DeepLearning mit Python und Jupyter-Notebooks

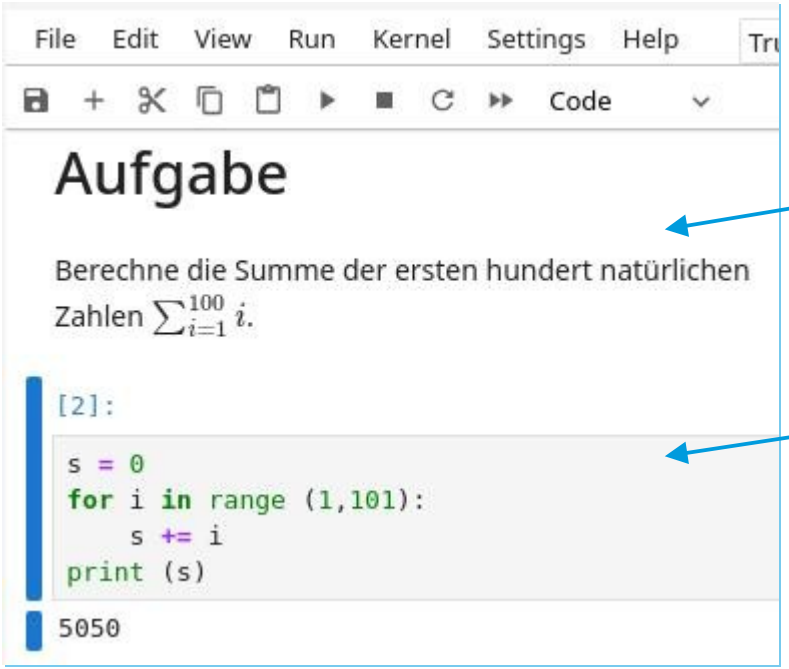
Ulla Diewald

inf-schule.de
Pädagogisches Landesinstitut RLP
Wiedtal-Gymnasium Neustadt (Wied)
cryptoparty.in/bonn

ul.di@posteo.de
ulla.diewald@pl.rlp.de

27.2.2025
Hessische Landesstelle für Technologiefortbildung

Jupyter-Notebooks



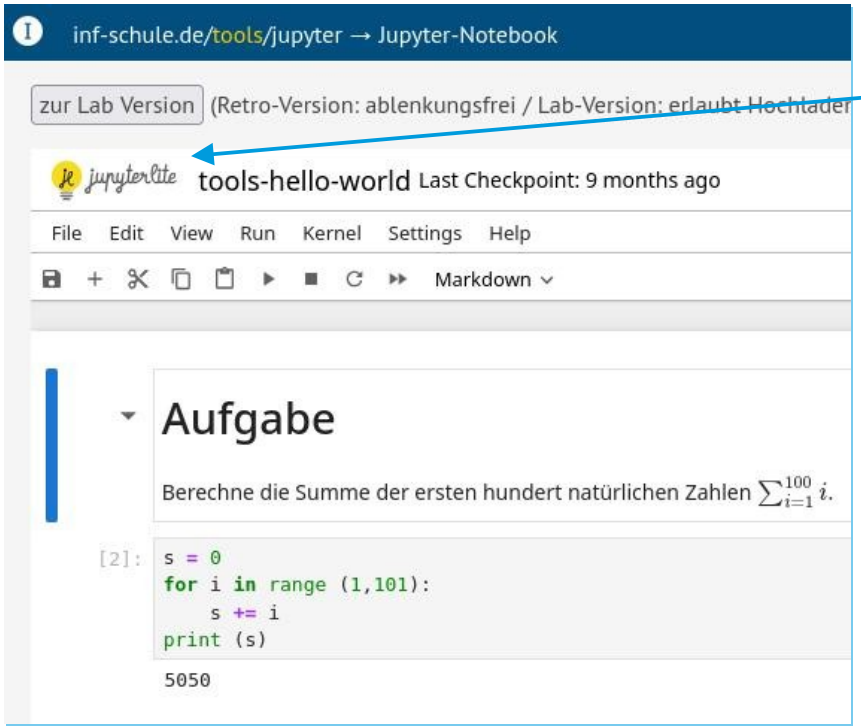
Markdown und Latex

Python-Code

Interaktive Arbeitsblätter

Rapid Prototyping

Jupyter-Lite



https://inf-schule.de/tools/jupyter

Online-Version (Webassembly)

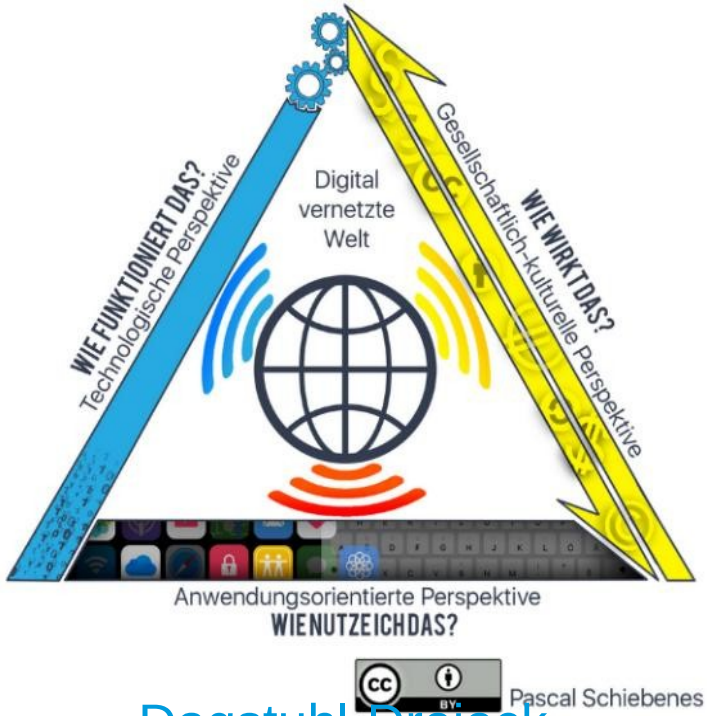
NB: Jupyter-Lite noch „experimental“, daher Alternativen für „Plan B“:
- Visual Studio Code
- PyCharme
- lokal installieren

DeepLearning

- Perzeptron, Seperationsgerade
- logische Gatter, Delta-Lernregel
- XOR-Problem, KNN
- künstl. Neuronen
- Forward- / Backward-Propagation
- MNIST-Datensatz, Ziffernerkennung
- Code from Scratch / Jupyter-Notebooks

