Neuronale Netze

Handgeschriebene Ziffern erkennen im MNIST-Dataset mit Python und PyTorch

UNIVERSITÄT DES SAARLANDES

SIC Saarland Informatics Campus

Bild: Pixabay

Bild: Pixabay

Künstliche Intelligenz

Definition von Künstlicher Intelligenz:

„Der Begriff **Künstliche Intelligenz** bezeichnet das Verhalten einer Maschine, das – wenn sich ein Mensch genauso verhält – als intelligent angesehen wird."

Simmons & Cappell, 1988

Definition von Maschinellem Lernen:

„**Maschinelles Lernen** bezeichnet – sehr allgemein formuliert – die Idee, Computern durch geeignete Algorithmen die Fähigkeit zu verleihen, aus Daten und Erfahrungen zu lernen.

Computer können damit ein **Modell ihrer Welt aufbauen** und die ihnen zugedachten Aufgaben besser lösen."

Sim BaFin-Studie KI, 2018

www.wissenschaftsjahr.de/2019/fileadmin/user_upload/Wissenschaftsjahr_2019/Jugendaktion/WJ19_LA_Material_Buch_CPS_barrRZ.pdf

Schwache und starke KI

Künstliche Intelligenz lässt sich in zwei Kategorien einteilen.

- **Schwache künstliche Intelligenz:** Sie ist nur auf einen bestimmten Bereich wie Bilderkennung spezialisiert. Innerhalb dieses Bereichs kann sie zwar lernen und eigenständig Probleme lösen. Jedoch kann sie die Lösungen auf andere Bereiche **nicht übertragen**.
- **Starke künstliche Intelligenz**: Sie kann, wie der Mensch, Probleme jeder Art lösen und ist nicht auf einen Bereich spezialisiert. Starke KI ist bisher jedoch **nicht möglich** und nur Science Fiction.

Das machen wir heute

https://studyflix.de/informatik/was-ist-kuenstliche-intelligenz-6919

Randnotiz: ChatGPT ist keine starke KI, auch wenn es über alles reden kann.

ChatGPT ist ein **Large Language Model (LLM)**, das auf Neuronalen Netzen basiert.

Bild: Pixabay

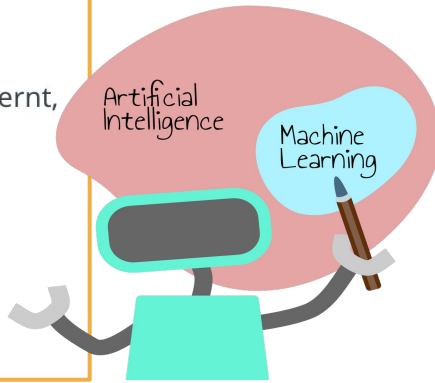
Maschinelles Lernen

Maschinelles Lernen (Machine Learning)

Machine Learning ist ein wichtiger Bestandteil der **künstlichen Intelligenz**. Dabei kann ein IT-System auf Basis von **Algorithmen** in Daten selbstständig Muster und Gesetzmäßigkeiten erkennen. So kann maschinelles Lernen mithilfe von **Daten** **Vorhersagen** treffen. Außerdem kann es durch Erfahrungen lernen, eigenständig neue Probleme zu lösen.

Anwendungsbeispiele für Machine Learning findest du viele:

- **Personalisierung im Marketing:** Machine Learning erkennt die Vorlieben und das Kaufverhalten von Kunden und kann so passende Produktempfehlungen liefern.
- **Autonomes Fahren:** Das Fahrzeug nimmt verschiedene Sensordaten aus dem Straßenverkehr auf und lernt, darauf passend zu reagieren.
- **IT-Security:** Machine Learning Modelle können mithilfe umfangreicher Datenanalysen schon früh Unregelmäßigkeiten erkennen. So wird betrügerisches Verhalten (*z. B. bei Kreditkartenbetrug*) schnell unterbunden.

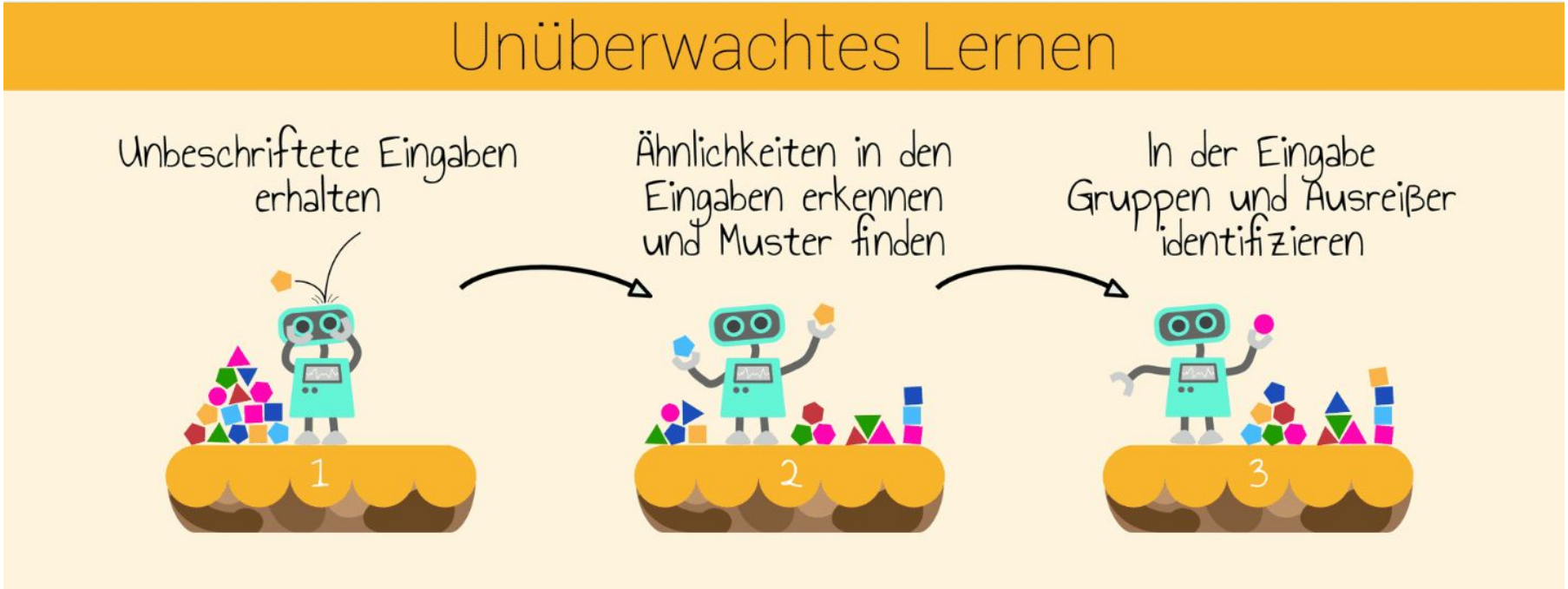


computingeducation.de/proj-ml-uebersicht/

https://studyflix.de/informatik/machine-learning-4356

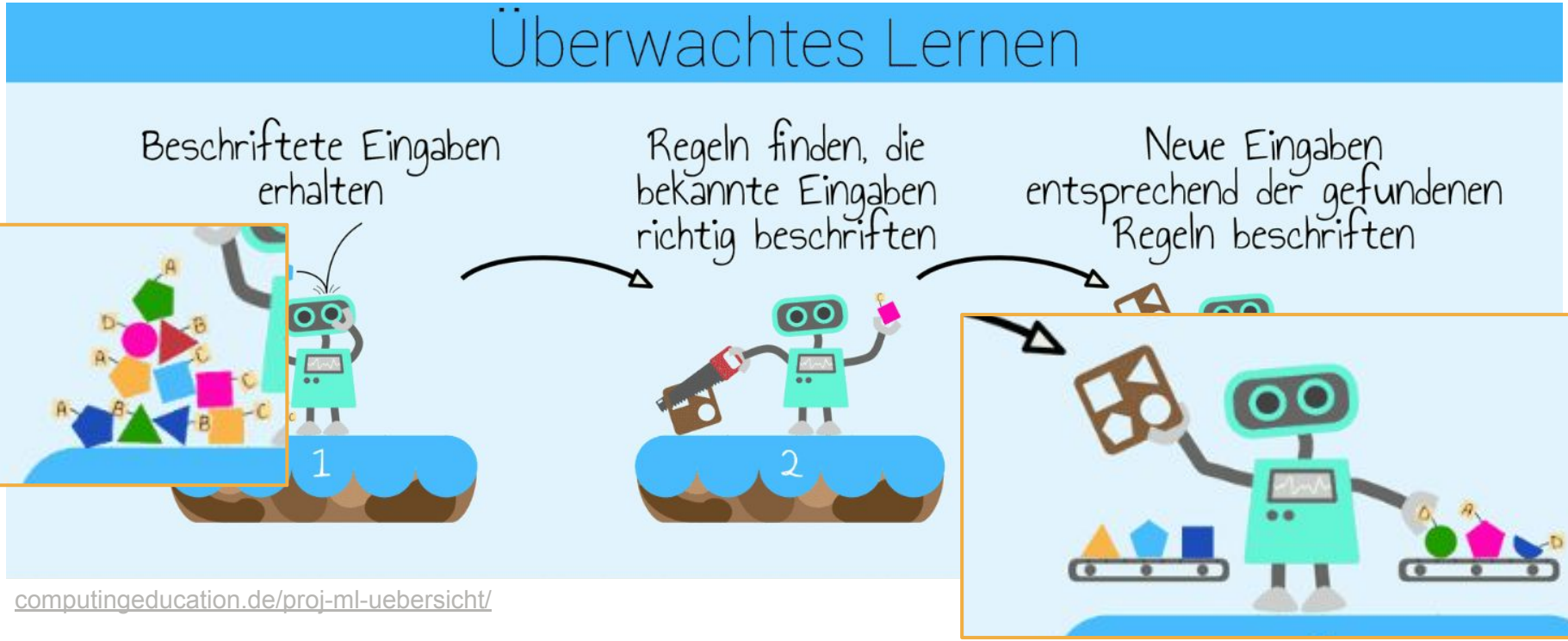
Paradigmen des maschinellen Lernens - Verstärkendes Lernen

Paradigmen des maschinellen Lernens - Unüberwachtes Lernen



computingeducation.de/proj-ml-uebersicht/

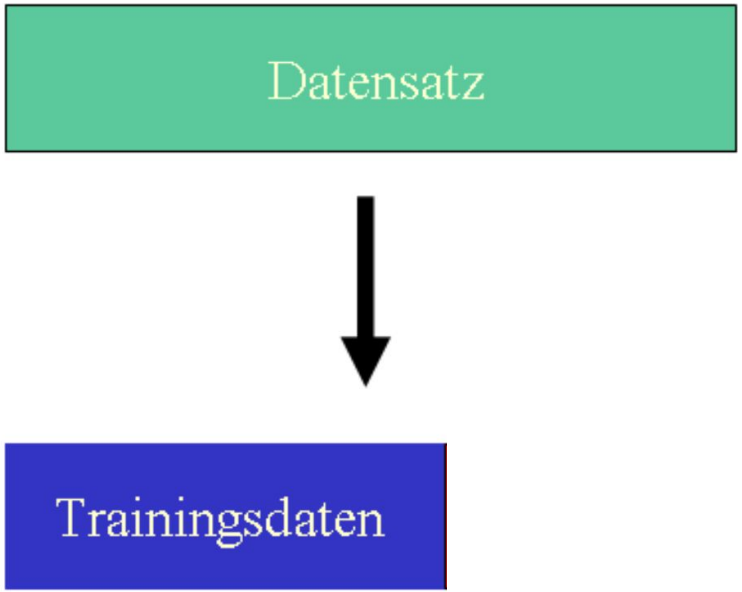
Paradigmen des maschinellen Lernens - Überwachtes Lernen



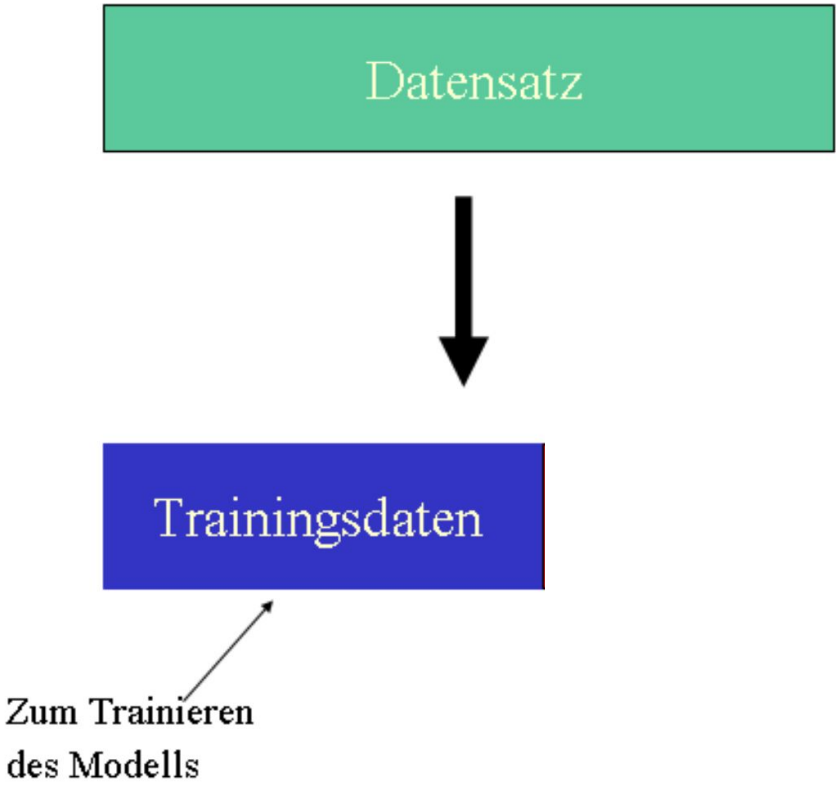
computingeducation.de/proj-ml-uebersicht/

Trainings- und Testdaten

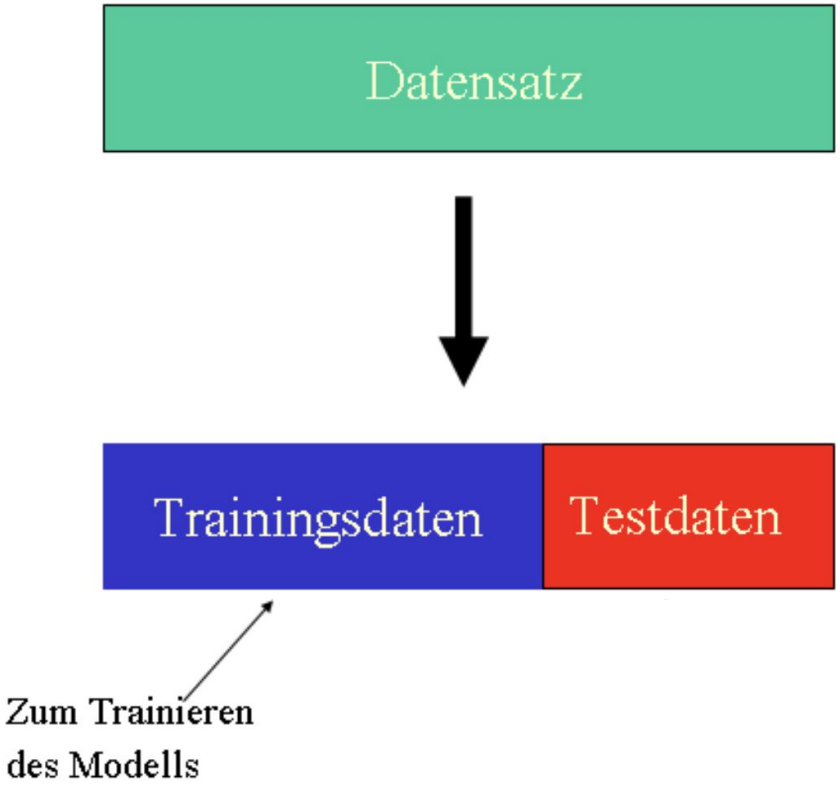
Trainings- und Testdaten



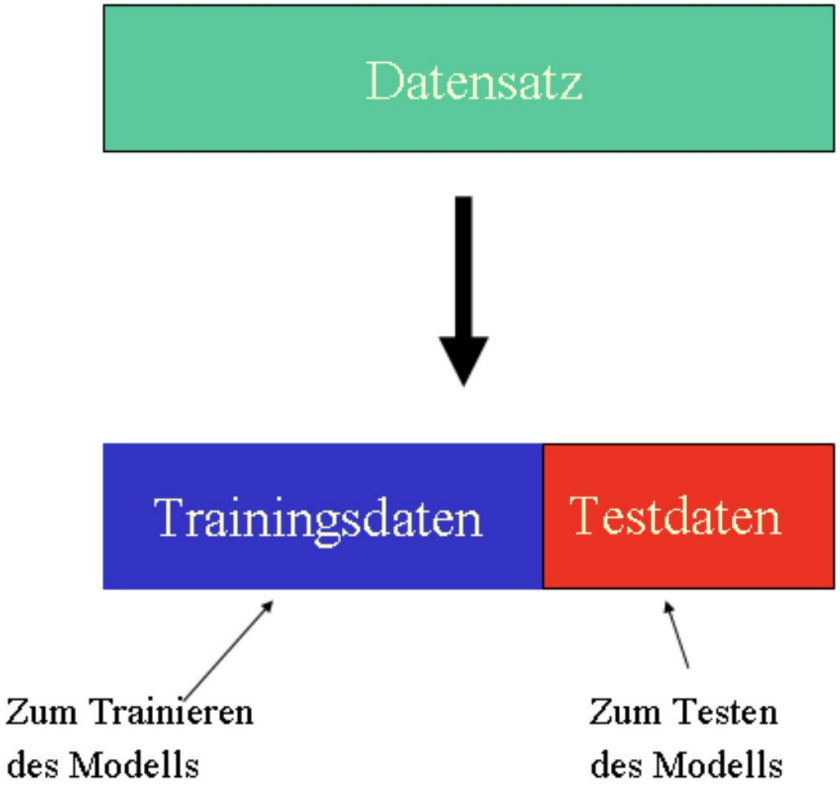
Trainings- und Testdaten



Trainings- und Testdaten



Trainings- und Testdaten



Aufgabe heute:

