Hahn Schickard

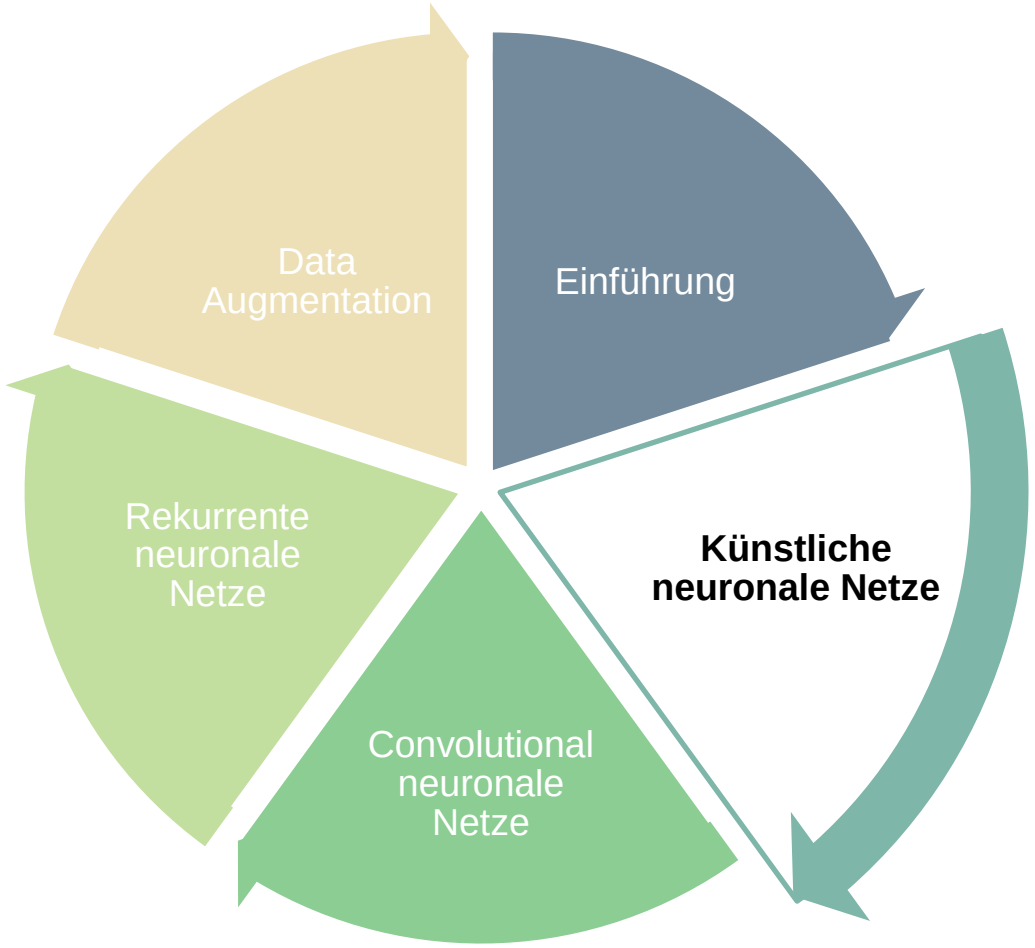
Visions to Products

Neural Networks

Marcus Rüb

Hahn-Schickard Villingen-Schwenningen
Marcus.rueb@hahn-schickard.de

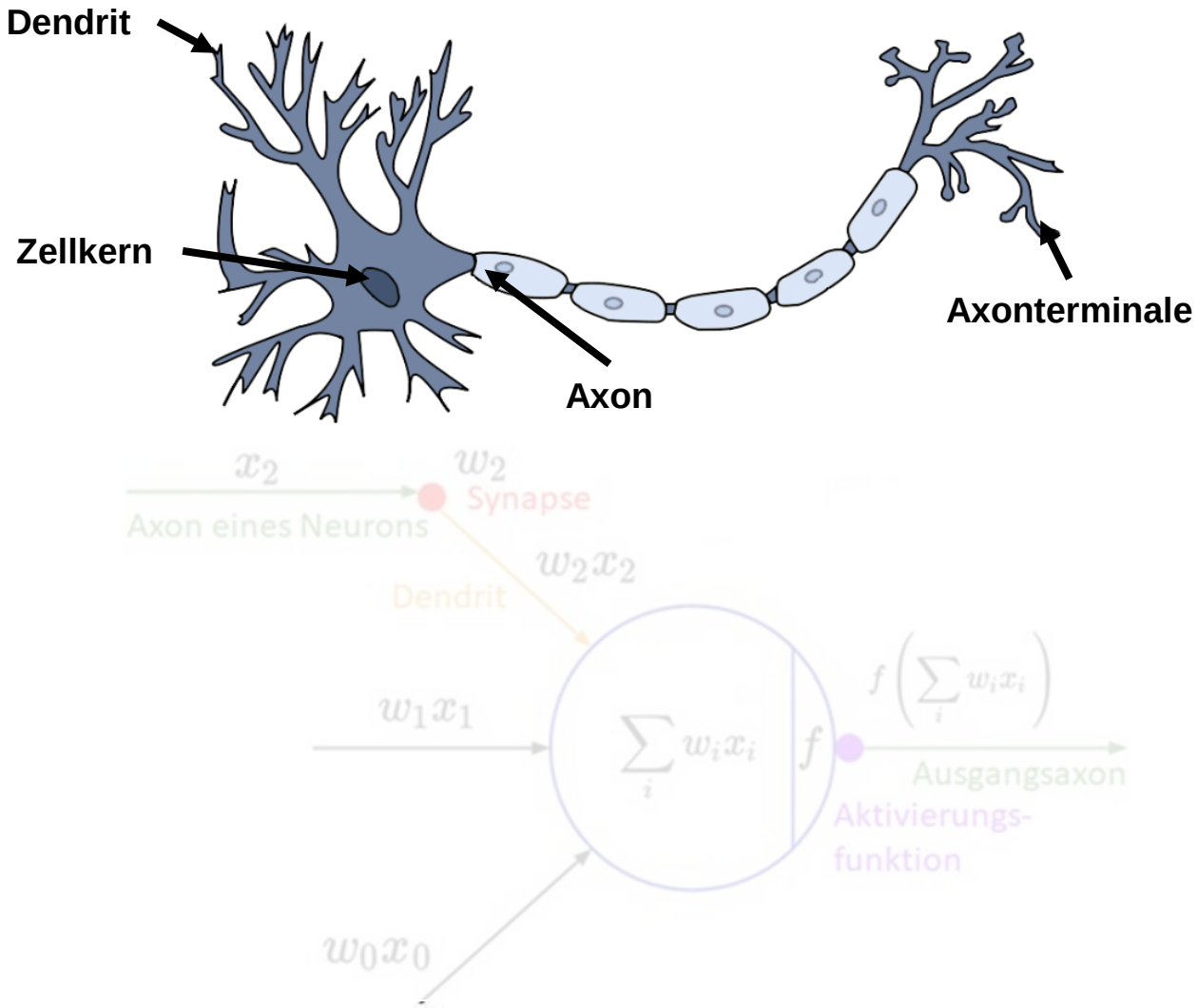
Deep Learning view



Künstliche neuronale Netze: Part 1

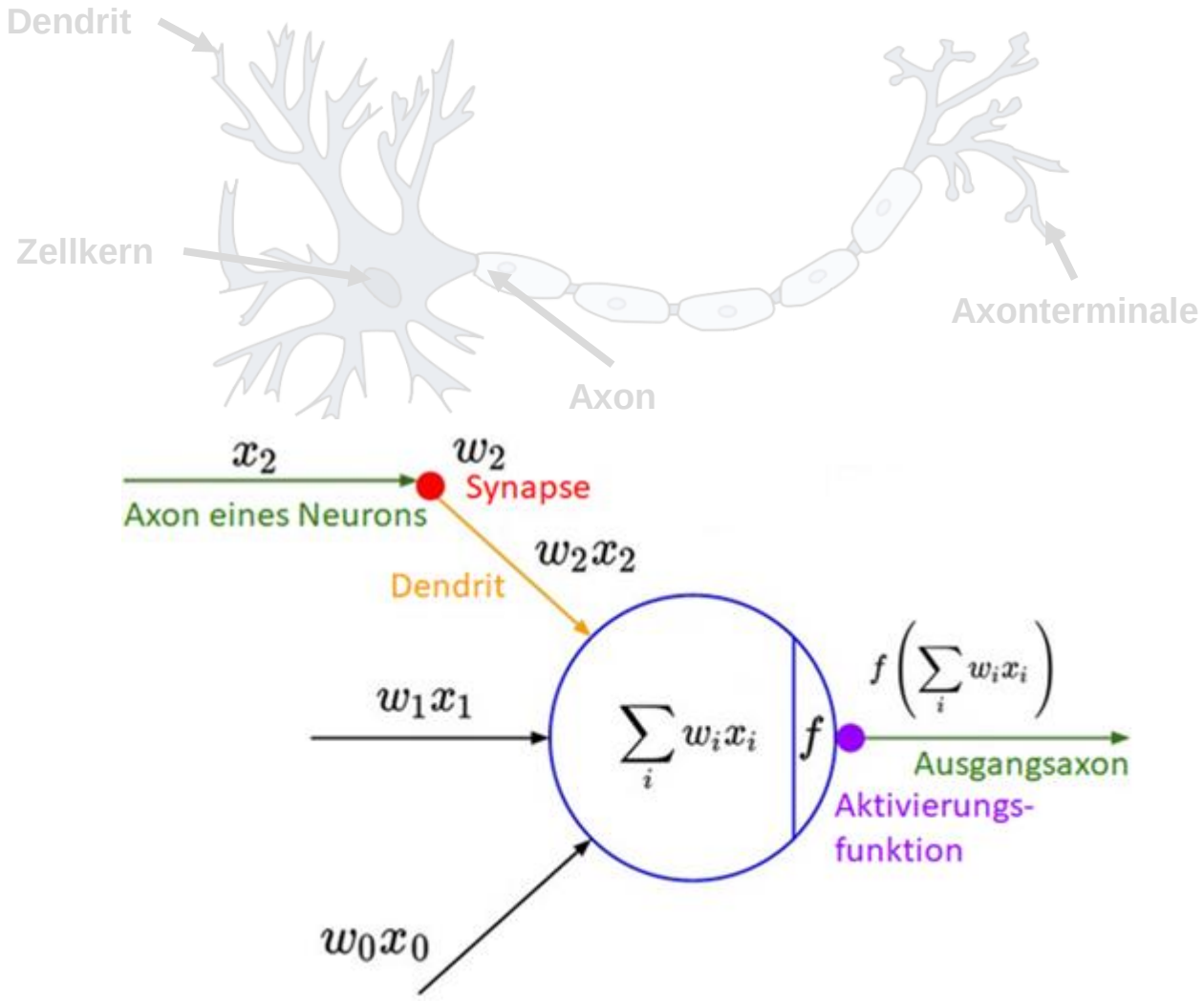
- Einführung
  - Das biologische Neuron
  - Das künstliche Neuron
  - Das XOR Problem
- Training neuronaler Netze

Artificial Neural Networks - KNN
The Artificial Neuron



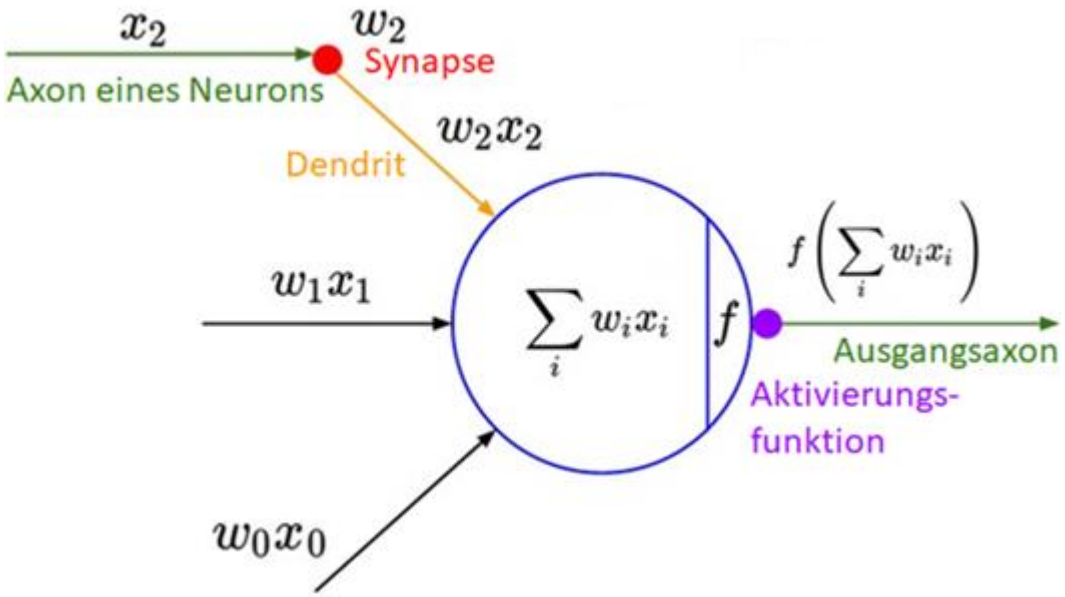
- **The Biological Neuron**
- Electrically excitable cell (recording, processing and forwarding of information)
- Basic unit of a biological brain
- Transmits electrical signals from the dendrites along the axon to the terminals
- Signals are passed from neurons to neurons
- Output signal only occurs when the input exceeds a threshold value:
- The neuron fires
- In this way we perceive light, sounds, pressure heat, etc.

Artificial Neural Networks - KNN
The Artificial Neuron



- **The Artificial Neuron - Perzeptron**
- Basic unit of a neural network
- Inputs and outputs are numbers (instead of binary on/off values)
- Each input connection is connected with a weight
- Calculation of the weighted sum of all inputs
- Applying an activation function
- Linking these building blocks form artificial neural networks

Artificial Neural Networks - KNN
The Artificial Neuron



Forward propagation

Weighted sum:

$$z = \sum_{i=0}^{n} \omega_i \cdot x_i = \boldsymbol{\omega}^{\mathrm{T}} \cdot \boldsymbol{x}$$

Bias is included in the weight vector

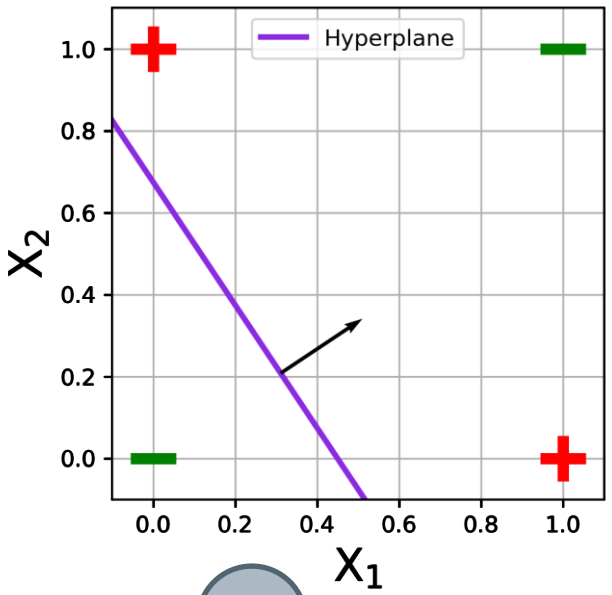
with $\boldsymbol{\omega} = \begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_n \end{pmatrix}$ $\mathbf{x} = \begin{pmatrix} 1 \\ x_1 \\ \vdots \\ x_n \end{pmatrix}$

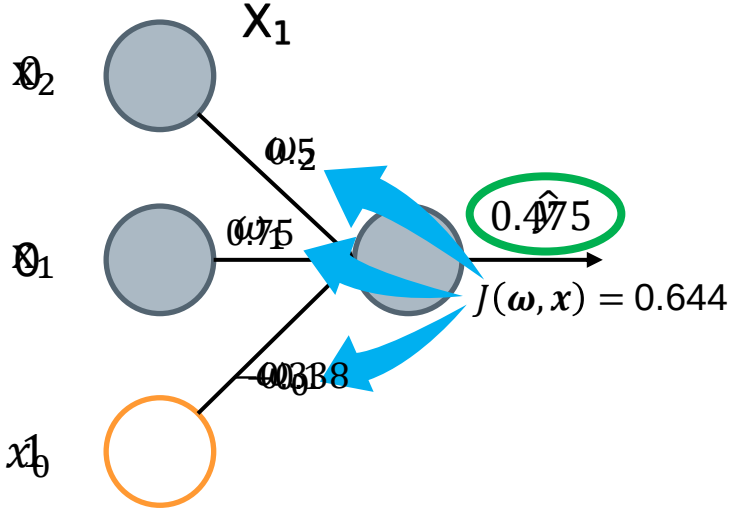
- Activation function
  (step function, signumfunction):

$$f(z) = f\left(\boldsymbol{\omega}^{\mathrm{T}} \cdot \boldsymbol{x}\right) = \begin{cases} 0 & \text{falls} \quad \boldsymbol{\omega}^{\mathrm{T}} \cdot \boldsymbol{x} < 0 \\ 1 & \text{sonst} \end{cases}$$

Artificial Neural Networks - KNN

The XOR Problem





The XOR Problem

Example of a nonlinear separable problem

Create data

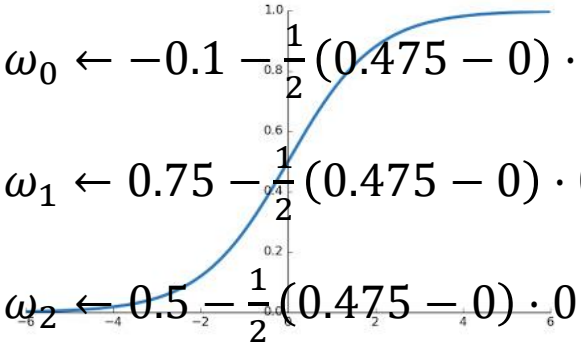
$$X = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} \quad y = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \quad W = \begin{bmatrix} \omega_0 \\ \omega_1 \\ \omega_2 \end{bmatrix}$$

1. Initialize weights to 0 or small random numbers
2. For each training epoch (iteration over the training set)
   a. For each training sample $x^{(i)}$
      i. Compute the output value $\hat{y}$ (binary class label or continuous value) via the unit step function: $\hat{y} = \begin{cases} 1 & \text{if } z \geq \theta \\ 0 & \text{otherwise} \end{cases}$ and $z = \omega_1 x_1^{(i)} + \omega_2 x_2^{(i)} + \cdots + \omega_m x_m^{(i)} = \boldsymbol{\omega}^T \boldsymbol{x}^{(i)}$
      ii. Update the weights: $\omega_j := \omega_j + \Delta\omega_j$ and $\Delta\omega_j = \eta(y^{(i)} - \hat{y}^{(i)})x_j^{(i)}$

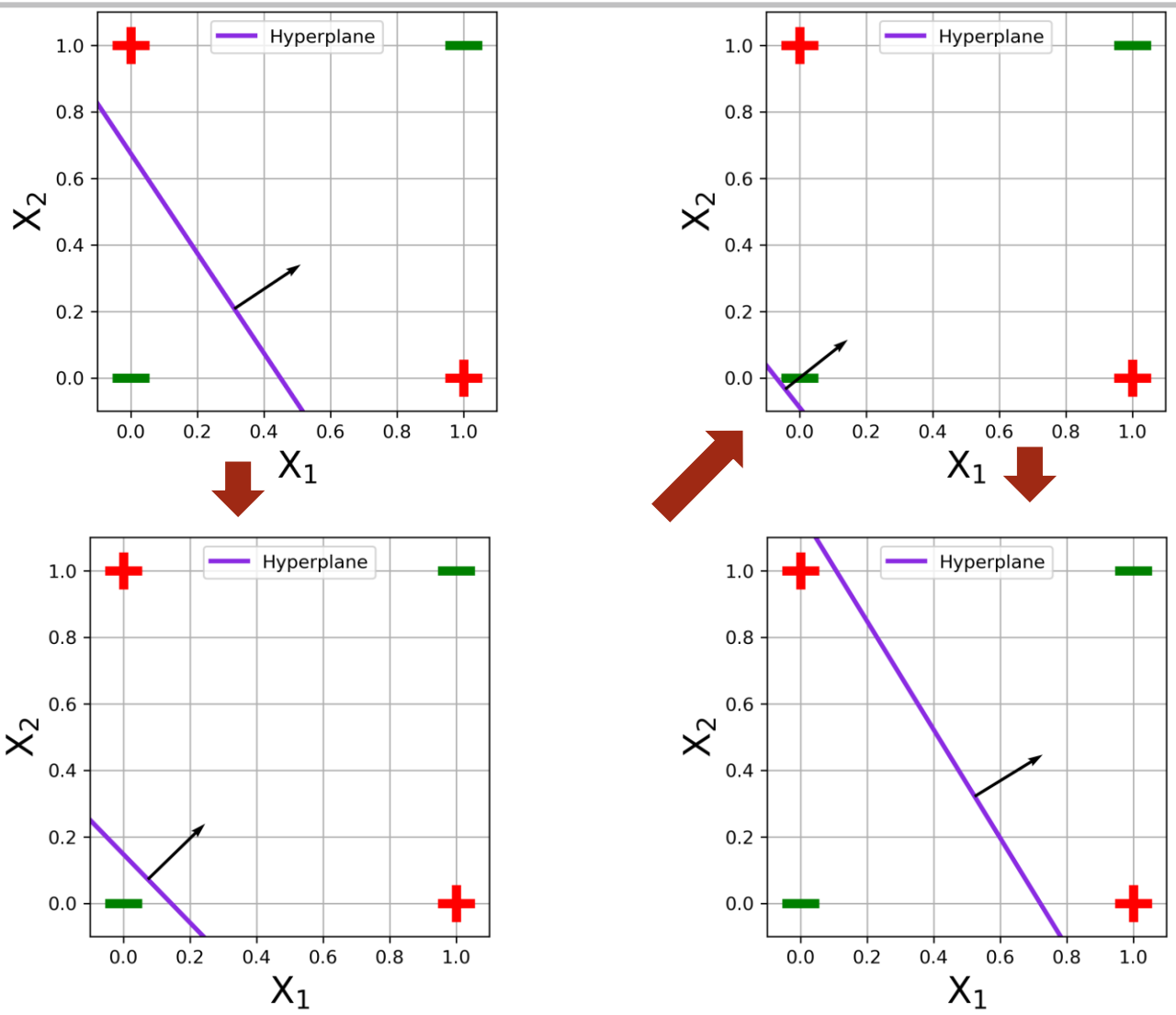
$$J(\boldsymbol{\omega}, \boldsymbol{x}) = -[y \log(\hat{y}) + (1-y)\log(1-\hat{y})]$$

