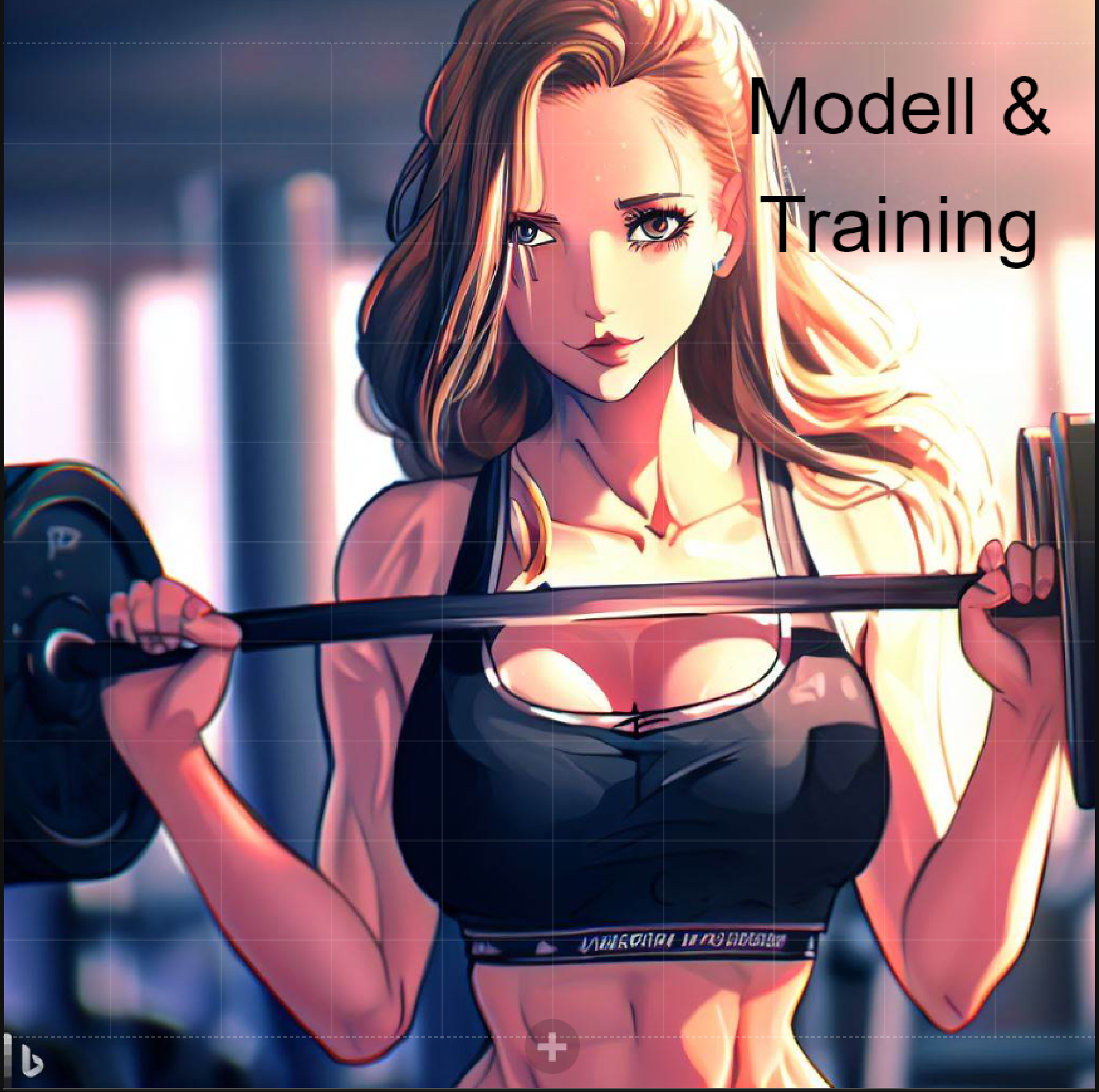


https://www.youtube.com/embed/O1ZQtTwPwYk?enablejsapi=1

Deep Learning Modell

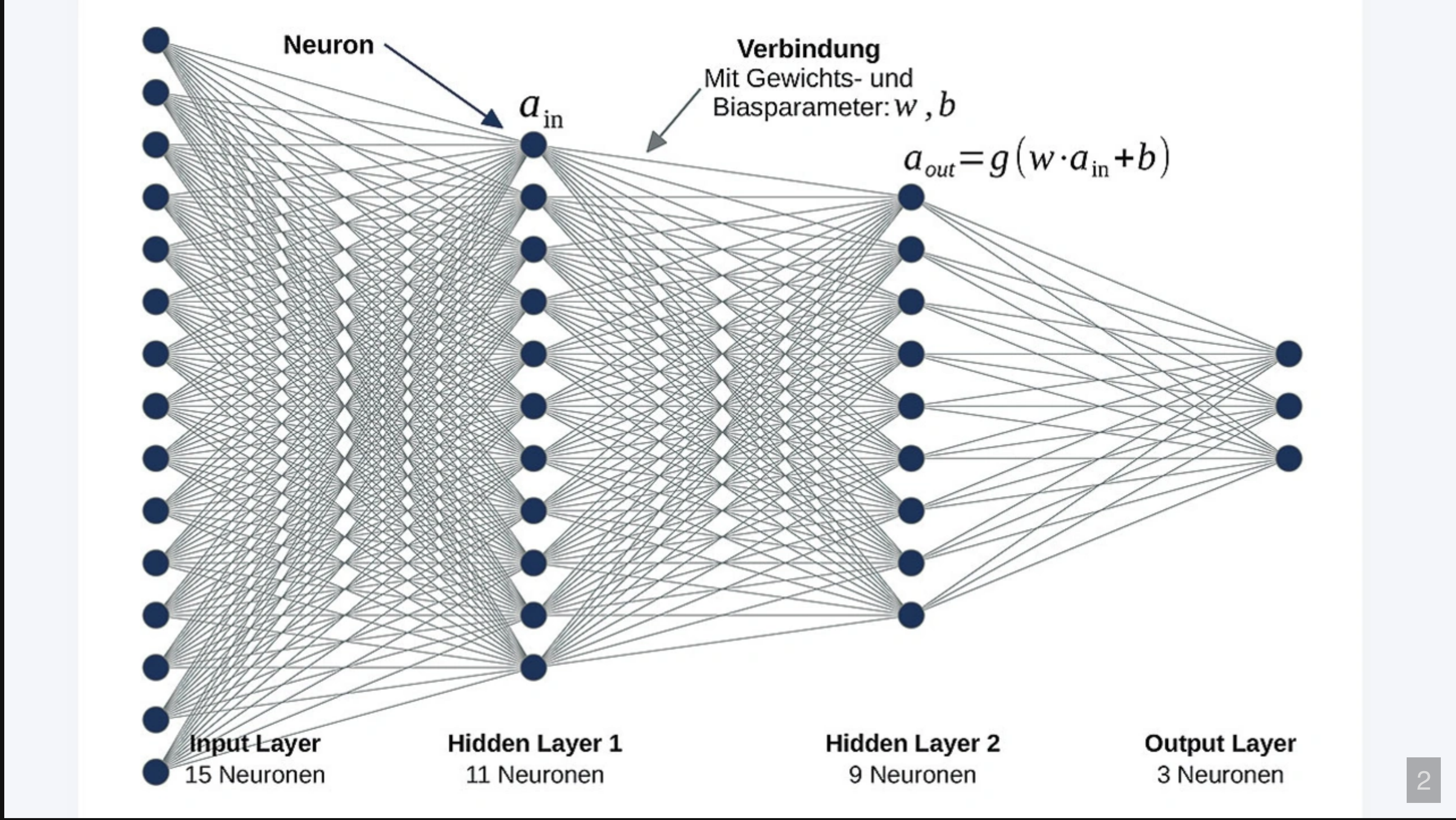
https://www.youtube.com/embed/qFES8S9D8RM?enablejsapi=1

Architektur: Neuronale Netzstruktur

Neuronales Netz = einfache Dartstellung komplexer Rechnung einfacher Bausteine

LinReg mit Basisfunktionen aus LinReg mit Basis aus Linreg mit Basis aus ...