Ein künstliches Neuronales Netz

zum überwachten Lernen implementieren

Gegeben / Kundenwunsch

- Daten für überwachtes Lernen (MNIST)
- Künstliches Neuronales Netz (KNN)
- Python
- PyTorch
- Jupyter Notebook
- Google Colab oder Visual Studio Code

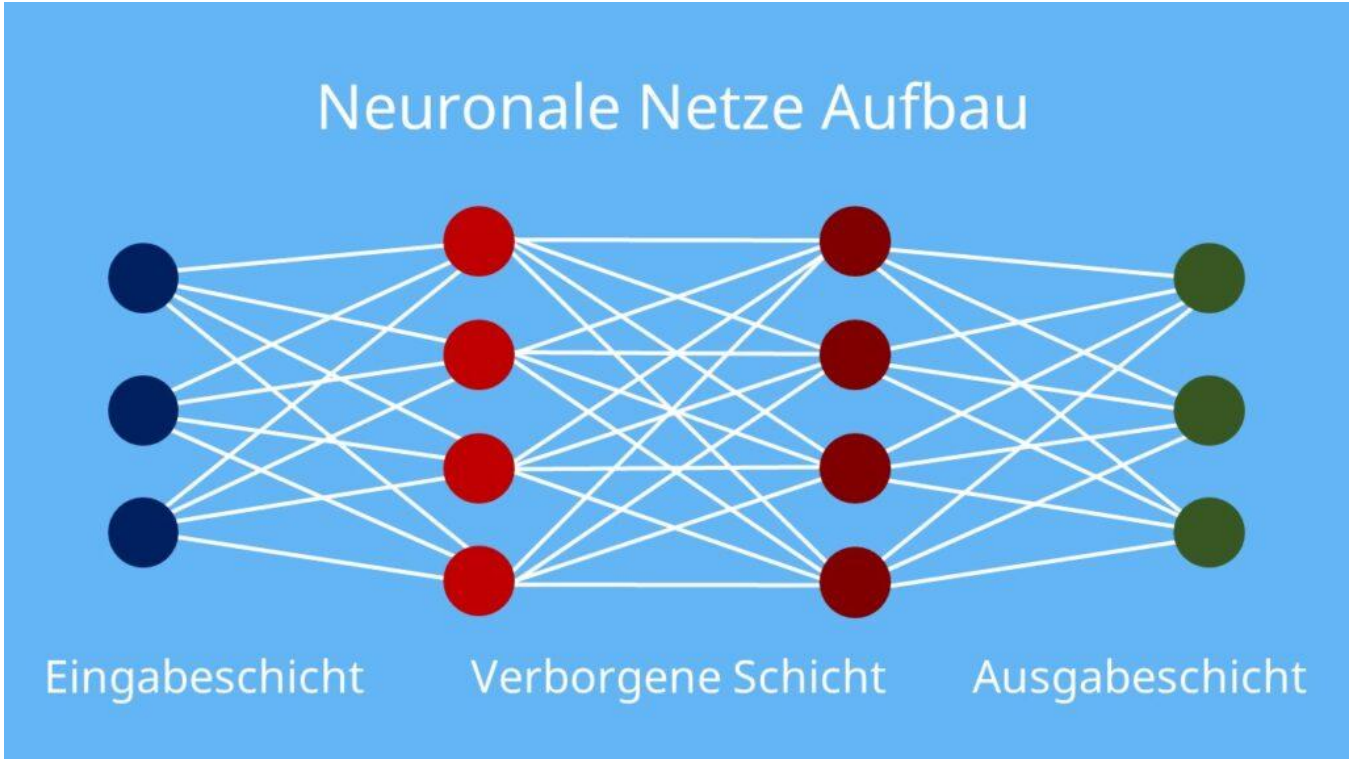
Gesucht / Euer Job

- Layout des KNN: Anzahl Neuronen (Input, Output, Hidden), Anzahl Layer
- Funktionen bei Benutzung des KNN, z.B. Aktivierungsfunktion
- Funktion für Verbesserung des KNN, z.B. Minimierung einer Verlustfunktion
- Anzahl Trainingszyklen, Viel hilft viel?
- Bewertung der Güte des KNN, z.B. Accuracy

(Künstliche) Neuronale Netze

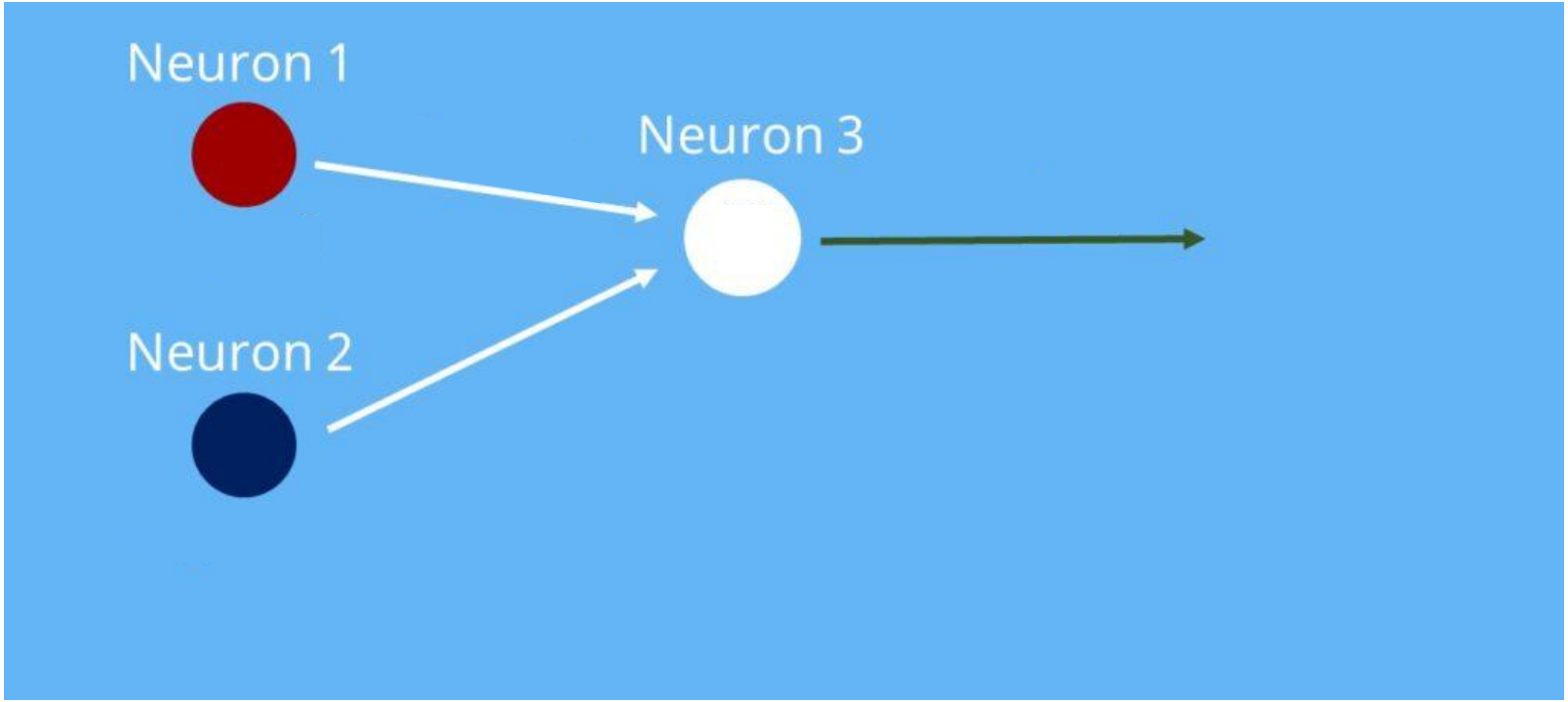
Bild: Pixabay

Neuronales Netz - Aufbau



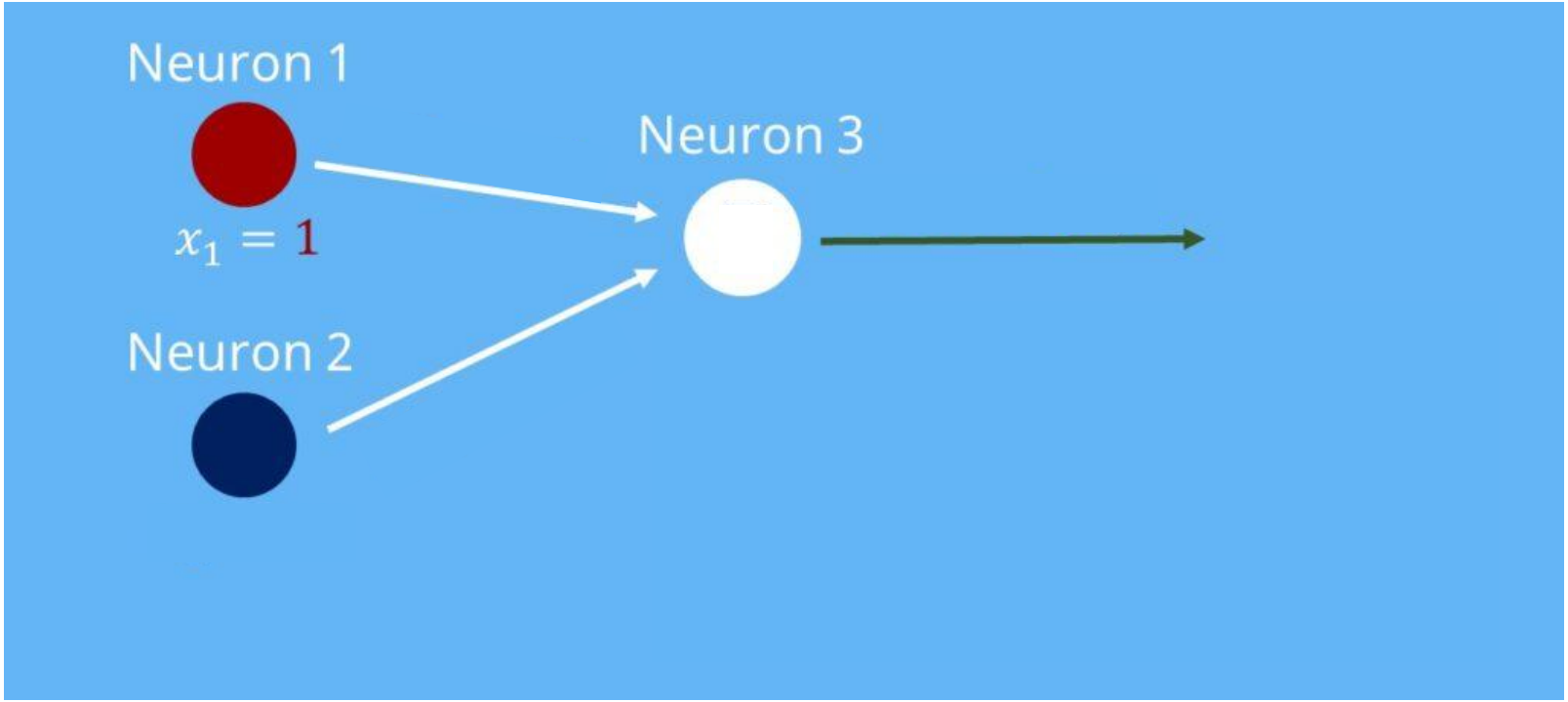
https://studyflix.de/informatik/neuronale-netze-4297

Aktivierung eines Neurons im Neuronalen Netz



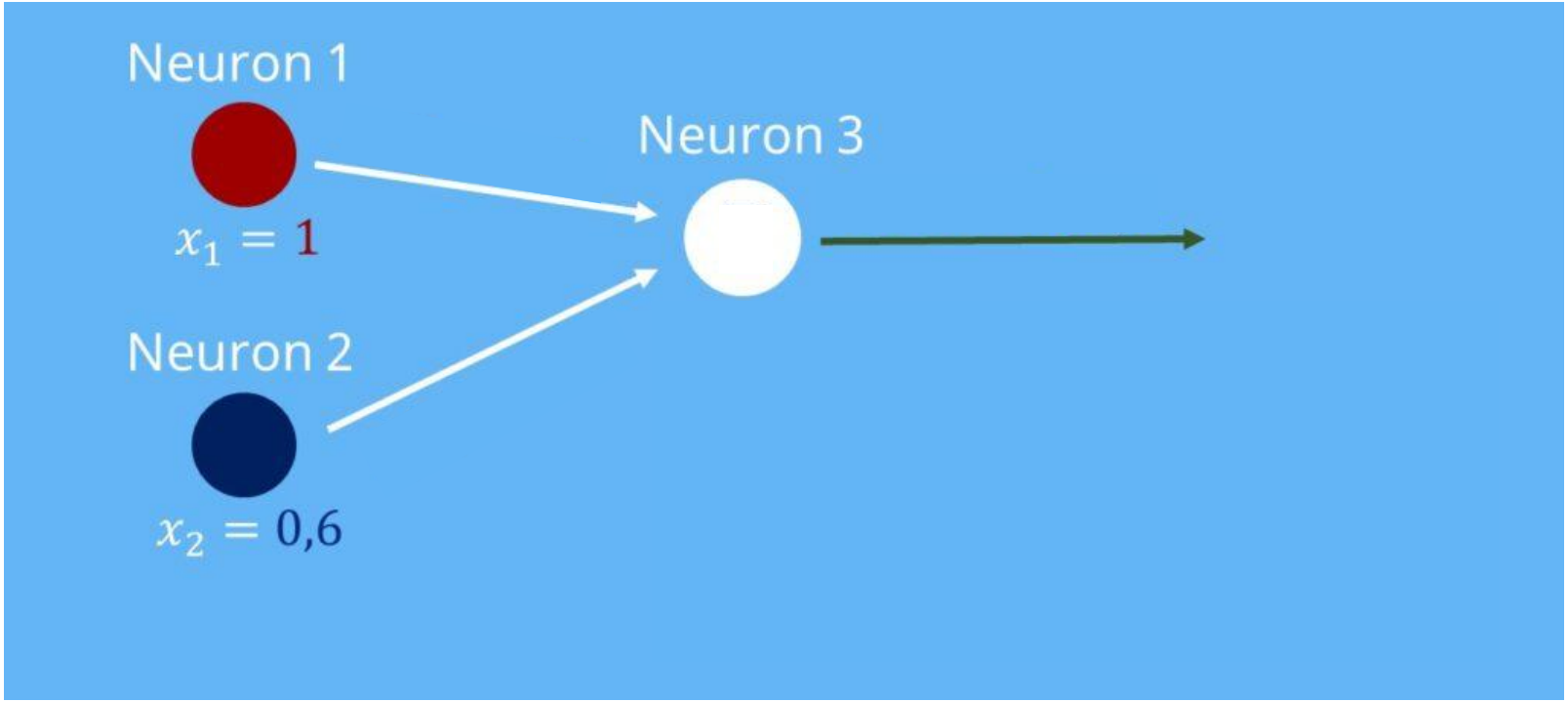
https://studyflix.de/informatik/neuronale-netze-4297