DeepLearning

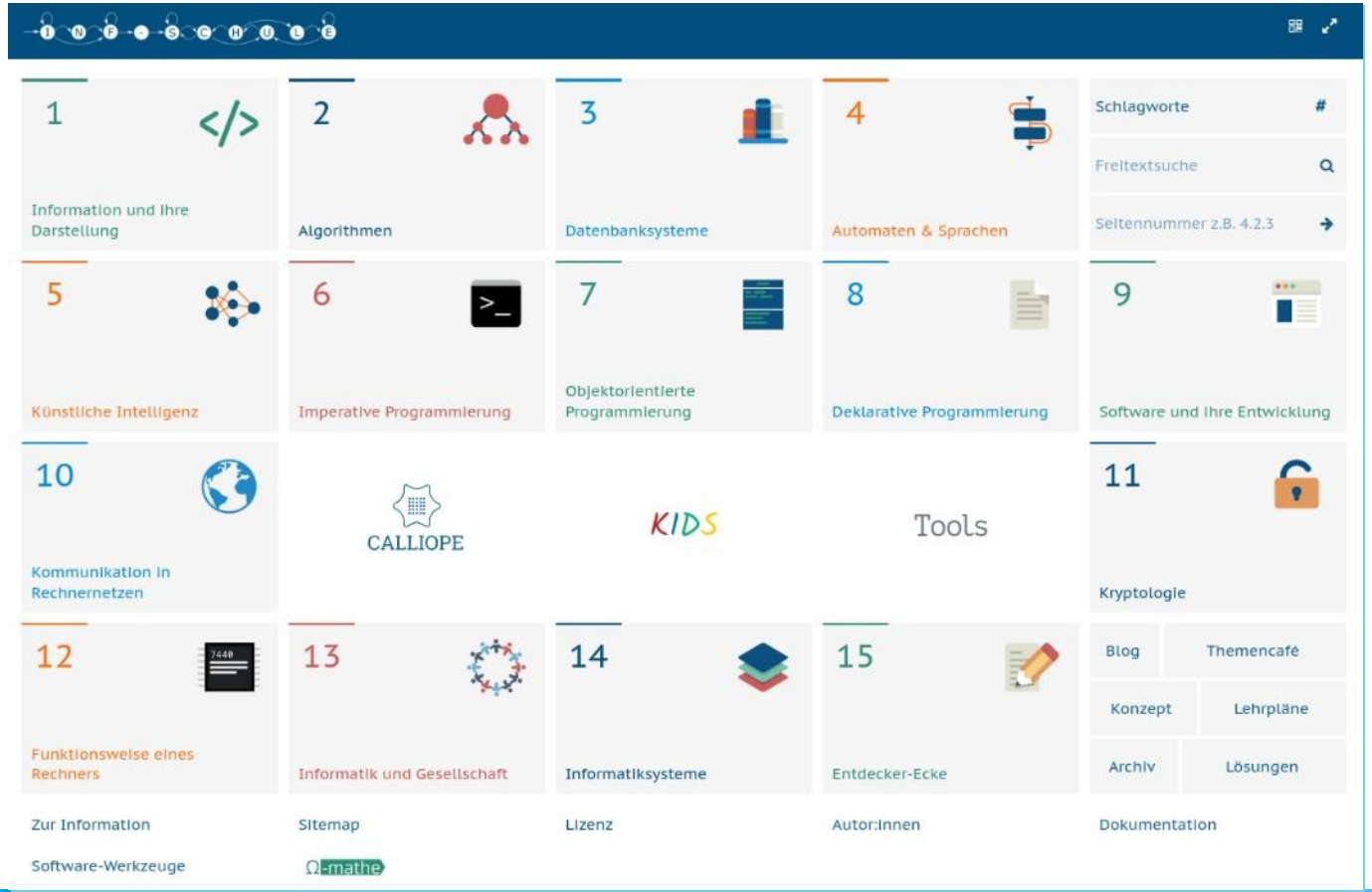
- Perzeptron, Seperationsgerade
- logische Gatter, Delta-Lernregel
- XOR-Problem, KNN
- künstl. Neuronen
- Forward- / Backward-Propagation
- MNIST-Datensatz, Ziffernerkennung
- Code from Scratch / Jupyter-Notebooks



Dagstuhl-Dreieck

Bild: Pascal Schiebenes, CC BY, https://bobblume.de/2019/10/02/digital-das-dagstuhl-dreieck-eine-grafische-aufbereitung-excitingedu/