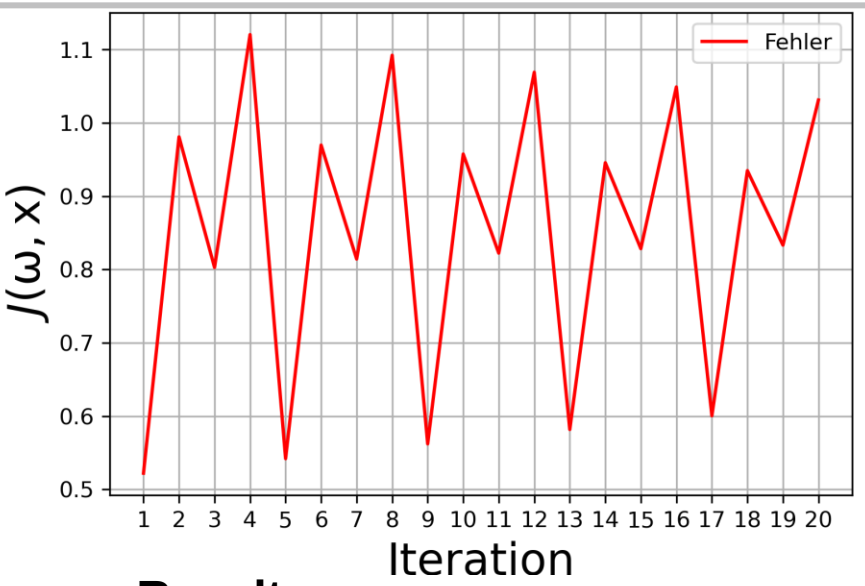
$$J(\boldsymbol{\omega}, \boldsymbol{x}) = -[0 \log(0.475) + (1-0)\log(1-0.475)] \approx 0.644$$

- $\omega_0 \leftarrow -0.1 - \frac{1}{2}(0.475 - 0) \cdot 1 = -0.338$
- $\omega_1 \leftarrow 0.75 - \frac{1}{2}(0.475 - 0) \cdot 0 = 0.75$
- $\omega_2 \leftarrow 0.5 - \frac{1}{2}(0.475 - 0) \cdot 0 = 0.5$



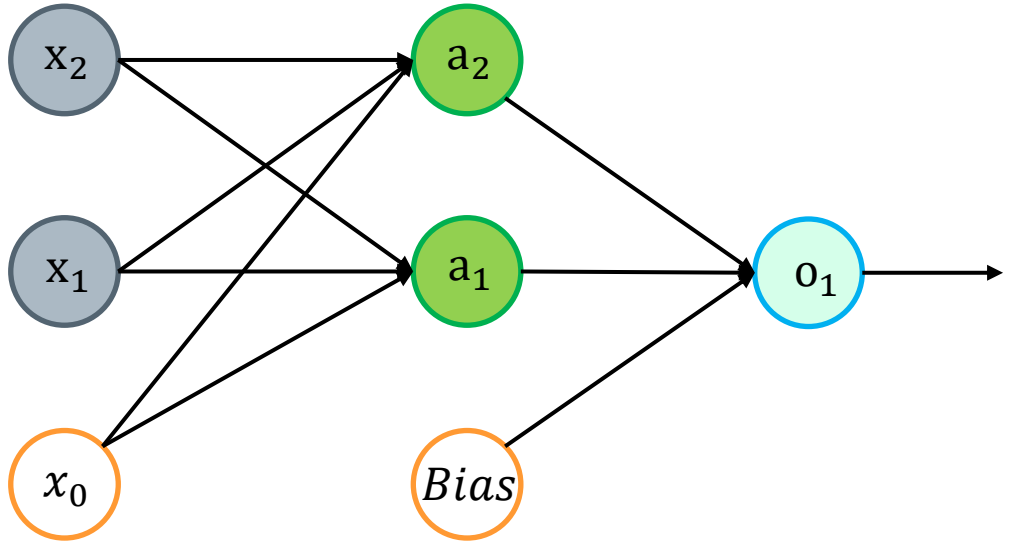
Artificial Neural Networks - KNN
The XOR Problem



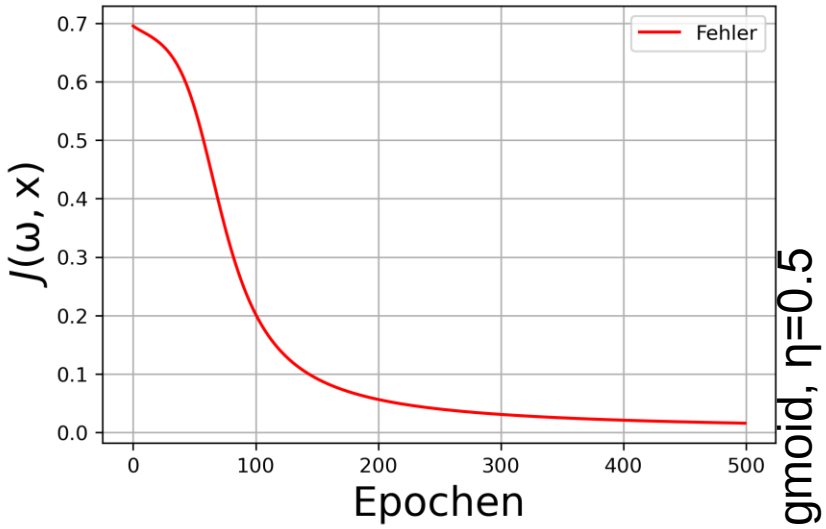


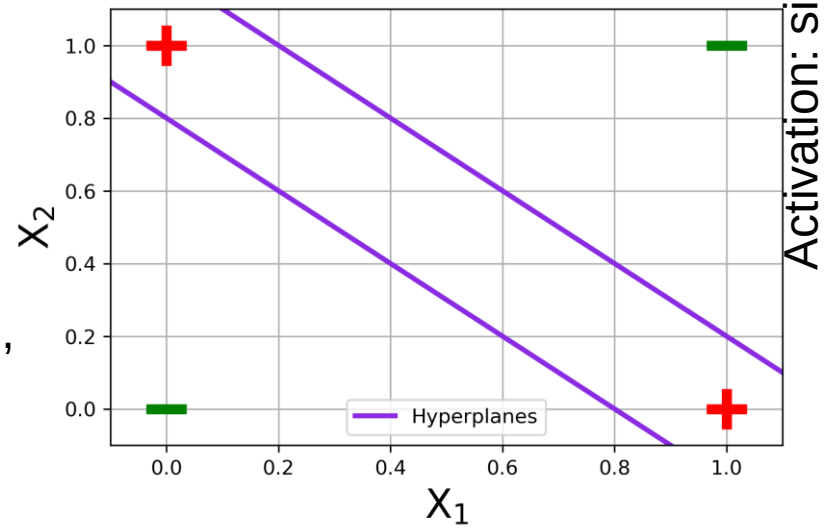
- **Result**
- Individual neurons cannot model the XOR problem
- What about the networking of several artificial neurons?

Artificial Neural Networks - KNN
The XOR Problem



Input layer Hidden layer output layer





Activation: sigmoid, η=0.5

Can be interconnecting into more complex structures

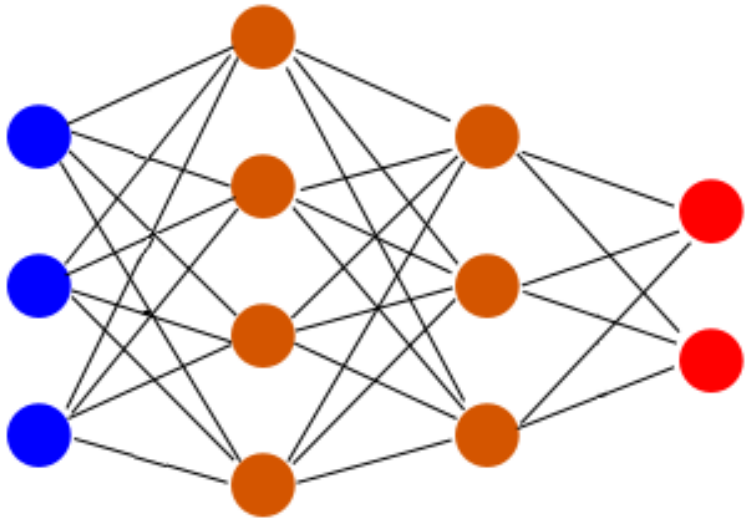
Many terms introduced: forward propagation, activation function, cost function
Backpropagation, Gradient Descent

Artificial Neural Networks - KNN
Problem of traditional computers

Problem	Computer	Mensch
Quickly multiply thousands of large numbers	easy	hard
Pick faces in a photo with a crowd	hard	easy

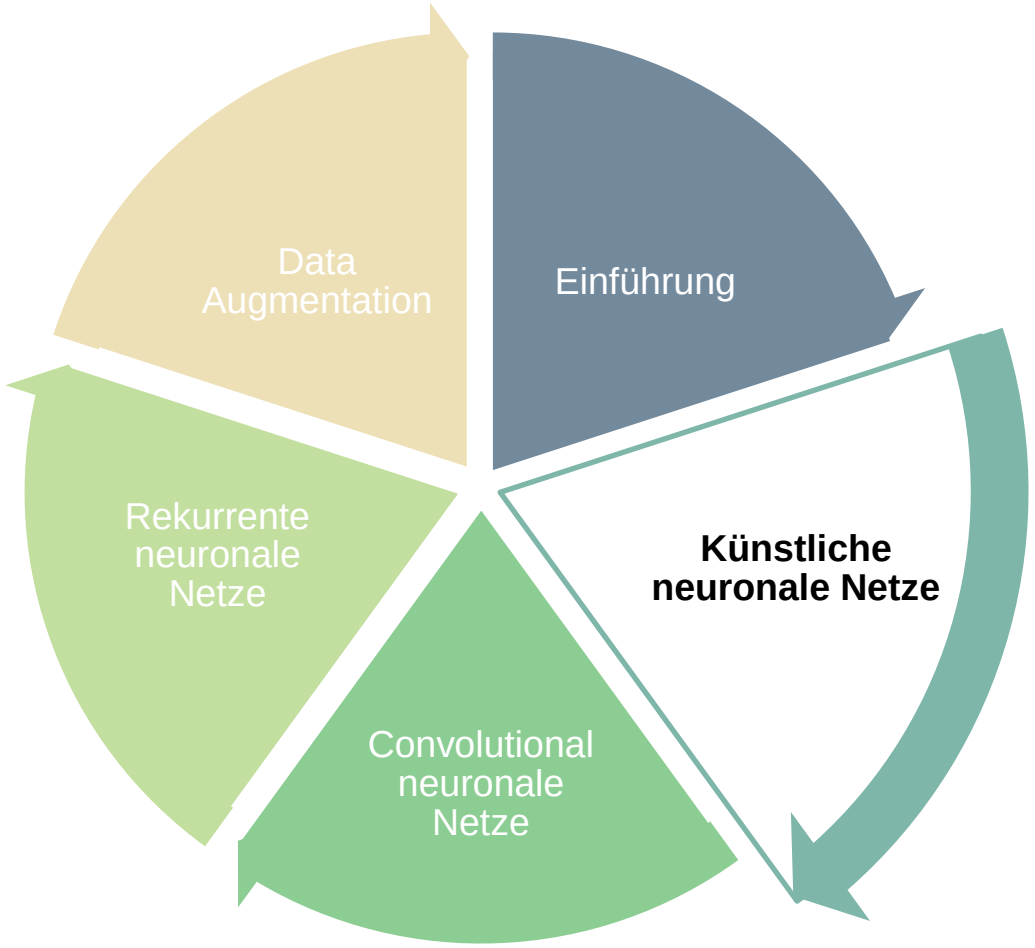
→ AI: find new algorithms that are trying to solve such problems in novel ways

Artificial neural network



- Takes an input and returns an output
- Inspired by the human brain
- Can detect patterns
- Must be trained
- KNNs: Heart of Deep Learning

Deep Learning view



Artificial neural networks: Part 1

introduction

Training of neural networks

Optimization algorithm

Activation features

Cost function

Performance metrics

Deep Learning - Ingredients

Hahn Schickard

Model architecture

- Number of shifts
- Activation function

Cost function

Defines what a good neural network is

Optimization algorithm

Minimizes cost function by varying NN weights

Training of the NN

Minimization of the cost function

AiF Mitglied
innBW INNOVATIONSALLIANZ BADEN-WÜRTTEMBERG

Programming neural networks in keras
The XOR sample

Hahn Schickard

```
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense
```

```
(x_train, y_train), (x_test, y_test)= load_data()
```

Importing data

```
model = Sequential()

model.add(Dense(units=2, activation='sigmoid', input_dim=2))
model.add(Dense(units=1, activation='sigmoid'))

model.compile(loss='binary_crossentropy', optimizer='sgd')
```

Create a model

```
model.fit(x_train, y_train, epochs=500)
```

Train model

```
classes = model.predict(x_test)
```

Testing the model

AiF Mitglied innBW INNOVATIONSALLIANZ BADEN-WÜRTTEMBERG

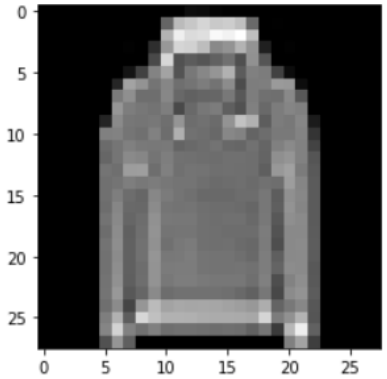
Artificial Neural Networks - KNN
Simple example

- Binary classification problem
- What influence does the number of layers have on the quality of the neural network?
- What influence does the activation function have on the quality of the neural network?
- What influence does the optimization algorithm have in order to obtain a good neural network as quickly as possible?

Artificial Neural Networks - KNN
Example image classifiers: Fashion MNIST

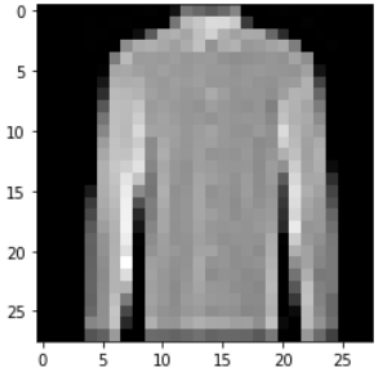
- Fashion MNIST: Zalando item
- 28x28 Grayscale Images
- 60,000 images
- 10 classes:
- T-shirt/top, pants, sweater, dress, coat, sandal, shirt, sneaker, bag, ankle boots

2 similar classes for binary classification



sweater

5507 pictures



shirt

5496 pictures

Programming neural networks in keras
Code for the Fashion MNIST example

Hahn

```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_1 (Flatten)          (None, 784)               0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 785
=================================================================
Total params: 785
Trainable params: 785
Non-trainable params: 0
_________________________________________________________________
```

```
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense

(x_train, y_train), (x_test, y_test)= load_data()
```

```
model = Sequential()

model.add(Flatten(input_shape=[28,28]))
model.add(Dense(units=1, activation='sigmoid'))

model.compile(loss='binary_crossentropy', optimizer='sgd')
```

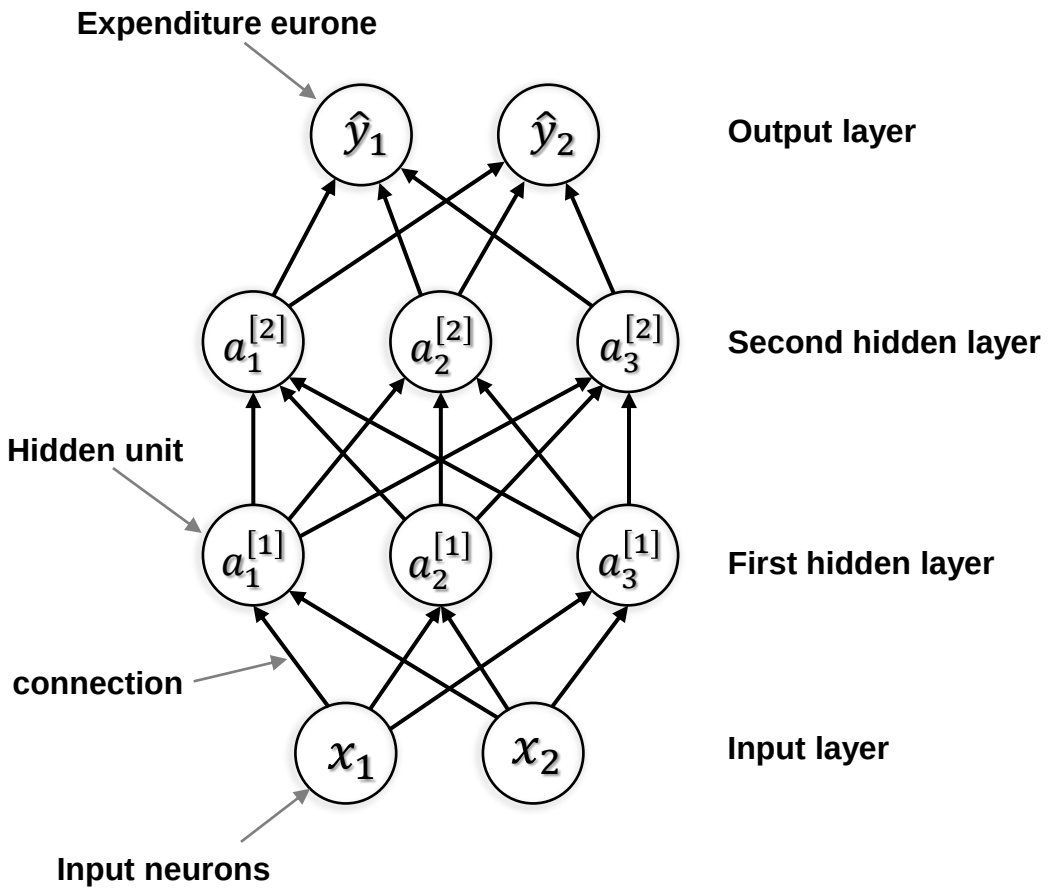
Create a model

```
model.fit(x_train, y_train, epochs=1000)
```

Train model

AiF Mitglied innBW INNOVATIONSALLIANZ BADEN-WÜRTTEMBERG

Artificial Neural Networks - KNN
Forward propagation of a multi-layerneural neural network



Forward propagation (with N hidden layers):

$$a_j^{[1]} = \sigma^{[1]}\left(\sum_i \omega_{ji}^{[1]} \cdot x_i\right)$$

$$a_j^{[\ell]} = \sigma^{[\ell]}\left(\sum_i \omega_{ji}^{[\ell]} \cdot a_i^{[\ell-1]}\right), \ell = 2, \dots, N$$

$$\hat{y}_j = \sigma^{[N+1]}\left(\sum_i \omega_{ji}^{[N+1]} \cdot a_j^{[N]}\right)$$

$a_j^{[\ell]}$: Activation function of the j-th neuron of the ℓ-th hidden layer

$\omega_{ji}^{[\ell]}$: weight of the i-th input of the j-th neuron of the ℓ-th hidden layer