Modell = Architektur mit Satz von Parametern



mehr zu Neuronalen Netzen

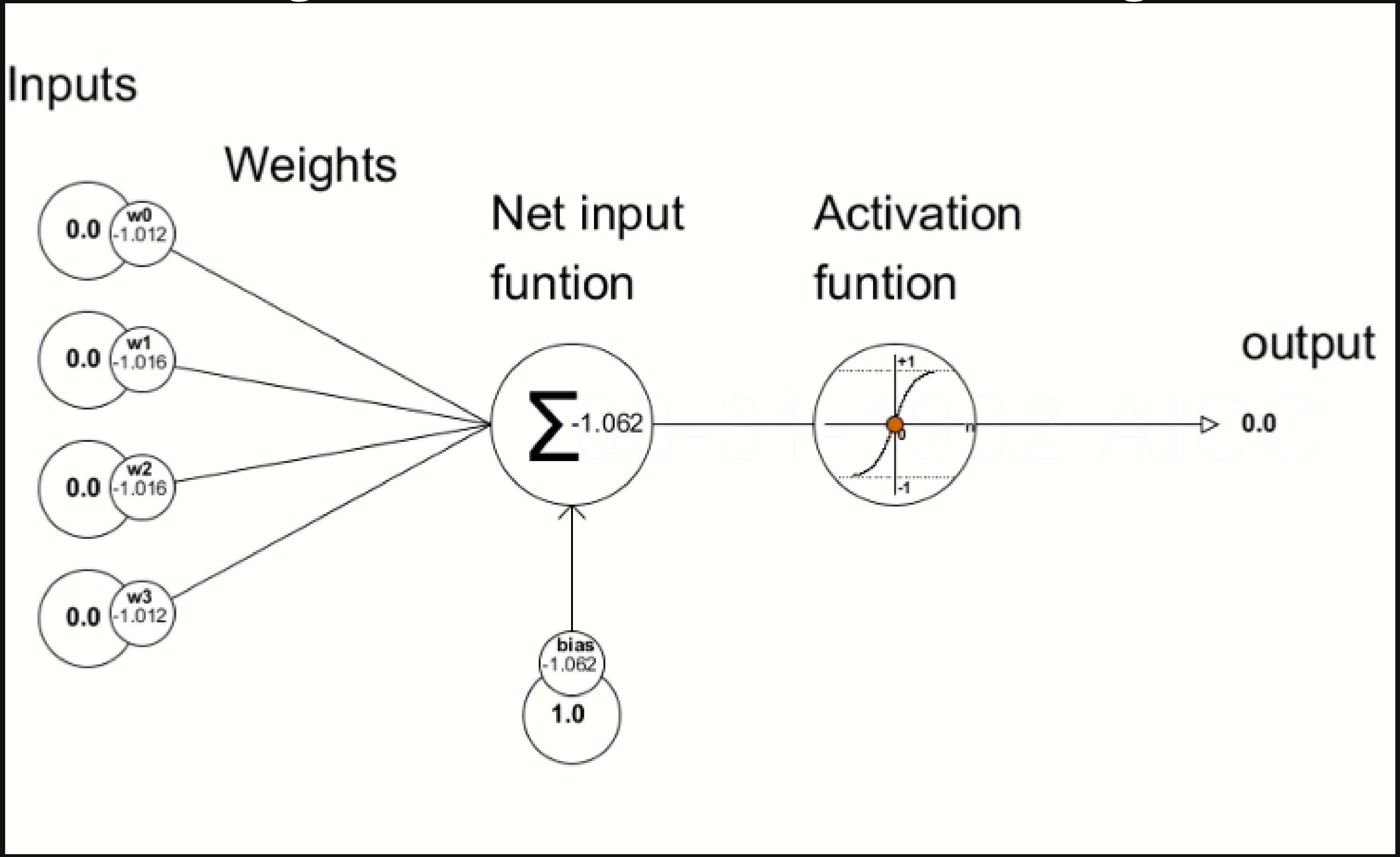
Layer

https://www.youtube.com/embed/ynFZ3wBqsao?enablejsapi=1

Layer = Level für Lineare Regression

Mehrere Knoten (Perceptronen)

Knoten = gewichtete Summe & Aktivierungsfunktion



Aktivierungsfunktion

Wofür Aktivierung?

Ohne Aktivierung: N Layer = 1 Layer

Lineare Kombination
von Linearkombinationen
ist eine Linearkombination

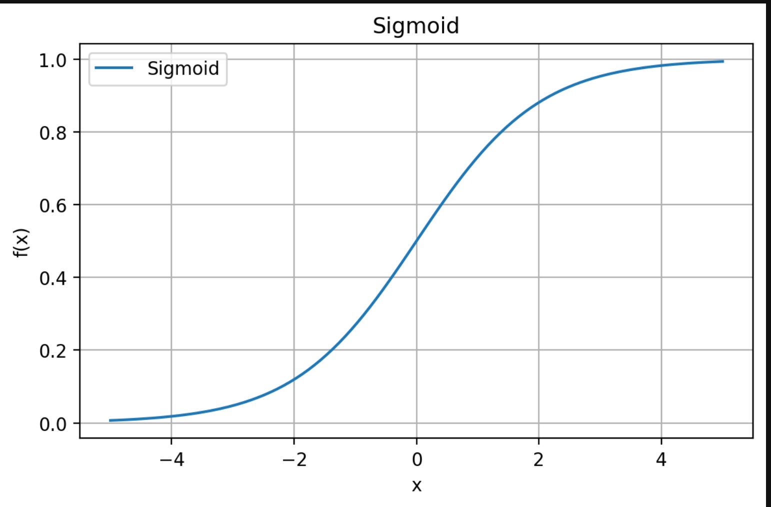
Nicht-Lineare Aktivierung
ermöglicht komplexe Modellierung