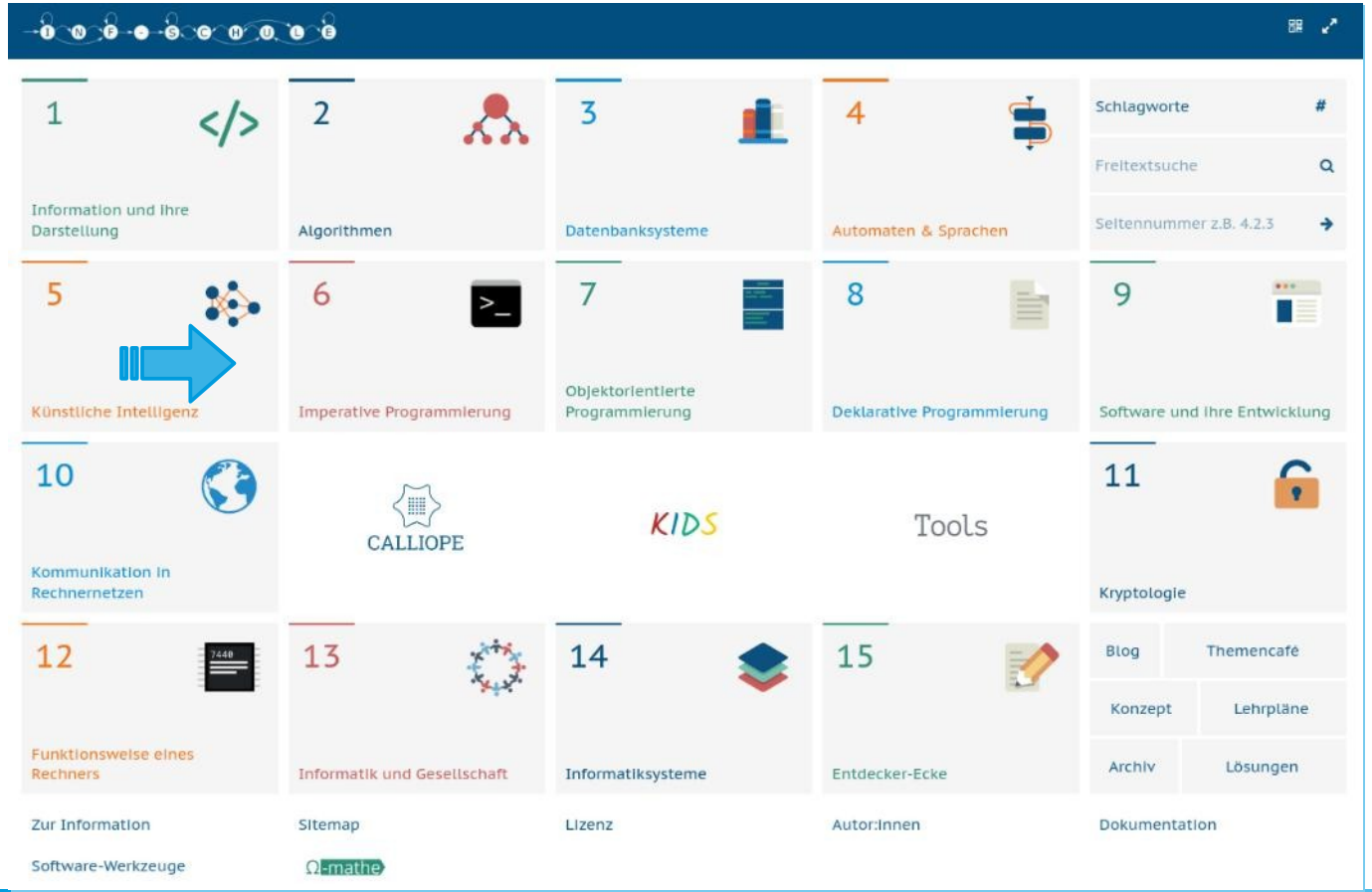
inf-schule.de



anerkanntes Schulbuch in RLP

OER

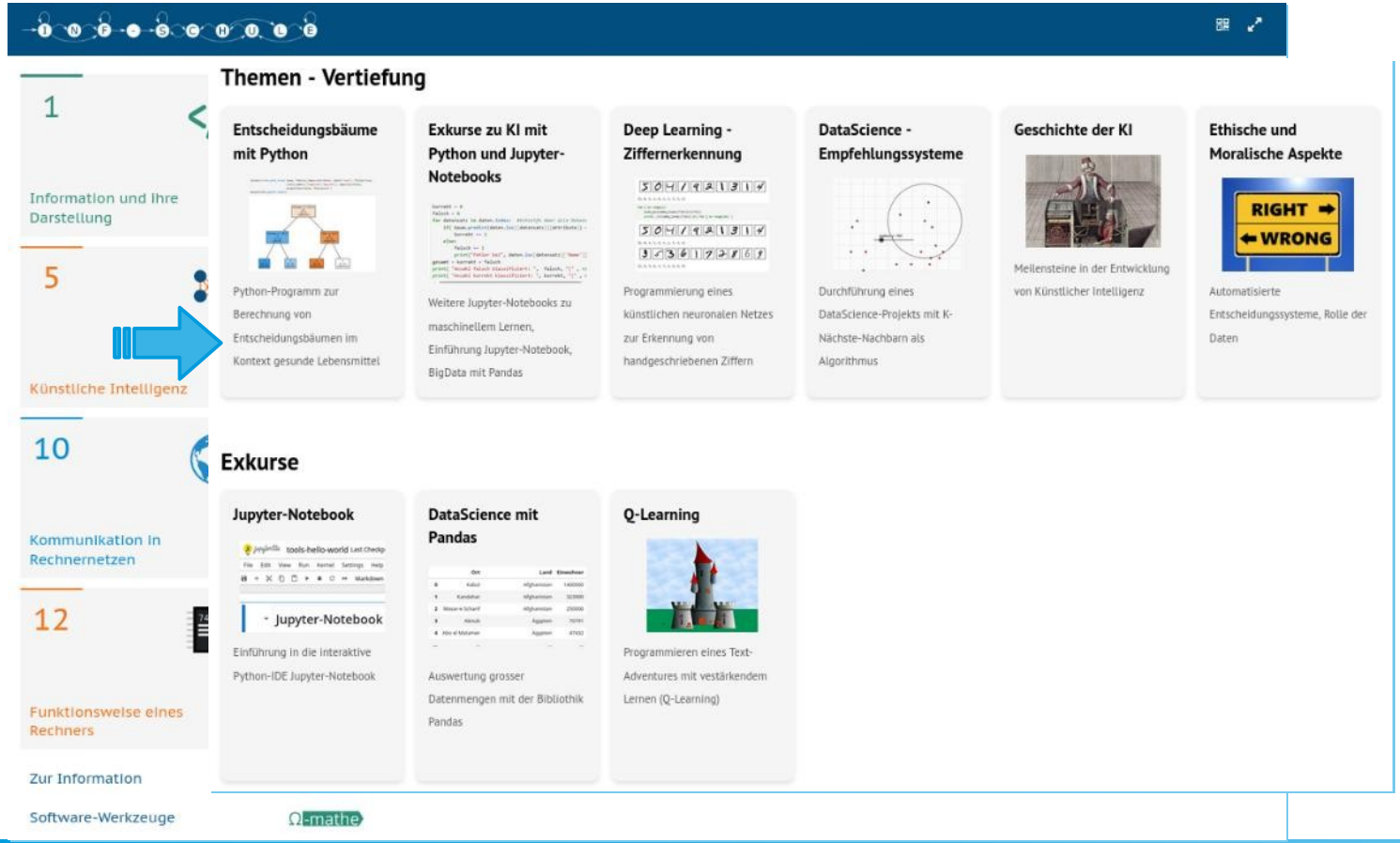
inf-schule.de



anerkanntes Schulbuch in RLP

OER

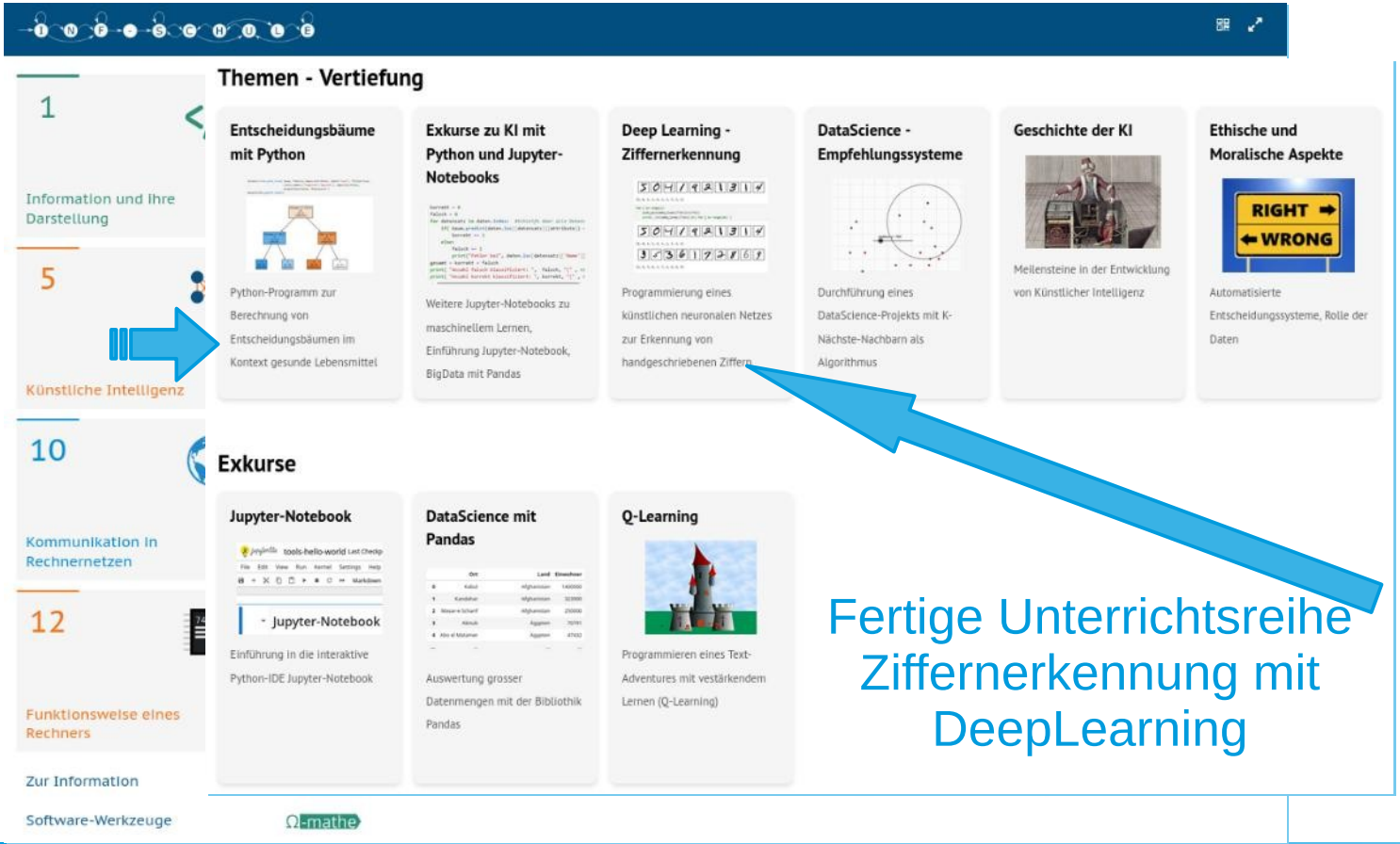
inf-schule.de



anerkanntes Schulbuch in RLP

OER

inf-schule.de



Fertige Unterrichtsreihe Ziffernerkennung mit DeepLearning

anerkanntes Schulbuch in RLP

OER

inf-schule.de/5.1.3.6.10

Perzeptron

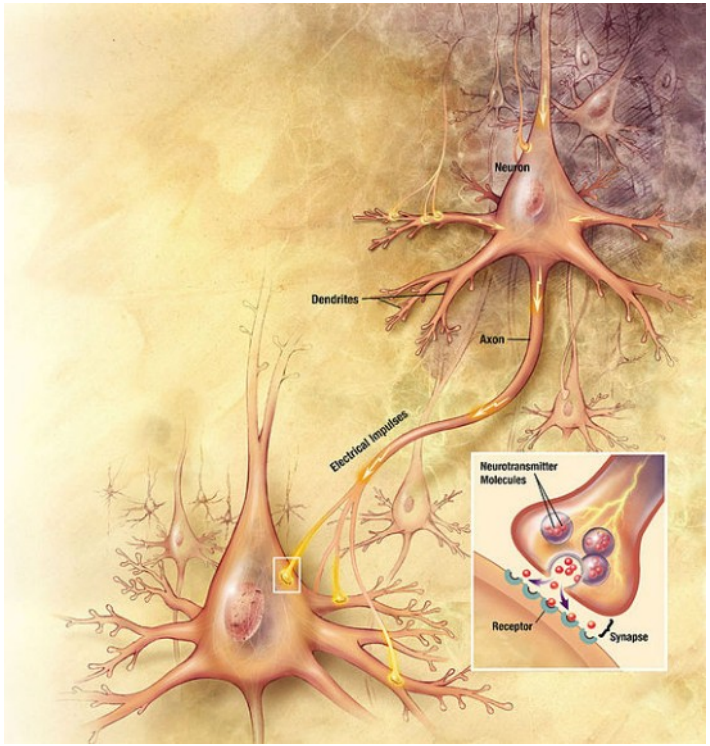
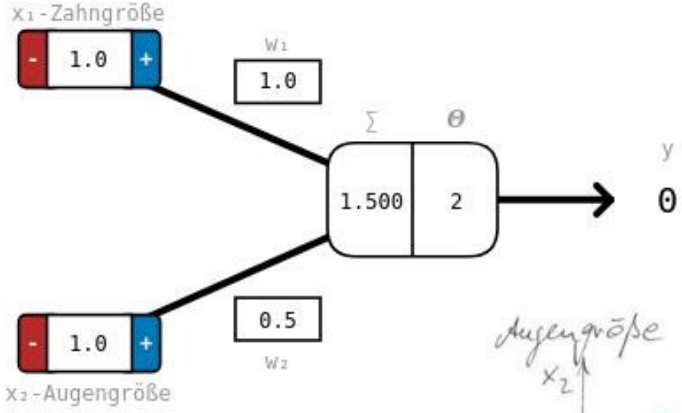
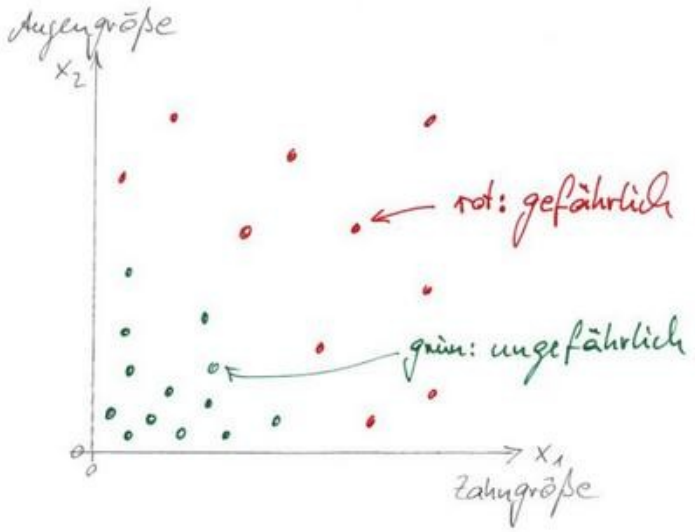