$\hat{y}_j$: Prediction of the j-th expenditure eurone

Artificial Neural Networks - KNN
Training neural networks

$$\omega \longleftarrow \omega - \eta \nabla_{\omega} J(\omega)$$

Hahn Schickard

Train: Search for the ideal activation

Baking propagation

Recirculation of the error through the entire network

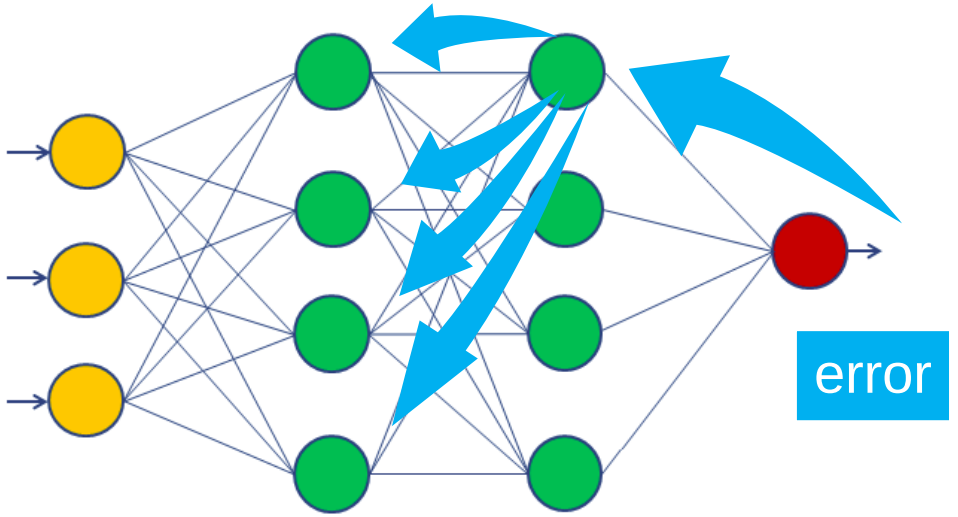
Proportional division of the error among all neurons

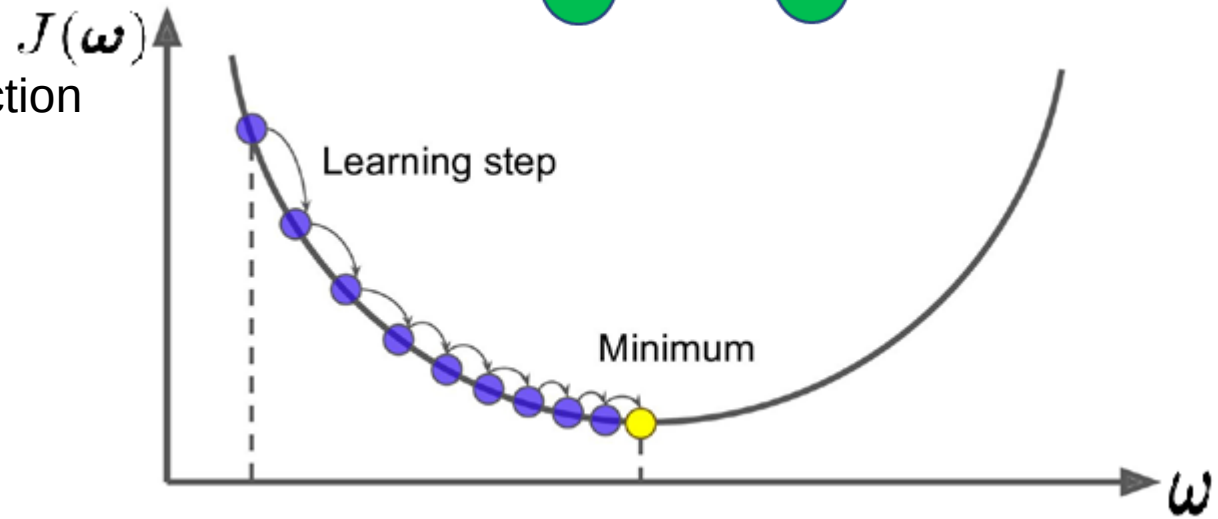
Gradient descent (optimization algorithm)

Updating parameters (weights, bias)

Contrary to the direction of the gradient of the cost function

Step η is called learning rate





$$\omega \leftarrow \eta \nabla_{\omega} J(\omega) \qquad \nabla_{\omega} J(\omega) = \begin{pmatrix} \frac{\partial J(\omega)}{\partial \omega_1} \\ \frac{\partial J(\omega)}{\partial \omega_2} \\ \frac{\partial J(\omega)}{\partial \omega_3} \end{pmatrix}$$

3-dimensional vector: points towards the steepest descent

Mitglied innBW INNOVATIONSALLIANZ BADEN-WÜRTTEMBERG

Example image classifier: Comparison of two activation functions

Hahn Schickard

Multi-layer mesh with an input layer (flattening), a hidden layer (Dense and 300 neurons) and an output layer (Dense, Sigmoid) :

hidden layer with linear activation
Introduction of the hyperparameter α

hidden layer with ReLU (Rectifier Linear Unit) activation

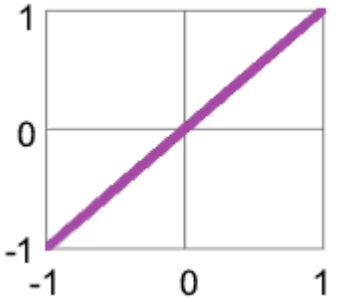
```
model_linear = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28,28]),
    keras.layers.Dense(300, activation='linear'),
    keras.layers.Dense(1, activation='sigmoid')
])
```

```
model_linear.compile(
    loss="binary_crossentropy",
    optimizer="sgd")
```

```
model_relu = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28,28]),
    keras.layers.Dense(300, activation='relu'),
    keras.layers.Dense(1, activation='sigmoid')
])
```

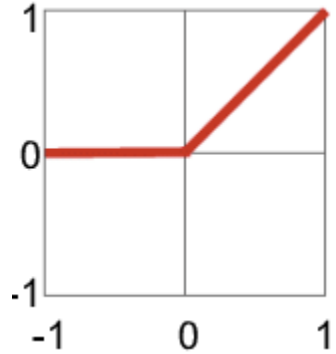
```
model_relu.compile(
    loss="binary_crossentropy",
    optimizer="sgd")
```

Linear



$$\sigma(z) = \alpha \cdot z$$

Rectified Linear Unit (ReLU)



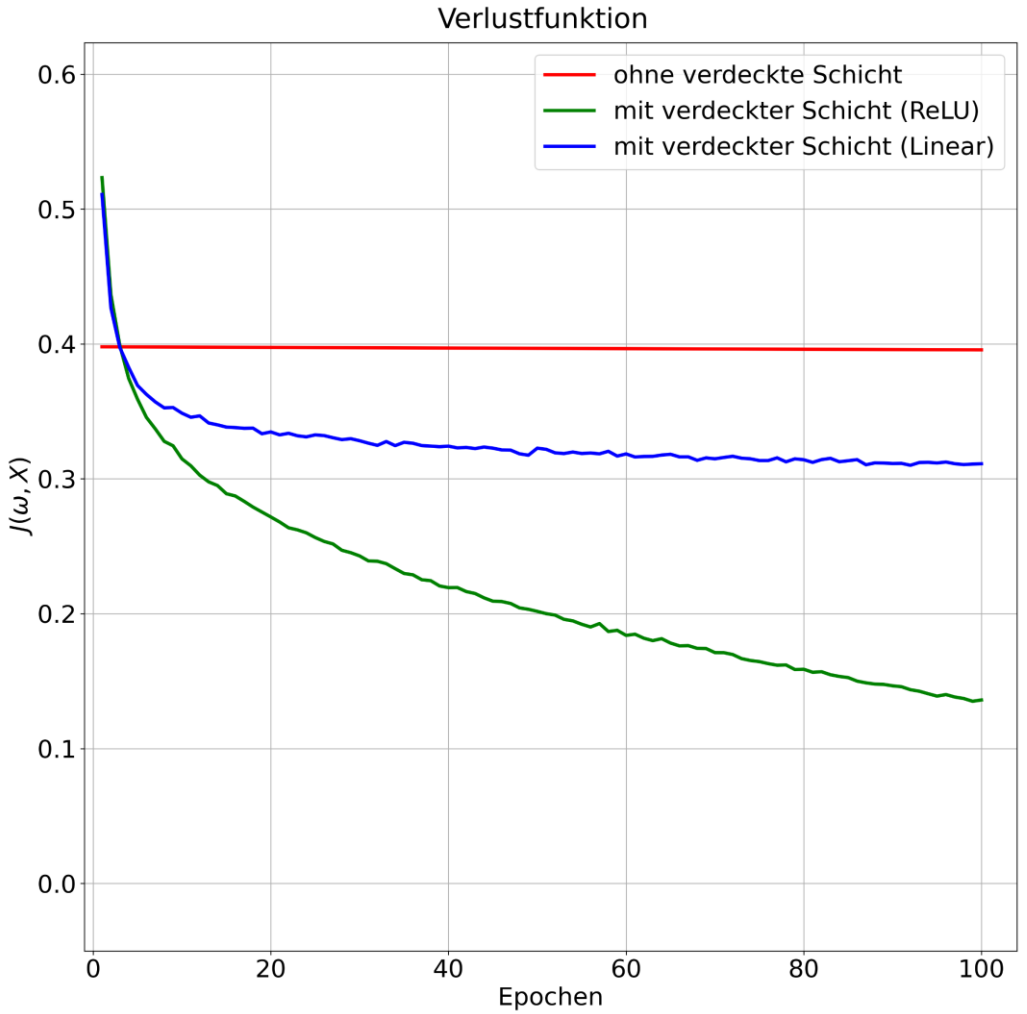
$$\sigma(z) = \max(0, z)$$

AIF Mitglied innBW INNOVATIONSALLIANZ BADEN-WÜRTTEMBERG

Artificial Neural Networks - KNN
Example image classifier: Comparison of two activation functions

```
model_relu.summary()
```

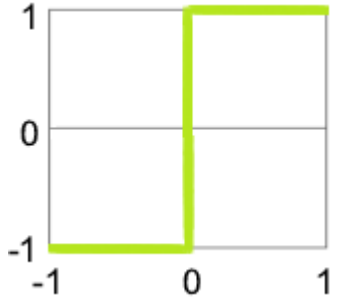
```
Model: "sequential_36"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_36 (Flatten)         (None, 784)               0
_________________________________________________________________
dense_68 (Dense)             (None, 300)               235500
_________________________________________________________________
dense_69 (Dense)             (None, 1)                 301
=================================================================
Total params: 235,801
Trainable params: 235,801
Non-trainable params: 0
_________________________________________________________________
```



Künstliche Neuronale Netze - KNN

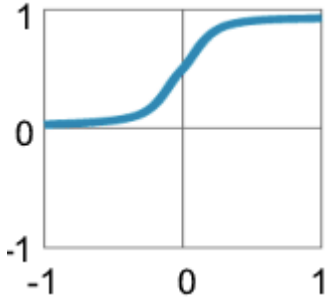
Aktivierungsfunktionen

Binary step function



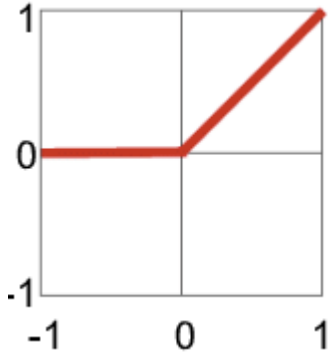
$$f(z) = \begin{cases} 0 & \text{falls } z < 0 \\ 1 & \text{sonst} \end{cases}$$

Sigmoid



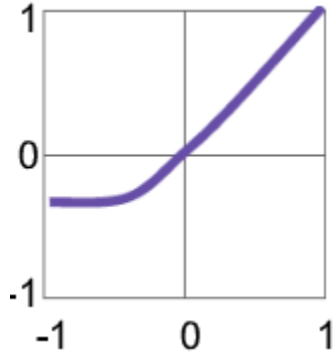
$$f(z) = \frac{1}{1+e^{-z}}$$

Rectified Linear Unit (ReLU)



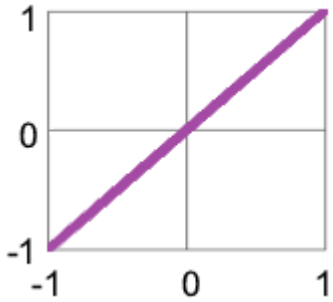
$$f(z) = \max(0, z)$$

Scaled Exp. LU (SELU)



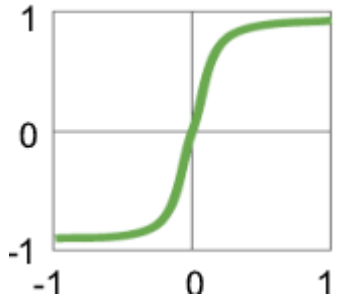
$$f(z) = \lambda \begin{cases} \alpha(e^z - 1) & \text{falls } z < 0 \\ z & \text{sonst} \end{cases}$$

Linear



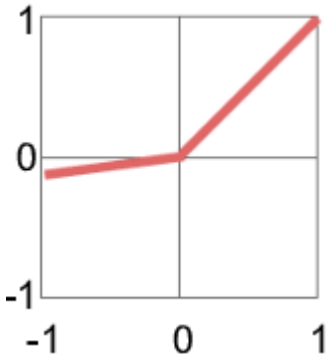
$$f(z) = \alpha z$$

Tangens hyperbolicus



$$f(z) = \tanh(z)$$

Leaky ReLU



$$f(z) = \max(\alpha z, z)$$

Softmax

$$f(z_k) = \text{softmax}(z_k) = \frac{e^{z_k}}{\sum_c e^{z_c}}$$

Mitglied innBW INNOVATIONSALLIANZ BADEN-WÜRTTEMBERG

Deep Learning - Ingredients

Hahn Schickard

Model architecture

- Number of shifts
- Activation function

Cost function

Defines what a good neural network is

Optimization algorithm

Minimizes cost function by varying NN weights

Training of the NN

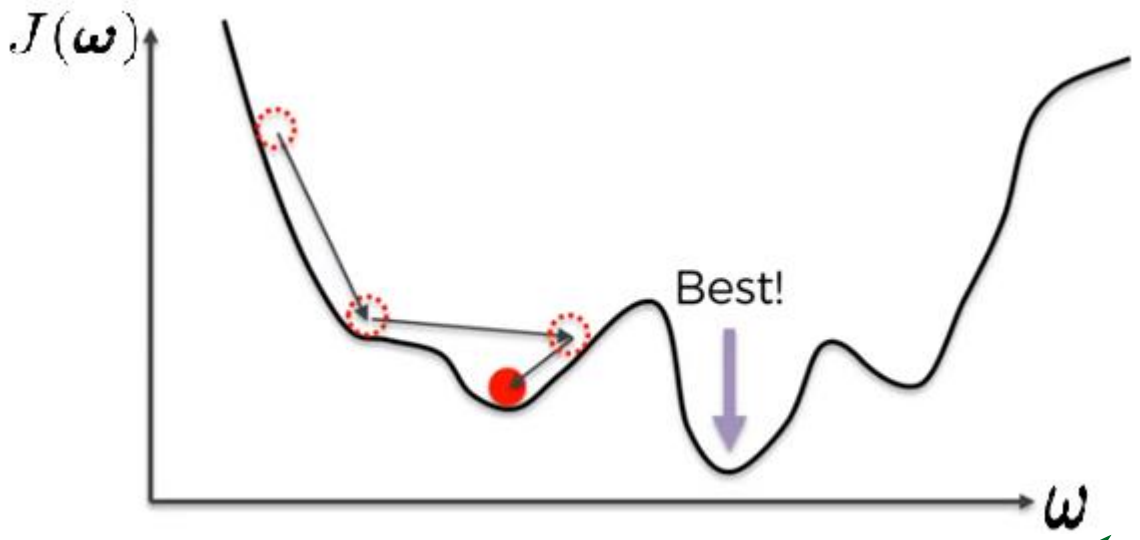
Minimization of the cost function

AiF Mitglied
innBW INNOVATIONSALLIANZ BADEN-WÜRTTEMBERG

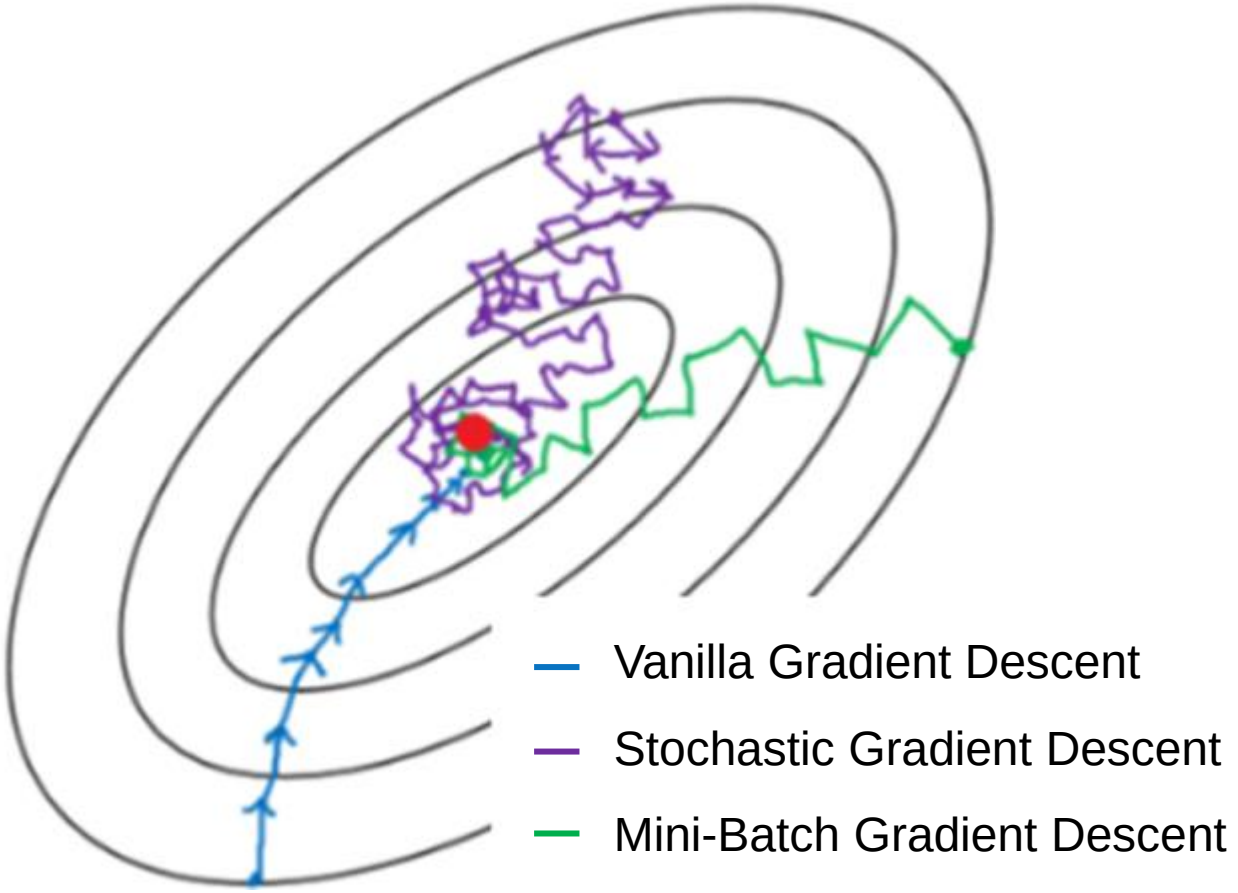
Artificial Neural Networks - KNN
Gradient Descent

- **Vanilla Gradient Descent (Batch Gradient Descent)**
- Iterated over all training instances
- Suitable only for convex functions
- Unsuitable for large training records
- **Stochastic Gradient Descent (SGD)**
- Takes a random training instance
- Escape from local minima possible
- Significantly faster training
- **Mini-Batch Gradient Descent**
- Uses random mini-batches
- Leverages the benefits of both previous methods

- $\boldsymbol{\omega} \leftarrow \boldsymbol{\omega} - \eta \cdot \frac{\partial J(\boldsymbol{\omega}, \boldsymbol{x})}{\partial \boldsymbol{\omega}} = \boldsymbol{\omega} - \eta \cdot \nabla_{\boldsymbol{\omega}} J(\boldsymbol{\omega}, \boldsymbol{x})$
- $\boldsymbol{\omega} \leftarrow \boldsymbol{\omega} - \eta \cdot \nabla_{\boldsymbol{\omega}} J\left(\boldsymbol{\omega}, \boldsymbol{x}^{(i)}; y^{(i)}\right)$
- $\boldsymbol{\omega} \leftarrow \boldsymbol{\omega} - \eta \cdot \nabla_{\boldsymbol{\omega}} J\left(\boldsymbol{\omega}, \boldsymbol{x}^{(i:i+n)}; y^{(i:i+n)}\right)$