Aktivierung eines Neurons im Neuronalen Netz



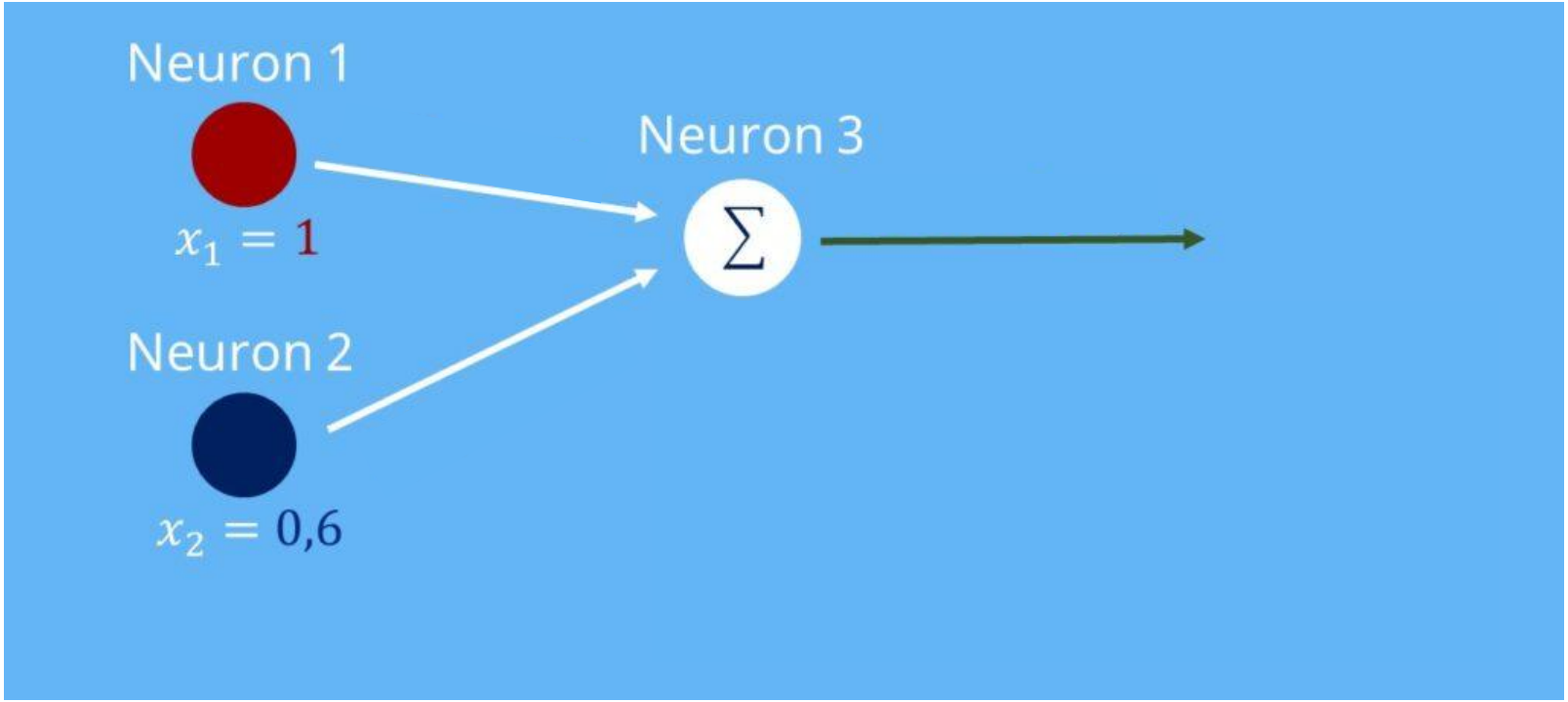
https://studyflix.de/informatik/neuronale-netze-4297

Aktivierung eines Neurons im Neuronalen Netz



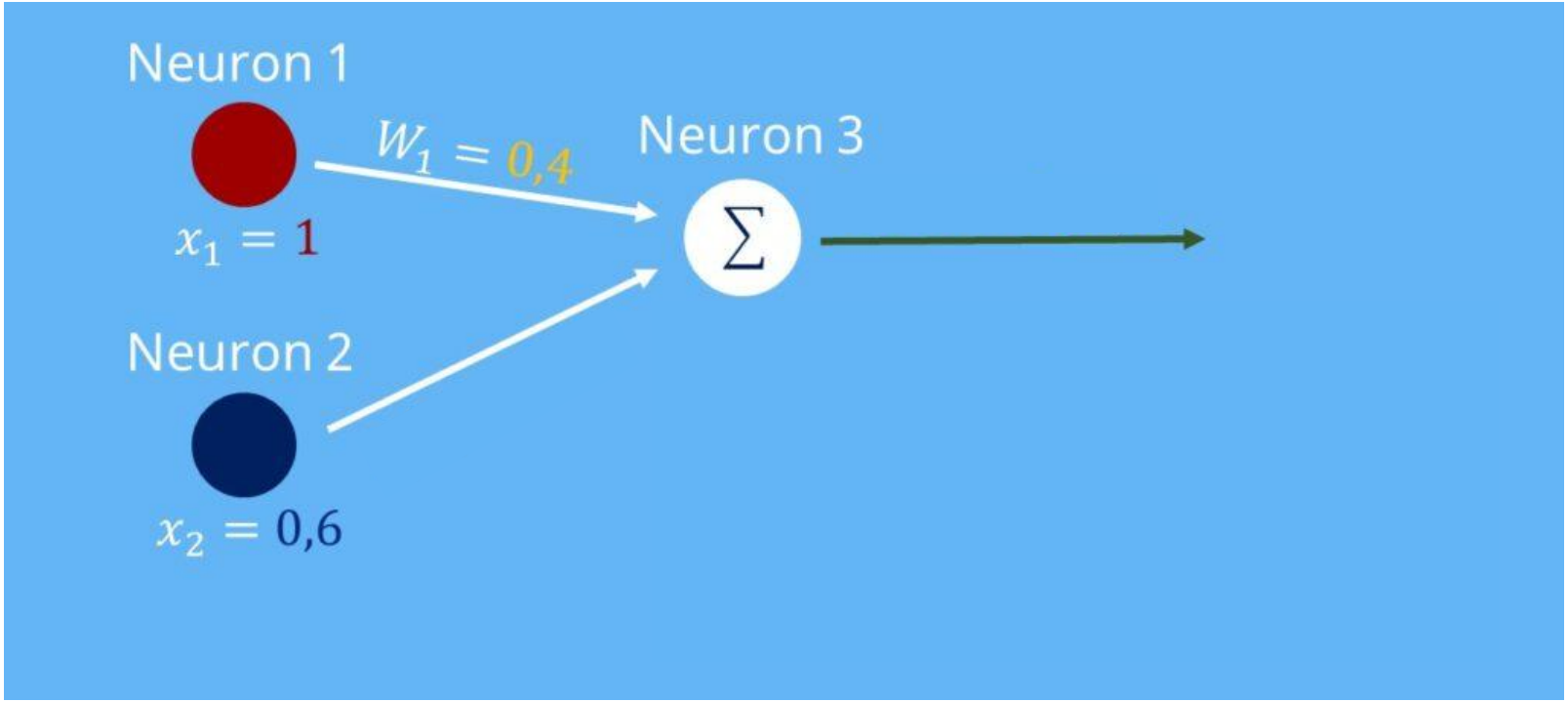
https://studyflix.de/informatik/neuronale-netze-4297

Aktivierung eines Neurons im Neuronalen Netz



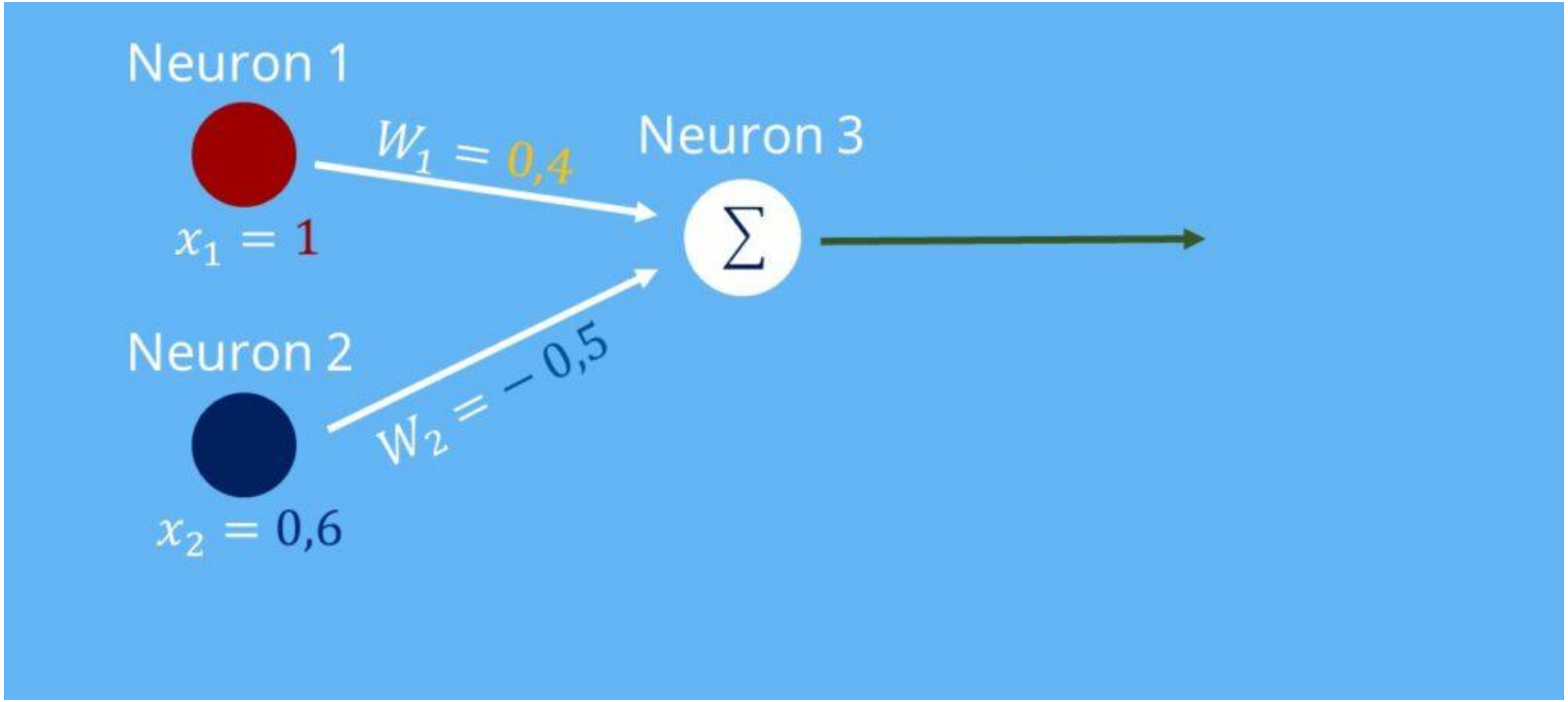
https://studyflix.de/informatik/neuronale-netze-4297

Aktivierung eines Neurons im Neuronalen Netz



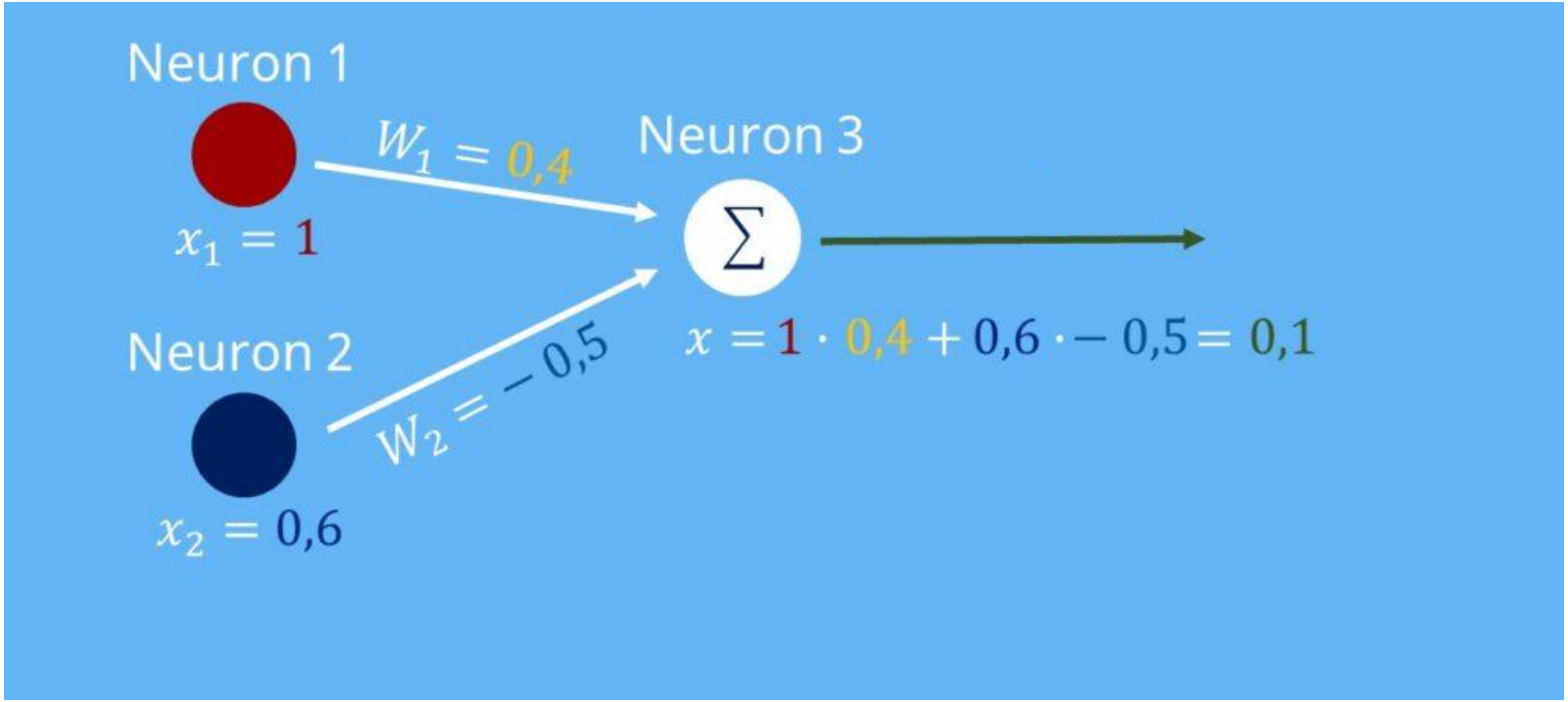
https://studyflix.de/informatik/neuronale-netze-4297

Aktivierung eines Neurons im Neuronalen Netz



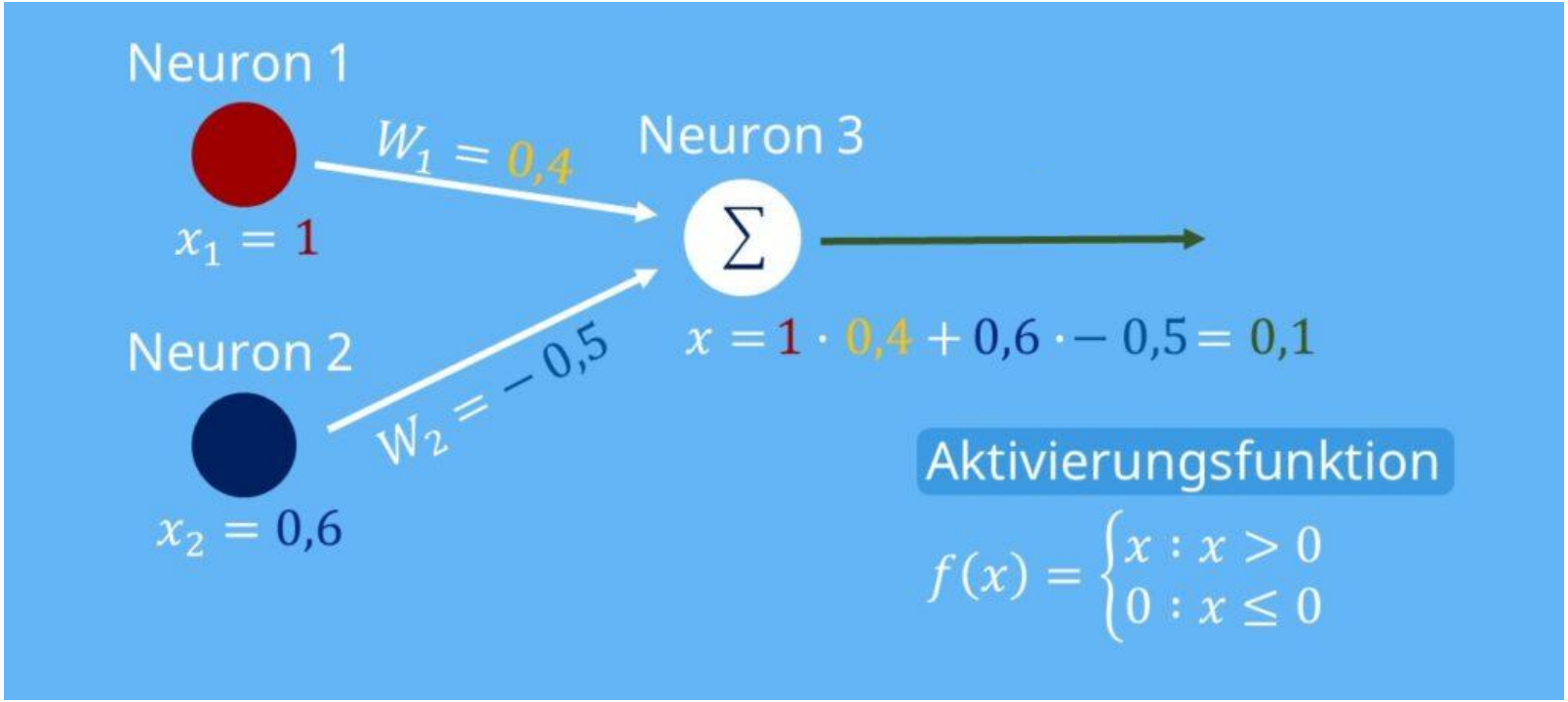
https://studyflix.de/informatik/neuronale-netze-4297