Bild: Neuron - Wikipedia: gemeinfrei



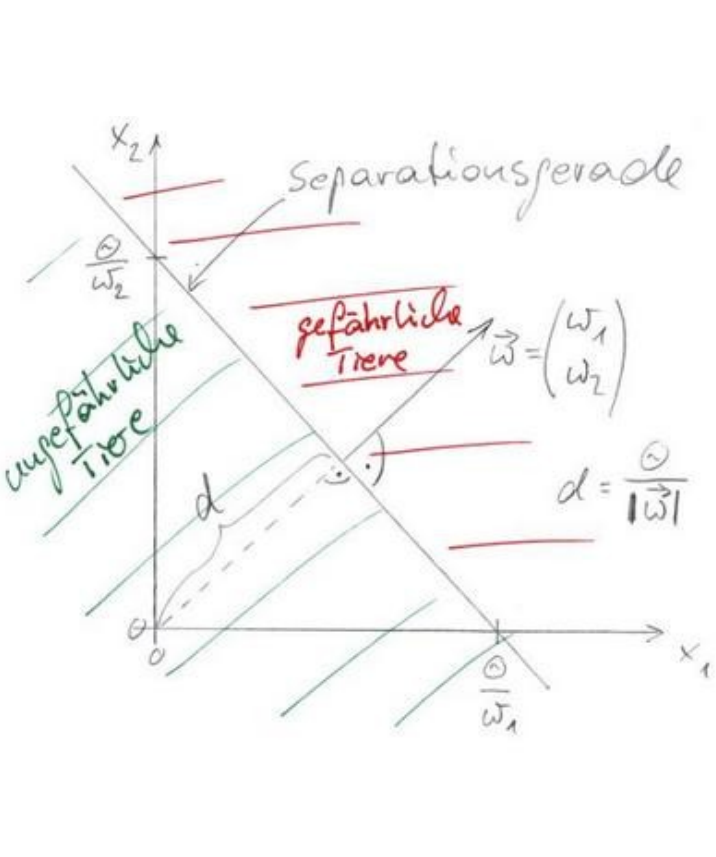
Perzeptron

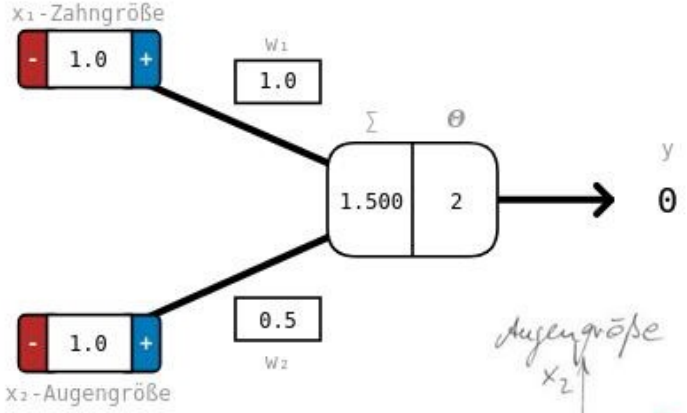
$$\sum_{i=1}^{n} w_i x_i \geq \Theta$$



inf-schule.de/5.1.3.6.2

Seperationsgerade

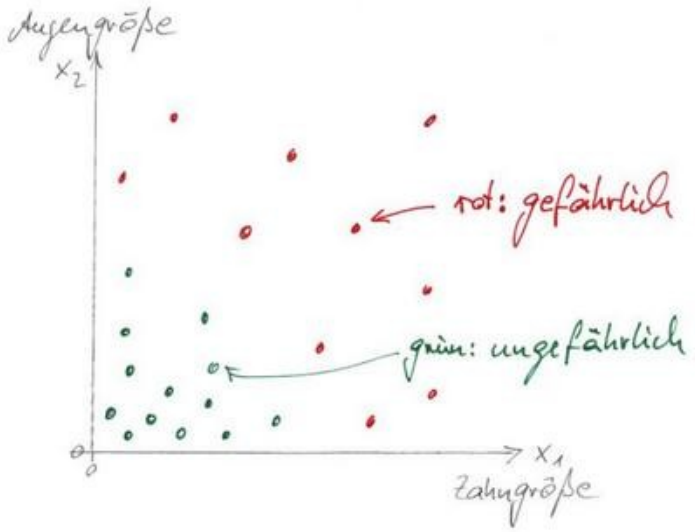




Perzeptron

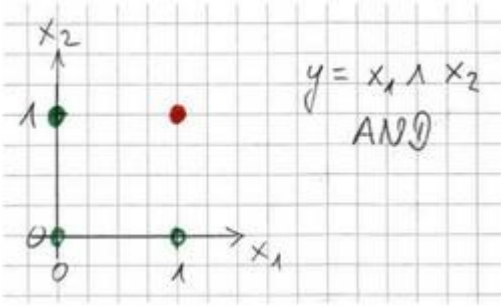
$$\sum_{i=1}^{n} w_i x_i \geq \Theta$$

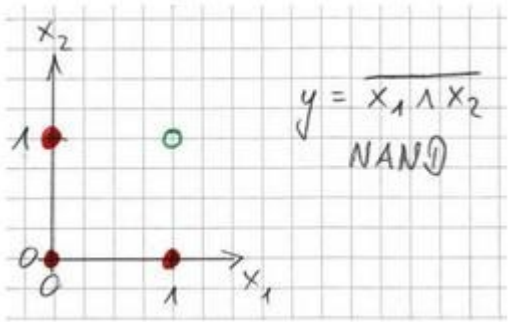
$$\hat{\vec{w}} \cdot \vec{x} - \frac{\Theta}{\|\vec{w}\|} = 0$$