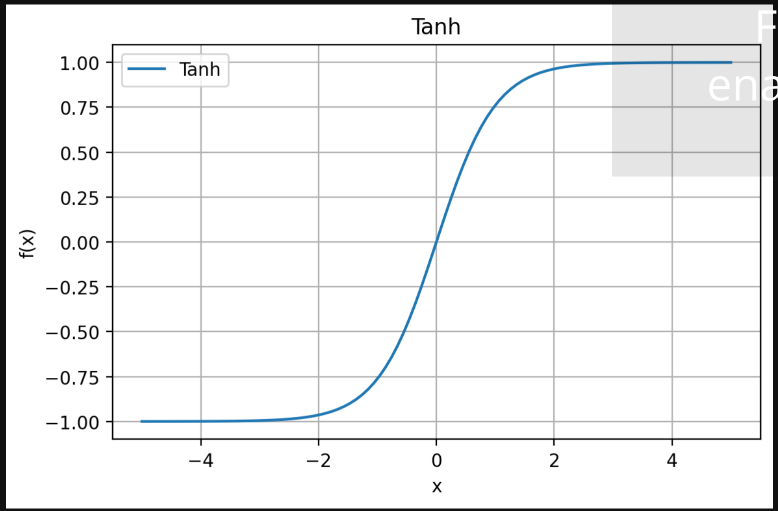
$$
\begin{aligned}
z_{1,j} &= \sum_{i=1}^{n} c_{1,ij} \cdot x_i \\
z_2 &= \sum_{j=1}^{m} c_{2,j} \cdot z_{1,j} \\
&= \sum_{j=1}^{m} c_{2,j} \cdot \left(\sum_{i=1}^{n} c_{1,ij} \cdot x_i \right) \\
&= \sum_{i=1}^{n} \left(\sum_{j=1}^{m} c_{2,j} \cdot c_{1,ij} \right) \cdot x_i \\
&= \sum_{i=1}^{n} c_{\text{ges},i} \cdot x_i
\end{aligned}
$$

$$
\begin{aligned}
\mathbf{z_1} &= \mathbf{W_1 x} + \mathbf{b_1} \\
\mathbf{z_2} &= \mathbf{W_2 z_1} + \mathbf{b_2} \\
&= \mathbf{W_2}(\mathbf{W_1 x} + \mathbf{b_1}) + \mathbf{b_2} \\
&= \mathbf{W_2 W_1 x} + \mathbf{W_2 b_1} + \mathbf{b_2} \\
&= \mathbf{W_{ges} x} + \mathbf{b_{ges}}
\end{aligned}
$$

Aktivierungsfunktion

https://www.youtube.com/embed/lRN7F4E25hk?enablejsapi=1



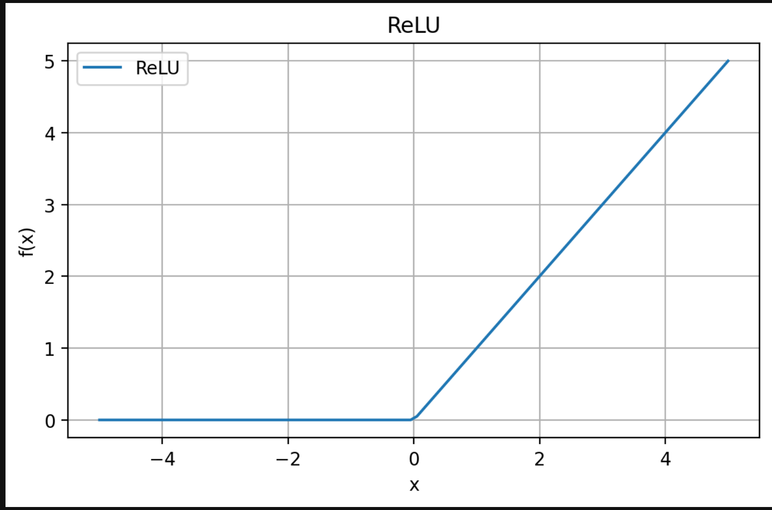


Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

$$\mathrm{ReLU}(x) = \max(0, x)$$



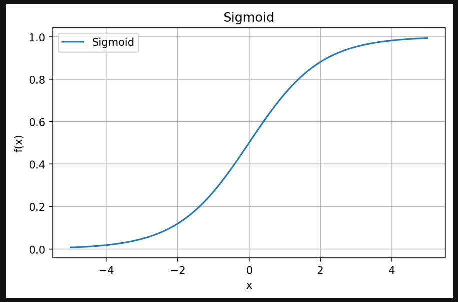
Wahl der Aktivierung:

- Output: Wertebereich Target
- Hidden: Effizienz (?)

Mehr zu Aktivierungen

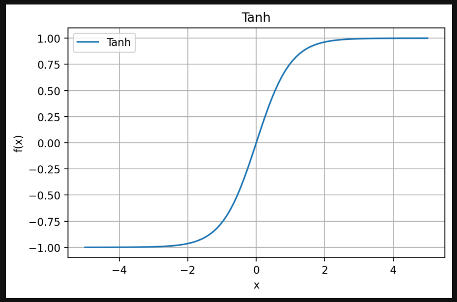
Aktivierungsfunktion

Sigmoid



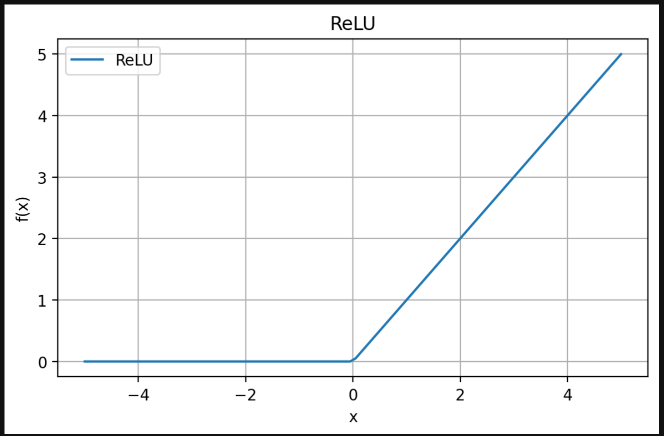
$$\sigma(x) = \frac{1}{1+e^{-x}}$$

$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$



$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

$$\tanh'(x) = 1 - \tanh^2(x)$$



$$\mathrm{ReLU}(x) = \max(0, x)$$

$$\mathrm{ReLU}'(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \end{cases}$$

Wahl der Aktivierung:

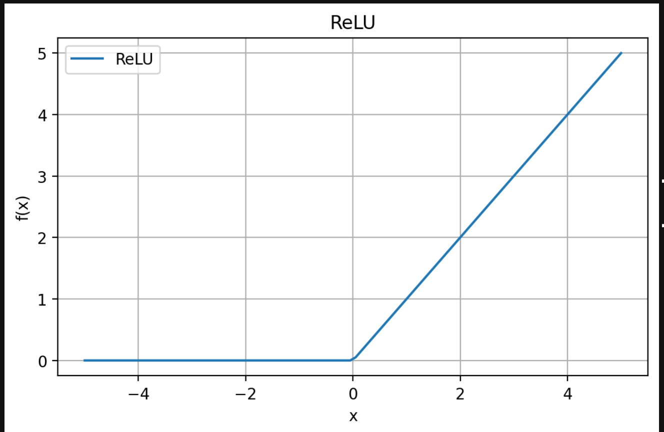
- Output: Wertebereich Target
- Hidden: Effizienz (ReLU)

Mehr zu Aktivierungen

Aktivierungsfunktion

Dying ReLU:

- Permanent x < 0
- ReLU & Ableitung = 0
- Perceptron lernt nicht
- Wird Unbrauchbar



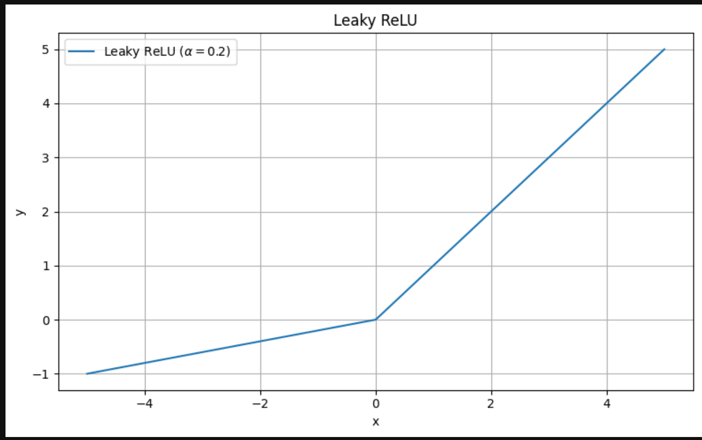
$$\mathrm{ReLU}(x) = \max(0, x)$$

$$\mathrm{ReLU}'(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \end{cases}$$