Aktivierung eines Neurons im Neuronalen Netz



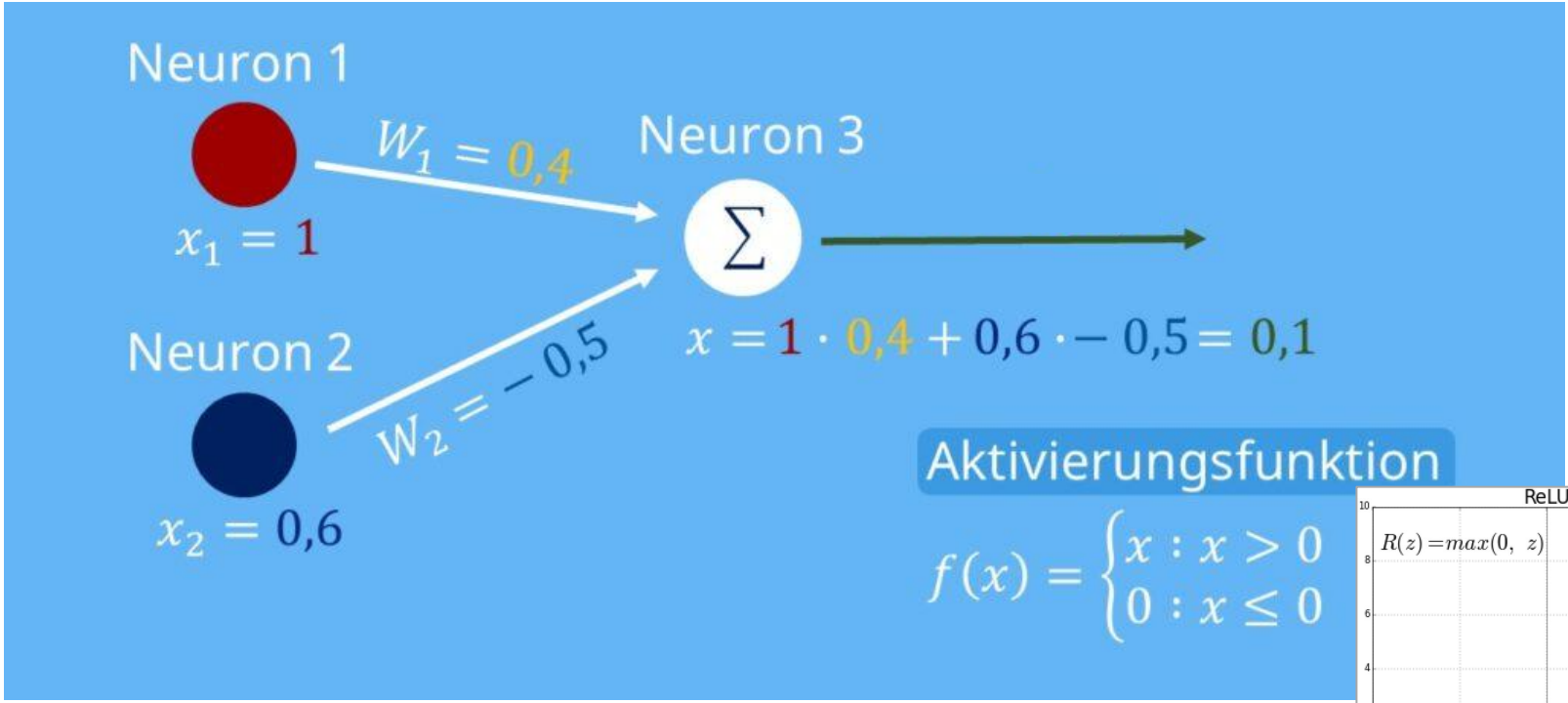
https://studyflix.de/informatik/neuronale-netze-4297

Aktivierung eines Neurons im Neuronalen Netz



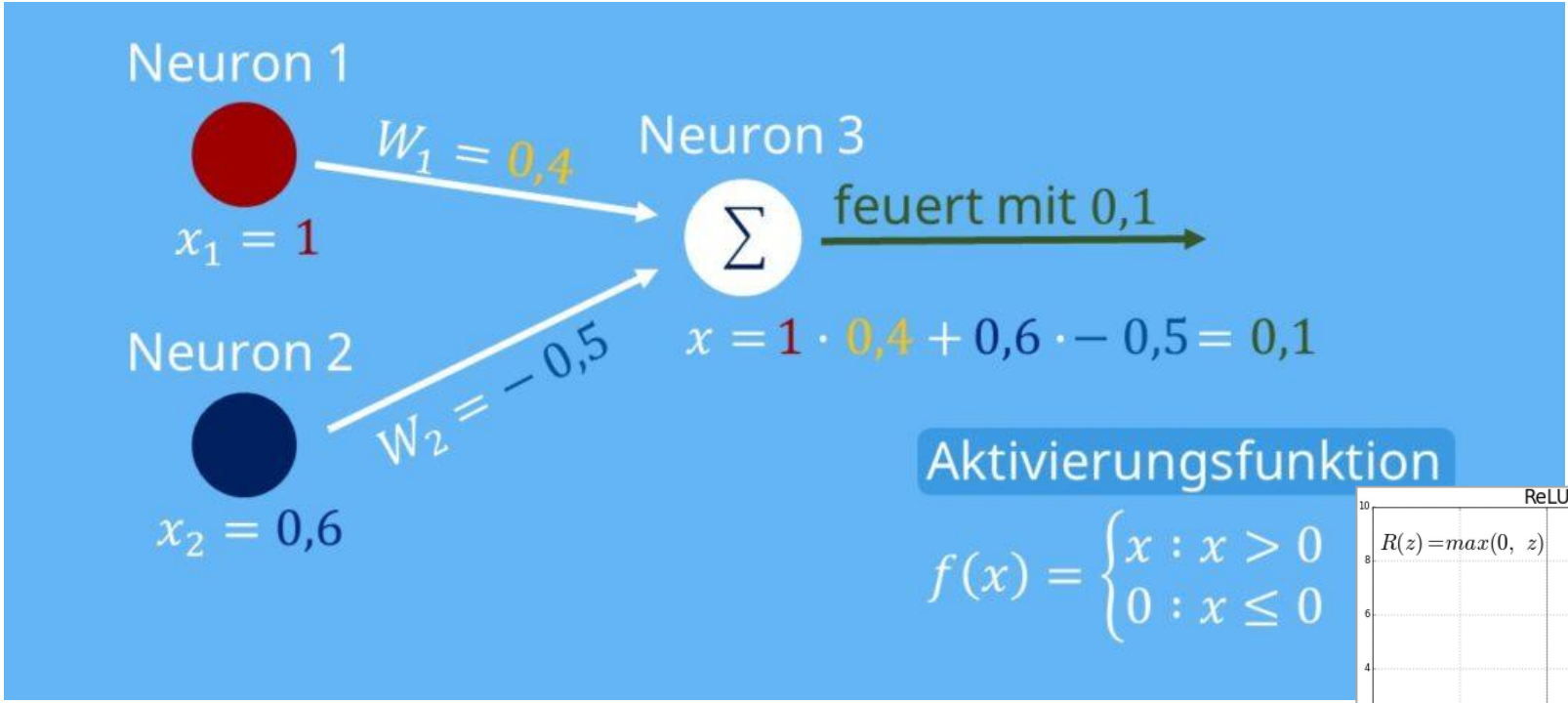
https://studyflix.de/informatik/neuronale-netze-4297

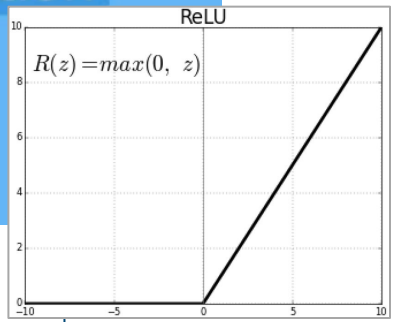
Aktivierung eines Neurons im Neuronalen Netz



https://studyflix.de/informatik/neuronale-netze-4297

Aktivierung eines Neurons im Neuronalen Netz

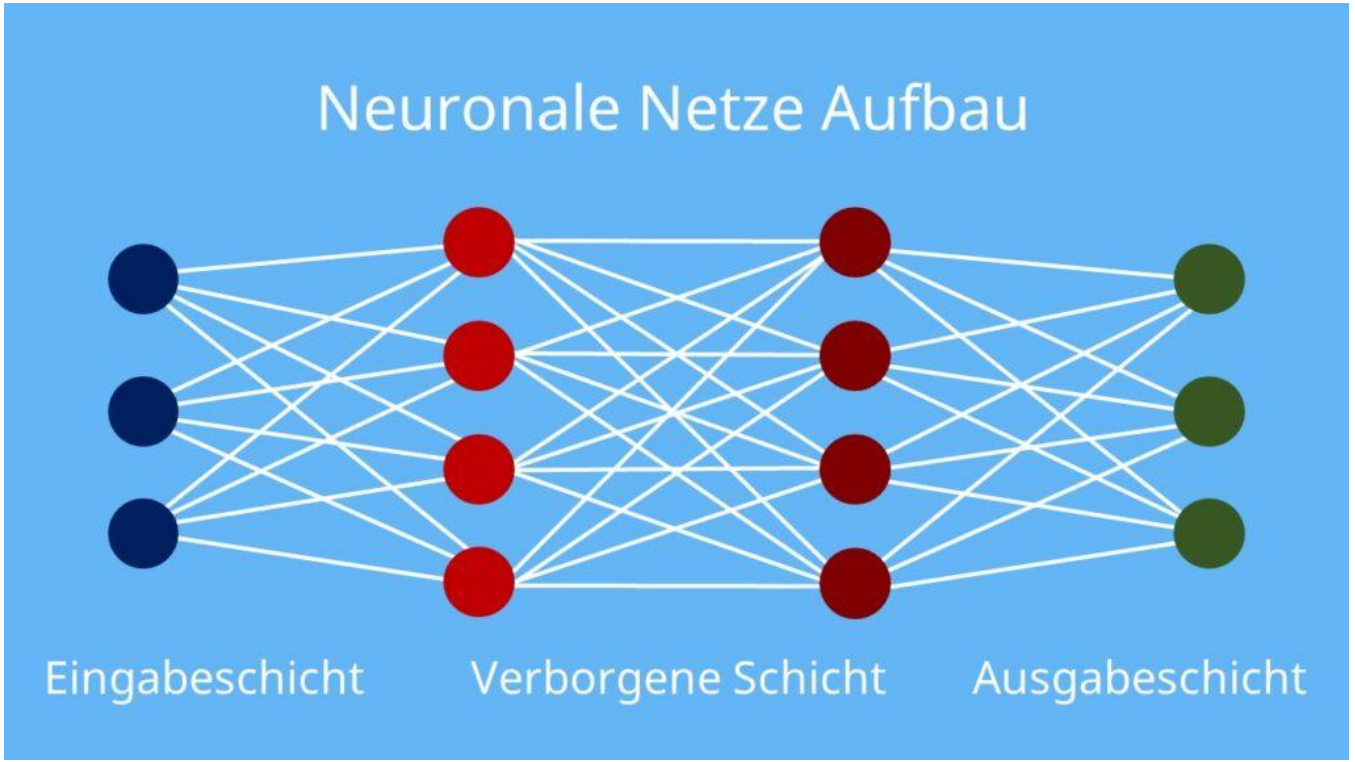




https://studyflix.de/informatik/neuronale-netze-4297

(Forward) Propagation

Bild einer Ziffer



Größte Aktivierung

8

https://studyflix.de/informatik/neuronale-netze-4297

Genauigkeit des Neuronalen Netzes (Accuracy)

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

https://developers.google.com/machine-learning/crash-course/classification/accuracy?hl=de

Es gibt weitere Metriken, um Modelle des maschinellen Lernens zu bewerten:

- Precision
- Recall / Sensitivität
- F1-Score
- Spezifität

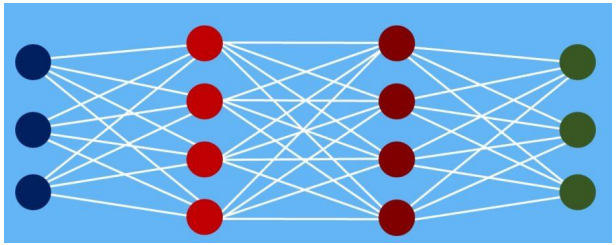
Diese Metriken betrachten wir heute nicht.

https://artemoppermann.com/de/accuracy-precision-recall-f1-score-und-specificity/

Um die **Accuracy zu berechnen**, werden **gelabelte Daten** benötigt, mit denen das Netz **nicht trainiert** worden ist.
-> Testdaten

Training des KNN: Back Propagation

Bild der Ziffer



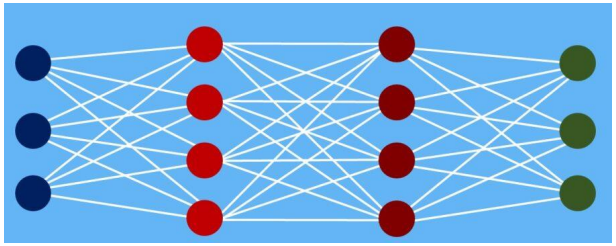
8

Größte Aktivierung

Richtig!

Gewichte verstärken!

Bild der Ziffer



3

Größte Aktivierung

Falsch!

Gewichte verringern!