Artificial Neural Networks - KNN
Gradient Descent



Artificial Neural Networks - KNN
Example image classifiers: Comparison of three optimization algorithms

Hahn Schickard

Multi-layer mesh with an input layer (flattening), a hidden layer (Dense) and a starting layer (Dense):

Vanilla Gradient Descent (1000, 35.14 seconds)

Stochastic Gradient Descent (200, 1713.81 seconds)

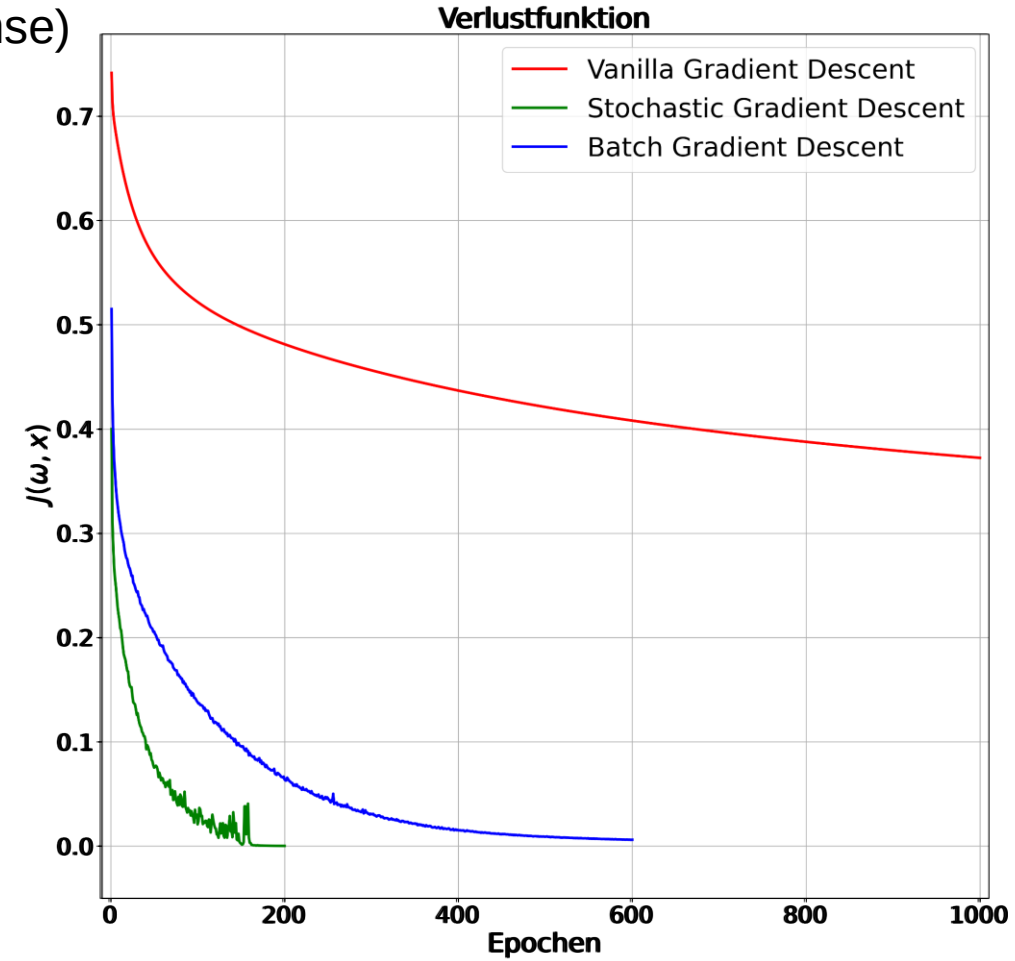
Mini-batch Gradient Descent (600, 216.22 seconds)

```
history_VGD = model_VGD.fit(X_train_two,
                            y_train_two,
                            batch_size=X_train_two.shape[0],
                            epochs=1000)
```

```
history_SGD = model_SGD.fit(X_train_two,
                            y_train_two,
                            batch_size=1,
                            epochs=200)
```

```
history_BGD = model_BGD.fit(X_train_two,
                            y_train_two,
                            epochs=600)
```

Standward value: 32



Mitglied
innBW INNOVATIONSALLIANZ BADEN-WÜRTTEMBERG

Artificial Neural Networks - KNN
Simple example - Results

- Binary classification problem
- What influence does the number of layers have on the quality of the neural network?
- Higher number of layers = More weights = More degrees of freedom
- What influence does the activation function have on the quality of the neural network?
- The activation function is the only nonlinear in the neural network
- Activation function also has a significant impact on the convergence of gradient descent (more on that later)
- What influence does the optimization algorithm have in order to obtain a good neural network as quickly as possible?
- Mini-Batch Gradient Descent speeds up training
- Later we will learn more optimization algorithms

Artificial Neural Networks - KNN
Baking propagation (mathematical)

Minimize the cost function:
(e.B. for a 2-dimensional parameter space)

$$\nabla_{\boldsymbol{\omega}} J(\boldsymbol{\omega}) = \begin{pmatrix} \frac{\partial}{\partial \omega_1} J(\omega_1, \omega_2) \\ \frac{\partial}{\partial \omega_2} J(\omega_1, \omega_2) \end{pmatrix}$$

For the weight $\omega_{ij}^{[\ell]}$ let's write:

$$\frac{\partial \mathrm{J}}{\partial \omega_{\mathrm{ij}}^{[\ell]}}$$

Cost function of the output layer

$$\mathrm{J}(\boldsymbol{\omega}; \mathbf{X}) = \frac{1}{N} \sum_j \left(y_j - \hat{y}_j\right)^2$$

Künstliche Neuronale Netze - KNN

Backpropagation (mathematisch)

Minimize the cost function: $\frac{\partial J}{\partial \omega_{ij}^{[\ell]}}$

Define the jth neuron error of the ℓ-th Schicht $\delta_j^{[\ell]}$:

$$\delta_j^{[\ell]} = \frac{\partial J}{\partial z_j^{[\ell]}}$$

Cost function of the output layer

$$J = \frac{1}{N} \sum_j \left(y_j - a_j^{[L]} \right)^2$$

General formula of the activation function

$$a_j^{[\ell]} = \sigma^{[\ell]} \left(\sum_k \omega_{jk}^{[\ell]} \, a_k^{[\ell-1]} \right)$$

with $z_j^{[\ell]} = \sum_k \omega_{jk}^{[\ell]} \, a_k^{[\ell-1]}$

Künstliche Neuronale Netze - KNN

Backpropagation (mathematisch)

Minimize the cost function: $\frac{\partial J}{\partial \omega_{ij}^{[\ell]}}$

Define the jth neuron error of the ℓ-th layer $\delta_j^{[\ell]}$:

$$\delta_j^{[\ell]} = \frac{\partial J}{\partial z_j^{[\ell]}}$$

Neuron error of a neuron j of the starting layer $\ell = L$:

$$\delta_j^{[L]} = \frac{\partial J}{\partial z_j^{[L]}} = \sum_i \underbrace{\frac{\partial J}{\partial a_i^{[L]}} \frac{\partial \mathrm{a}_i^{[L]}}{\partial z_j^{[L]}}}_{=0 \text{ für } i \neq j} = \frac{\partial J}{\partial a_j^{[L]}} \frac{\partial \mathrm{a}_j^{[L]}}{\partial z_j^{[L]}} = \frac{\partial J}{\partial a_j^{[L]}} \sigma^{[L]\prime}\left(z_j^{[L]}\right)$$

Activation functions must be differentiable!

Cost function of the output layer

$$J = \frac{1}{N} \sum_j \left(y_j - a_j^{[L]}\right)^2$$

General formula of the activation function

$$a_j^{[\ell]} = \sigma^{[\ell]}\left(\sum_k \omega_{jk}^{[\ell]}\, a_k^{[\ell-1]}\right)$$

with $z_j^{[\ell]} = \sum_k \omega_{jk}^{[\ell]}\, a_k^{[\ell-1]}$

Artificial Neural Networks - KNN
Baking propagation (mathematical)

Hahn Schickard

Minimize the cost function: $\frac{\partial J}{\partial \omega_{ij}^{[\ell]}}$

Define the j-th neuron error of the ℓ-th layer $\delta_j^{[\ell]}$:

$$\delta_j^{[\ell]} = \frac{\partial J}{\partial z_j^{[\ell]}}$$

Neuron error of a neuron j of the starting layer $\ell = L$:

$$\delta_j^{[L]} = \frac{\partial J}{\partial a_j^{[L]}} \sigma^{[L]\prime}(z_j^{[L]})$$

Neuron error of a neuron j of a hidden layer $\ell \neq L$

$$\delta_j^{[\ell]} = \frac{\partial J}{\partial z_j^{[\ell]}} = \sum_k \underbrace{\frac{\partial J}{\partial z_k^{[\ell+1]}}}_{=\delta_k^{[\ell+1]}} \frac{\partial z_k^{[\ell+1]}}{\partial z_j^{[\ell]}} = \sum_k \frac{\partial z_k^{[\ell+1]}}{\partial z_j^{[\ell]}} \delta_k^{[\ell+1]} = \sum_k \omega_{kj}^{[\ell+1]} \delta_k^{[\ell+1]} \sigma^{[\ell]\prime}\left(z_j^{[\ell]}\right)$$

Depends on the weights and neuron errors of the next layer!

Cost function of the output layer

$$J = \frac{1}{N} \sum_j \left(y_j - a_j^{[L]}\right)^2$$

General formula of the activation function

$$a_j^{[\ell]} = \sigma^{[\ell]}\left(\sum_k \omega_{jk}^{[\ell]}\, a_k^{[\ell-1]}\right)$$

with $z_j^{[\ell]} = \sum_k \omega_{jk}^{[\ell]}\, a_k^{[\ell-1]}$

$$\frac{\partial z_k^{[\ell+1]}}{\partial z_j^{[\ell]}} = \omega_{kj}^{[\ell+1]} \sigma^{[\ell]\prime}\left(z_j^{[\ell]}\right)$$

AIF Mitglied innBW INNOVATIONSALLIANZ BADEN-WÜRTTEMBERG

Artificial Neural Networks - KNN
Example image classifiers: Fashion MNIST

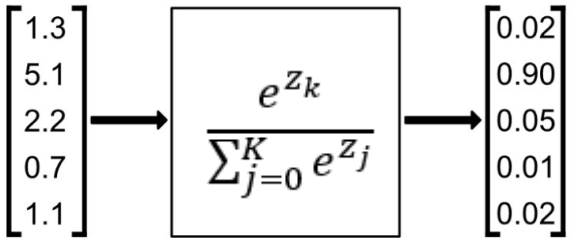
- Fashion MNIST: Zalando item
- 28x28 Grayscale Images
- 60,000 images
- 10 classes:
- T-shirt/top, pants, sweater, dress, coat, sandal, shirt, sneaker, bag, ankle boots
- Changes required:
- Activation function of the last layer
- Cost function

Artificial Neural Networks - KNN
Training neural networks - cost function $J(\omega)$

Cost function $\mathrm{J}(\boldsymbol{\omega}; \mathbf{X})$**:** measures the deviation between the outputs $\hat{\boldsymbol{y}}^{(i)} = h\left(\boldsymbol{x}^{(i)}\right)$ and the corresponding setpoints $\boldsymbol{y}^{(i)}$

- average deviation (*mean squared error*): $$\mathrm{J}(\boldsymbol{\omega}; \mathbf{X}) = \frac{1}{N}\sum_{i=0}^{N}\left(\hat{\boldsymbol{y}}^{(i)} - \boldsymbol{y}^{(i)}\right)^2$$
- Cross entropy / categorical cross entropy (*cross entropy / categorical cross entropy*):
  - Transforms a K-dimensional Vektor $\boldsymbol{z}$ into the value range (0,1) with a sum of 1

$$\mathrm{J}(\boldsymbol{\omega}; \mathbf{X}) = -\frac{1}{N}\sum_{i=0}^{N}\sum_{k=0}^{K} y_k^{(i)} log\left(\hat{p}_k^{(i)}\right) with\left(\hat{p}_k = \sigma(\boldsymbol{z})_k = \frac{e^{z_k}}{\sum_{j=0}^{K} e^{z_j}}\right)$$



- Binary cross entropy (2 classes: K=1) (*binary cross entropy*):

$$\mathrm{J}(\boldsymbol{\omega}; \mathbf{X}) = -\frac{1}{N}\sum_{i=0}^{N}\left(y^{(i)} log\left(\hat{p}^{(i)}\right) + \left(1 - y^{(i)}\right) log\left(1 - \hat{p}^{(i)}\right)\right) with\left(\hat{p} = \sigma(z) = \frac{1}{1 + e^{-z}}\right)$$

Programming neural networks in keras
Code for the Fashion MNIST example

```
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense
```

```
(x_train, y_train), (x_test, y_test)= load_data()
```

Importing data

```
model = Sequential()

model.add(Flatten(input_shape=[28,28]))
model.add(Dense(units=300, activation='relu'))
model.add(Dense(units=10, activation='softmax'))

model.compile(loss='sparse_categorical_crossentropy', optimizer='sgd')
```

Create a model

```
model.fit(x_train, y_train, epochs=1000)
```

Train model

sparse_categorical_crossentropy and categorical_crossentropy

Example: 5 samples with 3 classes

Classes are mutually exclusive

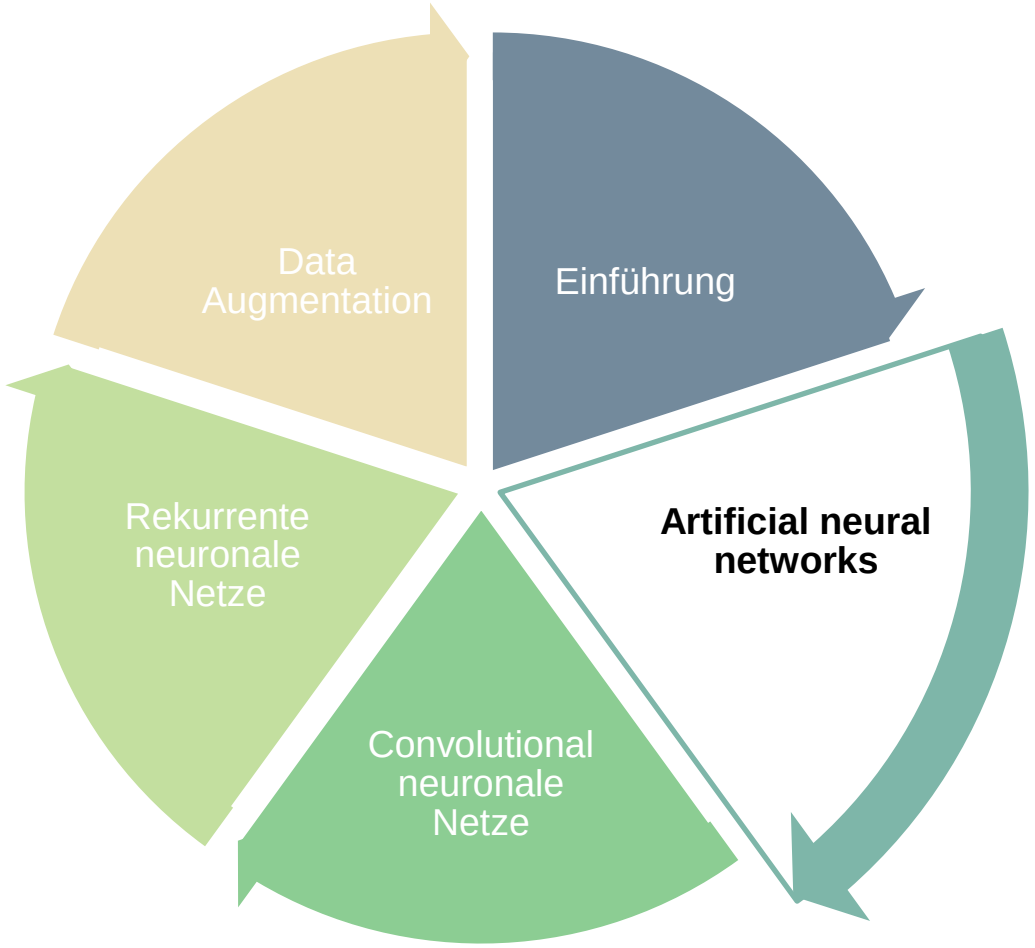
sparse_categorical_crossentropy:

$$y = \begin{pmatrix} 1 \\ 0 \\ 2 \\ 1 \\ 0 \end{pmatrix}$$

categorical_crossentropy:

$$y = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

Deep Learning view



Artificial neural networks: Part 1

introduction

Training of neural networks

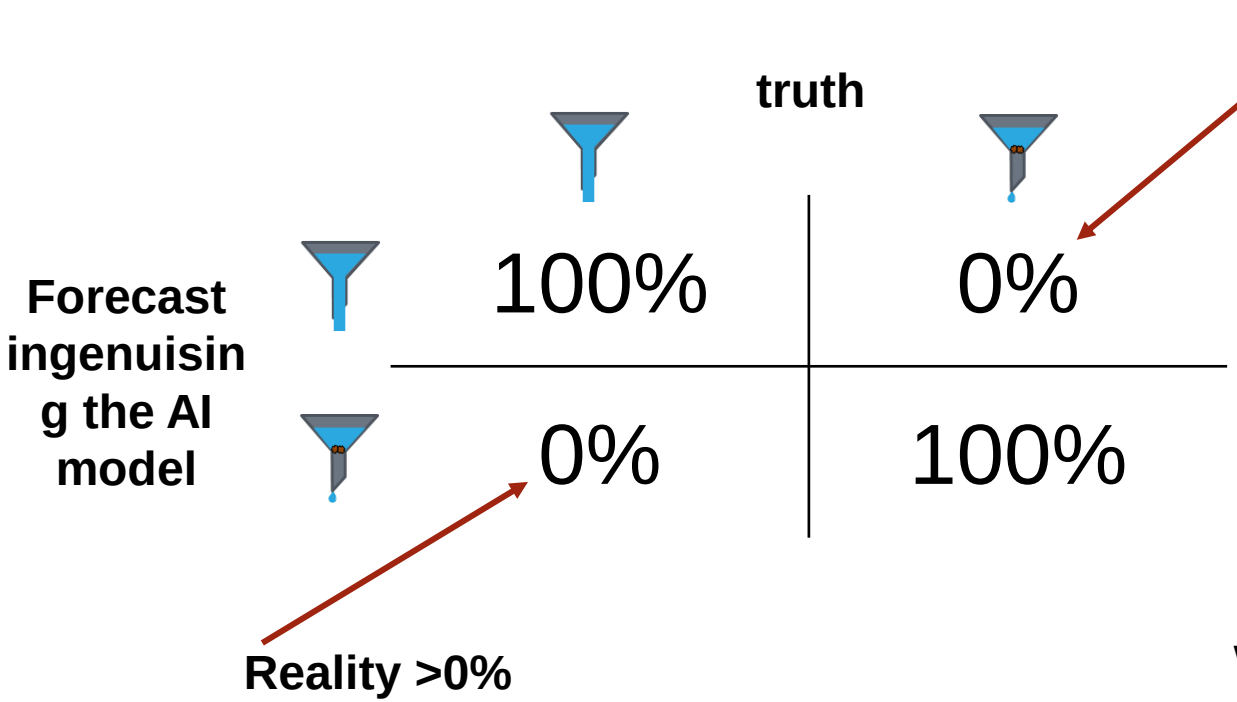
Optimization algorithm

Activation features

Cost function

Performance metrics

Artificial Neural Networks - KNN
Performance Metrics - How good is my neural network?



Reality >0%

If you want to keep the number small, the AI model has a high precision

Reality >0%

If you want to keep the number small, the AI model has a high sensitivity

What is the consequence if a required filter change is not predicted or a filter change is predicted that is not needed?

Artificial Neural Networks - KNN
Performance Metrics - How good is my neural network?