Aktivierungsfunktion

$$\text{Leaky ReLU}(x) = \max(\alpha x, x)$$

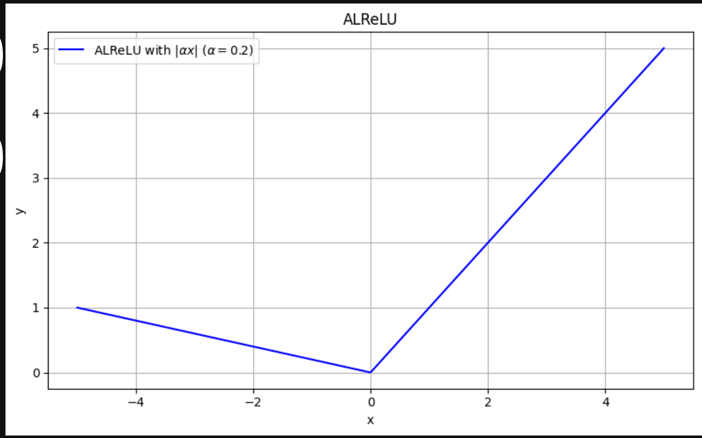
$$\text{Leaky ReLU}'(x) = \begin{cases} \alpha & \text{if } x < 0 \\ 1 & \text{if } x > 0 \end{cases}$$



Dying ReLU:

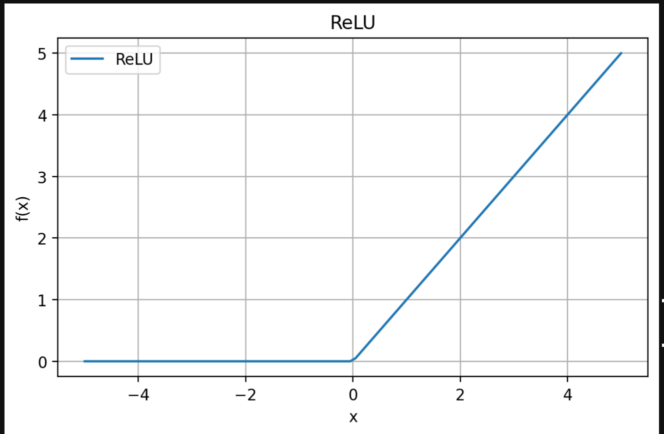
- Permanent x < 0
- ReLU & Ableitung = 0
- Perceptron lernt nicht
- Wird Unbrauchbar

$$\text{ALReLU}(x) = \begin{cases} x & \text{if } x > 0 \\ |\alpha \cdot x| & \text{if } x < 0 \end{cases}$$



nützlich bei sehr seltener Aktivierung

$$\text{ALReLU}'(x) = \begin{cases} 1 & \text{if } x > 0 \\ -\alpha & \text{if } x < 0 \end{cases}$$



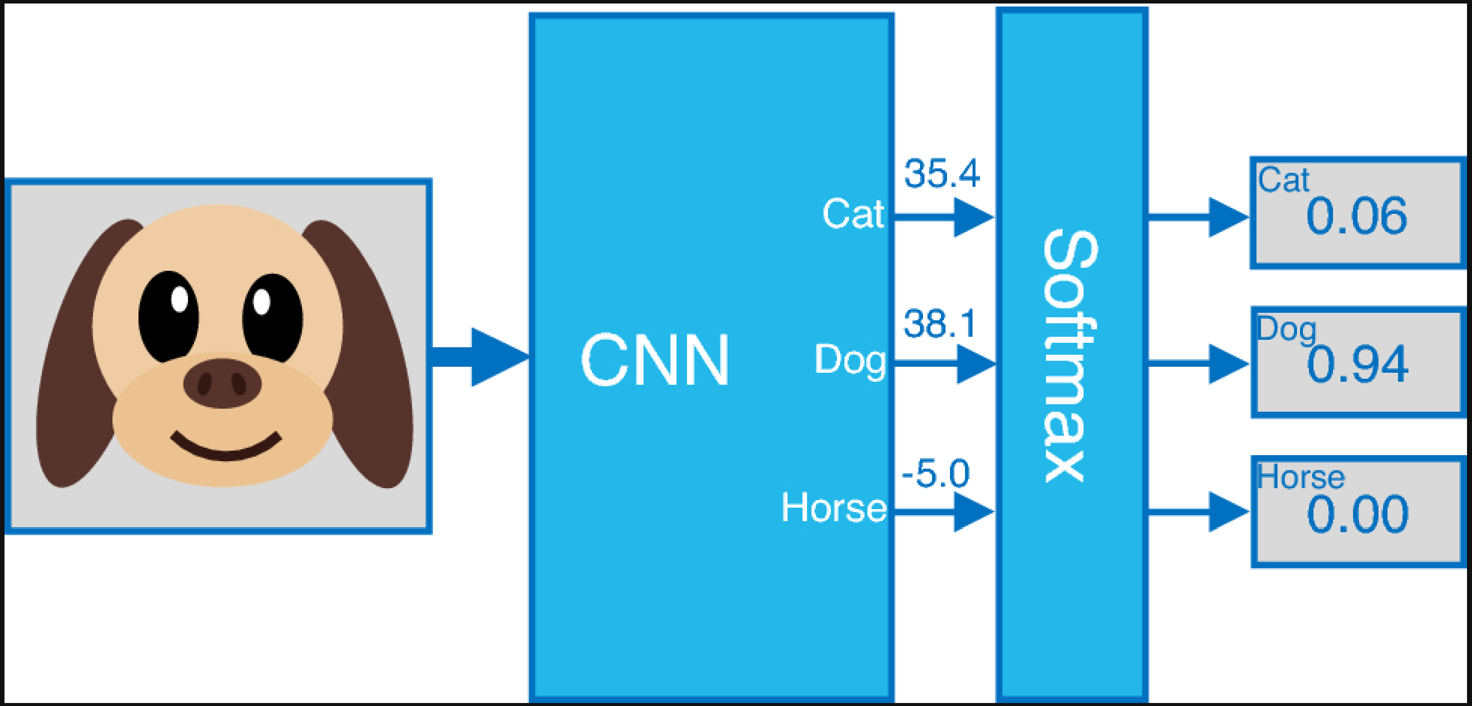
$$\text{ReLU}(x) = \max(0, x)$$

$$\text{ReLU}'(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \end{cases}$$

PReLU, RReLU, ELU, SELU, GELU, Swish, Mish, ReLU6, ...