Training = Gewichte ändern

Bild: Studyflix

Verlustfunktion (Loss)

Für Klassifikationsaufgaben, bei denen das neuronale Netz Wahrscheinlichkeitswerte im Intervall (0, 1) berechnen soll, wird die sog. Kreuzentropie verwendet (engl: Cross-Entropy) .
Die Kreuzentropie für eine berechnete Wahrscheinlichkeitsverteilung $\vec{y}^{(n)}$ und die tatsächliche Wahrscheinlichkeitsverteilung $\vec{t}^{(n)}$ ist definiert gemäß:

$$E_{cross} = -\sum_{m=1}^{M} t_m \cdot log(y_m)$$

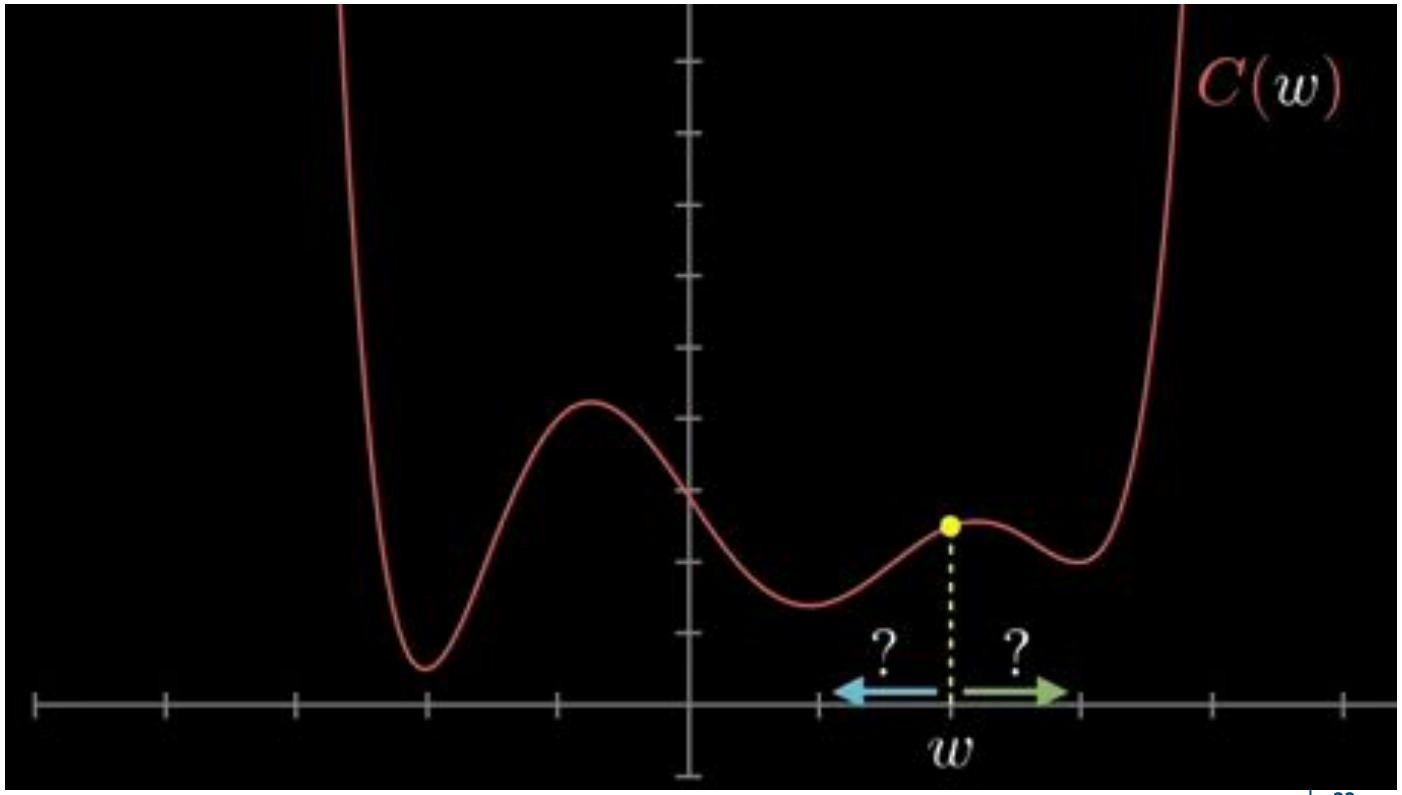
Aber heute: Kein Mathe, Ziel **Implementierung**.

https://artemoppermann.com/de/training-der-kuenstlichen-neuronalen-netze/

Der “Loss” ist **keine Prozentangabe**, die angibt, wie gut das Netz allgemein arbeitet.
Er ist ein **Gütekriterium**, dass die Entscheidung des KNN bewertet.

Erinnerung: Um die Korrektheit des KNN anzugeben, berechnet man die **Accuracy**.

Ausflug: Gradient Descent visualisiert (für Back Propagation)



https://youtu.be/IHZwWFHWa-w?si=5T-jOeEYAoULButi

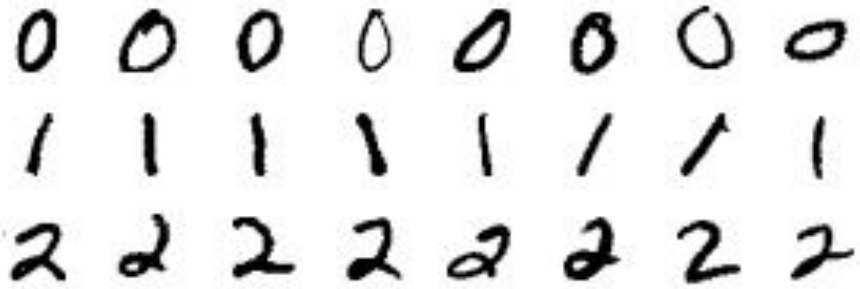
Bild: Wikipedia

MNIST Dataset

Das MNIST Dataset

Die **MNIST-Datenbank** (*Modified National Institute of Standards and Technology database*[1]) ist eine öffentlich verfügbare Datenbank von handgeschriebenen Ziffern. Wobei jede Ziffer als 28 × 28 Pixel großes Graustufen-Bild gespeichert ist[1]. Die MNIST-Datenbank besteht aus 60.000 Beispielen im Trainingsdatensatz und 10.000 Beispielen im Testdatensatz.

https://de.wikipedia.org/wiki/MNIST-Datenbank



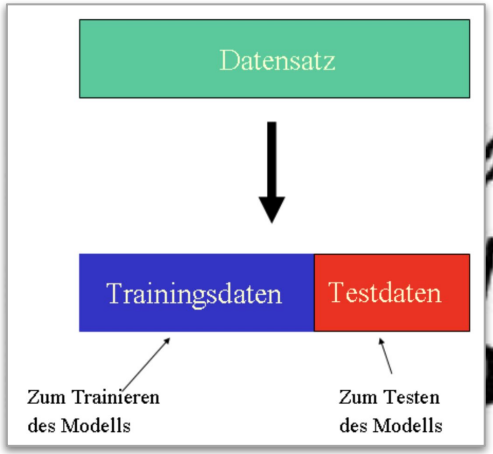
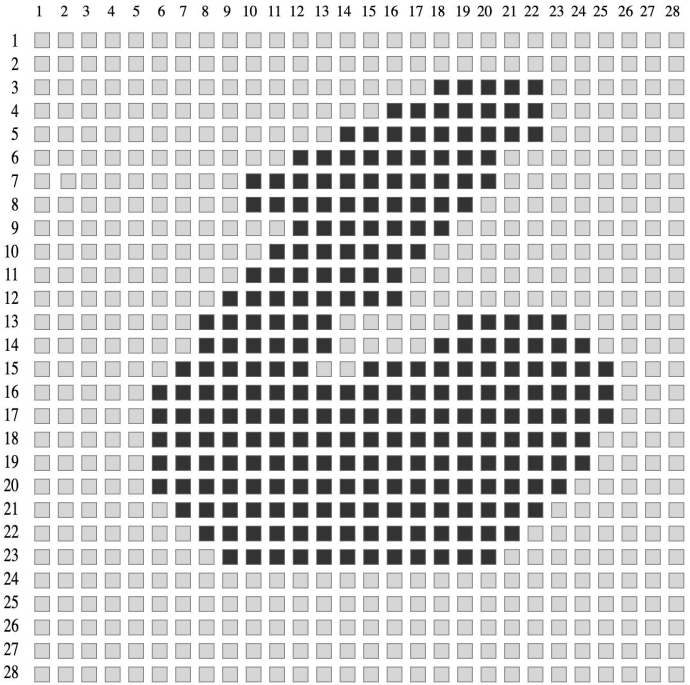


Bild: Wikipedia

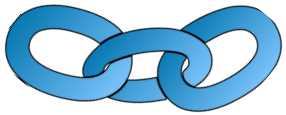
Ein Eintrag aus dem MNIST-Dataset

Bild



28 x 28 = 784 Pixel

Label



6

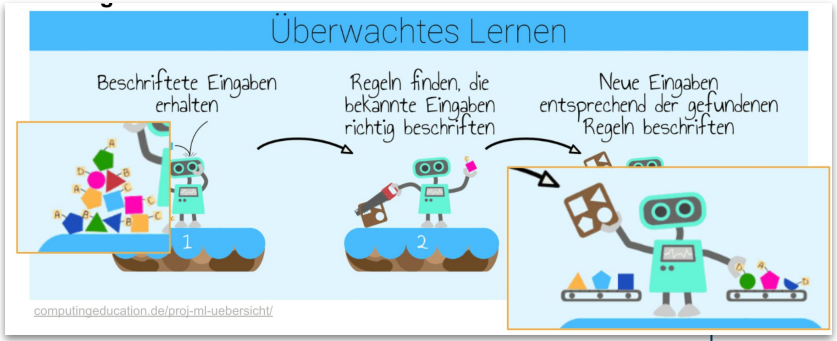


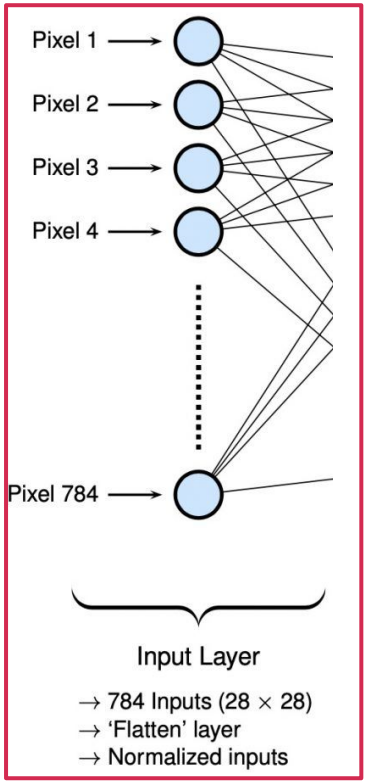
Bild: Pixabay

Aufgabe: Handgeschriebene Ziffern erkennen

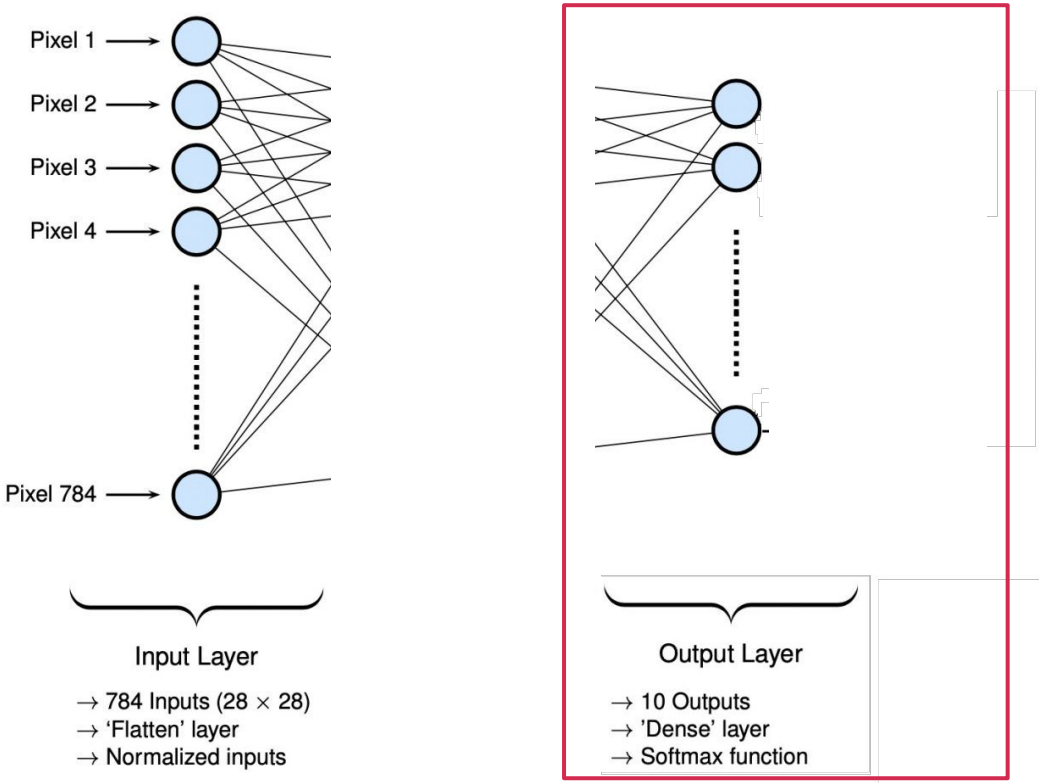
Mit einem Teil der MNIST-Ziffern ein Neuronales Netz trainieren (**Trainingsdaten**).

Danach mit einem anderen Teil der MNIST-Ziffern bewerten, wie gut das Neuronale Netz arbeitet (**Testdaten**).

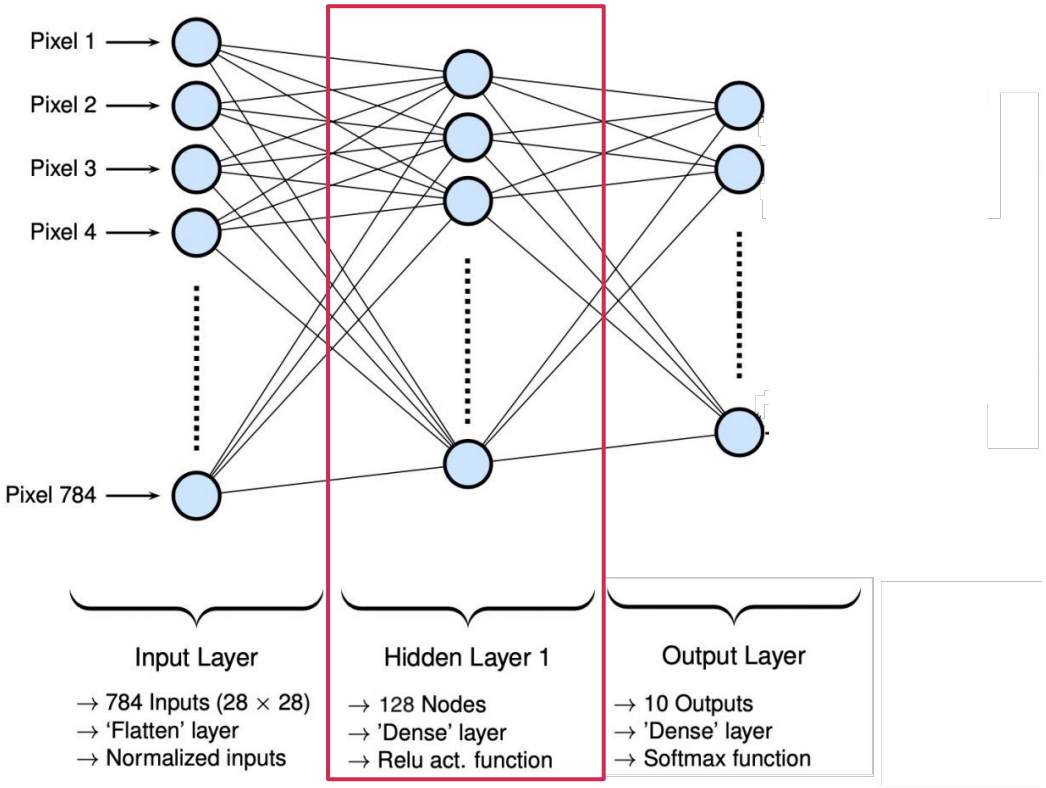
Neuronales Netzwerk für die MNIST-Aufgabe - Input Layer

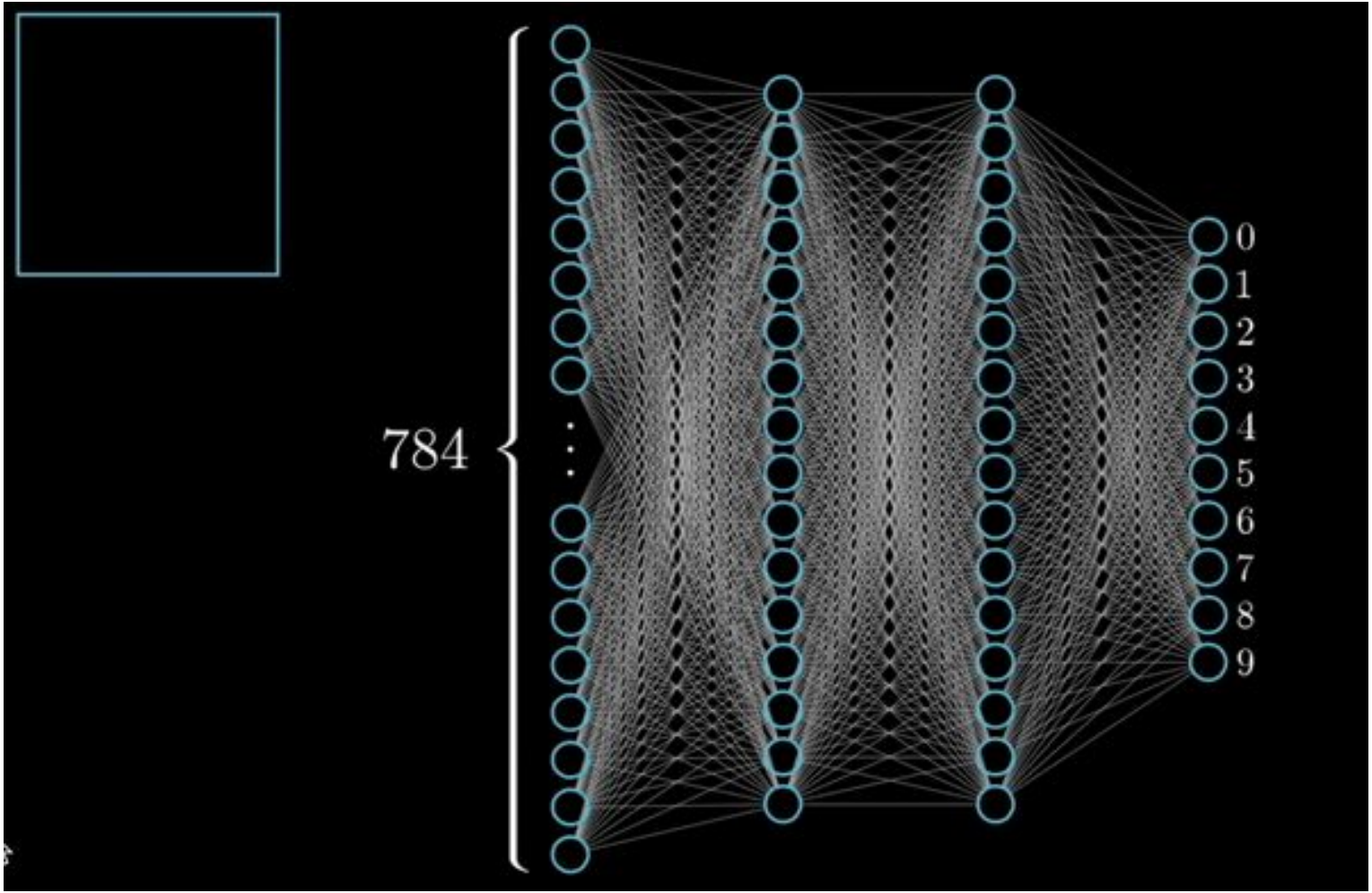


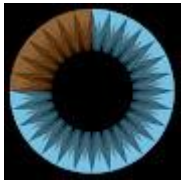
Neuronales Netzwerk für die MNIST-Aufgabe - Output Layer