Hahn Schickard

Confusion matrix (determination of classification quality)

Classifiers \ reality	yes	No
yes	Tp correct positive	Fp false positive
No	*false negative Fn*	*correct negative Tn*

100	100
0	0

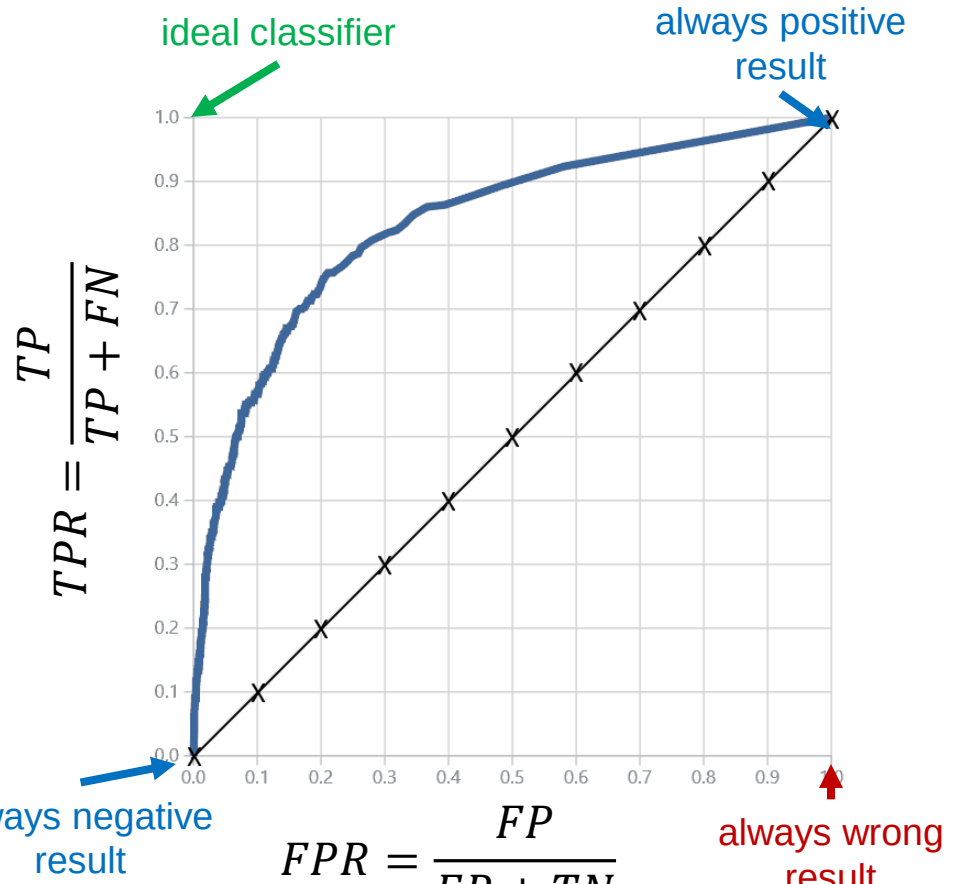
$y = 1/0$: Patient is sick / healthy	
$f(x) = 1/0$: classified as sick / healthy	
really positive (TP)	$y = 1, f(x) = 1$
really negative (TN)	$y \neq 1, f(x) \neq 1$
false positive (FP)	$y \neq 1, f(x) = 1$
false negative (FN)	$y = 1, f(x) \neq 1$

$$\text{precision} = \frac{TP}{TP+FP} = 50\%$$

$$\text{sensitivity} = \frac{TP}{TP+FN} = 100\%$$

$$accuracy = \frac{TP+TN}{TP+TN+FP+FN} = 50\%$$

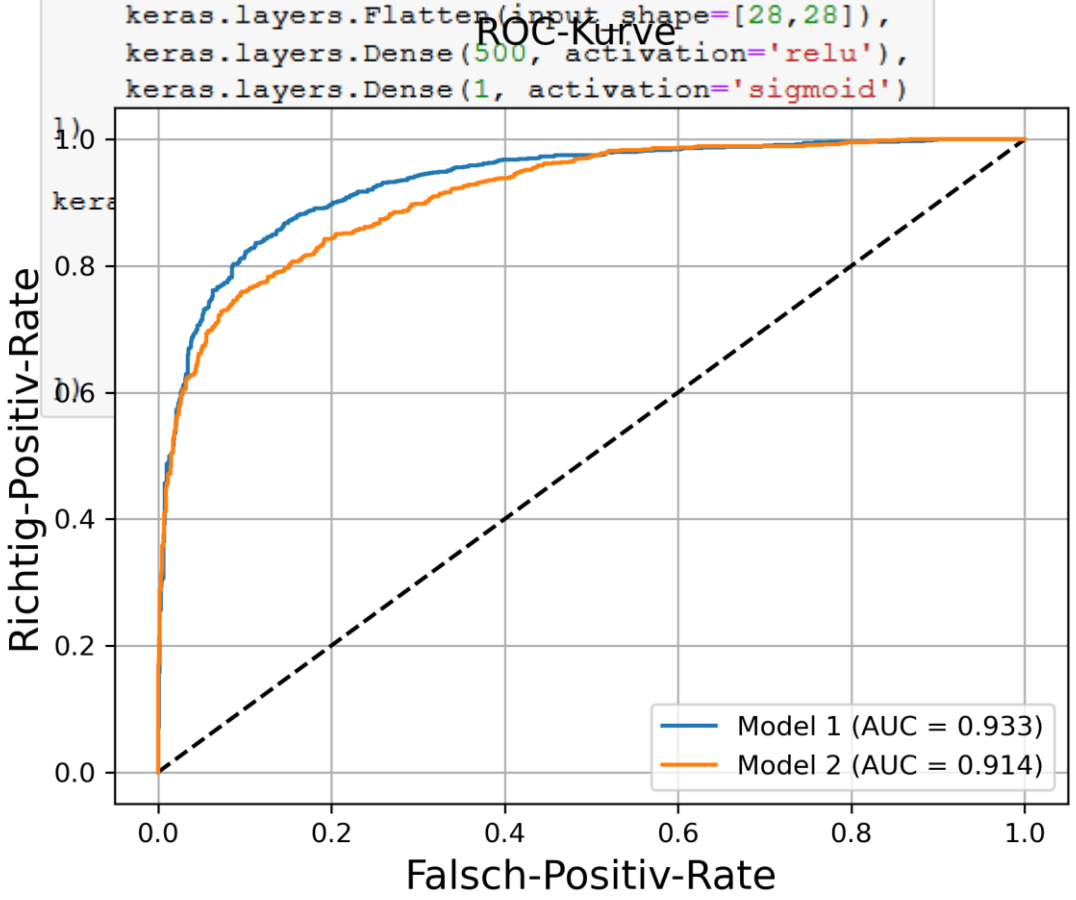
ROC Chart

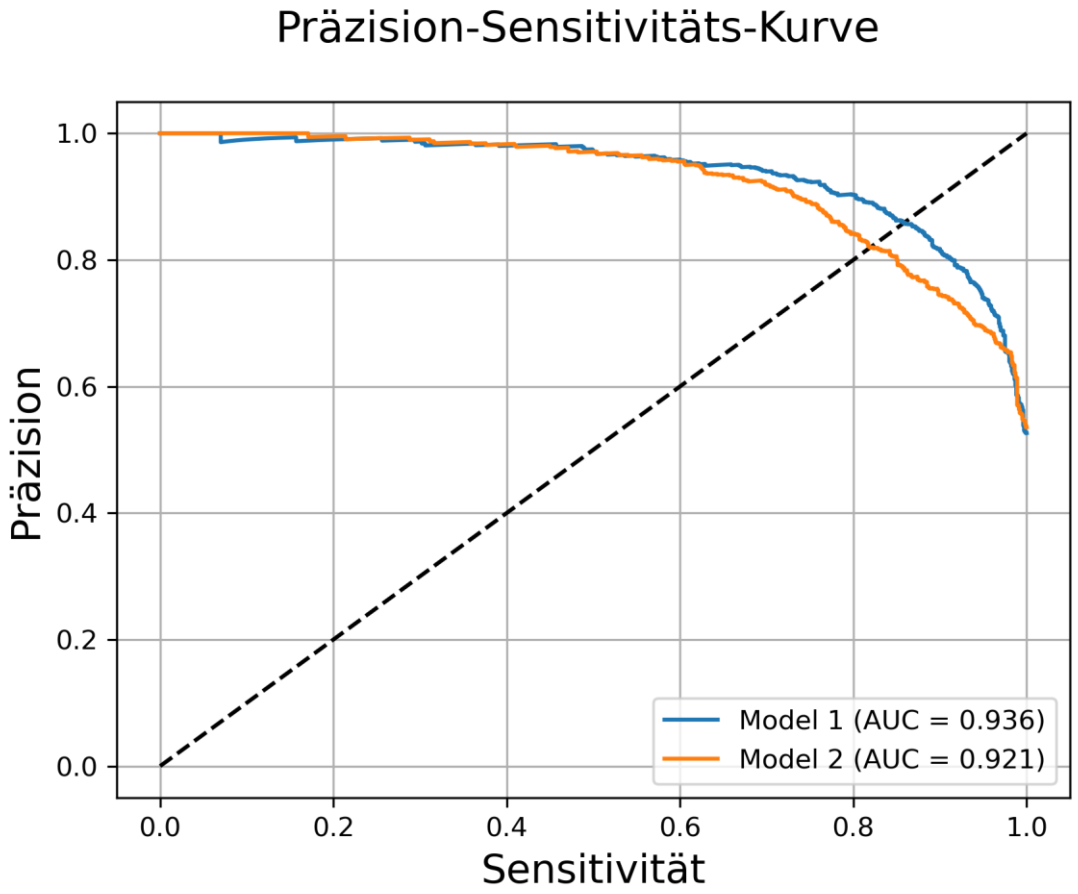


AiF Mitglied
innBW INNOVATIONSALLIANZ BADEN-WÜRTTEMBERG

Artificial Neural Networks - KNN
Example Binary Image Classifier: Precision Vs. Recall & ROC Curve

```
keras_metrik1 = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28,28]),
    keras.layers.Dense(500, activation='relu'),
    keras.layers.Dense(1, activation='sigmoid')
])
```

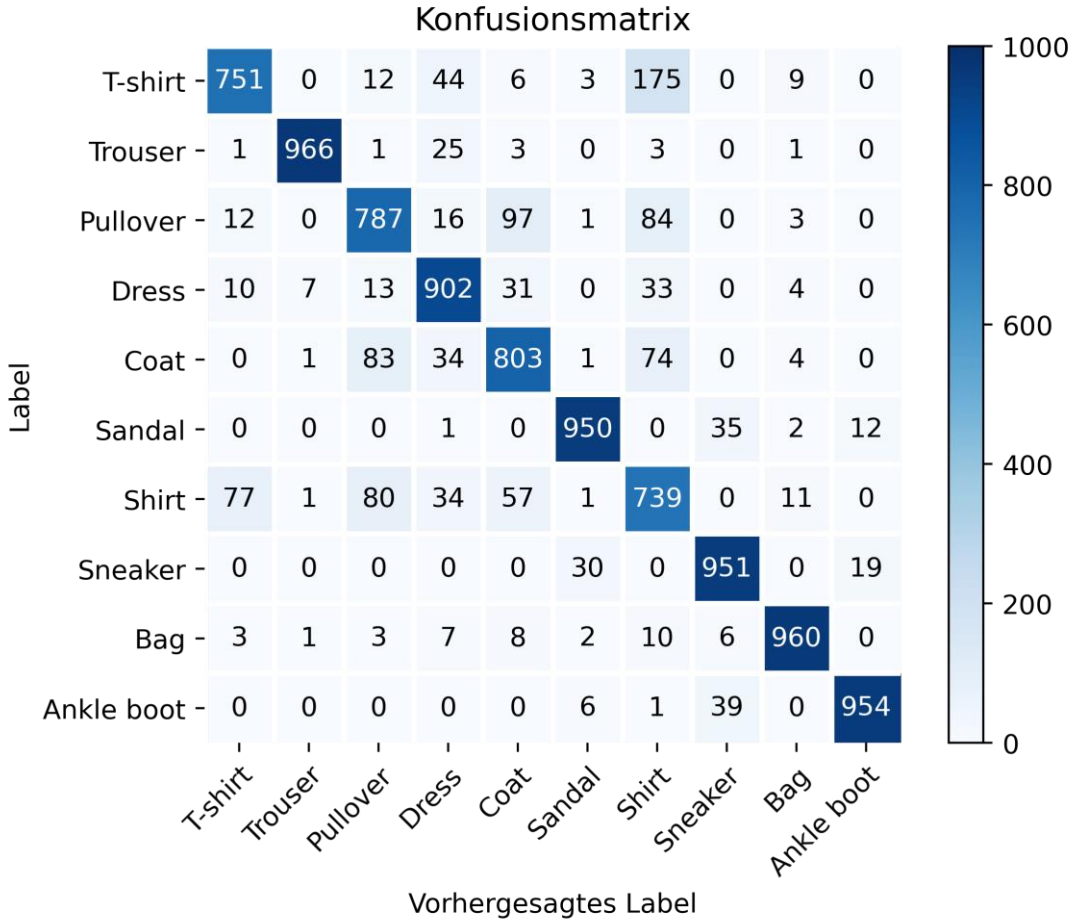




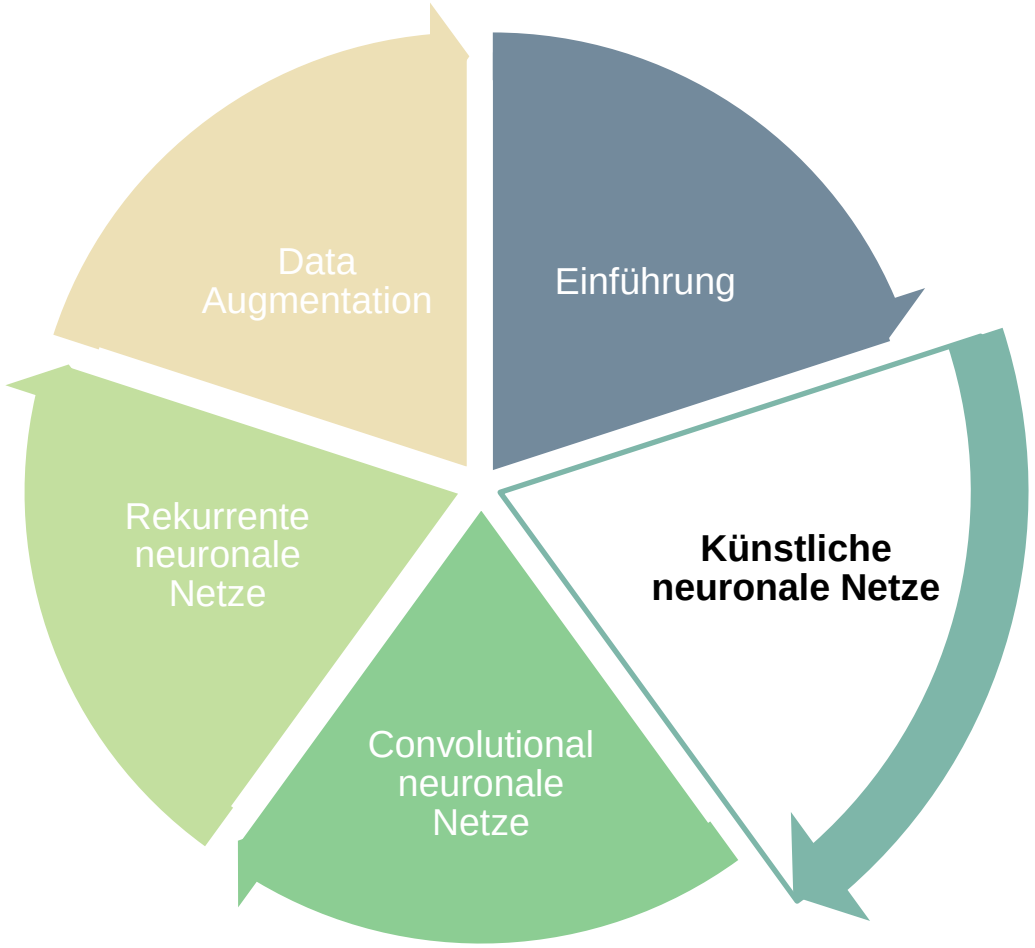
Artificial Neural Networks - KNN
Example image classifier: confusion matrix

```
# Predict the label and search the maximum
pred = model_all_classes(X_test)
pred = np.argmax(pred, axis=1)
label = np.argmax(to_categorical(y_test),axis=1)

# Calculate the confusion matrix
conf_matrix = tf.math.confusion_matrix(label, pred)
conf_matrix = conf_matrix.numpy()
conf_matrix
```

```
array([[751,   0,  12,  44,   6,   3, 175,   0,   9,   0],
       [  1, 966,   1,  25,   3,   0,   3,   0,   1,   0],
       [ 12,   0, 787,  16,  97,   1,  84,   0,   3,   0],
       [ 10,   7,  13, 902,  31,   0,  33,   0,   4,   0],
       [  0,   1,  83,  34, 803,   1,  74,   0,   4,   0],
       [  0,   0,   0,   1,   0, 950,   0,  35,   2,  12],
       [ 77,   1,  80,  34,  57,   1, 739,   0,  11,   0],
       [  0,   0,   0,   0,   0,  30,   0, 951,   0,  19],
       [  3,   1,   3,   7,   8,   2,  10,   6, 960,   0],
       [  0,   0,   0,   0,   0,   6,   1,  39,   0, 954]])
```



Deep Learning view



Künstliche neuronale Netze: Part 2

- Training, test and validation data
- Regularization
  - **Early Stopping**
  - ℓ_1 & ℓ_2**-Regularisierung**
  - **Dropout**
  - Weight initialization
  - Batch Normalisierung
- Optimization algorithms

Artificial Neural Networks - KNN
Training, validation and test data

Hahn Schickard

Split of the data set in training, validation and test data:

z.B. with a ratio of 75:10:15 (There must be disjuncent sets)

Train the neural network with the training data

Validate the neural network with the validation data

Final check on the test data (test may only be carried out once)

```
X_valid, X_train = X_train_full[:5000], X_train_full[5000:]
y_valid, y_train = y_train_full[:5000], y_train_full[5000:]
```

What does validation mean?

Neural networks have hyperparameters (number of layers and neurons, initialization of weights, data preprocessing, ...) which need to be adjusted

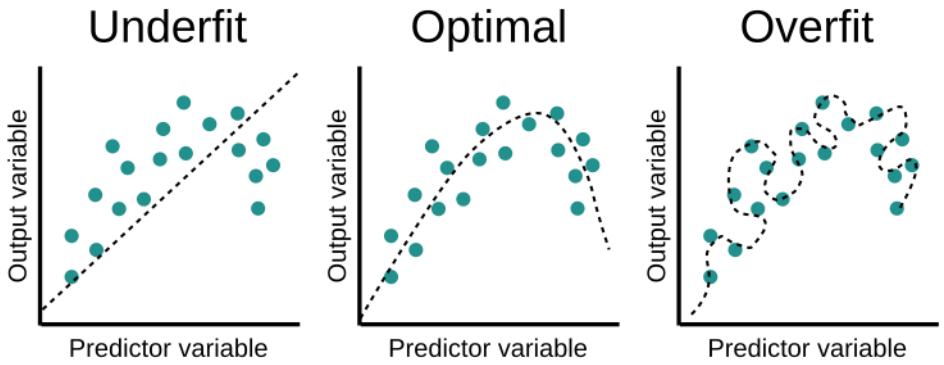
When should I end the iterative training? (see next slide)

AF Mitglied innBW INNOVATIONSALLIANZ BADEN-WÜRTTEMBERG

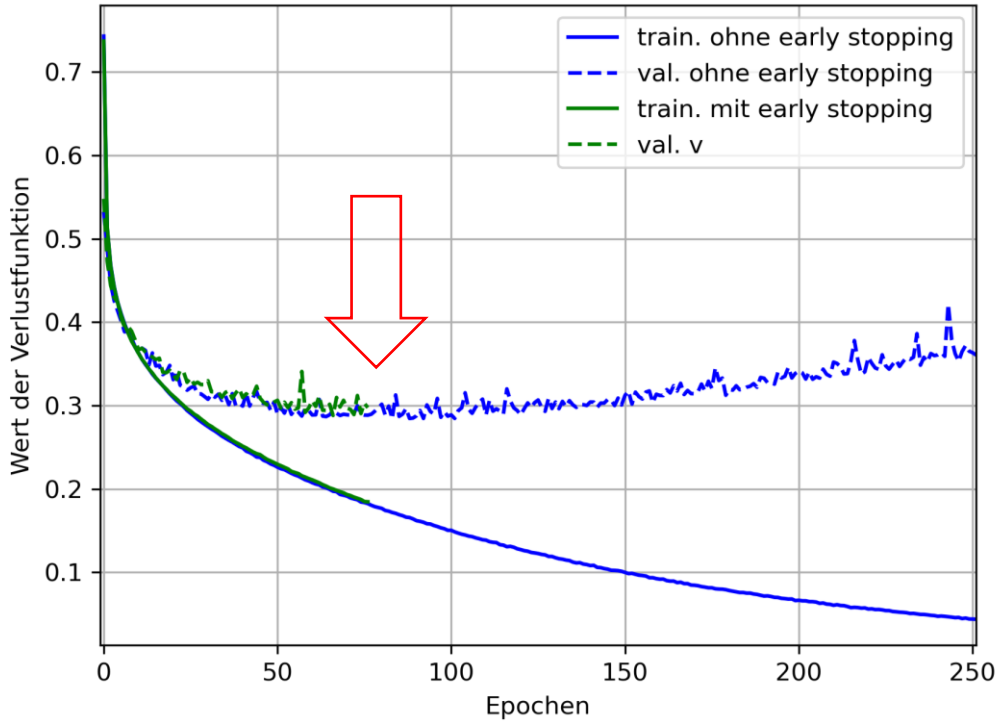
Artificial Neural Networks - KNN
Regularization

problem:

Overfitting on training data ⟶ **Regularization**



- Early Stopping
  Determines the best model based on the validation error

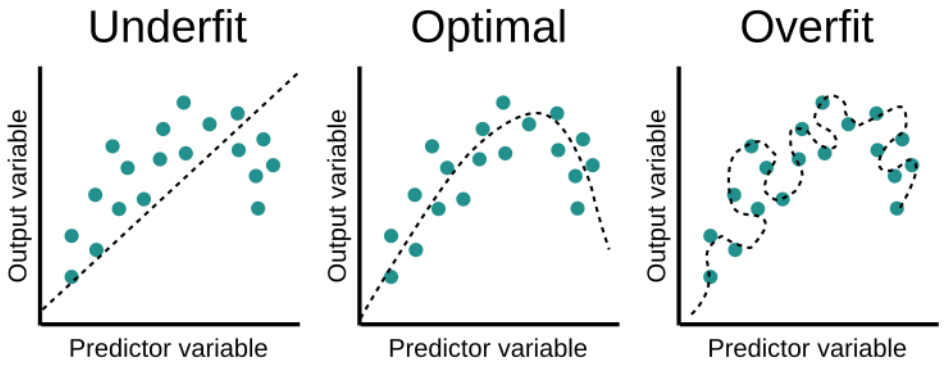



```
history_early_stopping = model_early_stopping.fit(
    X_train,
    y_train,
    epochs=400,
    callbacks=[tf.keras.callbacks.EarlyStopping(
        monitor='val_loss',
        min_delta=1e-3,
        patience=10,
        restore_best_weights=True)],
    validation_data=(X_valid, y_valid))
```

Artificial Neural Networks - KNN Regularization

problem:

Overfitting on training data ⟶ **Regularization**



- $\ell_1 \& \ell_2$ - Regularization
  - Extend the cost function with a regularization term $\Omega(\mathbf{W})$

$$\ell_2: \quad \Omega(\mathbf{W}) = \frac{\alpha}{2}||\mathbf{W}||_2^2 = \frac{\alpha}{2}\sum_i\sum_j w_{ij}^2 \qquad \ell_1: \quad \Omega(\mathbf{W}) = \alpha||\mathbf{W}||_1 = \alpha\sum_i\sum_j |w_{ij}|$$

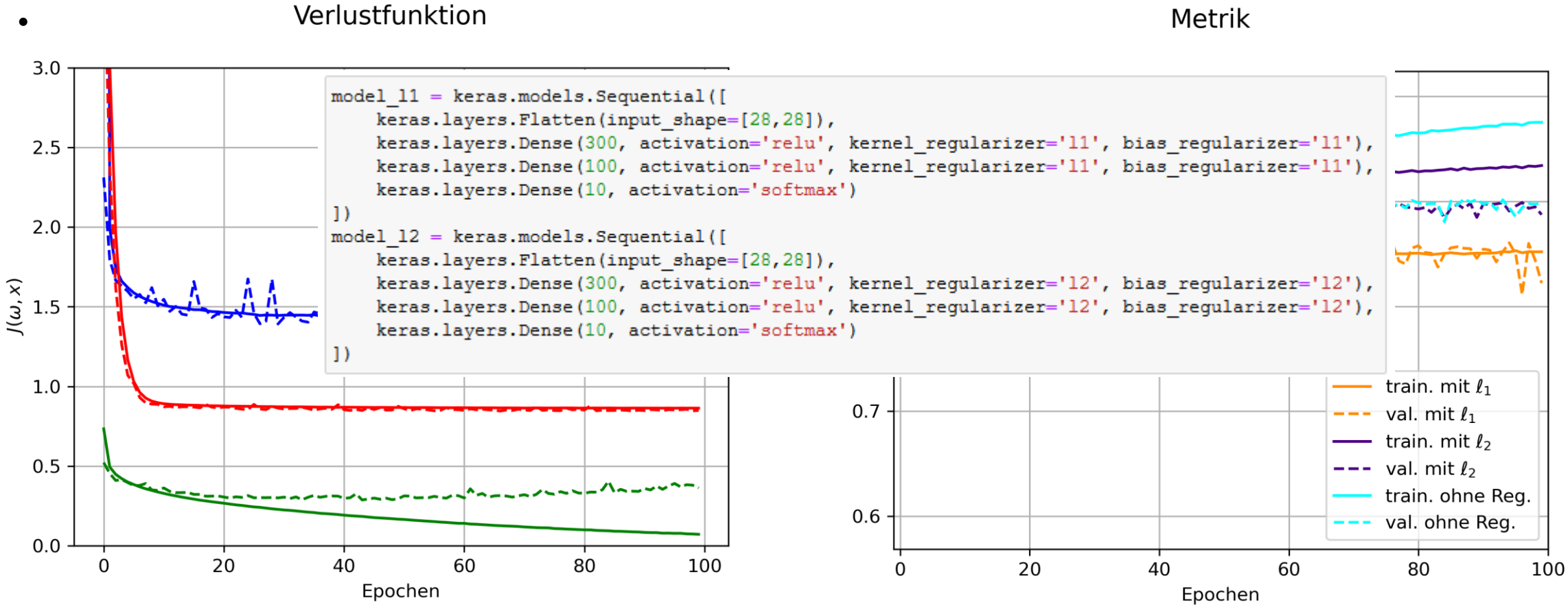
example: binary cross entropy with ℓ_2-Regularization:

$$J_{\text{reg}}(\boldsymbol{\omega}, \mathbf{X}) = J(\boldsymbol{\omega}, \mathbf{X}) + \Omega(\boldsymbol{\omega}) = -\frac{1}{N}\sum_{i=0}^{N}\left(y^{(i)}\log\left(\hat{p}^{(i)}\right) + \left(1-y^{(i)}\right)\log\left(1-\hat{p}^{(i)}\right)\right) + \frac{\alpha}{2}\frac{1}{N}\sum_l\sum_i\sum_j\left(\omega_{j,i}^{[l]}\right)^2$$

Artificial Neural Networks - KNN
Example image classifier: regularization

- Comparison of two models, one with regularization of one without
-

```
model_l1 = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28,28]),
    keras.layers.Dense(300, activation='relu', kernel_regularizer='l1', bias_regularizer='l1'),
    keras.layers.Dense(100, activation='relu', kernel_regularizer='l1', bias_regularizer='l1'),
    keras.layers.Dense(10, activation='softmax')
])
model_l2 = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28,28]),
    keras.layers.Dense(300, activation='relu', kernel_regularizer='l2', bias_regularizer='l2'),
    keras.layers.Dense(100, activation='relu', kernel_regularizer='l2', bias_regularizer='l2'),
    keras.layers.Dense(10, activation='softmax')
])
```

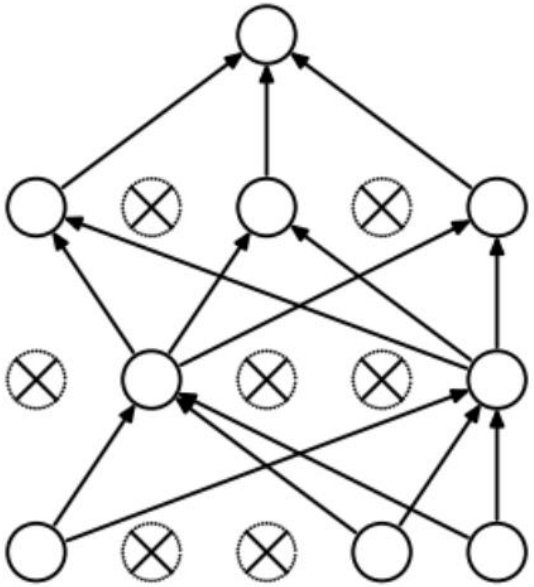


Artificial Neural Networks - KNN
Regularization

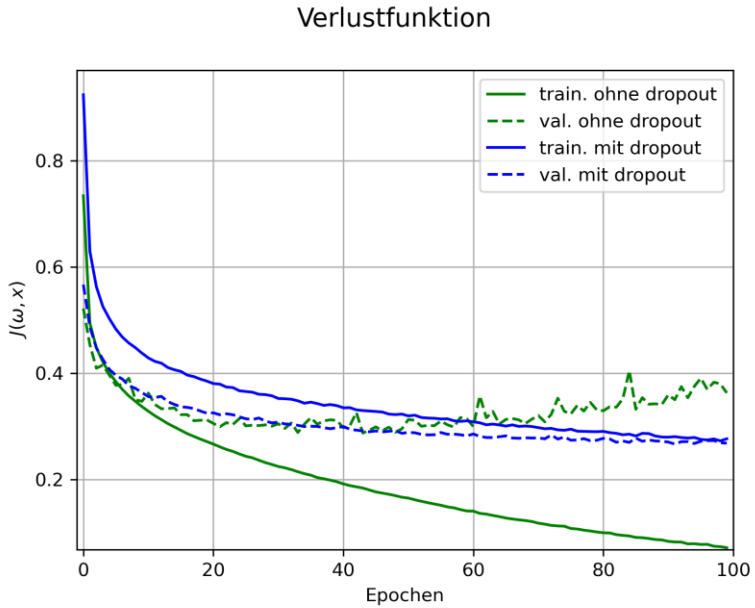
problem:

Overfitting on training data ⟶ **Regularization**

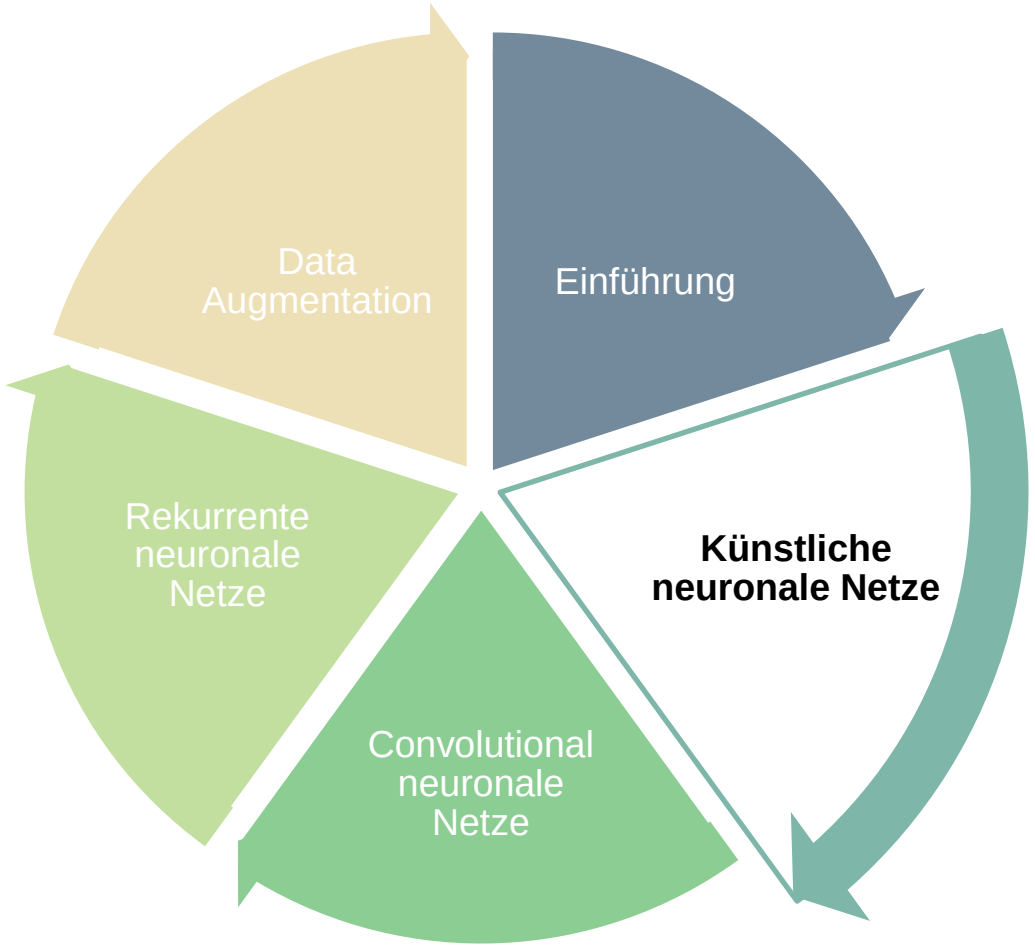
Dropout: Random disable of neurons during training.
Reduces the complexity of the neural network.



```
model_drop = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28,28]),
    keras.layers.Dropout(0.2),
    keras.layers.Dense(300, activation='relu'),
    keras.layers.Dropout(0.2),
    keras.layers.Dense(100, activation='relu'),
    keras.layers.Dropout(0.2),
    keras.layers.Dense(10, activation='softmax'
])
```



Deep Learning
Ausblick

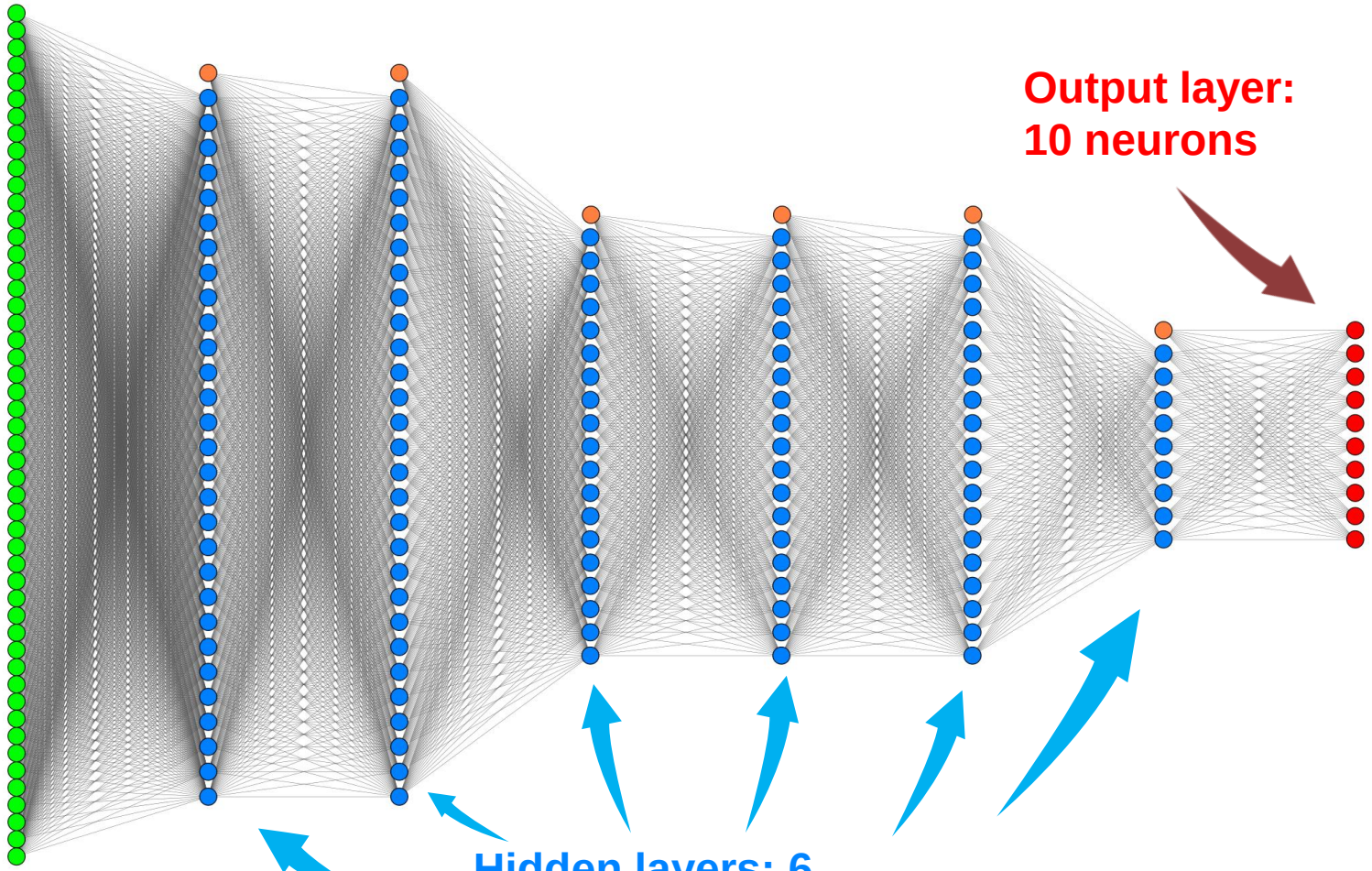


Künstliche neuronale Netze: Part 2

- Training, test and validation data
- Regularization
  - Early Stopping
  - ℓ_1 & ℓ_2-Regularization
  - Dropout
  - **Weight initialization**
  - **Batch normalization**
- Optimization algorithms

Artificial Neural Networks - KNN
Problems during training - motivation weight initialization

Hahn Schickard



Input layer:
28×28=784
neurons

Output layer:
10 neurons

Hidden layers: 6
30, 30, 20, 20, 20, 10 neurons
+ Bias

Model 1
Activation: tanh

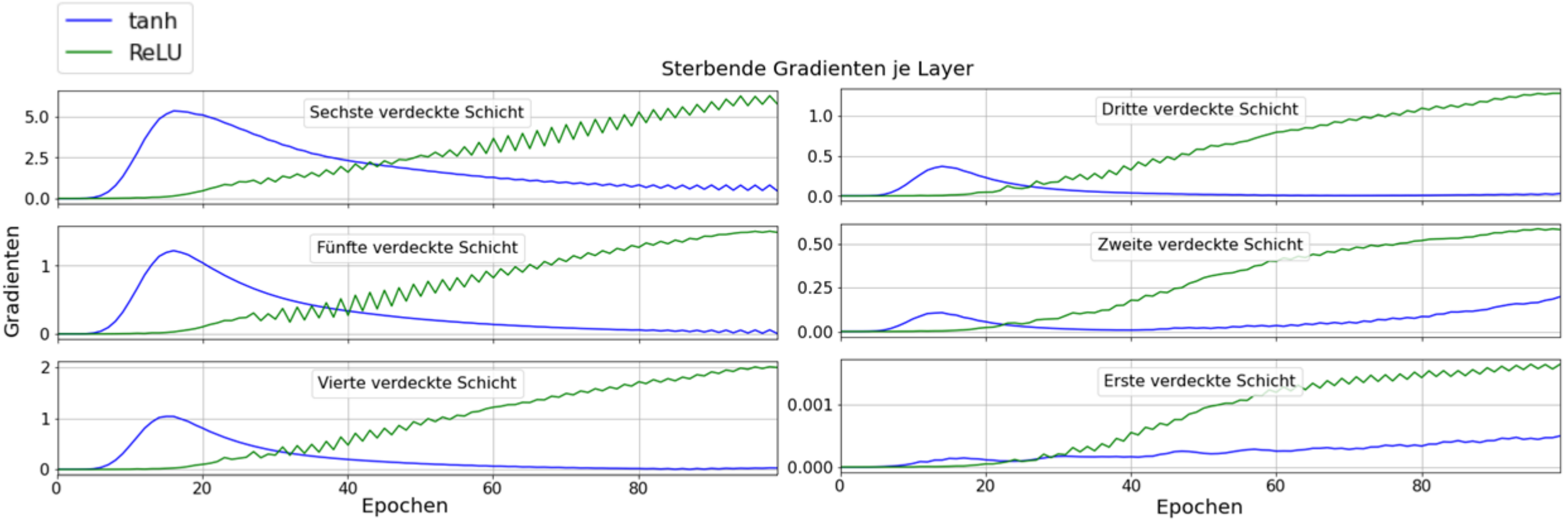
Model 2
Activation: ReLU

Weight initialization:
Normally distributed

How do the gradients of both models change over 100 epochs of the 6 hidden layers?