Aktivierungsfunktion

https://www.youtube.com/embed/32bEK-m_30I?enablejsapi=1



Implementation

	Pytorch	**Tensorflow**
Eigenschaften	Pythonic, einfache Syntax schnelleres Training dynamischer Berenchnungsgraph höhere Flexibilität	Skalierbar Speichereffizient statisch oder dynamisch
Hauptanwendun g	Forschung Prototyping	Grossprojekte Produktion
Community	Forschung	Industrie
Pakete	TorchText, TorchVision, TorchAudio	TF Extended, TF Lite, TF Serving

Beide Frameworks sehr nützlich & weit verbreitet

Mathematik identisch & Aufbau sehr ähnlich

Wahl meist durch Arbeitsumfeld bestimmt

Architektur entwerfen

https://www.youtube.com/embed/VakbvsvzbD4?enablejsapi=1

Wenn möglich, bereits existierende Architektur / Modelle verwenden

1. Aufgabe klar definieren (Klassifikation, Regression, Erkennung, ...)
2. Ein- und Ausgabedimension festlegen (MNIST: In: 784; Out: 10)
3. Geeignete Art von Schichten bestimmen (Linear, Convolutional, ...)
4. Anzahl Schichten und Neuronen pro Schicht festlegen
5. Aktivierungsfunktionen festlegen (Hidden & Output)

Implementation: Architektur

- MNIST Classifier

Implementation: Architektur

- MNIST Classifier

```python
from torch import nn
import torch.nn.functional as F

class Classifier(nn.Module):
    def __init__(self):
        super().__init__()





    def forward(self, x):




        return x

model = Classifier()
output = model(data)
```

Pytorch
(Meta)

Implementation: Architektur

- MNIST Classifier
- 10 Outputs

```python
from torch import nn
import torch.nn.functional as F

class Classifier(nn.Module):
    def __init__(self):
        super().__init__()



        self.fc4 = nn.Linear(784, 10)

    def forward(self, x):



        x = self.fc4(x)
        return x

model = Classifier()
output = model(data)
```

Pytorch
(Meta)

Implementation: Architektur

- MNIST Classifier
- 10 Outputs
- 3 Hidden Layer

```python
from torch import nn
import torch.nn.functional as F

class Classifier(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(784, 64)
        self.fc2 = nn.Linear(64, 64)
        self.fc3 = nn.Linear(64, 64)
        self.fc4 = nn.Linear(64, 10)

    def forward(self, x):
        x = self.fc1(x)
        x = self.fc2(x)
        x = self.fc3(x)
        x = self.fc4(x)
        return x

model = Classifier()
output = model(data)
```

Pytorch
(Meta)

Implementation: Architektur

- MNIST Classifier
- 10 Outputs
- 3 Hidden Layer

```python
from torch import nn
import torch.nn.functional as F

class Classifier(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(784, 64)
        self.fc2 = nn.Linear(64, 64)
        self.fc3 = nn.Linear(64, 64)
        self.fc4 = nn.Linear(64, 10)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = F.relu(self.fc3(x))
        x = self.fc4(x)
        return x

model = Classifier()
output = model(data)
```

Pytorch
(Meta)

Implementation: Architektur

- MNIST Classifier
- 10 Outputs
- 3 Hidden Layer
- Softmax activation

```python
from torch import nn
import torch.nn.functional as F

class Classifier(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(784, 64)
        self.fc2 = nn.Linear(64, 64)
        self.fc3 = nn.Linear(64, 64)
        self.fc4 = nn.Linear(64, 10)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = F.relu(self.fc3(x))
        x = F.softmax(self.fc4(x), dim=1)
        return x

model = Classifier()
output = model(data)
```

Pytorch
(Meta)

Implementation: Architektur

https://www.youtube.com/embed/rQ6v-xkh4cA?enablejsapi=1

```python
import tensorflow as tf
from tensorflow.keras import layers

class Classifier(tf.keras.Model):
    def __init__(self):
        super(Classifier, self).__init__()
        self.fc1 = layers.Dense(64, activation='relu')
        self.fc2 = layers.Dense(64, activation='relu')
        self.fc3 = layers.Dense(64, activation='relu')
        self.fc4 = layers.Dense(10, activation='softmax')

    def call(self, x):
        x = self.fc1(x)
        x = self.fc2(x)
        x = self.fc3(x)
        x = self.fc4(x)
        return x

model = Classifier()
model.build((None, 784))
model(data)
```

Tensorflow (Google)

```python
from torch import nn
import torch.nn.functional as F

class Classifier(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(784, 64)
        self.fc2 = nn.Linear(64, 64)
        self.fc3 = nn.Linear(64, 64)
        self.fc4 = nn.Linear(64, 10)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = F.relu(self.fc3(x))
        x = F.softmax(self.fc4(x), dim=1)
        return x

model = Classifier()
output = model(data)
```

Pytorch (Meta)

- MNIST Classifier
- 10 Outputs
- 3 Hidden Layer
- Softmax activation

Implementation: Architektur

https://www.youtube.com/embed/OtPB-4Y0iWQ?enablejsapi=1

```python
from tensorflow.keras import models

model = models.Sequential([
    layers.Dense(64, activation='relu', input_shape=(784,)),
    layers.Dense(64, activation='relu'),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')
])

model(data)
```

Tensorflow (Google)

```python
import torch.nn as nn

model = nn.Sequential(
    nn.Linear(784, 64),
    nn.ReLU(),
    nn.Linear(64, 64),
    nn.ReLU(),
    nn.Linear(64, 64),
    nn.ReLU(),
    nn.Linear(64, 10),
    nn.Softmax(dim=1)
)

output = model(data)
```

Pytorch (Meta)

- MNIST Classifier
- 10 Outputs
- 3 Hidden Layer
- Softmax activation

Trainingsloop

https://www.youtube.com/embed/nWn_5fPEf_E?enablejsapi=1

```
trainloader = DataLoader(trainset, batch_size=256, shuffle=True)
```

1. Daten laden (batch)

```
for images, labels in trainloader:
```

2. Modell anwenden (forward)

```
    prediction = model(images)
```

3. Loss berechnen

```
    loss = criterion(prediction, labels)
```

4. Updates berechnen (backward)

```
    optimizer.zero_grad()
    loss.backward()
```

5. Update durchfüren

```
    optimizer.step()
```

Trainingsloop entwerfen

https://www.youtube.com/embed/StQ9tHHgxaM?enablejsapi=1

1. Aufgabe klar definieren
2. Lossfunktion bestimmen
3. Berechnungsschritte definieren

Loss Funktion

https://www.youtube.com/embed/FVhKvumlyZY?enablejsapi=1

- Definiert das Ziel des Trainings
- Ziel: Loss minimieren
- erlaubt Vergleich von Modellen
- verschiedene Losses für verschiedene Aufgaben

Loss Funktion

- Mean Squared Error (MSE):
  mittlerer quadratische Abweichung

$$MSE = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

Loss Funktion

- Mean Squared Error (MSE):
  mittlerer quadratische Abweichung

$$MSE = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

- Binäre Cross-Entropy (BCE):
  vergleich von Wahrscheinlichkeit einer Klasse

$$BCE = -\sum_{i=1}^{N} [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)]$$

Loss Funktion

https://www.youtube.com/embed/Y_Smxn0Wfss?enablejsapi=1

- Mean Squared Error (MSE):
  mittlerer quadratische Abweichung

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

- Binäre Cross-Entropy (BCE):
  vergleich von Wahrscheinlichkeit einer Klasse

$$BCE = -\sum_{i=1}^{N}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)]$$

- Cross-Entropy (CE):
  vergleich von Wahrscheinlichkeiten mehrerer Klassen

$$CE = -\sum_{i=1}^{N}\sum_{c=1}^{C} y_{ic}\log(\hat{y}_{ic})$$

mehr zu Hintergrund und Varianten

Implementation: Training

- Loss: CrossEntropy

Implementation: Training

- Loss: CrossEntropy

```
criterion = nn.CrossEntropyLoss()
```

Pytorch

Implementation: Training

- Loss: CrossEntropy
- Optimizer: Adam

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.003)
```

Pytorch

Implementation: Training

- Loss: CrossEntropy
- Optimizer: Adam

```python
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.003)

for e in range(epochs):
```

Pytorch

Epoche: alle Daten trainieren

Implementation: Training

- Loss: CrossEntropy
- Optimizer: Adam