Neuronales Netzwerk für die MNIST-Aufgabe - Hidden Layer







3Blue1Brown

https://youtu.be/aircAruvnKk?si=CqhTWazY_Vq0UmVa

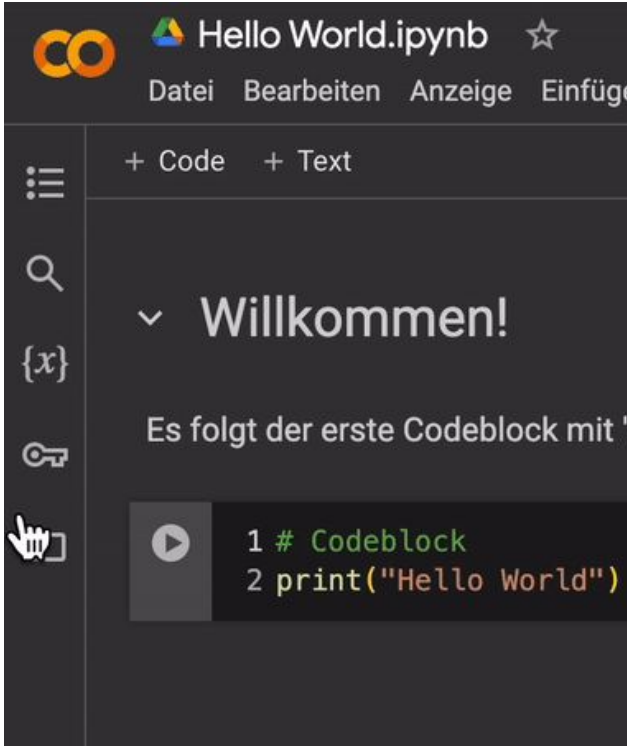
Implementierung

Bilder: Pixabay

Jupyter Notebook (in Google Colab)

- Jupyter Notebooks sind eine Laufzeitumgebung für **Python.** Sie laufen in verschiedenen **Entwicklungsumgebungen**, z.B. Visual Studio Code, als auch im **Webbrowser**
- Es können sich **Code- und Textblöcke** abwechseln. Projekte können erklärt werden.
- Codeblöcke können **ausgeführt** werden. Die Ausgabe erscheint unter dem Block.
- Viele KI-Projekte insbesondere zu **Datascience**, wo oft Daten visualisiert werden müssen, nutzen Jupyter Notebooks.

- **Google Colab** ist eine **Online-Umgebung** für Jupyter Notebooks
- Genutzt wird die Rechenleistung der **GoogleServer**. Das eigene Endgerät wird nicht genutzt.
- Colab kann **kostenlos** genutzt werden. Gegen **Aufpreis** kann Serverleistung dazugekauft werden.

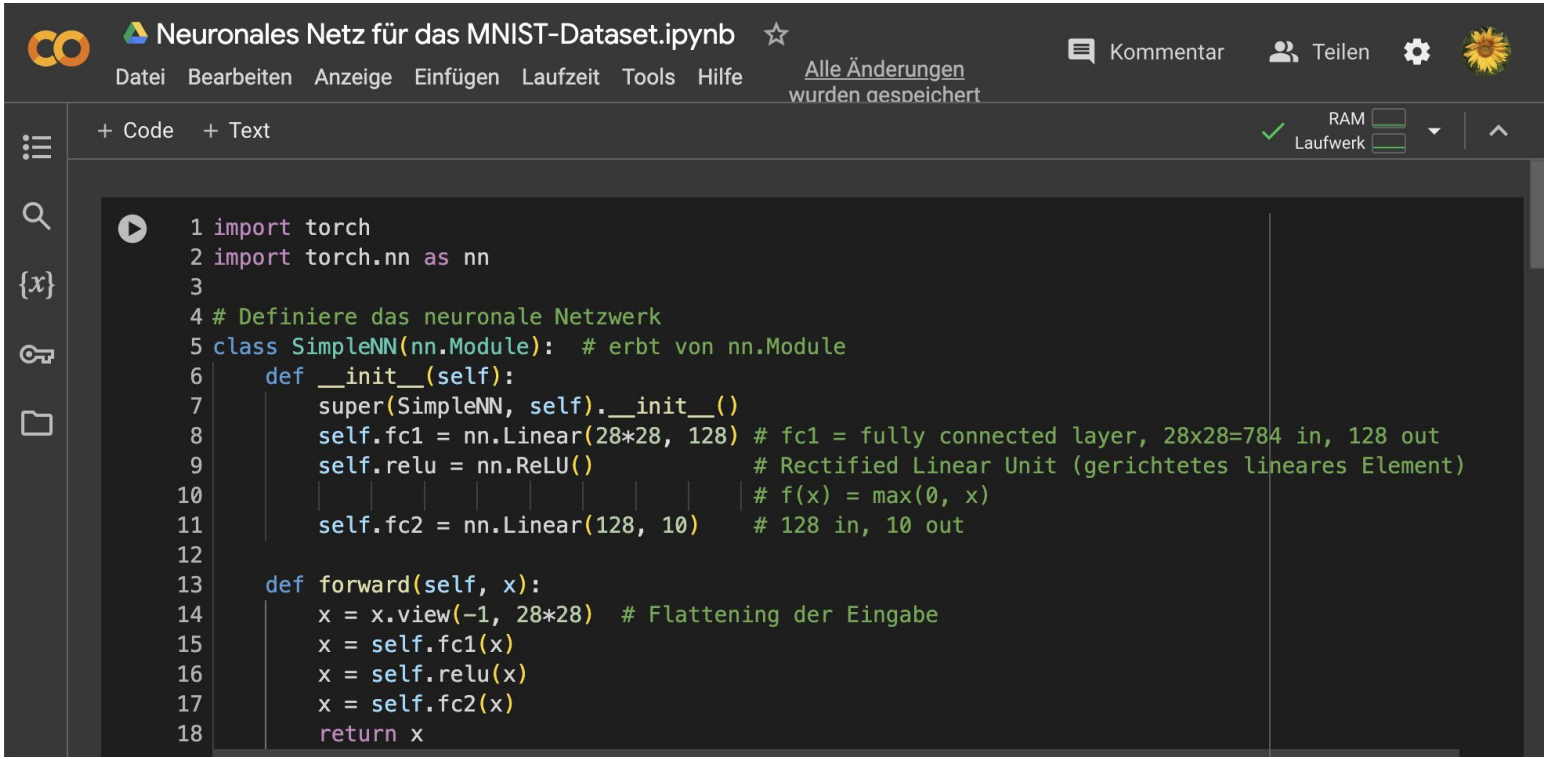


Bibliothek PyTorch

PyTorch ist eine auf Maschinelles Lernen ausgerichtete Open-Source-Programmbibliothek für die Programmiersprache Python, basierend auf der in Lua geschriebenen Bibliothek Torch, die bereits seit 2002 existiert.[2][3][4] Entwickelt wurde PyTorch von dem Facebook-Forschungsteam für künstliche Intelligenz.[5][6][7] Im September 2022 wurde Pytorch mittels der neugegründeten Pytorch Foundation Teil der Linux Foundation.[8]

https://de.wikipedia.org/wiki/PyTorch

Implementierung im Jupyter Notebook (in Google Colab)



https://colab.research.google.com/drive/1siDxH4qgLsR64tf8Qfmn3jM6j16pCHs9?usp=sharing

Block 1 Definition des Neuronalen Netzes (Klasse definieren)

```python
# Block 1
import torch
import torch.nn as nn

# Definiere das neuronale Netzwerk
class SimpleNN(nn.Module):  # erbt von nn.Module
    def __init__(self):
        super(SimpleNN, self).__init__()
        self.fc1 = nn.Linear(28*28, 128) # fc1 = fully connected layer, 28x28=784 in, 128 out
        self.relu = nn.ReLU()            # Rectified Linear Unit (gerichtetes lineares Element)
                                         # f(x) = max(0, x)
        self.fc2 = nn.Linear(128, 10)    # 128 in, 10 out

    def forward(self, x):
        x = x.view(-1, 28*28)  # Flattening der Eingabe
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        return x

# Model für das Neuronales Netz instantiieren
model = SimpleNN()
```

Selber programmieren nach Block 1 - Beispiel

```
1 # Selber programmieren nach Block 1
2 print(model)
```

```
SimpleNN(
  (fc1): Linear(in_features=784, out_features=128, bias=True)
  (relu): ReLU()
  (fc2): Linear(in_features=128, out_features=10, bias=True)
)
```

Block 2 Transformationsfunktion definieren

```
# Block 2
from torchvision import transforms

# Lade das MNIST-Dataset und wende Transformationen an
transform_function = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.5,), (0.5,))])
```

Selber programmieren nach Block 2 - Beispiel

```
# Selber programmieren nach Block 2
print(transform_function)
```

```
Compose(
    ToTensor()
    Normalize(mean=(0.5,), std=(0.5,))
)
```

Block 3 Trainings- und Testdaten herunterladen

```python
# Block 3
from torchvision import datasets

training_data = datasets.MNIST(root='./data', train=True, download=True, transform=transform_function)
test_data = datasets.MNIST(root='./data', train=False, download=True, transform=transform_function)
```

```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz to ./data/MNIST/raw/train-images-idx3-ubyte.gz
100%|██████████| 9912422/9912422 [00:00<00:00, 222824510.25it/s]Extracting ./data/MNIST/raw/train-images-idx3-ubyte.gz to


Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz to ./data/MNIST/raw/train-labels-idx1-ubyte.gz
100%|██████████| 28881/28881 [00:00<00:00, 69180864.55it/s]
Extracting ./data/MNIST/raw/train-labels-idx1-ubyte.gz to ./data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz to ./data/MNIST/raw/t10k-images-idx3-ubyte.gz
100%|██████████| 1648877/1648877 [00:00<00:00, 65769796.36it/s]
Extracting ./data/MNIST/raw/t10k-images-idx3-ubyte.gz to ./data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz to ./data/MNIST/raw/t10k-labels-idx1-ubyte.gz
100%|██████████| 4542/4542 [00:00<00:00, 5729482.34it/s]
Extracting ./data/MNIST/raw/t10k-labels-idx1-ubyte.gz to ./data/MNIST/raw
```

https://pytorch.org/vision/stable/generated/torchvision.datasets.MNIST.html#torchvision.datasets.MNIST

Selber programmieren nach Block 3 - Beispiel

```python
# Selber programmieren nach Block 3
print(len(training_data))
print(len(test_data))
print(training_data[0])
```

```
60000
10000
(tensor([[[-1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,
          -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,
          -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,
          -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000],
         [-1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,
          -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,
          -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,
          -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000],
         [-1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,
          -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,
          -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,
          -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000],
         [-1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,
          -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,
```

Block 4 Trainings- und Testdaten in Datenstruktur laden

```
1 # Block 4
2 from torch.utils.data import DataLoader
3
4 training_loader = DataLoader(dataset=training_data, batch_size=64, shuffle=True)
5 test_loader = DataLoader(dataset=test_data, batch_size=64, shuffle=False)
```

Dataset stores the samples and their corresponding labels, and **DataLoader** wraps an iterable around the Dataset.

https://pytorch.org/tutorials/beginner/basics/quickstart_tutorial.html?highlight=dataloader

Bei Batchsize 64 ergeben sich 938 Batches im Tainingsset.

Selber programmieren nach Block 4 - Beispiel

```python
# Selber programmieren nach Block 4
print(training_loader)
print(len(training_loader))
print(len(test_loader))
print(len(training_loader)*64)
print(len(test_loader)*64)
```

```
<torch.utils.data.dataloader.DataLoader object at 0x7df9683db5e0>
938
157
60032
10048
```

Block 5 Neuronales Netz instantiieren, Loss und Optimierer definieren

```
1 # Block 5
2 import torch.optim as optim
3
4 # Verlustfunktion (Loss) und Optimierer definieren
5 loss_function = nn.CrossEntropyLoss()
6 optimization_algorithm = optim.SGD(model.parameters(), lr=0.01) # SGD = stochastic gradient descent,
7                                                                 # lr = learning rate
```

https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html

https://pytorch.org/docs/stable/generated/torch.optim.SGD.html

SGD Implements stochastic gradient descent

Selber programmieren nach Block 5 - Beispiel

```
1 # Selber programmieren nach Block 5
2 print(loss_function)
3 print(optimization_algorithm)
```

```
CrossEntropyLoss()
SGD (
Parameter Group 0
    dampening: 0
    differentiable: False
    foreach: None
    lr: 0.01
    maximize: False
    momentum: 0
    nesterov: False
    weight_decay: 0
)
```

Block 6 Neuronales Netz trainieren

```python
# Block 6
# Trainingsloop
num_epochs = 1

print("Starte Training")
for epoch in range(num_epochs):
    for batch_index, (images, labels) in enumerate(training_loader):
        optimization_algorithm.zero_grad()
        outputs = model(images)     # forward wird aufgerufen, wenn data durch das Model geleitet wird
                # outputs = bis zu 64 (batch_size) Wahrscheinlichkeitsverteilungen für jeweils all 10 Ziffern
                # [-4.8741e+00,  6.0430e+00,  1.4008e+00,  1.1718e+00, -2.7590e+00,
                #  3.0154e-01, -4.9669e-01,  6.0514e-01,  8.4430e-01, -7.3287e-01]
        loss = loss_function(outputs, labels)
        loss.backward()
        optimization_algorithm.step()

        if batch_index % 100 == 0:
            print(f'Epoch {epoch+1}/{num_epochs}, Batch {batch_index}/{len(training_loader)}, Loss: {loss.item()}')
print("Traning fertig!")
```

```
Epoch 1/1, Batch 0/938, Loss: 0.16773580014705658
Epoch 1/1, Batch 100/938, Loss: 0.22459541261196136
Epoch 1/1, Batch 200/938, Loss: 0.2589246332645416
Epoch 1/1, Batch 300/938, Loss: 0.2684001922607422
```

https://inf-schule.de/informatiksysteme/kuenstliche-intelligenz/maschinelles-lernen/maschinelles_lernen

Selber programmieren nach Block 6 - Beispiel

```python
# Selber programmieren nach Block 6
print(len(outputs))
print(outputs[0])
print(labels[0])
print(len(labels))
print(images[0])
```

```
32
tensor([ 0.7932, -2.6793,  2.3830, -0.8346,  2.2291, -0.6511,  6.6040, -3.6512,
         0.9268, -2.0978], grad_fn=<SelectBackward0>)
tensor(6)
32
tensor([[[-1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,
          -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,
          -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,
          -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000],
         [-1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,
          -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,
          -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,
          -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000],
```

Block 7 Trainiertes Neuronales Netz mit Testdaten evaluieren

```python
# Block 7
# Evaluierung auf dem Testdatensatz
model.eval()
correct = 0
total = 0

with torch.no_grad():
    for image, label in test_loader:
        outputs = model(image)
        _, predicted = torch.max(outputs.data, 1)
        total += label.size(0)
        correct += (predicted == label).sum().item()

accuracy = correct / total
print(f'Genauigkeit auf dem Testdatensatz: {accuracy * 100:.2f}%')
```

```
Genauigkeit auf dem Testdatensatz: 92.98%
```