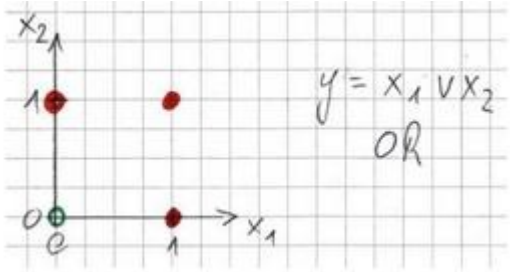


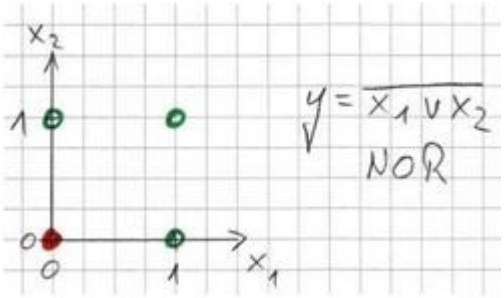
inf-schule.de/5.1.3.6.3

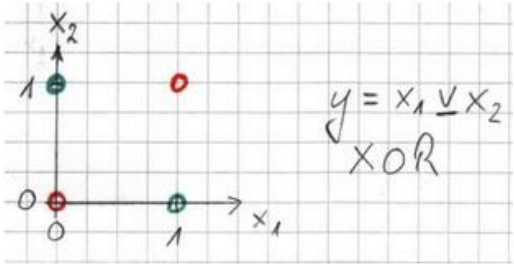
Aufgaben: Logische Operatoren





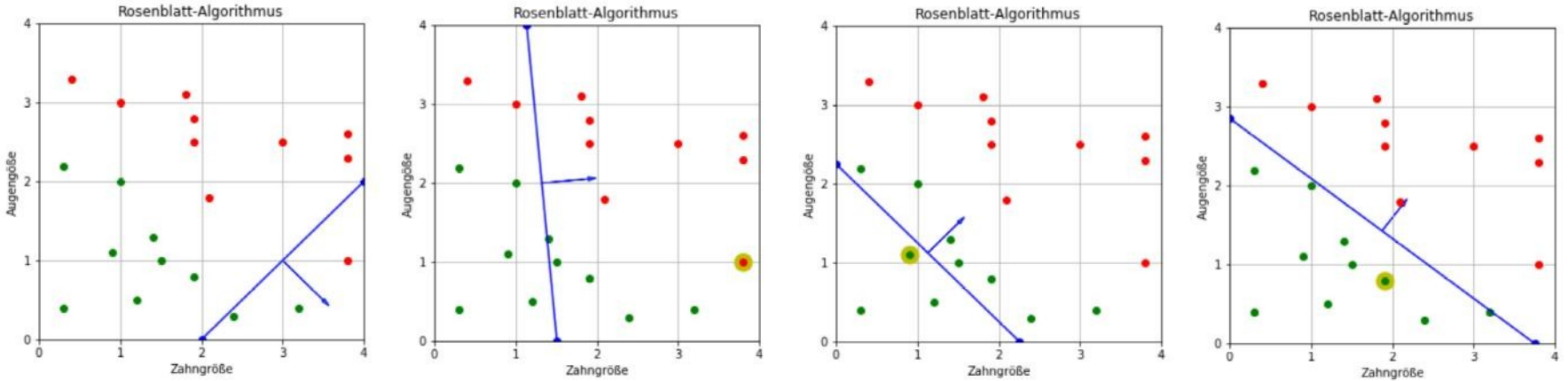






inf-schule.de/5.1.3.6.3.1

Delta-Lernregel (Rosenblatt 1958)



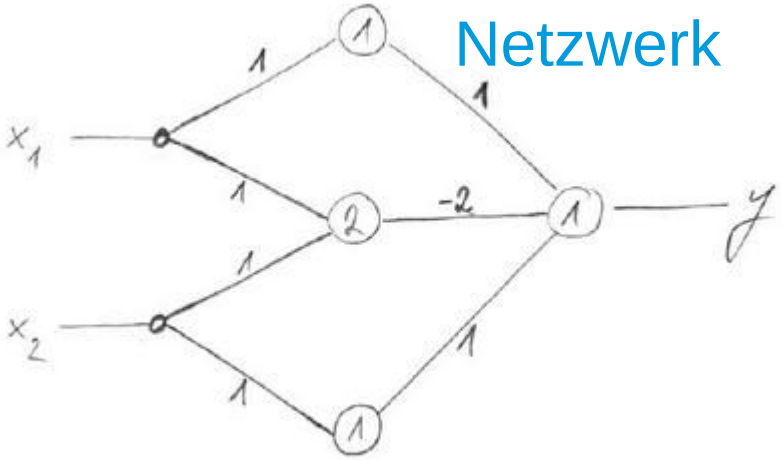
Leider nur für separierbare Daten...

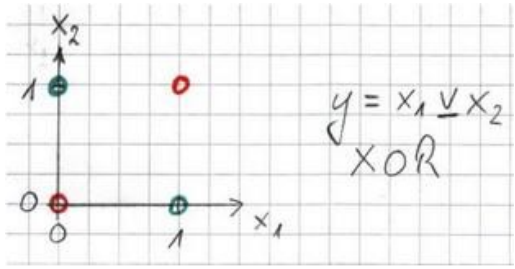
inf-schule.de/5.1.3.6.5

Frank Rosenblatt: The perceptron – a probabilistic model for information storage and organization in the brain. Psychological Review 65, 1958

XOR

Künstliches Neuronales Netzwerk





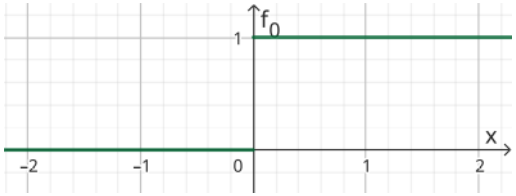
Lernregel...?! ---> Backward-Propagation (1986)

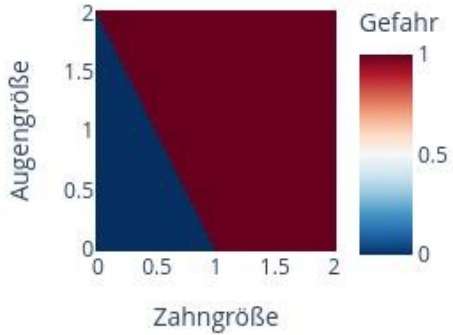
Künstliches Neuron

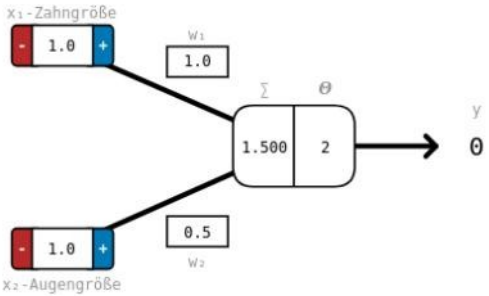
Perzeptron

treshold

$$y(\vec{x}) = f_0(\vec{w} \cdot \vec{x} - \Theta)$$







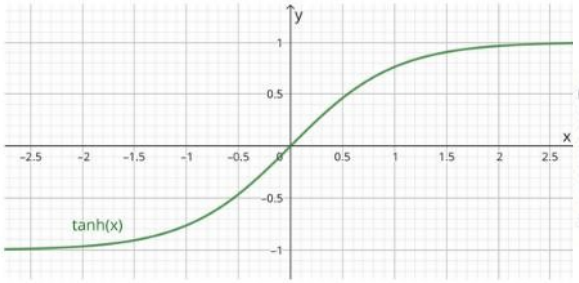
Künstliches Neuron

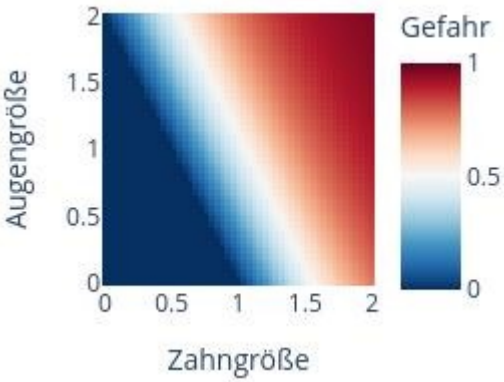
$$z = w_1 \cdot x_1 + w_2 \cdot x_2 + b$$

bias

$b = -\Theta$

$$a(z) = tanh(z) := \frac{e^z - e^{-z}}{e^z + e^{-z}}$$





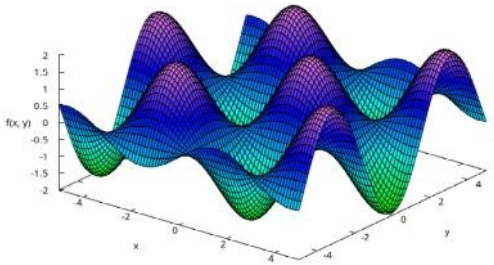
stetige Aktivierungsfunktion
mit „einfacher“ Ableitung

$$tanh'(z) = 1 - tanh^2(z)$$

Gradientenabstieg

Künstliches Neuron

$$z(x_1, x_2) = w_1 \cdot x_1 + w_2 \cdot x_2 + b$$
$$a(z) = tanh(z)$$

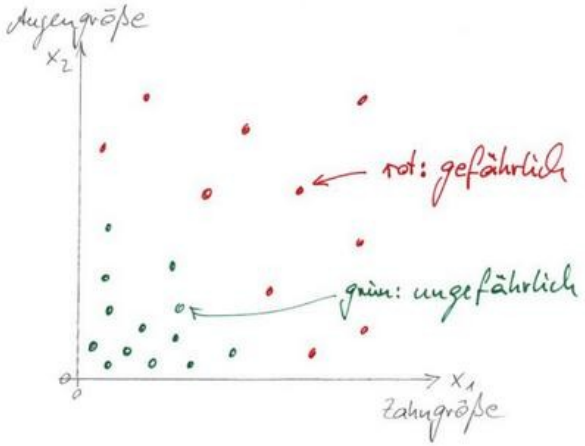


Targetfunktion

$$t(x_1, x_2) = \begin{cases} +1 & \text{gefährlich} \\ -1 & \text{ungefährlich} \end{cases}$$