```python
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.003)

for e in range(epochs):

    for images, labels in trainloader:
```

Pytorch

Batchweise Input & Target

Implementation: Training

- Loss: CrossEntropy
- Optimizer: Adam

```python
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.003)

for e in range(epochs):

    for images, labels in trainloader:
        prediction = model(images)
        loss = criterion(prediction, labels)
```

Pytorch

Forward-Pass

Implementation: Training

- Loss: CrossEntropy
- Optimizer: Adam

```python
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.003)

for e in range(epochs):

    for images, labels in trainloader:
        prediction = model(images)
        loss = criterion(prediction, labels)

        optimizer.zero_grad()
        loss.backward()

```

Pytorch

Backward-Pass

Implementation: Training

- Loss: CrossEntropy
- Optimizer: Adam

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.003)

for e in range(epochs):

    for images, labels in trainloader:
        prediction = model(images)
        loss = criterion(prediction, labels)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

```

Pytorch

Update

Implementation: Training

- Loss: CrossEntropy
- Optimizer: Adam

```python
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.003)

for e in range(epochs):
    running_loss = 0
    for images, labels in trainloader:
        prediction = model(images)
        loss = criterion(prediction, labels)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
```

Pytorch

Implementation: Training

https://www.youtube.com/embed/sjga-FLKP9k?enablejsapi=1

- Loss: CrossEntropy
- Optimizer: Adam

```python
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.003)

for e in range(epochs):
    running_loss = 0
    for images, labels in trainloader:
        prediction = model(images)
        loss = criterion(prediction, labels)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
```

Pytorch

Tensorflow

```python
model.compile(optimizer=optimizers.Adam(learning_rate=0.003),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

history = model.fit(train_images, train_labels, epochs=1, batch_size=64)

print(f'Training loss: {history.history["loss"][0]}')
```

Update

Ableitung der Kosten -> Richtung für Verbesserung -> Update

Gradient Descent $\theta \leftarrow \theta - \alpha \cdot \frac{\partial \mathrm{Loss}}{\partial \theta}$ $\mathrm{Lernrate}\ \alpha$

Update

Ableitung der Kosten -> Richtung für Verbesserung -> Update

Gradient Descent $\theta \leftarrow \theta - \alpha \cdot \frac{\partial \text{Loss}}{\partial \theta}$ $\text{Lernrate } \alpha$

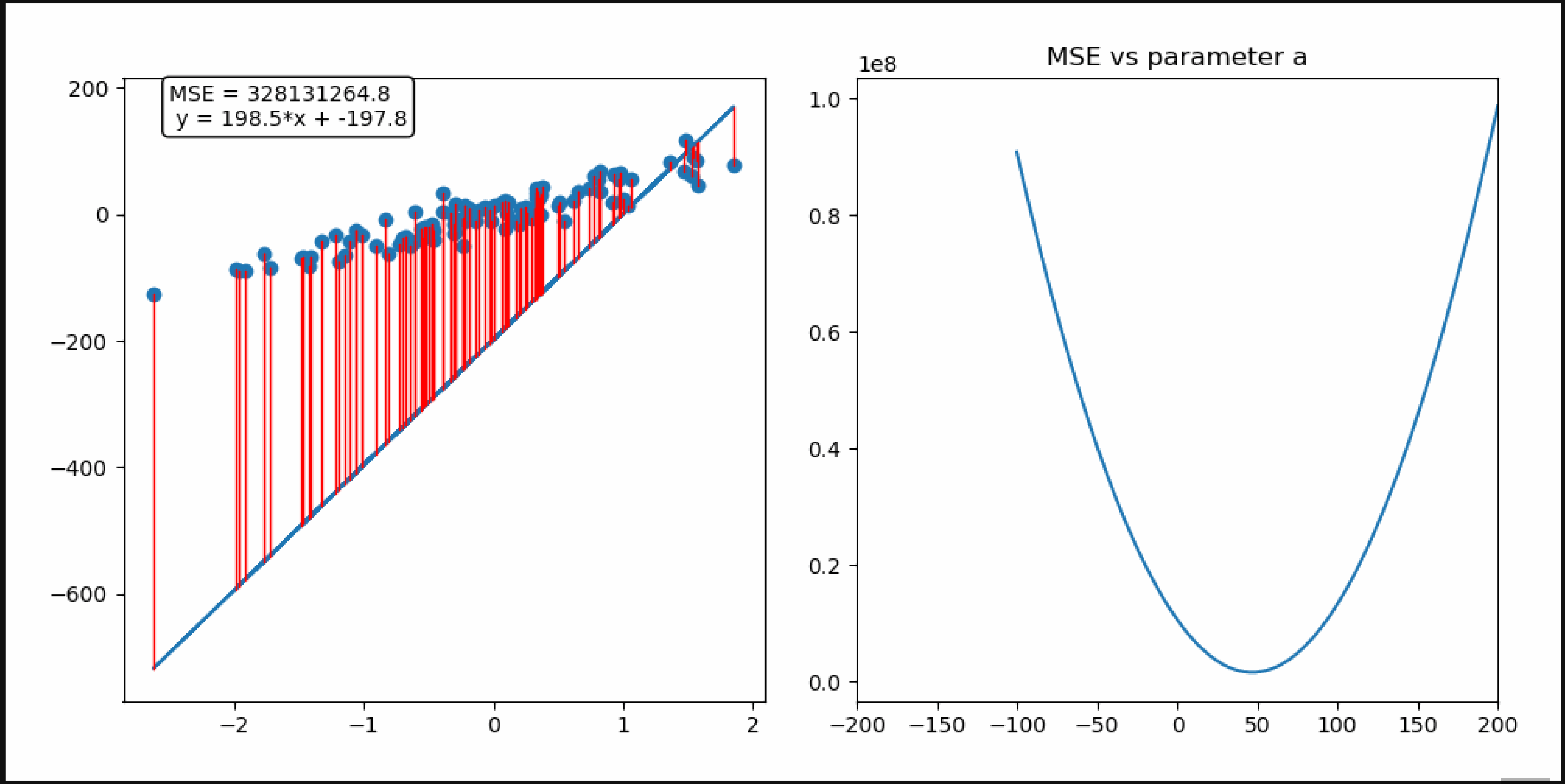
Update

https://www.youtube.com/embed/0S70j3Lh2Jg?enablejsapi=1

Ableitung der Kosten -> Richtung für Verbesserung -> Update

Gradient Descent $\theta \leftarrow \theta - \alpha \cdot \frac{\partial \text{Loss}}{\partial \theta}$

Lernrate α



Update

https://www.youtube.com/embed/wyPHES7Sr_U?enablejsapi=1

Ableitung der Kosten -> Richtung für Verbesserung -> Update

Gradient Descent $\theta \leftarrow \theta - \alpha \cdot \frac{\partial \text{Loss}}{\partial \theta}$

