import numpy as np
import matplotlib.pyplot as plt

def f(a, x):
    return a * x

def J(a, x, y):
    return (y - a * x) ** 2

def J_ableitung_a(a, x, y):
    return -2 * x * (y - a * x)

point = (1, 4)
lr = 0.05
a = 1
for i in range(0, 20):
    da = J_ableitung_a(a, point[0], point[1])
    a = a - lr * da
    print("Kosten J = " + str(round(J(a, point[0], point[1]),3)) + " wenn a = " +
    str(round(a,3)))

xs = np.arange(-2, 2, 0.1)
ys = f(a, xs)
plt.plot(xs, ys)


plt.scatter(point[0], point[1])
plt.show()

```

```

Kosten J = 7.29 wenn a = 1.3
Kosten J = 5.905 wenn a = 1.57
Kosten J = 4.783 wenn a = 1.813
Kosten J = 3.874 wenn a = 2.032
Kosten J = 3.138 wenn a = 2.229
Kosten J = 2.542 wenn a = 2.406
Kosten J = 2.059 wenn a = 2.565
Kosten J = 1.668 wenn a = 2.709
Kosten J = 1.351 wenn a = 2.838
Kosten J = 1.094 wenn a = 2.954
Kosten J = 0.886 wenn a = 3.059
Kosten J = 0.718 wenn a = 3.153
Kosten J = 0.581 wenn a = 3.237
Kosten J = 0.471 wenn a = 3.314
Kosten J = 0.382 wenn a = 3.382
Kosten J = 0.309 wenn a = 3.444
Kosten J = 0.25 wenn a = 3.5
Kosten J = 0.203 wenn a = 3.55
Kosten J = 0.164 wenn a = 3.595
Kosten J = 0.133 wenn a = 3.635

```

 C:/Users/Martin/Desktop/jupyter\_Book\_data/2020-jupyterbook-with-turing-way-master/my\_jupyter-book/\_build/jupyter\_execute/lineare Regression1\_1\_1.png

## Gradientenabstiegsverfahren

```

import numpy as np
import matplotlib.pyplot as plt

def f(x):
    return x ** 2 - 4 * x + 5

def f_ableitung(x):
    return 2 * x - 4

```

```

x = 5
plt.scatter(x, f(x), c="r")
for i in range(0, 25):
    steigung_x = f_ableitung(x)
    x = x - 0.05 * steigung_x
    plt.scatter(x, f(x), c="r")
    print(round(x,3))

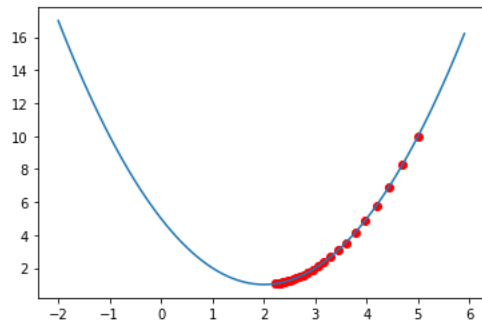
xs = np.arange(-2, 6, 0.1)
ys = f(xs)
plt.plot(xs, ys)
plt.show()

```

```

4.7
4.43
4.187
3.968
3.771
3.594
3.435
3.291
3.162
3.046
2.941
2.847
2.763
2.686
2.618
2.556
2.5
2.45
2.405
2.365
2.328
2.295
2.266
2.239
2.215

```



Lineare Regression (mehrere Punkte)

```

import numpy as np
import matplotlib.pyplot as plt

def f(a, x):
    return a * x

def J(a, x, y):
    return (y - a * x) ** 2

def J_ableitung_a(a, x, y):
    return -2 * x * (y - a * x)

point1 = (1, 4)
point2 = (1.5, 5)

lr = 0.05
a = 1
for i in range(0, 20):
    da = J_ableitung_a(a, point1[0], point1[1])
    a = a - lr * da

    da = J_ableitung_a(a, point2[0], point2[1])
    a = a - lr * da

    cost = J(a, point1[0], point1[1]) + J(a, point2[0], point2[1])
    print("Kosten wenn a = " + str(round(a,3)) + ": " + str(round(cost,3)))

xs = np.arange(-2, 2, 0.1)
ys = f(a, xs)
plt.plot(xs, ys)


plt.scatter(point1[0], point1[1], color="red")
plt.scatter(point2[0], point2[1], color="green")
plt.show()

```

```

Kosten wenn a = 1.758: 10.616
Kosten wenn a = 2.286: 5.407
Kosten wenn a = 2.654: 2.848
Kosten wenn a = 2.911: 1.585
Kosten wenn a = 3.091: 0.959
Kosten wenn a = 3.216: 0.646
Kosten wenn a = 3.303: 0.488
Kosten wenn a = 3.364: 0.407
Kosten wenn a = 3.406: 0.364
Kosten wenn a = 3.436: 0.342
Kosten wenn a = 3.457: 0.33
Kosten wenn a = 3.471: 0.323
Kosten wenn a = 3.481: 0.318
Kosten wenn a = 3.488: 0.316
Kosten wenn a = 3.493: 0.314
Kosten wenn a = 3.496: 0.313
Kosten wenn a = 3.499: 0.313
Kosten wenn a = 3.5: 0.312
Kosten wenn a = 3.501: 0.312
Kosten wenn a = 3.502: 0.312

```

 C:/Users/Martin/Desktop/jupyter\_Book\_data/2020-jupyterbook-with-turing-way-master/my\_jupyter-book/\_build/jupyter\_execute/lineare Regression2\_1\_1.png

Lineare Regression (vektorisert)

```

import numpy as np
import matplotlib.pyplot as plt

def f(a, x):
    return a * x

def J(a, x, y):
    return np.mean((y - a * x) ** 2)

def J_ableitung_a(a, x, y):
    return np.mean(-2 * x * (y - a * x))

points = np.array([
    [1, 4],
    [1.5, 5],
    [2, 8],
    [0.5, 3]
])

lr = 0.05
a = 1
for i in range(0, 50):
    da = J_ableitung_a(a, points[:, 0], points[:, 1])
    a = a - lr * da

    cost = J(a, points[:, 0], points[:, 1])
    print("Kosten wenn a = " + str(round(a,3)) + ": " + str(round(cost,3)))

xs = np.arange(-2, 2, 0.1)
ys = f(a, xs)
plt.plot(xs, ys)

plt.scatter(points[:, 0], points[:, 1], color="red")
plt.show()

```