Kostenfunktion

$$C_x = \frac{1}{2}(t_x - a_x)^2$$



Beachte

$$C_x = C_x(w_1, w_2, b)$$

zufällig vorbelegt

Gradient (partielle Ableitungen)

$$\frac{\partial C_x}{\partial w_1} = -(t-a)\left(1-a^2\right) \cdot x_1$$
$$\frac{\partial C_x}{\partial w_2} = -(t-a)\left(1-a^2\right) \cdot x_2$$
$$\frac{\partial C_x}{\partial b} = -(t-a)\left(1-a^2\right)$$

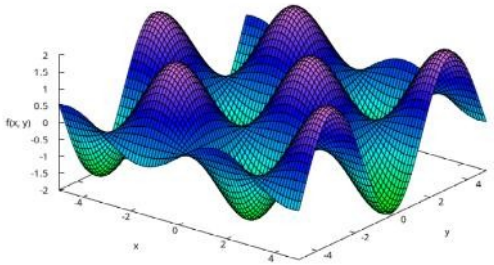
Iterationsvorschrift

$$w_1^{neu} := w_1^{alt} - \frac{\partial C_x}{\partial w_1}$$
$$w_2^{neu} := w_2^{alt} - \frac{\partial C_x}{\partial w_2}$$
$$b^{neu} := b^{alt} - \frac{\partial C_x}{\partial b}$$

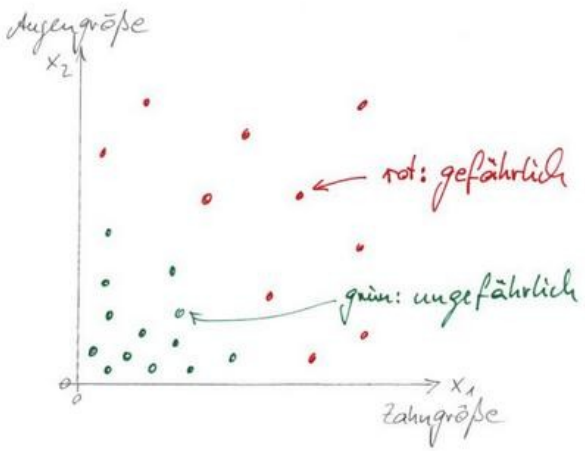
Gradientenabstieg

Künstliches Neuron

$$z(x_1, x_2) = w_1 \cdot x_1 + w_2 \cdot x_2 + b$$
$$a(z) = tanh(z)$$



```python
# Gradientenabstiegsverfahren
def gewichte_update(x1, x2, t, w1, w2, b):

    # Forward-Propagation
    # Berechnung der Neuronenaktivierung bis zum Output
    z = w1*x1+w2*x2+b  # Propagierungsfunktion
    a = tanh(z)        # Aktivierung

    # Backward-Progation
    # Aktualisierung der Gewichte
    w1 += lr*(t-a)*(1.0-a**2)*x1
    w2 += lr*(t-a)*(1.0-a**2)*x2
    b  += lr*(t-a)*(1.0-a**2)

    return w1, w2, b
```



Beachte

$$C_x = C_x(w_1, w_2, b)$$

zufällig vorbelegt

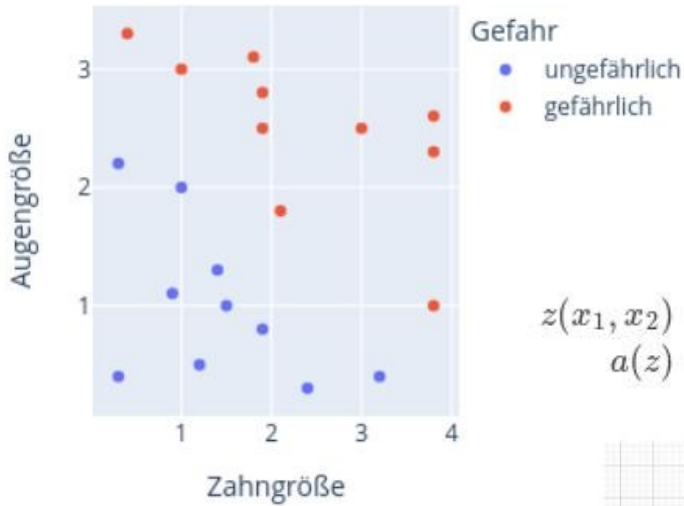
Gradient (partielle Ableitungen)

$$\frac{\partial C_x}{\partial w_1} = -(t-a)\left(1-a^2\right) \cdot x_1$$
$$\frac{\partial C_x}{\partial w_2} = -(t-a)\left(1-a^2\right) \cdot x_2$$
$$\frac{\partial C_x}{\partial b} = -(t-a)\left(1-a^2\right)$$

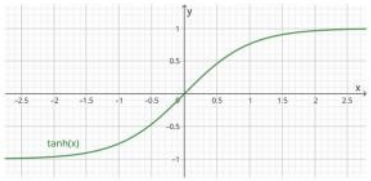
Iterationsvorschrift

$$w_1^{neu} := w_1^{alt} - \frac{\partial C_x}{\partial w_1}$$
$$w_2^{neu} := w_2^{alt} - \frac{\partial C_x}{\partial w_2}$$
$$b^{neu} := b^{alt} - \frac{\partial C_x}{\partial b}$$