$\text{Lernrate } \alpha$

Optimizer

https://www.youtube.com/embed/DvTnakOFZos?enablejsapi=1

Gradient Descent

$$\theta \leftarrow \theta - \alpha \cdot \frac{\partial \mathrm{Loss}}{\partial \theta}$$

$$Loss = \sum_i (y_i - \hat{y}_i)^2$$

$$\text{Forward: } \hat{y} = f(x)$$

x: Input Daten

y: Ziel Daten

$\hat{y}$: Vorhersage

f: Modell

θ: Parameter

Optimizer

https://www.youtube.com/embed/QiVIuvMSrCs?enablejsapi=1

Gradient Descent

$$\theta \leftarrow \theta - \alpha \cdot \frac{\partial \text{Loss}}{\partial \theta}$$

b: Bias

w: Gewichte

a: Aktivierung

$$Loss = \sum_i (y_i - \hat{y}_i)^2$$

$$\text{Forward: } \hat{y} = f(x) = a(z) = a(b + w \cdot x)$$

Optimizer

Gradient Descent

$$\theta \leftarrow \theta - \alpha \cdot \frac{\partial \mathrm{Loss}}{\partial \theta}$$

$$Loss = \sum_i (y_i - \hat{y}_i)^2$$

$$\text{Forward: } \hat{y} = f(x) = a(z) = a(b + w \cdot x)$$

$$\text{Backward: } \frac{\partial Loss}{\partial b}$$

Optimizer

Gradient Descent

$$\theta \leftarrow \theta - \alpha \cdot \frac{\partial \text{Loss}}{\partial \theta}$$

$$Loss = \sum_i (y_i - \hat{y}_i)^2$$

$$\text{Forward: } \hat{y} = f(x) = a(z) = a(b + w \cdot x)$$

$$\text{Backward: } \frac{\partial Loss}{\partial b} = \frac{\partial Loss}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial a} \cdot \frac{\partial a}{\partial z} \cdot \frac{\partial z}{\partial b}$$

Optimizer

https://www.youtube.com/embed/hvf555EhBsw?enablejsapi=1

Gradient Descent

$$\theta \leftarrow \theta - \alpha \cdot \frac{\partial \text{Loss}}{\partial \theta}$$

$$Loss = \sum_i (y_i - \hat{y}_i)^2$$

$$\text{Forward: } \hat{y} = f(x) = a(z) = a(b + w \cdot x)$$

$$\text{Backward: } \frac{\partial Loss}{\partial b} = \frac{\partial Loss}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial a} \cdot \frac{\partial a}{\partial z} \cdot \frac{\partial z}{\partial b}$$

$$= \sum_i 2(y_i - \hat{y}_i) \cdot 1 \cdot \frac{\partial a}{\partial z} \cdot 1$$

Optimizer

Gradient Descent

$$\theta \leftarrow \theta - \alpha \cdot \frac{\partial \text{Loss}}{\partial \theta}$$

$$Loss = \sum_i (y_i - \hat{y}_i)^2$$

$$\text{Forward: } \hat{y} = f(x) = a(z) = a(b + w \cdot x)$$

$$\text{Backward: } \frac{\partial Loss}{\partial b} = \frac{\partial Loss}{\partial a} \cdot \frac{\partial a}{\partial z} \cdot \frac{\partial z}{\partial b}$$

$$= \sum_i 2(y_i - \hat{y}_i) \cdot \frac{\partial a}{\partial z}$$

Optimizer

https://www.youtube.com/embed/kPKfXFgClu0?enablejsapi=1

Gradient Descent

$$\theta \leftarrow \theta - \alpha \cdot \frac{\partial \text{Loss}}{\partial \theta}$$

$$Loss = \sum_i (y_i - \hat{y}_i)^2$$

$$\text{Forward: } \hat{y}_i = f(x_i) = a_1(z_1(x_i)) = a_1(b_1 + w_1 \cdot x_i)$$

$$\text{Backward: } \frac{\partial Loss}{\partial b_1} = \frac{\partial Loss}{\partial a_1} \cdot \frac{\partial a_1}{\partial z_1} \cdot \frac{\partial z_1}{\partial b_1}$$

$$= \sum_i 2(y_i - \hat{y}_i) \frac{\partial a_1}{\partial z_1}$$

Optimizer

https://www.youtube.com/embed/TXWRUcSuars?enablejsapi=1

Gradient Descent

$$\theta \leftarrow \theta - \alpha \cdot \frac{\partial \text{Loss}}{\partial \theta}$$

$$Loss = \sum_i (y_i - \hat{y}_i)^2$$

$$\text{Forward: } \hat{y}_i = f(x_i) = a_2(z_2(a_1(z_1(x_i)))) = a_2(b_2 + w_2 \cdot a_1(b_1 + w_1 \cdot x_i))$$

$$\text{Backward: } \frac{\partial Loss}{\partial b_1} = \frac{\partial Loss}{\partial a_2} \cdot \frac{\partial a_2}{\partial z_2} \cdot \frac{\partial z_2}{\partial a_1} \cdot \frac{\partial a_1}{\partial z_1} \cdot \frac{\partial z_1}{\partial b_1}$$

$$= \sum_i 2(y_i - \hat{y}_i) \cdot \frac{\partial a_2}{\partial z_2} \cdot w_2 \cdot \frac{\partial a_1}{\partial z_1}$$

Optimizer

Gradient Descent

https://www.youtube.com/embed/Fn_U_GPg6iI?enablejsapi=1

$$\theta \leftarrow \theta - \alpha \cdot \frac{\partial \text{Loss}}{\partial \theta}$$

$$Loss = \sum_i (y_i - \hat{y}_i)^2$$

$$\text{Forward: } \hat{y}_i = f(x_i) = a_3(z_3(a_2(z_2(a_1(z_1(x_i)))))) = a_3(b_3 + w_3 \cdot a_2(b_2 + w_2 \cdot a_1(b_1 + w_1 \cdot x_i)))$$

$$\text{Backward: } \frac{\partial Loss}{\partial b_1} = \frac{\partial Loss}{\partial a_3} \cdot \frac{\partial a_3}{\partial z_3} \cdot \frac{\partial z_3}{\partial a_2} \cdot \frac{\partial a_2}{\partial z_2} \cdot \frac{\partial z_2}{\partial a_1} \cdot \frac{\partial a_1}{\partial z_1} \cdot \frac{\partial z_1}{\partial b_1}$$

$$= \sum_i 2(y_i - \hat{y}_i) \cdot \frac{\partial a_3}{\partial z_3} \cdot w_3 \cdot \frac{\partial a_2}{\partial z_2} \cdot w_2 \cdot \frac{\partial a_1}{\partial z_1}$$

Optimizer

Gradient Descent

$$\theta \leftarrow \theta - \alpha \cdot \frac{\partial \mathrm{Loss}}{\partial \theta}$$

Berechnet Ableitung $\frac{\partial \mathrm{Loss}}{\partial \theta}$
für jedes Update
mit gesamten Datensatz

folgt exakt steilstem Abstieg

Bei Millionen von Daten
sehr rechenintensiv...

Optimizer

$$\theta \leftarrow \theta - \alpha \cdot \frac{\partial \mathrm{Loss}}{\partial \theta}$$

Gradient Descent

Berechnet Ableitung $\frac{\partial \mathrm{Loss}}{\partial \theta}$
für jedes Update
mit gesamten Datensatz

folgt exakt steilstem Abstieg

Bei Millionen von Daten
sehr rechenintensiv...