AiF Mitglied innBW INNOVATIONSALLIANZ BADEN-WÜRTTEMBERG

Artificial Neural Networks - KNN
Problems during training - Example: Vanishing Gradients



The gradients of the first model (tanh) converge to 0 the further back the error is propagated back in the mesh

→ problem of vanishing gradients

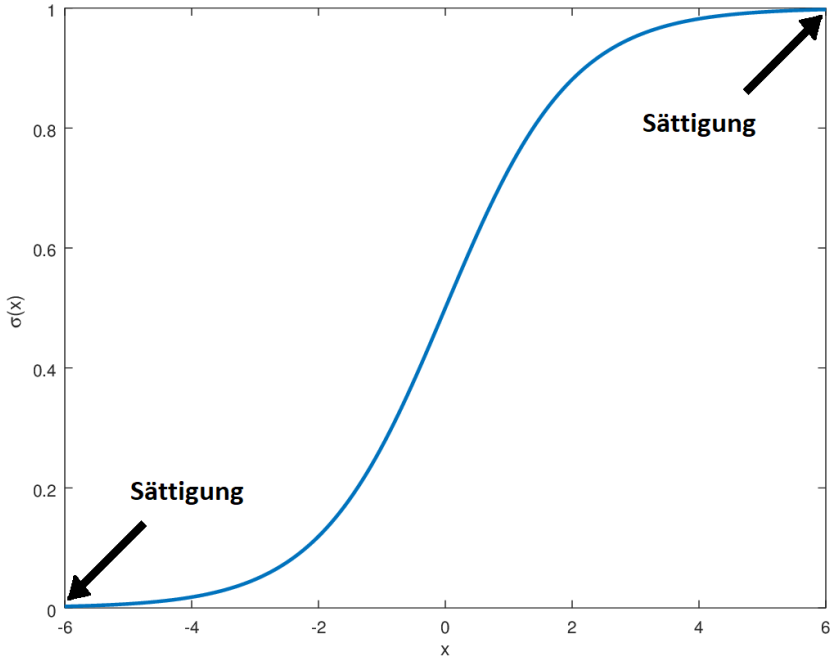
Artificial Neural Networks - KNN

Problems during training - weight initialization

Hahn Schickard

Possible problems

- *Vanishing Gradients:* Weights become smaller and smaller
  - Extinct signal (forward and backward) through the network
- *Exploding Gradients:* Weights get bigger and bigger
  - Weights are greatly changed in the individual steps, unstable network
- Saturation leads to small gradients and slows down training (e.B. by large weights)
- No symmetry break (e.B. due to zero initialization)



Initialisierung	Aktivierungsfunktion	σ^2 (Normal)
Xavier/Glorot	None, tanh, Sigmoid, Softmax	$1 \,/\, fan_{\text{avg}}$
He Normal	ReLU und Varianten	$2 \,/\, fan_{\text{in}}$
LeCun	SELU	$1 \,/\, fan_{\text{in}}$

$$fan_{\text{avg}} = \frac{fan_{\text{in}} + fan_{\text{out}}}{2}$$

- fan_{in} Eingangsneuronen
- fan_{out} Ausgangsneuronen

AiF Mitglied innBW INNOVATIONSALLIANZ BADEN-WÜRTTEMBERG

Artificial Neural Networks - KNN

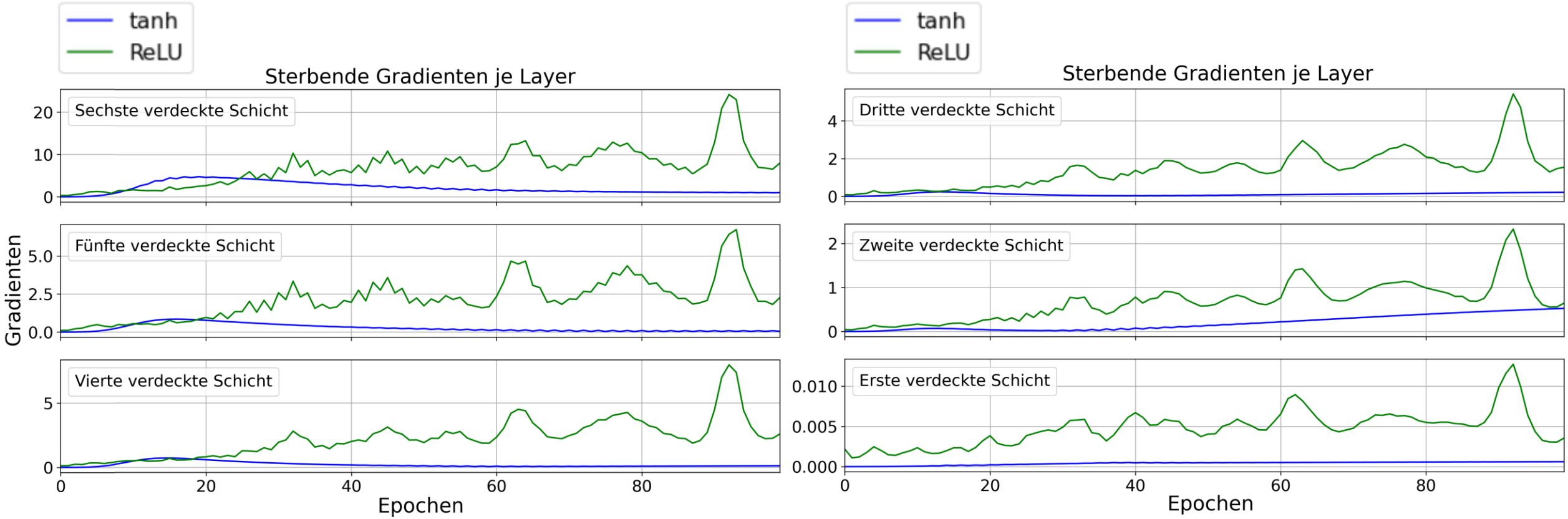
Problems during training - Example: Vanishing Gradients

Setup	Modell 1	Modell 2
Training data	MNIST	MNIST
Anzahl versteckter Schichten (Neuronen)	6 (30,30,20,20,20,10)	6 (30,30,20,20,20,10)
Activation, hidden layers	tanh	ReLU
Activation, output layer	Softmax	Softmax
Weight initialization	Uniform (random)	Uniform (Glorot)

How do the gradients of both models change over 100 epochs of the 6 hidden layers?

Künstliche Neuronale Netze - KNN

Problems during training - Example: Vanishing Gradients



The gradients of the first model (tanh) converge to 0 the further back the error is propagated back in the mesh

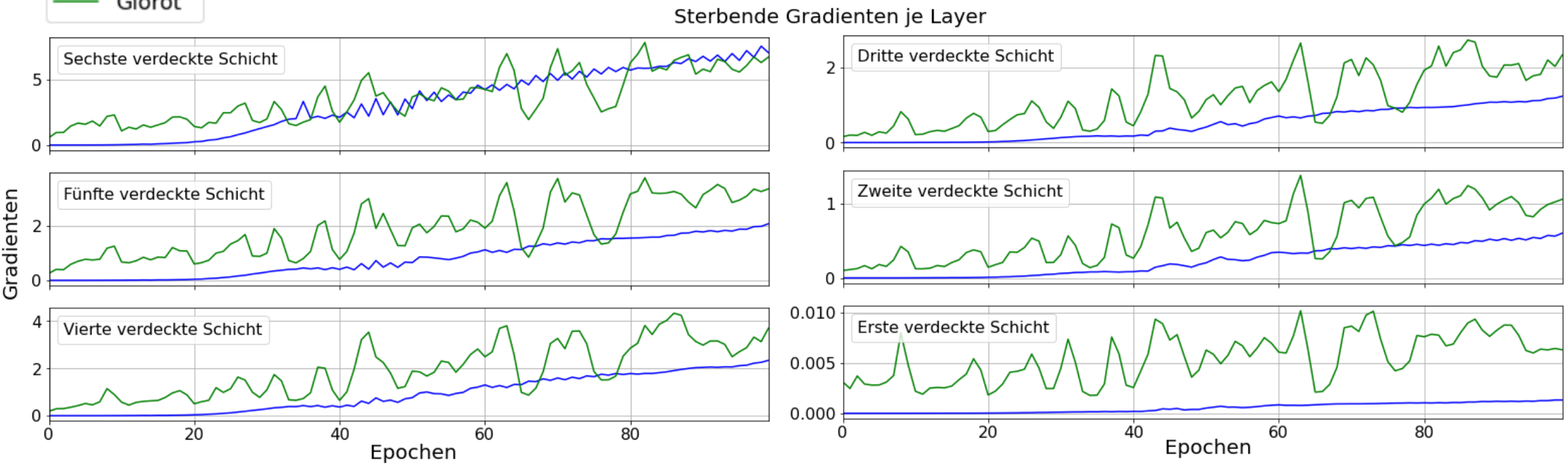
→ **Problem of vanishing gradients**

Artificial Neural Networks - KNN

Problems during training - comparison of two weight initializations

Comparison between normal distribution and Glorot weight initialization

```
= keras.models.Sequential([
yers.Flatten(input_shape=[28,28]),
```




```
])
```

Artificial Neural Networks - KNN
Batch Normalization (BN)

Hahn Schickard

***Vanishing/Exploding* Gradient Problem**

- Start of training
  - Greatly reduced by He-initialization with ELU (or a variant)
- During training?

  → **Batch normalization**!

Batch normalization

- Applied before or after the activation function of each layer
- Calculate mean and variance via the minibatches
- Normalize shift input with the previously calculated statistics
- Scale and move to get the output of the layer
- Used on very large deep neural networks (as they are very sensitive to weight initialization)

$$\boldsymbol{\mu}_B = \frac{1}{m_B}\sum_{i=1}^{m_B}\boldsymbol{x}^{(i)}$$

$$\boldsymbol{\sigma}_B^2 = \frac{1}{m_B}\sum_{i=1}^{m_B}\left(\boldsymbol{x}^{(i)} - \boldsymbol{\mu}_B\right)^2$$

$$\hat{\boldsymbol{x}}^{(i)} = \frac{\boldsymbol{x}^{(i)} - \boldsymbol{\mu}_B}{\sqrt{\boldsymbol{\sigma}_B^2 + \epsilon}}$$

$$\boldsymbol{z}^{(i)} = \boldsymbol{\gamma} \otimes \hat{\boldsymbol{x}}^{(i)} + \boldsymbol{\beta}$$

$\boldsymbol{\gamma}$: Scaleparamerer
$\boldsymbol{\beta}$: Shift parameters

What about $\boldsymbol{\mu}_B, \boldsymbol{\sigma}_B^2$ at the test time?

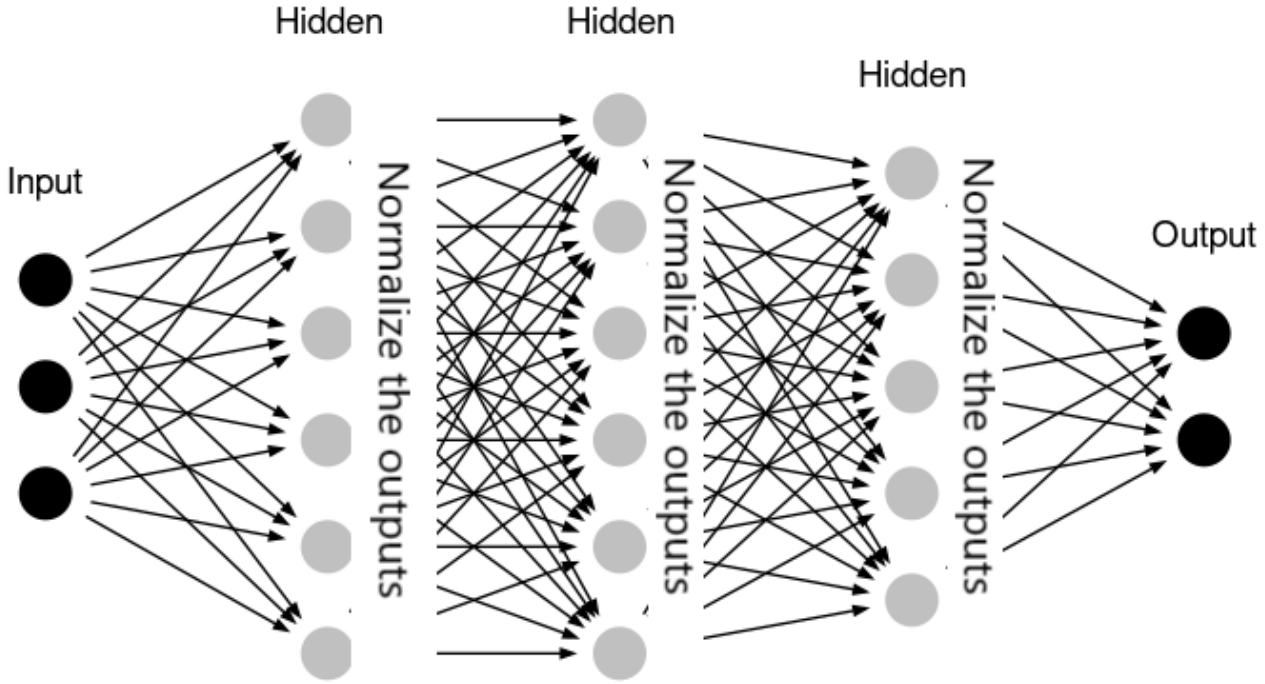
z.B.: exponential moving average

AIF Mitglied innBW INNOVATIONSALLIANZ BADEN-WÜRTTEMBERG

Artificial Neural Networks - KNN
Batch normalization

advantages

- Reduces the risk of vanishing/exploding gradient problem
- Minimizes the problem of covariance shift (covariance shift)
- Less sensitivity to weight initialization and increases the stability of the entire network
- Use of higher learning rates possible
- Reduces the number of training epochs
- The latter and the latter speed up the learning process
- Acts as a regulator and thus reduces the risk of overfitting



disadvantages:

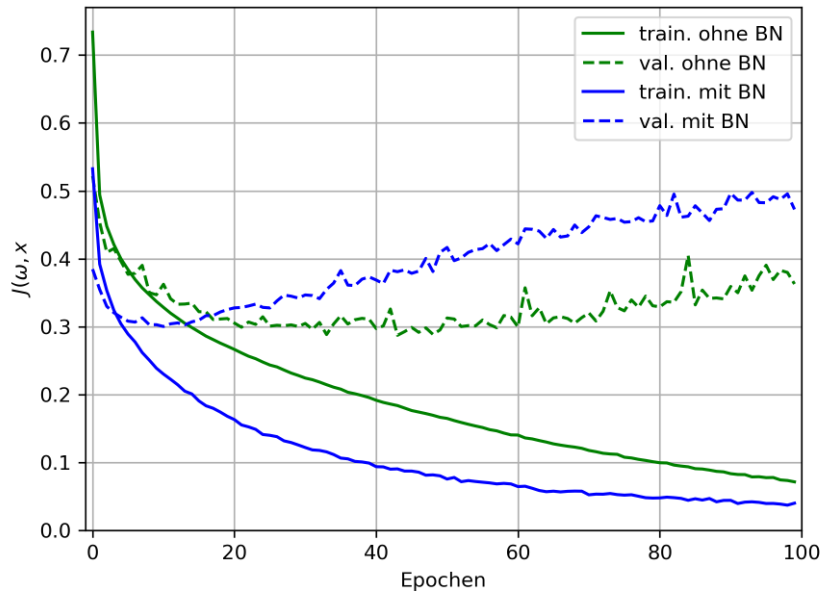
- adds complexity, slower predictions through extra calculations

Artificial Neural Networks - KNN
Example Image Classifier: Batch Normalization

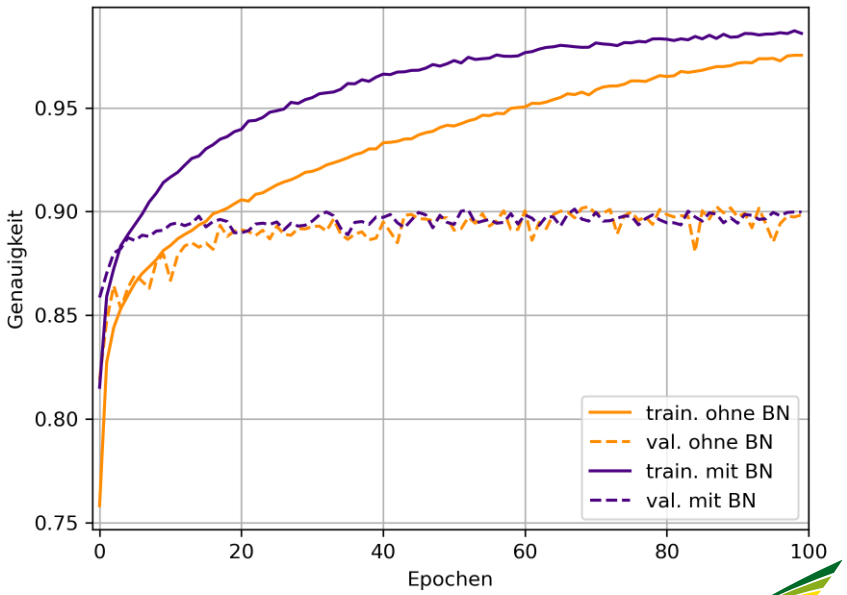
Batch normalization

```
model_batch = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28,28]),
    keras.layers.BatchNormalization(),
    keras.layers.Dense(300, activation='relu'),
    keras.layers.BatchNormalization(),
    keras.layers.Dense(100, activation='relu'),
    keras.layers.BatchNormalization(),
    keras.layers.Dense(10, activation='softmax')
])
```

Verlustfunktion



Metrik



Deep Learning - Ingredients

Hahn Schickard

Model architecture

- Number of shifts
- Activation function

Cost function

Defines what a good neural network is

Optimization algorithm

Minimizes cost function by varying NN weights

Training of the NN

Minimization of the cost function

AiF Mitglied innBW INNOVATIONSALLIANZ BADEN-WÜRTTEMBERG

Artificial Neural Networks - KNN
Optimization Algorithms I

- **Vanilla SGD Gradient Descent**

1. $\boldsymbol{\omega} \leftarrow \boldsymbol{\omega} - \eta \nabla_{\boldsymbol{\omega}} J(\boldsymbol{\omega})$

- **SGD with Momentum**
  - Considers previous gradients (moment vector), 0<β<1
  - Faster convergence and less oscillation

1. $\boldsymbol{m} \leftarrow \beta \boldsymbol{m} - \eta \nabla_{\boldsymbol{\omega}} J(\boldsymbol{\omega})$
2. $\boldsymbol{\omega} \leftarrow \boldsymbol{\omega} + \boldsymbol{m}$

- **Nesterov Accelerated Gradient (NAG)**
  - Measurement of the gradient on $\boldsymbol{\omega} + \beta \boldsymbol{m}$ Instead of $\boldsymbol{\omega}$
  - $\boldsymbol{m}$ usually points towards Optimum
  - Almost always faster than Vanilla Momentum optimization
  - Reduces oscillations

1. $\boldsymbol{m} \leftarrow \beta \boldsymbol{m} - \eta \nabla_{\boldsymbol{\omega}} J(\boldsymbol{\omega} + \beta \boldsymbol{m})$
2. $\boldsymbol{\omega} \leftarrow \boldsymbol{\omega} + \boldsymbol{m}$

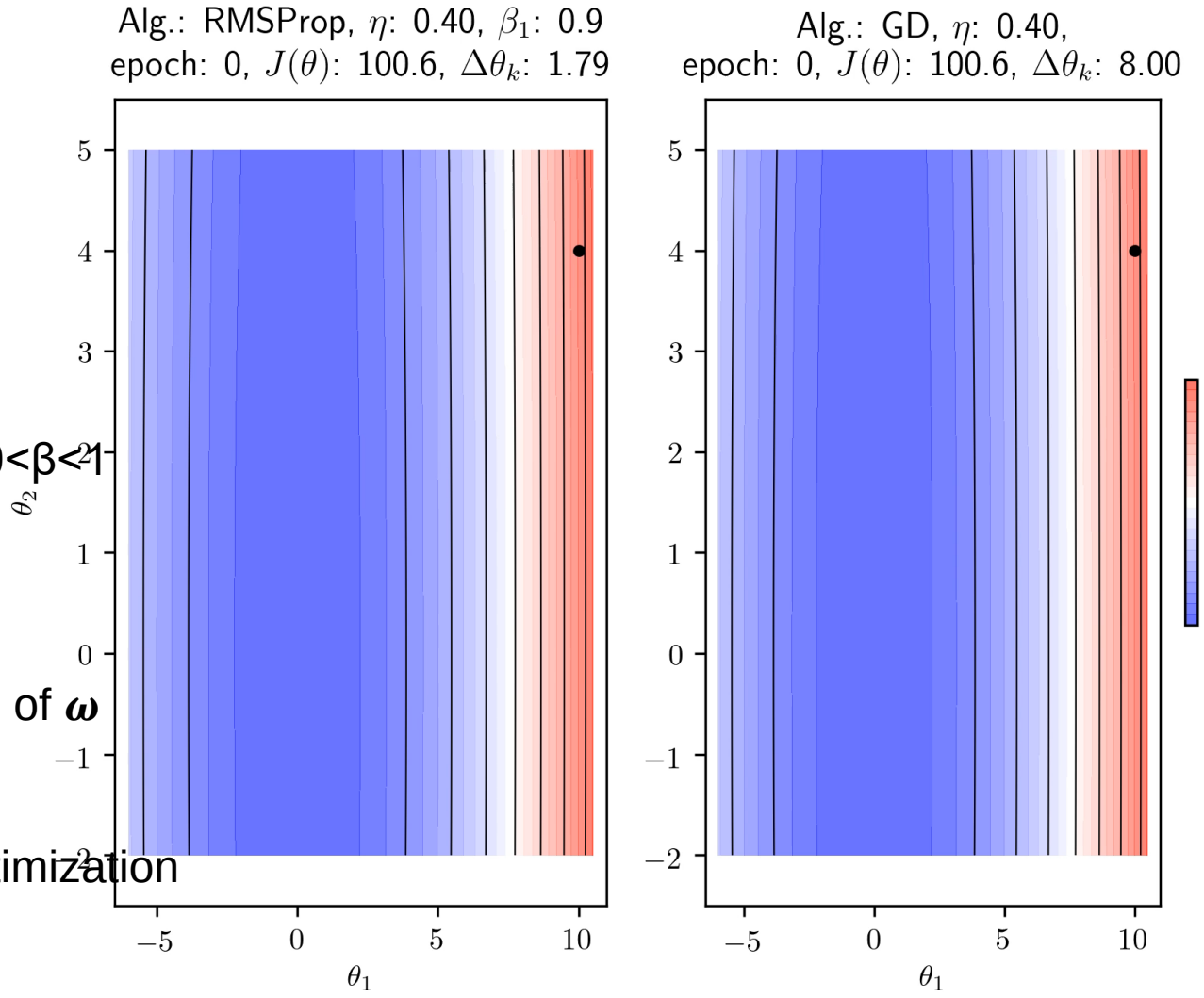
- **RMSProp**
  - Replaces gradient accumulation with exponentially weighted moving average