Gradientenabstieg

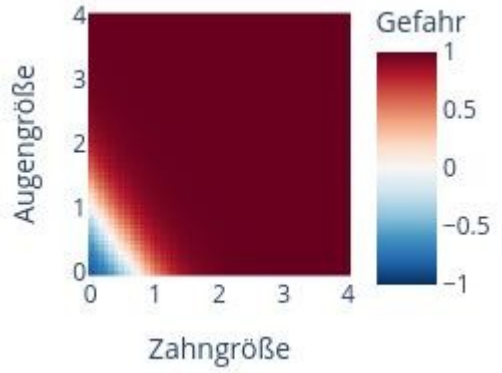


$$z(x_1, x_2) = w_1 \cdot x_1 + w_2 \cdot x_2 + b$$
$$a(z) = tanh(z)$$

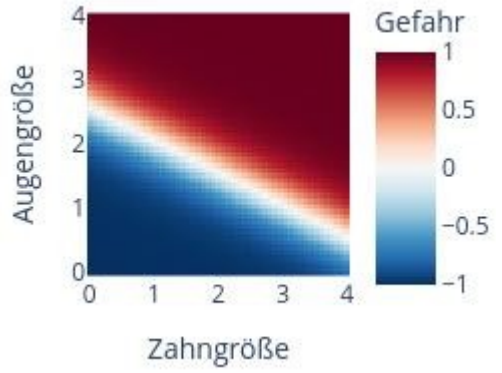


$$C_x = C_x(w_1, w_2, b)$$

zufällig vorbelegt



Startbelegung

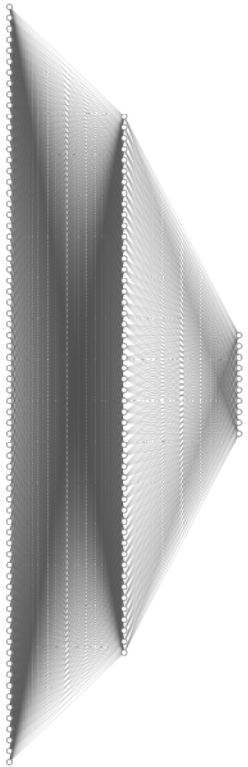


Nach 10 Lernepochen

DeepLearning: Hello World

MNIST-Datensatz
70.000 Bilder

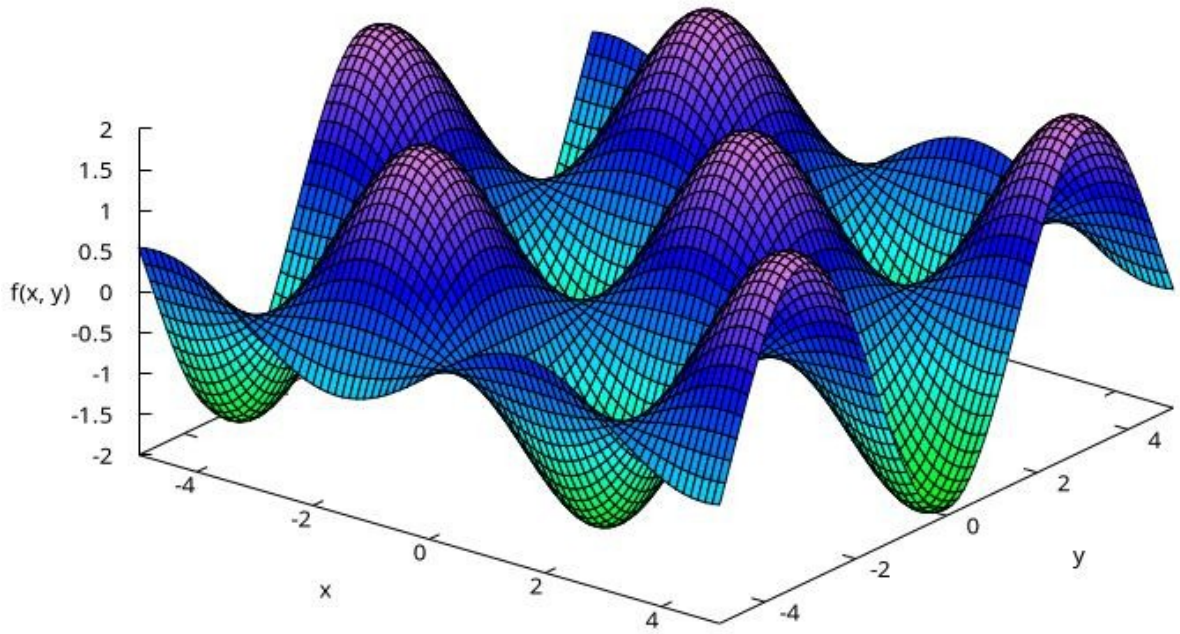
28 x 28



~~70~~ 784-50-10

0: 0%
1: 0%
2: 0%
3: 100%
4: 0%
5: 0%
6: 0%
7: 0%
8: 0%
9: 0%

Gradientenabstieg

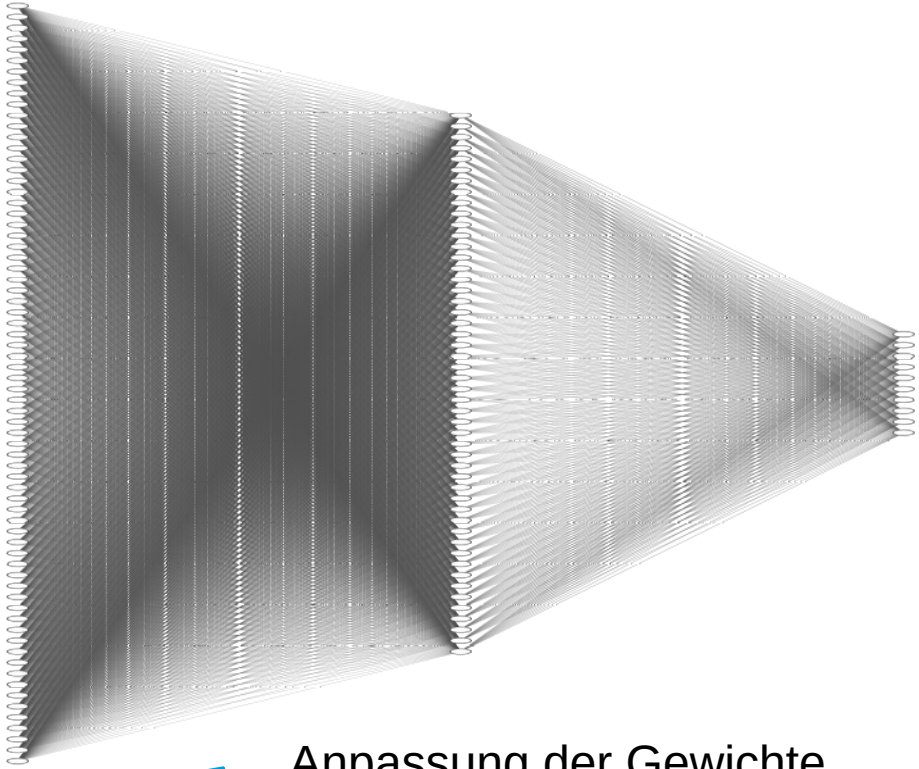


784*50*10 = 392.000

Backprop vorgegeben
als Blackbox-Code
(14 Zeilen netto)

```
def train(self, inputs_list, targets_list):
    # Inputs in 2-dimensionales Array konvertieren (Inputwerte und Zielwerte)
    inputs  = np.array(inputs_list, ndmin=2).T # stehender Vektor
    targets = np.array(targets_list, ndmin=2).T # stehender Vektor

    # Forward-Propagation
    hidden_inputs  = np.dot(self.wih, inputs) + self.bias_hidden # stehender Vektor
    hidden_outputs = self.activation_function(hidden_inputs) # stehender Vektor
    final_inputs  = np.dot(self.who, hidden_outputs) + self.bias_output
    final_outputs = self.activation_function(final_inputs)

    # Fehlerberechnung
    output_errors = (final_outputs - targets)
    output_delta = output_errors * final_outputs  * (1.0 - final_outputs)

    # Backpropagation der Fehler
    hidden_errors = np.dot(self.who.T, output_delta)
    hidden_delta = hidden_errors * hidden_outputs * (1.0 - hidden_outputs)

    # Update der Kantengewichte
    # NB: hiddenoutputs.T ist ein liegender Vektor,
    #     daher ist np.dot(output_delta , hidden_outputs.T)
    #     ein so genanntes Hadamard-Produkt.
    self.who -= self.lr * np.dot(output_delta , hidden_outputs.T)
    self.wih -= self.lr * np.dot(hidden_delta , inputs.T)
    self.bias_output -= self.lr * output_delta
    self.bias_hidden -= self.lr * hidden_delta
```

Gradientenabstieg: Backpropagation



Anpassung der Gewichte

0: 10%
1: 10%
2: 10%
3: 10%
4: 10%
5: 10%
6: 10%
7: 10%
8: 10%
9: 10%

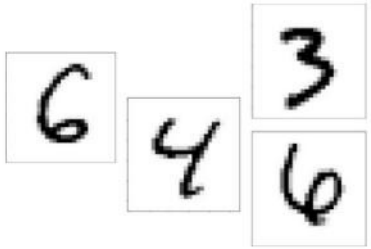
Ziffernerkennung mit Jupyter-Notebooks

File Edit View Run Kernel Settings Help

Markdown JupyterLab Python (Pyodide)

1. Der MNIST-Datensatz

Der **MNIST** (Modified National Institute of Standards) Datensatz ist ein Datensatz mit 70.000 Bildern von handgeschriebenen Ziffern. Dieser Datensatz wird weltweit als Standarddatensatz genutzt, um zu prüfen wie gut Machine Learning Verfahren die Bilderkennung beherrschen. Darüber hinaus zeigt es ein Anwendungbeispiel bei dem man mit herkömmlichen Programmiermet
das aber mit Künstlichen Neuronalen Netzen sehr gut lösbar ist.



File Edit View Run Kernel Settings Kernel Not

Markdown

1.3. Festlegen der Parameter und Erstellen des KNN

[ ]:

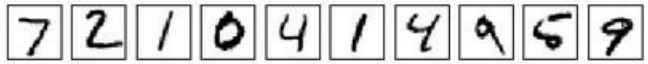
```
# Anzahl der Neuronen in der Eingangsschicht, der verste
input_nodes = 784 # 28x28 Pixel-Bild als Input
hidden_nodes = 50
output_nodes = 10 # 10 Ziffern (0-9) als Output

# Lernrate
learning_rate = 0.1

# Neuronales Netz erstellen
knn = neuralNetwork( input_nodes, hidden_nodes, output_no
```

Launcher | mnist_test_100.c | 10.2-ziffernerke | mnist_train_100

Markdown Notebook Python (Pyodide)

[28]:

```
# predictions anzeigen
for i in range(1):
    plot_list(test_data_lines[i*10:(i+1)*10])
    print( predict_list(test_data_lines[i*10:(i+1)*10]))
```



[7, 2, 1, 0, 4, 1, 4, 9, 2, 7]