Selber programmieren nach Block 7 - Beispiel

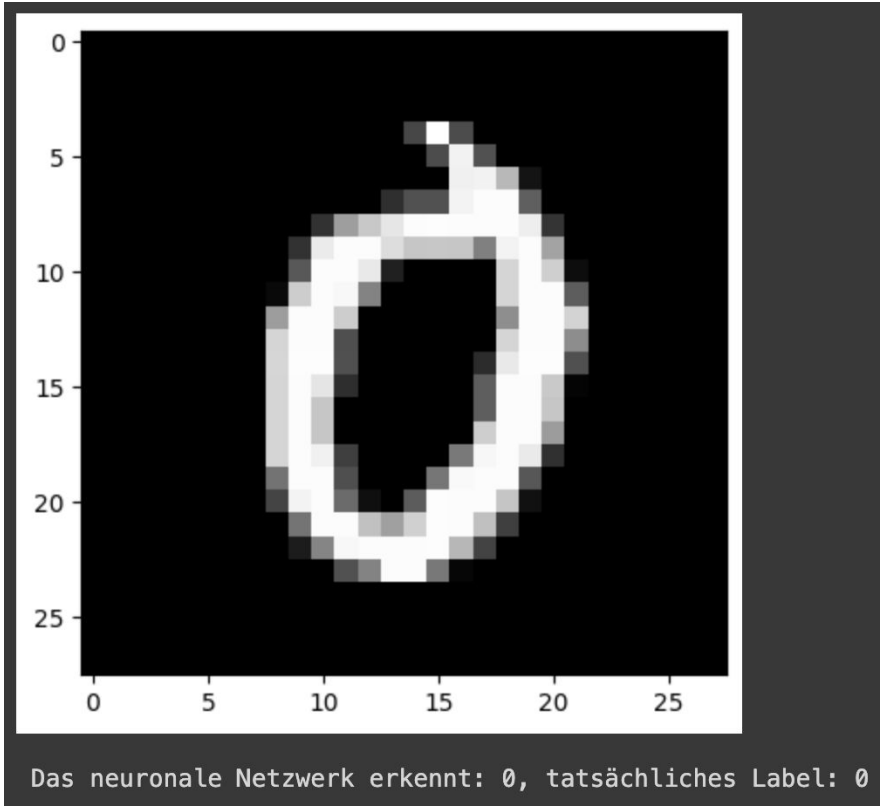
```
1 # Selber programmieren nach Block 7
2 print(correct)
3 print(total)
```

```
8936
10000
```

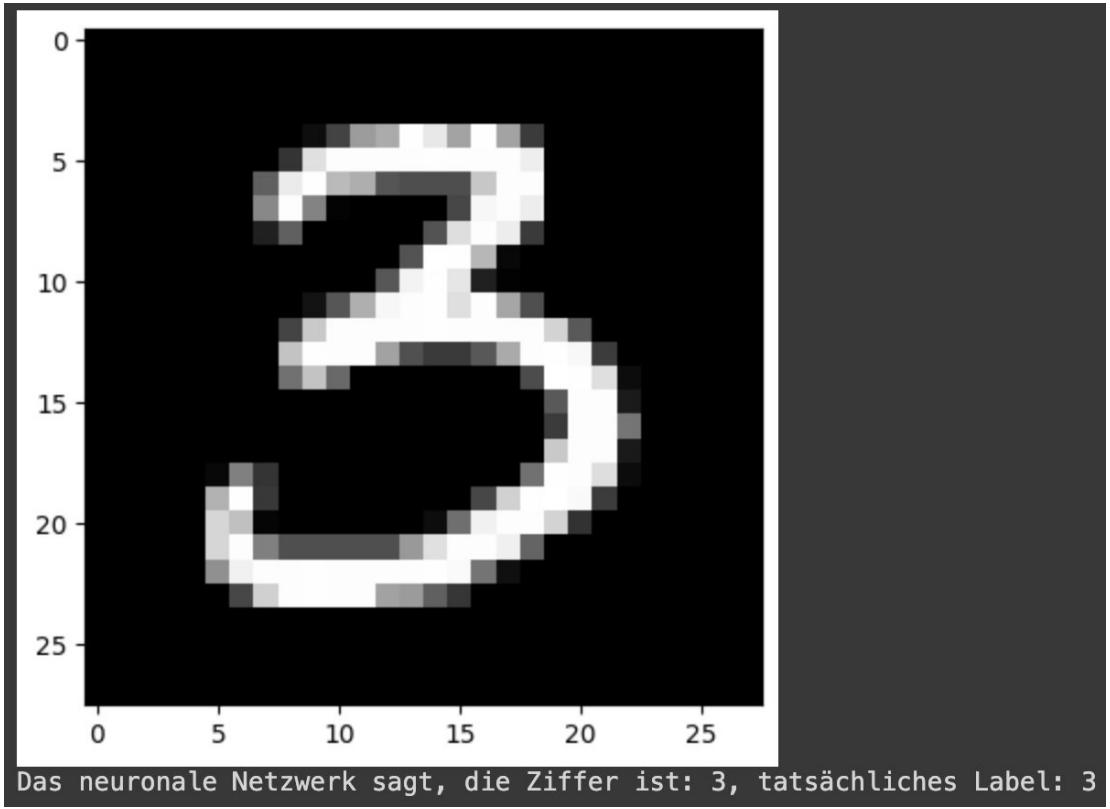
8 Zufälliges Bild nehmen und vom NN erkennen lassen

```python
# Block 8
import matplotlib.pyplot as plt
import random

random_index = random.randint(0, len(test_data) - 1)
random_image, random_label = test_data[random_index]

plt.imshow(random_image[0], cmap='gray')
plt.show()

# Vorhersage für das zufällige Bild
model.eval()
with torch.no_grad():
    output = model(random_image.view(-1, 28*28))
    _, predicted = torch.max(output, 1)

print("\n", f'Das neuronale Netzwerk erkennt: {predicted.item()}, tatsächliches Label: {random_label}')
```

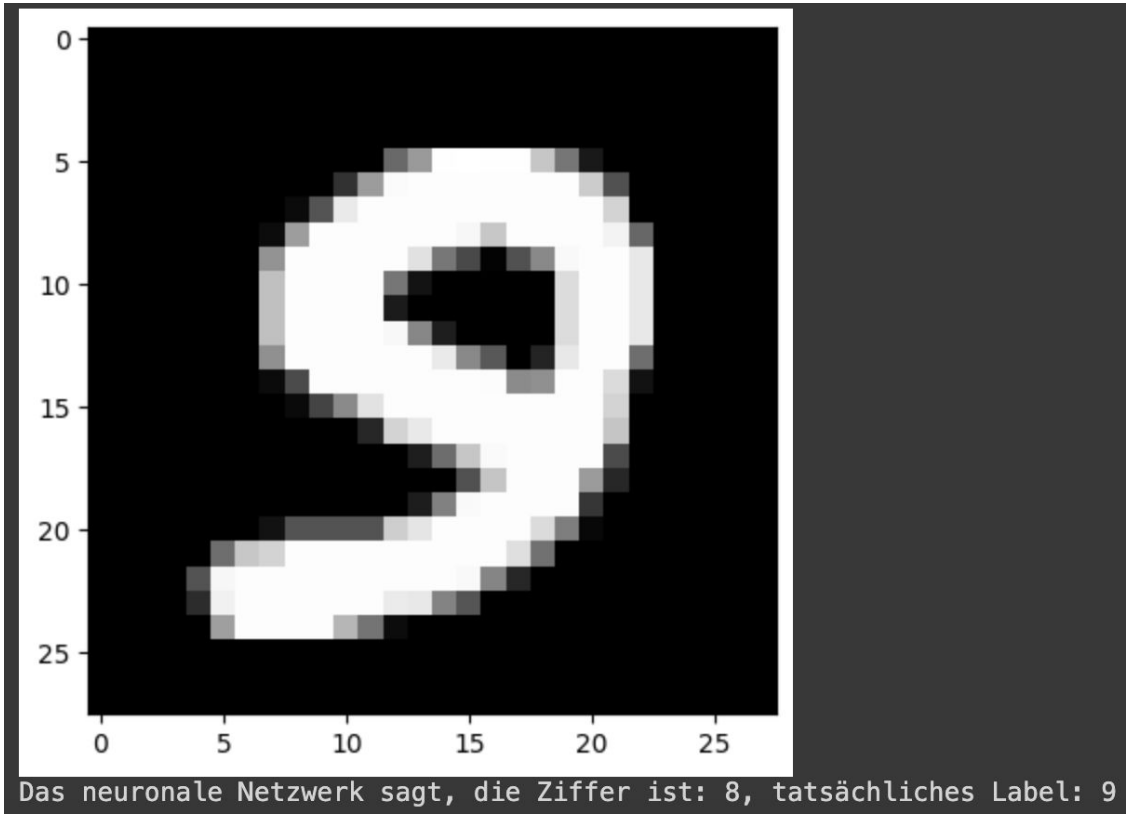
Gewähltes Bild und erkannte Zahl - richtig



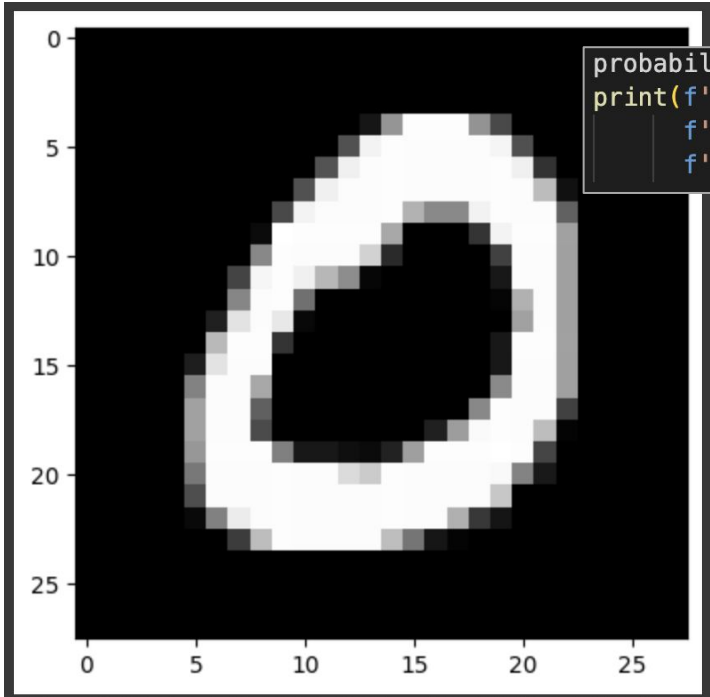
Gewähltes Bild und erkannte Zahl - richtig



Gewähltes Bild und erkannte Zahl - falsch



Gewähltes Bild und erkannte Zahl - richtig mit Wahrscheinlichkeit




```
probability_of_predicted_class = softmax_probs[0, predicted.item()] * 100
print(f'Das neuronale Netzwerk sagt, die Ziffer ist: {predicted.item()}, '
      f'tatsächliches Label: {random_label}, '
      f'Wahrscheinlichkeit der Erkennung: {probability_of_predicted_class:.8f}%')
```

```
Das neuronale Netzwerk sagt, die Ziffer ist: 0, tatsächliches Label: 0
Das neuronale Netzwerk sagt, die Ziffer ist: 0, tatsächliches Label: 0, Wahrscheinlichkeit der Erkennung: 0.00001000%
```

Abschluss

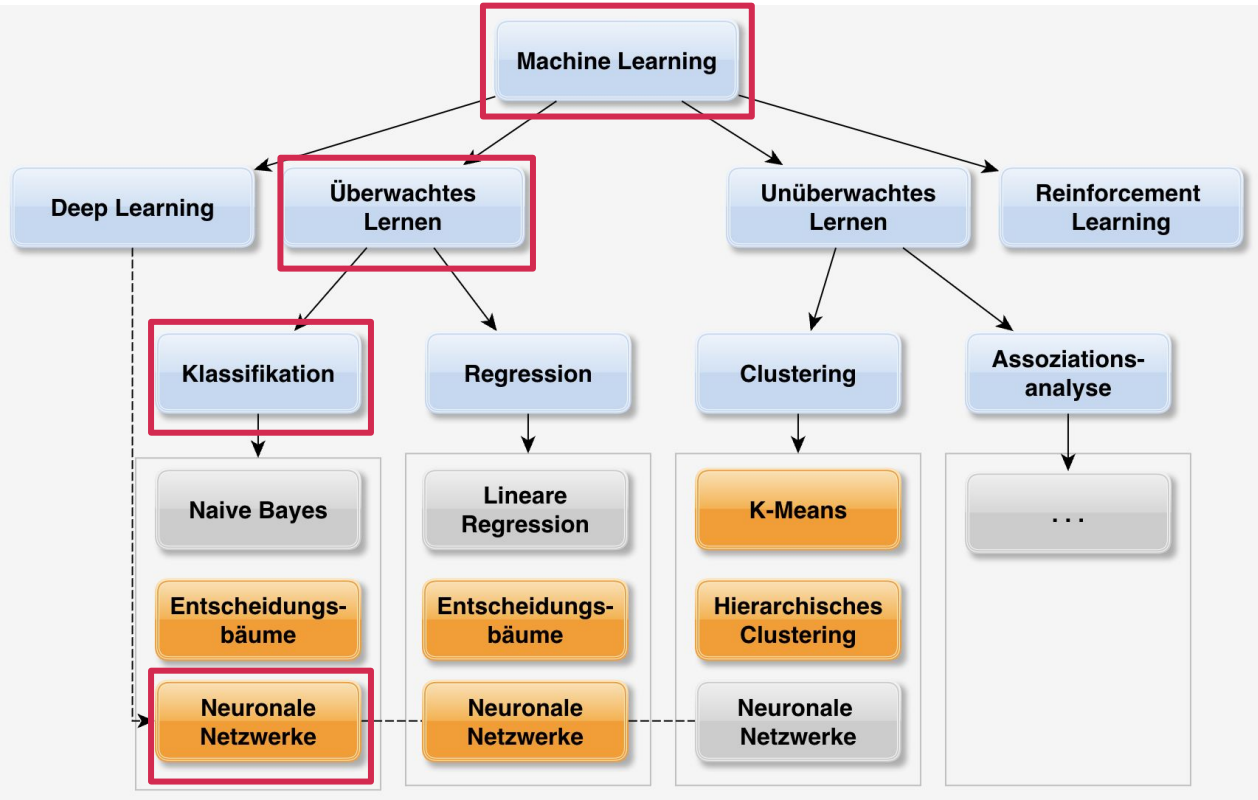
Bemerkungen

- Das “Netz” ist nur eine Visualisierung einer mathematischen Funktion mit **sehr vielen** Parametern.
- Die Gewichte im Neuronalen Netz haben keine “Bedeutung”. Damit sind die Entscheidungen des Neuronalen Netzes auch **nicht erklärbar.** Das ist ein Nachteil - insbesondere, wenn das Netz sensible Entscheidungen treffen soll, z.B. im medizinischen Bereich.
- Manche Fragen sollte man keinem Computer stellen. Bei KI-Systemen ist man sehr schnell bei **Ethik und Moral**, z.B. beim autonomen Fahren.

CPEC CENTER FOR PERSPICUOUS COMPUTING

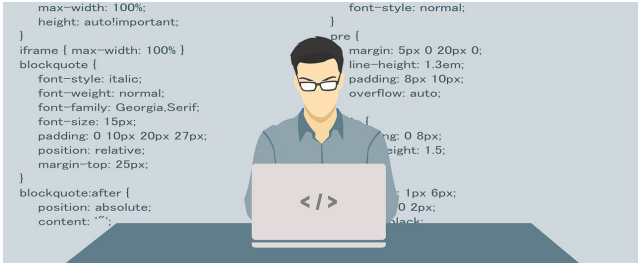
perspicuous /pə ˈspɪkjʊəs/
adjective *formal*
clearly expressed and easily understood; lucid.
able to give an account or express an idea clearly.

Was haben wir heute implementiert?

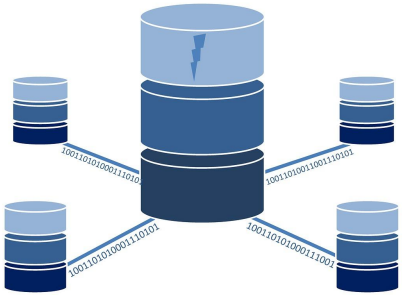


Welche Jobs werden im Umfeld des maschinellen Lernens gebraucht?