Stochastic Gradient Descent (SGD)

Zerlegt Datensatz in Batches
Update nach jeder Batch

folgt Schätzung des steilsten Abstiegs
Forschritt kann Schwanken

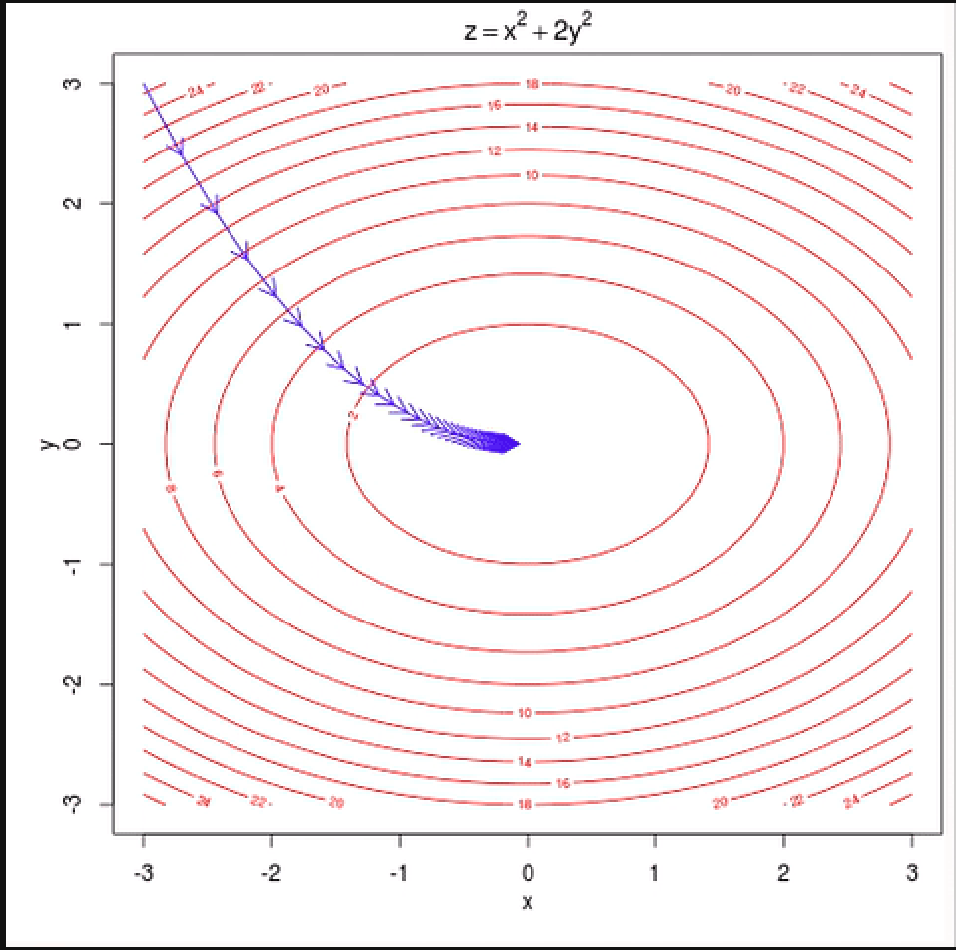
Schnelle Konvergenz
dank schneller Rechnung

Optimizer

https://www.youtube.com/embed/WVUapM2JeJ0?enablejsapi=1

Gradient Descent

$$\theta \leftarrow \theta - \alpha \cdot \frac{\partial \mathrm{Loss}}{\partial \theta}$$



Optimizer

https://www.youtube.com/embed/213nGdgPjx4?enablejsapi=1

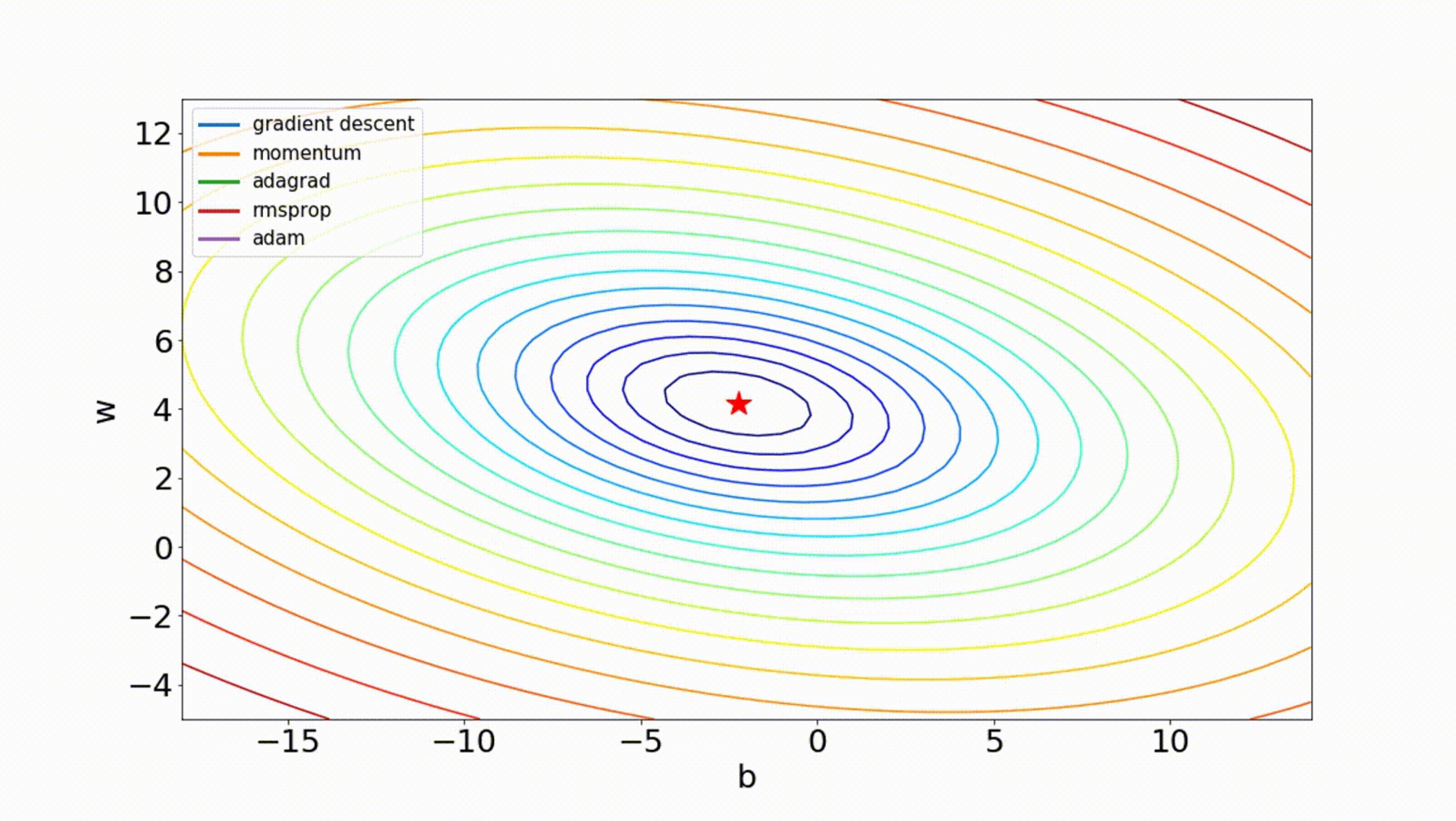
SGD

$$\theta \leftarrow \theta - \alpha \cdot \frac{\partial \mathrm{Loss}}{\partial \theta}$$

Momentum

$$v \leftarrow \beta \cdot v + \alpha \cdot \frac{\partial \mathrm{Loss}}{\partial \theta}$$

$$\theta \leftarrow \theta - v$$



mehr zu Optimizern

Model & Trainingsloop

https://www.youtube.com/embed/m6pkOvmWXdo?enablejsapi=1

Hands-On: MNIST Classifier

Bearbeiten Sie dieses Notebook

- Erstellen Sie einen MNIST Classifier
- Definieren Sie die Trainingsloop
- Testen sie die Trainingsloop ueber zwei Epochen

Die Lösung finden Sie in diesem Notebook