1. $\boldsymbol{s} \leftarrow \beta \boldsymbol{s} + (1 - \beta) \nabla_{\boldsymbol{\omega}} J(\boldsymbol{\omega}) \otimes \nabla_{\boldsymbol{\omega}} J(\boldsymbol{\omega})$
2. $\boldsymbol{\omega} \leftarrow \boldsymbol{\omega} - \eta \nabla_{\boldsymbol{\omega}} J(\boldsymbol{\omega}) \oslash \sqrt{\boldsymbol{s} + \epsilon}$

Künstliche Neuronale Netze - KNI

Optimierungsalgorithmen I

- **Vanilla SGD Gradient Descent**
- **SGD with Momentum**
  - Considers previous gradients (moment vector), 0<β<1
  - Faster convergence and less oscillation
- **Nesterov Accelerated Gradient (NAG)**
  - Measurement of the gradient on $\boldsymbol{\omega} + \beta \boldsymbol{m}$ Instead of $\boldsymbol{\omega}$
  - $\boldsymbol{m}$ usually points towards Optimum
  - Almost always faster than Vanilla Momentum optimization
  - Reduces oscillations
- **RMSProp**
  - Replaces gradient accumulation with exponentially weighted moving average



1. $\boldsymbol{s} \leftarrow \beta \boldsymbol{s} + (1-\beta)\nabla_{\omega} J(\boldsymbol{\omega}) \otimes \nabla_{\boldsymbol{\omega}} J(\boldsymbol{\omega})$
2. $\boldsymbol{\omega} \leftarrow \boldsymbol{\omega} - \eta \nabla_{\boldsymbol{\omega}} J(\boldsymbol{\omega}) \oslash \sqrt{\boldsymbol{s} + \epsilon}$

Component-by-component multiplication and division

Mitglied innBW INNOVATIONSALLIANZ BADEN-WÜRTTEMBERG

Artificial Neural Networks - KNN
Optimization Algorithms II

Hahn Schickard

1 Adam Algorithmus

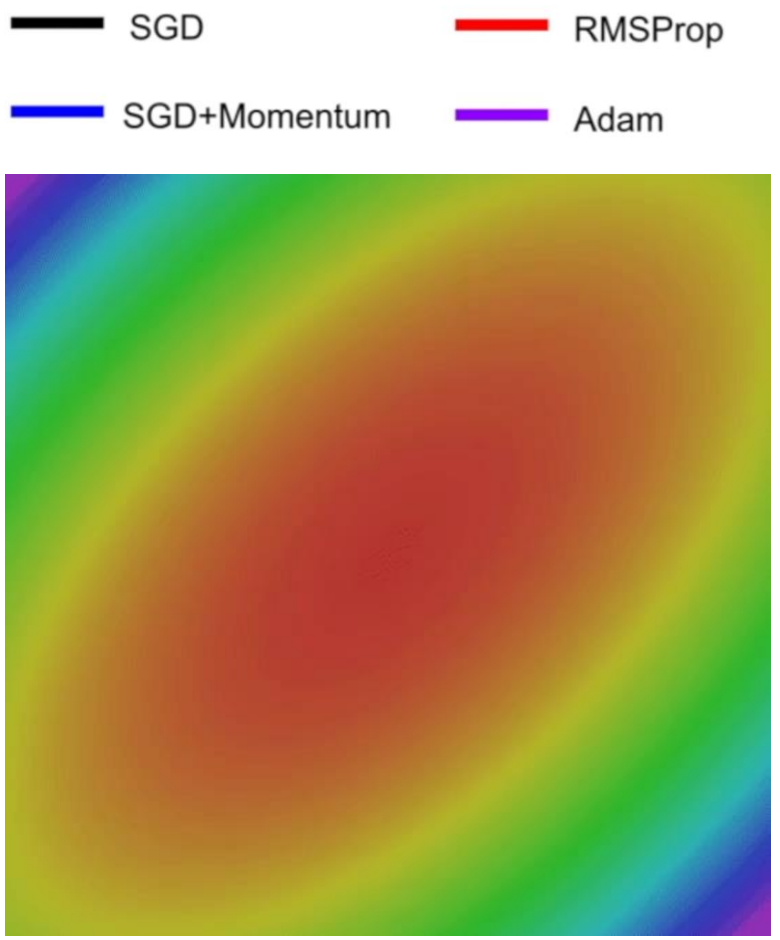
Der Adam Algorithmus wird in Algorithmus 1 dargestellt. Die Berechnung des 1. und des 2. Moments sind aus dem Momentum Optimierung Algorithmus bzw. des RMSProp Algorithmus bekannt.

Algorithm 1 Adam Algorithmus

Require: $\eta > 0$: Schrittweite
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponentielle Abbauraten für 1. und 2. Moment
Require: $J(\boldsymbol{\omega})$: Zielfunktion
Require: $\boldsymbol{\omega}_0$: Startvektor
$\boldsymbol{m}_0 \leftarrow 0$ (1. Moment auf Null initialisieren)
$\boldsymbol{v}_0 \leftarrow 0$ (2. Moment auf Null initialisieren)
$t \leftarrow 0$ (Zeitschritt auf Null initialisieren)
while $\boldsymbol{\omega}_t$ not converged **do**
$t \leftarrow t + 1$ ▷ Aktualsisiere Zeitschritt
$\boldsymbol{g}_t \leftarrow \nabla_{\boldsymbol{\omega}} J(\boldsymbol{\omega}_{t-1})$ ▷ Erhalte Gradienten in Schritt t
$\boldsymbol{m}_t \leftarrow \beta_1 \cdot \boldsymbol{m}_{t-1} + (1 - \beta_1) \cdot \boldsymbol{g}_t$ ▷ Aktualisiere 1. Moment
$\boldsymbol{v}_t \leftarrow \beta_2 \cdot \boldsymbol{v}_{t-1} + (1 - \beta_2) \cdot \boldsymbol{g}_t \odot \boldsymbol{g}_t$ ▷ Aktualisiere 2. Moment

$\widehat{\boldsymbol{m}}_t \leftarrow \boldsymbol{m}_t / (1 - \beta_1^t)$ ▷ Bias im 1. Moment korrigieren
$\widehat{\boldsymbol{v}}_t \leftarrow \boldsymbol{v}_t / (1 - \beta_2^t)$ ▷ Bias im 2. Moment korrigieren

$\boldsymbol{\omega}_t \leftarrow \boldsymbol{\omega}_{t-1} - \eta \cdot \widehat{\boldsymbol{m}}_t \oslash \left(\sqrt{\widehat{\boldsymbol{v}}_t} + \epsilon\right)$ ▷ Aktualisiere die Parameter
end while
return $\boldsymbol{\omega}_t$ ▷ Rückgabe der Endparameter



AIF Mitglied
innBW INNOVATIONSALLIANZ BADEN-WÜRTTEMBERG

Künstliche Neuronale Netze - KNN

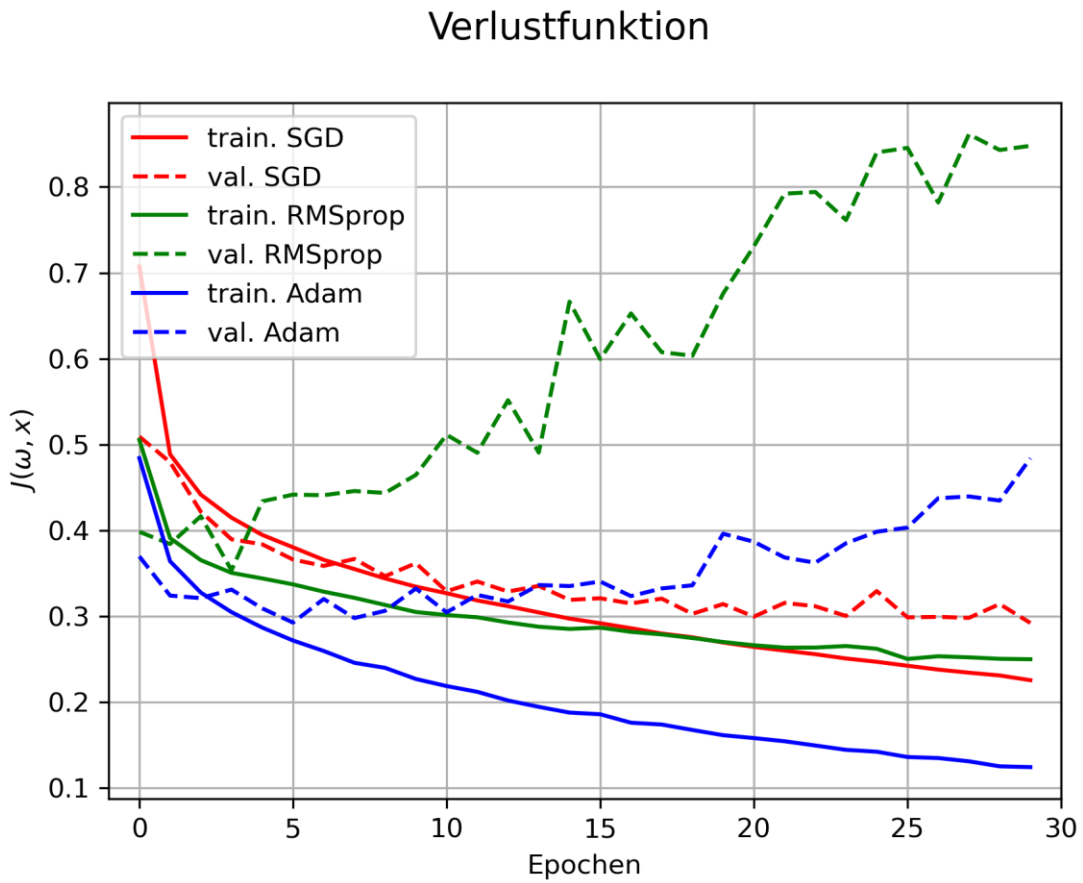
Optimierungsalgorithmen in Keras

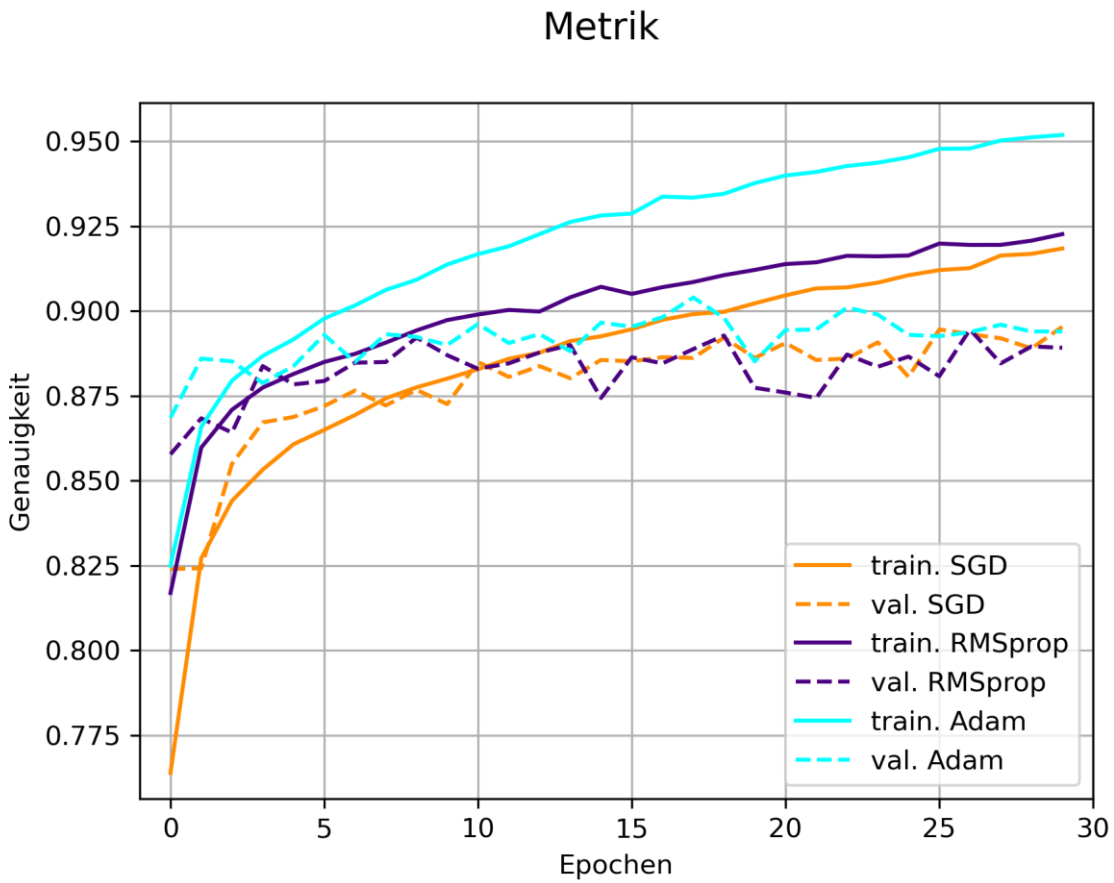
```
# Adadelta
...
    optimizer="Adadelta"
    optimizer=keras.optimizers.Adadelta()
...

# Adagrad
...
    optimizer="Adagrad"
    optimizer=keras.optimizers.Adagrad()
...

# Adam
...
    optimizer="Adam"
    optimizer=keras.optimizers.Adam()
...
```

```
# Nadam
...
    optimizer="Nadam"
    optimizer=keras.optimizers.Nadam()
...

# RMSprop
...
    optimizer="RMSprop"
    optimizer=keras.optimizers.RMSprop()
...

# SGD
...
    optimizer="SGD"
    optimizer=keras.optimizers.SGD()
...
```

Artificial Neural Networks - KNN
Example Image Classifiers: Optimization Algorithms

Hahn Schickard





AIF Mitglied
innBW INNOVATIONSALLIANZ BADEN-WÜRTTEMBERG

Artificial Neural Networks - KNN
Training procedure Summary

Hahn Schickard

Training procedures

1. Initialize weights (randomly or according to a scheme)
2. Propagate training data forward through the net
3. Calculate the error between the target output and the actual output
4. Propagate the error backwards and calculate the proportional error of all neurons
5. Update the weights using an optimization process (gradient descent)
6. If the error is greater than a desired amount, jump to point 2.
   Otherwise, the training ends

AiF Mitglied innBW INNOVATIONSALLIANZ BADEN-WÜRTTEMBERG

Artificial Neural Networks - KNN Practice Guide

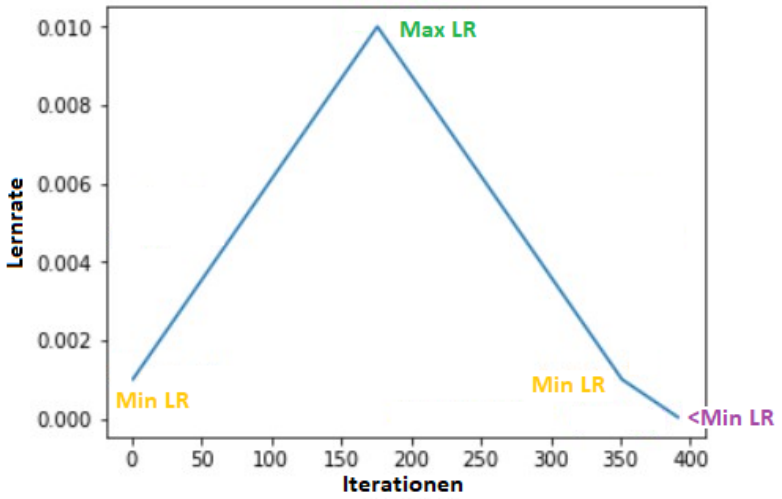
Hahn Schickard

Hyperparameter	default
Weight initialization	HE Normal
Activation function	ELU
normalization	Batch normalization bei Deep
Regularization	Early Stopping ($+\ell_2$ falls nötig)
optimizer	SGD with Momentum (or RMSProp or Nadam)
Learning rate	*1Cycle policy*

One-cycle policy

Maximum learning rate: LR range test

- Minimum learning rate: $\frac{1}{5}$ or $\frac{1}{10}$ maximum learning rate
- Cycle length: < Total number of epochs
- Last iterations: LR < $\frac{1}{10}$ or $\frac{1}{100}$



AIF Mitglied innBW INNOVATIONSALLIANZ BADEN-WÜRTTEMBERG

Artificial Neural Networks - KNN
Image classifiers with Keras

Hahn Schickard

MNIST Fashion Record

1. Importing the packages
2. Loading the MNIST Fashion Record

```
import tensorflow as tf
from tensorflow import keras
import numpy as np
```

```
fashion_mnist = keras.datasets.fashion_mnist
(X_train_full, y_train_full), (X_test, y_test) = fashion_mnist.load_data()
```


AiF Mitglied innBW INNOVATIONSALLIANZ BADEN-WÜRTTEMBERG

Artificial Neural Networks - KNN
Image classifiers with Keras

Hahn Schickard

1. Examine the form of the data
2. Investigate data type
3. Create validation and training data, scale the input data
4. Create a sequential model

```
X_train_full.shape
```

```
(60000, 28, 28)
```

```
X_train_full.dtype
```

```
dtype('uint8')
```

```
X_valid, X_train = X_train_full[:5000] / 255., X_train_full[5000:] / 255.
y_valid, y_train = y_train_full[:5000], y_train_full[5000:]
X_test = X_test / 255.
```

```
model = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28, 28]),
    keras.layers.Dense(300, activation="relu"),
    keras.layers.Dense(100, activation="relu"),
    keras.layers.Dense(10, activation="softmax")])
```


AIF Mitglied innBW INNOVATIONSALLIANZ BADEN-WÜRTTEMBERG

Artificial Neural Networks - KNN
Image classifiers with Keras

Hahn Schickard

5. Summary of the model and list the parameters

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 784)               0
_________________________________________________________________
dense (Dense)                (None, 300)               235500
_________________________________________________________________
dense_1 (Dense)              (None, 100)               30100
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1010
=================================================================
Total params: 266,610
Trainable params: 266,610
Non-trainable params: 0
_________________________________________________________________
```


AIF Mitglied innBW INNOVATIONSALLIANZ BADEN-WÜRTTEMBERG

Artificial Neural Networks - KNN
Image classifiers with Keras

1. Evaluation of the model on the test data
2. Make predictions

- Class probabilities
- Class with the highest estimated probability

```
model.evaluate(X_test, y_test)
```

```
10000/10000 - loss: 0.3361 - accuracy: 0.8780
```

```
X_new = X_test[:3]
y_proba = model.predict(X_new)
y_proba.round(2)
```

```
array([[0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.01, 0.  , 0.99],
       [0.  , 0.  , 0.99, 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  ],
       [0.  , 1.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  ]],
      dtype=float32)
```

```
y_pred = model.predict_classes(X_new)
y_pred
```

```
array([9, 2, 1], dtype=int64)
```

Hands-on

https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/tutorials/keras/classification.ipynb#scrollTo=2tRmdq_8CaXb

Additional Infos

Questions

Artificial Neural Networks - KNN
Questions I

The perceptron fires whenever

-All inputs greater than 0 are

-If at least one input is less than 0

-Sum of weighted input greater than 0 is

A two-intake perceptron solves the following problems

-Linearly separable problem in a two-dimensional plane

-Linearly separable problem in a three-dimensional plane

-Nonlinear separable problem in a two-dimensional plane (XOR problem)

-None of the above problems

Artificial Neural Networks - KNN Questions II

What is true in terms of backward propagation?

-The error in the output is propagated only backwards to set weight updates

-A learning algorithm for neural networks with only one feedforward layer

-Local minima and slow convergence can limit backward propagation

Is learning back-propagation based on a gradient descent along the error surface?

Artificial Neural Networks - KNN
Questions III

What statements about linear activation functions are true?

- Only binary outputs are possible
- Allows non-linear mapping between inputs and outputs
- Gradient descent is not possible because the derivation is a constant

What is true in terms of backward propagation?

- The error in the output is propagated only backwards to set weight updates
- It is a learning algorithm for neural networks with only one feedforward layer
- Local minima and slow convergence can limit backward propagation
- Is learning back-propagation based on a gradient descent along the error surface?