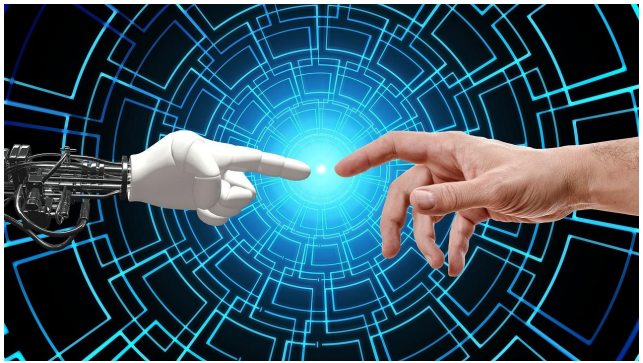
Kundenkontakt / Beratung / Verständnis des Problembereichs



Programmierung / Architektur



Data Analyst / Data Engineer



Human Computer Interaction

Bilder: Pixabay

Welche Studiengänge gibt es zum Thema KI an der UdS?

- Informatik - Bachelor/Lehramt
- Data Science & AI
- Cybersecurity
- Bioinformatik
- Wirtschaftsinformatik
- Medieninformatik
- Human Computer Interaction
- ...

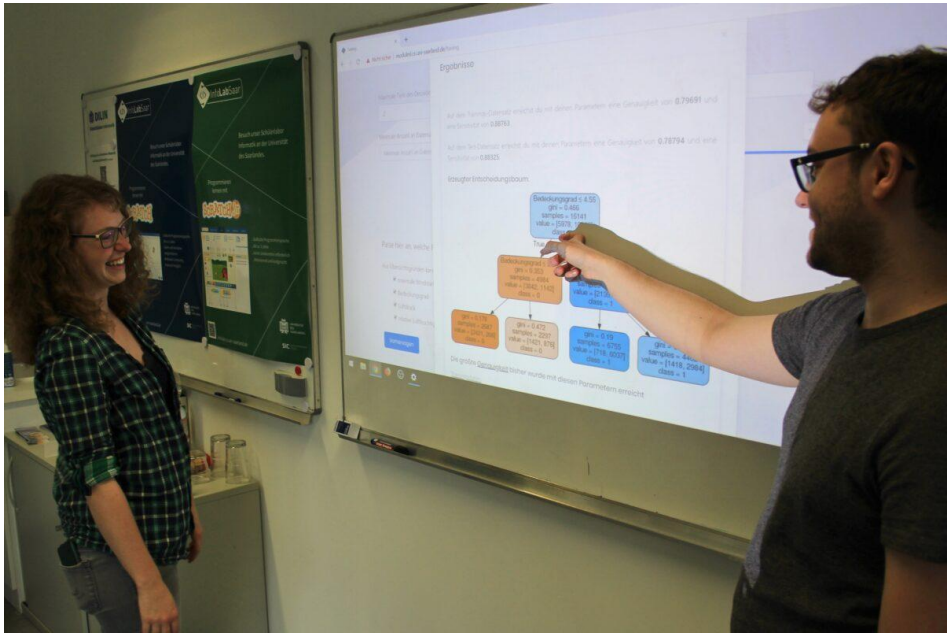


Bild: InfoLab Saar

Neuronale Netze im Studium an der UdS

Beispielstudienplan Bachelor Data Science and Artificial Intelligence

1	Programmierung 1 (9 CP)	Mathematik für Informatiker 1 (9 CP)	Ringvorlesung (2 CP)	Elements of Data Science and Artificial Intelligence (9 CP)	29
2	Programmierung 2 (9 CP)	Mathematik für Informatiker 2 (9 CP)	Statistics Lab	Anwendungsfach	31
3	Elements of Machine Learning (6 CP)	Mathematik für Informatiker 3 (9 CP)			
4	Big Data Engineering (6 CP)	Projektseminar Data Science and Artificial Intelligence (9 CP)			
5	Wahlpflicht-bereich (8 CP)	Stammvorlesung DSAI (9 CP)			
6	Bachelor-Seminar (9 CP)	Bachelor-Arbeit (12 CP)	Vertiefungs-vorlesung DSAI (6 CP)		

UNIVERSITÄT DES SAARLANDES

FAKULTÄT FÜR MATHEMATIK UND INFORMATIK

MODULHANDBUCH

Data Science and Artificial Intelligence BSc

Elements of Data Science and Artificial Intelligence — EIDSAI

Studiensem.	Regelst.sem.	Turnus	Dauer	SWS	ECTS
1	6	every winter semester	1 semester	6	9

Modulverantwortliche/r Prof. Dr. Jörg Hoffmann, Prof. Dr. Jens Dittrich, Prof. Dr. Bernt Schiele, Prof. Dr. Vera Demberg

Dozent/inn/en Prof. Dr. Jörg Hoffmann, Prof. Dr. Jens Dittrich, Prof. Dr. Bernt Schiele, Prof. Dr. Vera Demberg

Zulassungsvoraussetzungen keine

Leistungskontrollen / Prüfungen Weekly assignments, Exam (qualification for exam depends on performance in assignments)

Lehrveranstaltungen / SWS
4 h lectures
+ 2 h tutorial
= 6 h (weekly)

Arbeitsaufwand
90 h of classes
+ 180 h private study
= 270 h (= 9 ECTS)

Modulnote Based on exam. The exact modalities are specified by the lecturers.

Sprache English

Lernziele / Kompetenzen

Overview of challenges and methods in Data Science and AI. Basic knowledge of key concepts and algorithms.

Inhalt

Introduction to history and concepts of Data Science and AI

- Machine Learning (supervised, unsupervised, reinforcement, neural networks)
- (adversarial) Search, Planning
- Reasoning

• Machine Learning (supervised, unsupervised, reinforcement, neural networks)

ving, Language Processing, Social Networks.

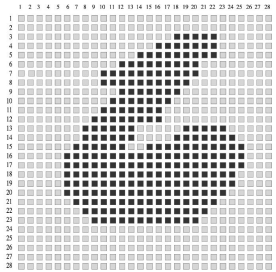
The exercises will cover methodological, algorithmic, as well as practical aspects. Where basic programming or scripting skills are required, the lecture and exercises will introduce these skills.

Was ihr sofort machen könnt

- Online-Kurse, z.B. KI Campus oder 3Blue1Brown
- Bundeswettbewerb Künstliche Intelligenz
- Bundeswettbewerb Informatik
- Gasthörer an der UdS - ohne formale Zulassungskriterien
- Juniorstudium an der UdS - mit Bewerbung, Start im Wintersemester
- Saar Hackathon: Frühjahr und Herbst in Dudweiler
- CoderDojo Saar - Teilnehmer*in / Mentor*in, jeden 4. Samstag im Monat
- Hacksaar: wöchentliche Treffen in Dudweiler, hardwarenah

Ein Eintrag aus dem MNIST-Dataset

Bild



Label

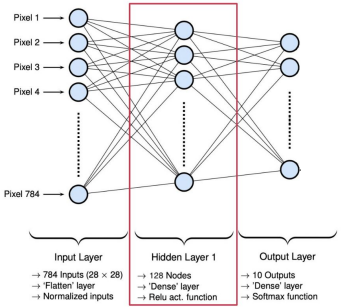
6

28 x 28 = 784 Pixel

37

Bild: Pixabay

Neuronales Netzwerk für die MNIST-Aufgabe - Hidden Layer



41

Block 6 Neuronales Netz trainieren

```python
# Block 6
# Trainingsloop
num_epochs = 1

print("Starte Training")
for epoch in range(num_epochs):
    for batch_index, (images, labels) in enumerate(training_loader):
        optimization_algorithm.zero_grad()
        outputs = model(images)     # forward wird aufgerufen, wenn data durch das Model geleitet wird
                # outputs = bis zu 64 (batch_size) Wahrscheinlichkeitsverteilungen für jeweils all 10 Ziffern
                # [-4.8741e+00,  6.0430e+00,  1.4008e+00,  1.1718e+00, -2.7590e+00,
                #  3.0154e-01, -4.9669e-01,  6.0514e-01,  8.4430e-01, -7.3287e-01]
        loss = loss_function(outputs, labels)
        loss.backward()
        optimization_algorithm.step()

        if batch_index % 100 == 0:
            print(f'Epoch {epoch+1}/{num_epochs}, Batch {batch_index}/{len(training_loader)}, Loss: {loss.item()}')
print("Traning fertig!")
```

```
Epoch 1/1, Batch 0/938, Loss: 0.16773580014705658
Epoch 1/1, Batch 100/938, Loss: 0.22459541261196136
Epoch 1/1, Batch 200/938, Loss: 0.2589246332645416
Epoch 1/1, Batch 300/938, Loss: 0.2684001922607422
```

https://inf-schule.de/informatiksysteme/kuenstliche-intelligenz/maschinelles-lernen/maschinelles_lernen

56

Welche Studiengänge gibt es zum Thema KI an der UdS?

- Informatik - Bachelor/Lehramt
- Data Science & AI
- Cybersecurity
- Bioinformatik
- Wirtschaftsinformatik
- Medieninformatik
- Human Computer Interaction
- …

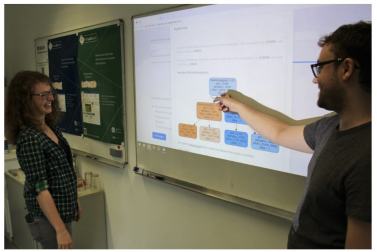


Bild: InfoLab Saar

69

```
(tensor(
[[[-1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,-1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,-1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,-1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000],
  [-1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,-1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,-1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,-1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000],
  [-1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,-1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,-1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,-1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000],
  [-1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,-1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,-1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,-1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000],
  [-1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,-1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,-1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,-1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000],
  [-1.0000, -1.0000, -1.0000, -1.0000, -1
  [-1.0000, -1.0000, -1.0000, -1.0000, -1
  [-1.0000, -1.0000, -1.0000, -1.0000, -1
  [-1.0000, -1.0000, -1.0000, -1.0000, -1
  [-1.0000, -1.0000, -1.0000, -1.0000, -1
  [-1.0000, -1.0000, -1.0000, -1.0000, -1
  [-1.0000, -1.0000, -1.0000, -1.0000, -1
  [-1.0000, -1.0000, -1.0000, -1.0000, -1
  [-1.0000, -1.0000, -1.0000, -1.0000, -1
  [-1.0000, -1.0000, -1.0000, -1.0000, -1
  [-1.0000, -1.0000, -1.0000, -1.0000, -1
  [-1.0000, -1.0000, -1.0000, -1.0000, -1
  [-1.0000, -1.0000, -1.0000, -1.0000, -1
  [-1.0000, -1.0000, -1.0000, -1.0000, -1
  [-1.0000, -1.0000, -1.0000, -1.0000, -1
  [-1.0000, -1.0000, -1.0000, -1.0000, -1
  [-1.0000, -1.0000, -1.0000, -1.0000, -1
  [-1.0000, -1.0000, -1.0000, -1.0000, -0
  [-1.0000, -1.0000, -1.0000, -1.0000,  0
  [-1.0000, -1.0000, -1.0000, -1.0000, -1
  [-1.0000, -1.0000, -1.0000, -1.0000, -1
  [-1.0000, -1.0000, -1.0000, -1.0000, -1
]]), 5)
```

```
(tensor(
[[[-1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,-1.0000, -1.0000, -1.0000, -1.0000,
  [-1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,-1.0000, -1.0000, -1.0000, -1.0000,
  [-1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,-1.0000, -1.0000, -1.0000, -1.0000,
  [-1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,-1.0000, -1.0000, -1.0000, -1.0000,
  [-1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,-1.0000, -1.0000, -1.0000, -1.0000,
  [-1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,-1.0000, -1.0000, -1.0000, -1.0000,
  [-1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,-1.0000, -0.7647, -0.7176, -0.2627,
  [-1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,-0.6157,  0.8667,  0.9843,  0.9843,
  [-1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,-0.8588,  0.7176,  0.9843,  0.9843,
  [-1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,-1.0000, -0.3725,  0.2235, -0.1608,
  [-1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,-1.0000, -1.0000, -0.8902, -0.9922,
  [-1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,-1.0000, -1.0000, -1.0000, -1.0000,
  [-1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,-1.0000, -1.0000, -1.0000, -1.0000,
  [-1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,-1.0000, -1.0000, -1.0000, -1.0000,
  [-1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,-1.0000, -1.0000, -1.0000, -1.0000,
  [-1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,-1.0000, -1.0000, -1.0000, -1.0000,
  [-1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,-1.0000, -1.0000, -1.0000, -1.0000,
  [-1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,-1.0000, -1.0000, -1.0000, -1.0000,
  [-1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,-1.0000, -1.0000, -1.0000, -1.0000,
  [-1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,-1.0000, -1.0000, -1.0000, -1.0000,
  [-1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,-1.0000, -1.0000, -1.0000, -0.8118,
  [-1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,-1.0000, -0.8196, -0.4824,  0.6706,
  [-1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -0.8588, 0.3412,  0.7176,  0.9843,  0.9843,
  [-1.0000, -1.0000, -1.0000, -1.0000, -0.5686,  0.3490,  0.7725, 0.9843,  0.9843,  0.9843,  0.9843,
  [-1.0000, -1.0000, -1.0000, -1.0000,  0.0667,  0.9843,  0.9843, 0.9843,  0.6627,  0.0588,  0.0353,
  [-1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,-1.0000, -1.0000, -1.0000, -1.0000,
  [-1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,-1.0000, -1.0000, -1.0000, -1.0000,
  [-1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,-1.0000, -1.0000, -1.0000, -1.0000,
]]), 5)
```

1. Neuronales Netz definieren und instantiieren

2. - 4. Daten herunterladen, transformieren und in Batches aufteilen

5. Verlustfunktion und Optimierungsalgorithmus definieren

6. Neuronales Netz trainieren

7. Accuracy berechnen

```python
# Version 1.0
# Block 1
import torch
import torch.nn as nn

# Definiere das neuronale Netzwerk
class SimpleNN(nn.Module):  # erbt von nn.Module
    def __init__(self):
        super(SimpleNN, self).__init__()
        self.fc1 = nn.Linear(28*28, 128) # fc1 = fully connected layer, 28x28=784 in, 128 out
        self.relu = nn.ReLU()                    # Rectified Linear Unit (gerichtetes lineares Element)
                                                 # f(x) = max(0, x)
        self.fc2 = nn.Linear(128, 10)    # 128 in, 10 out

    def forward(self, x):
        x = x.view(-1, 28*28)  # Flattening der Eingabe
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        return x

# Model für das Neuronales Netz instantiieren
model = SimpleNN()

# Block 2
from torchvision import transforms

# Lade das MNIST-Dataset und wende Transformationen an
transform_function = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.5,), (0.5,))])

# Block 3
from torchvision import datasets

training_data = datasets.MNIST(root='./data', train=True, download=True, transform=transform_function)
test_data = datasets.MNIST(root='./data', train=False, download=True, transform=transform_function)

# Block 4
from torch.utils.data import DataLoader

training_loader = DataLoader(dataset=training_data, batch_size=64, shuffle=True)
test_loader = DataLoader(dataset=test_data, batch_size=64, shuffle=False)

# Block 5
import torch.optim as optim

# Verlustfunktion (Loss) und Optimierer definieren
loss_function = nn.CrossEntropyLoss()
optimization_algorithm = optim.SGD(model.parameters(), lr=0.01) # SGD = stochastic gradient descent,
                                                                # lr = learning rate

# Block 6
# Trainingsloop
num_epochs = 1

print("Starte Training")
for epoch in range(num_epochs):
    for batch_index, (images, labels) in enumerate(training_loader):
        optimization_algorithm.zero_grad()
        outputs = model(images)     # forward wird aufgerufen, wenn data durch das Model geleitet wird
                # outputs = bis zu 64 (batch_size) Wahrscheinlichkeitsverteilungen für jeweils all 10 Ziffern
                # [-4.8741e+00,  6.0430e+00,  1.4008e+00,  1.1718e+00, -2.7590e+00,
                #   3.0154e-01, -4.9669e-01,  6.0514e-01,  8.4430e-01, -7.3287e-01]
        loss = loss_function(outputs, labels)
        loss.backward()
        optimization_algorithm.step()

        if batch_index % 100 == 0:
            print(f'Epoch {epoch+1}/{num_epochs}, Batch {batch_index}/{len(training_loader)}, Loss: {loss.item()}')
print("Traning fertig!")

# Block 7
# Evaluierung auf dem Testdatensatz
model.eval()
correct = 0
total = 0

with torch.no_grad():
    for image, label in test_loader:
        outputs = model(image)
        _, predicted = torch.max(outputs.data, 1)
        total += label.size(0)
        correct += (predicted == label).sum().item()

accuracy = correct / total
print(f'Genauigkeit auf dem Testdatensatz: {accuracy * 100:.2f}%')
```