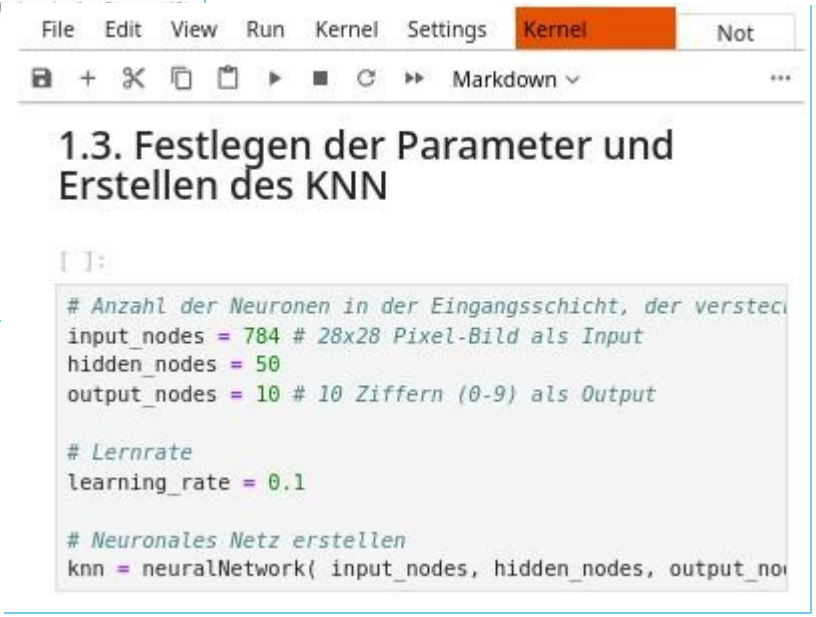
1.9.1. Aufgabe 6

Teste, wie gut das KNN auf den unbekannten Testdaten abschneidet.
Stelle Vermutungen über die Grenzen unseres bisherigen Modells an.

Ergebnisse bitte hierher

inf-schule.de/5.1.3.6.10

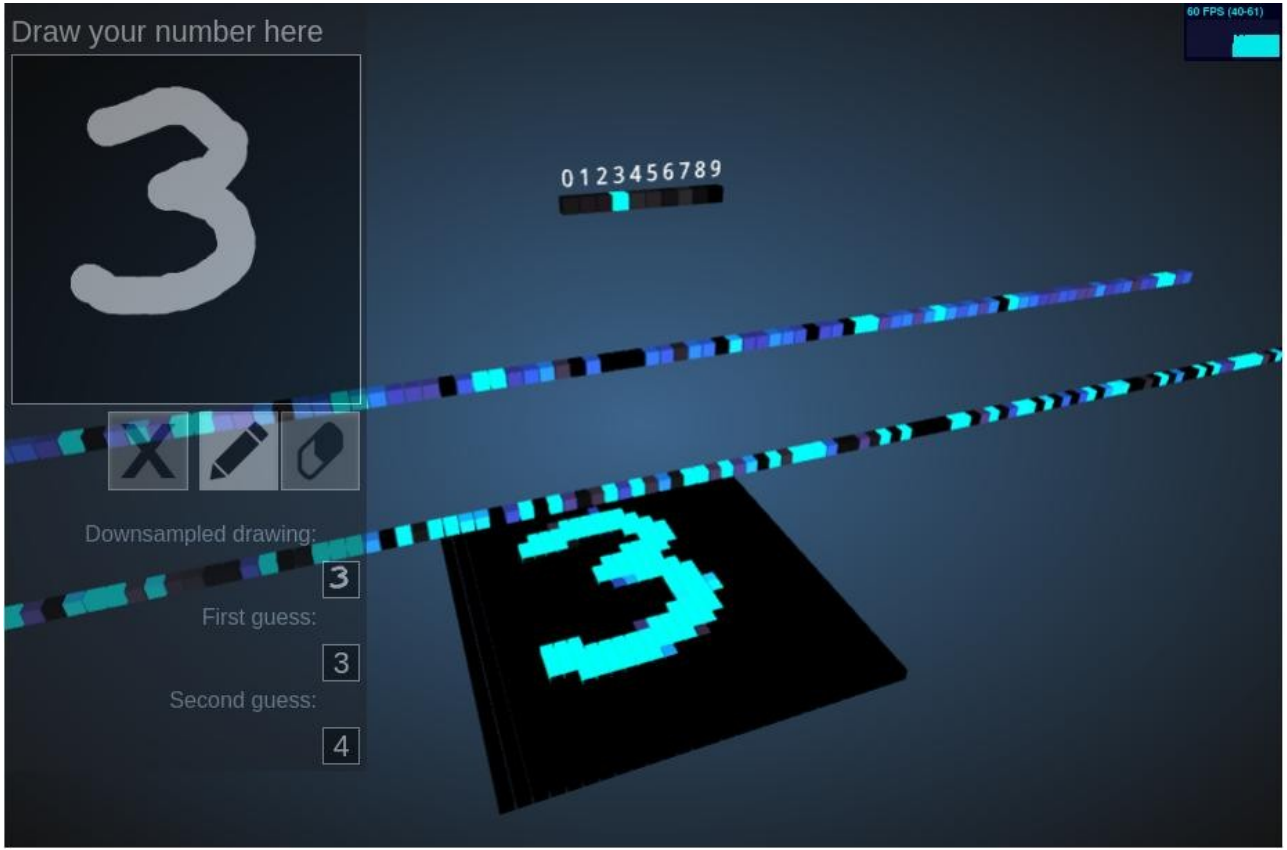
Ziffernerkennung mit Jupyter-Notebooks



Experimente zu
- Dimensionen des KNN
- Lernrate
- Overfitting

inf-schule.de/5.1.3.6.10

Visualisierung



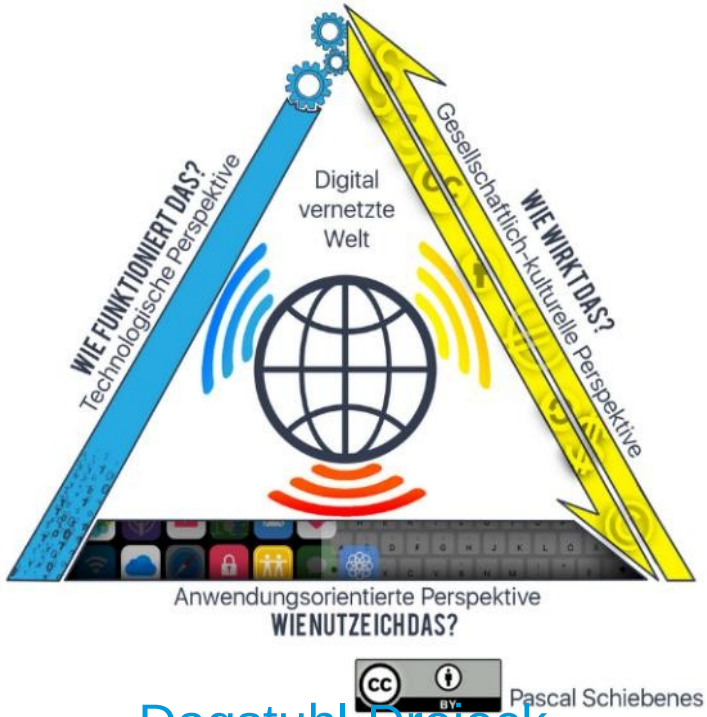
inf-schule.de/5.1.3.6.10.3.2

DeepLearning

- Perzeptron, Seperationsgerade
- logische Gatter, Delta-Lernregel
- XOR-Problem, KNN
- künstl. Neuronen
- Forward- / Backward-Propagation
- MNIST-Datensatz, Ziffernerkennung
- Code from Scratch / Jupyter-Notebooks

DeepLearning

- Perzeptron, Seperationsgerade
- logische Gatter, Delta-Lernregel
- XOR-Problem, KNN
- künstl. Neuronen
- Forward- / Backward-Propagation
- MNIST-Datensatz, Ziffernerkennung
- Code from Scratch / Jupyter-Notebooks



Dagstuhl-Dreieck

Bild: Pascal Schiebenes, CC BY, https://bobblume.de/2019/10/02/digital-das-dagstuhl-dreieck-eine-grafische-aufbereitung-excitingedu/

Ulla Diewald – Fortbildungen 2025

Ein nicht-technischer Einstieg in Algorithmen der "KI-Technologie"	Darmstadt	22.5.
DeepLearning mit Python und Jupyter-Notebooks	Darmstadt	23.5.
Python für Umsteiger:innen aus andern Programmiersprachen	Darmstadt	24.3.
Jupyter-Notebooks/Markdown mit Anwendungen Big-Data/Machine-Learning	Darmstadt	25.3.

https://technologiefortbildung.hlft.hessen.de/fruehjahr-2025-1

Vielen Dank!
Fragen?