Hochschule Düsseldorf
University of Applied Sciences

Fachbereich Medien
Faculty of Media

MIREVI

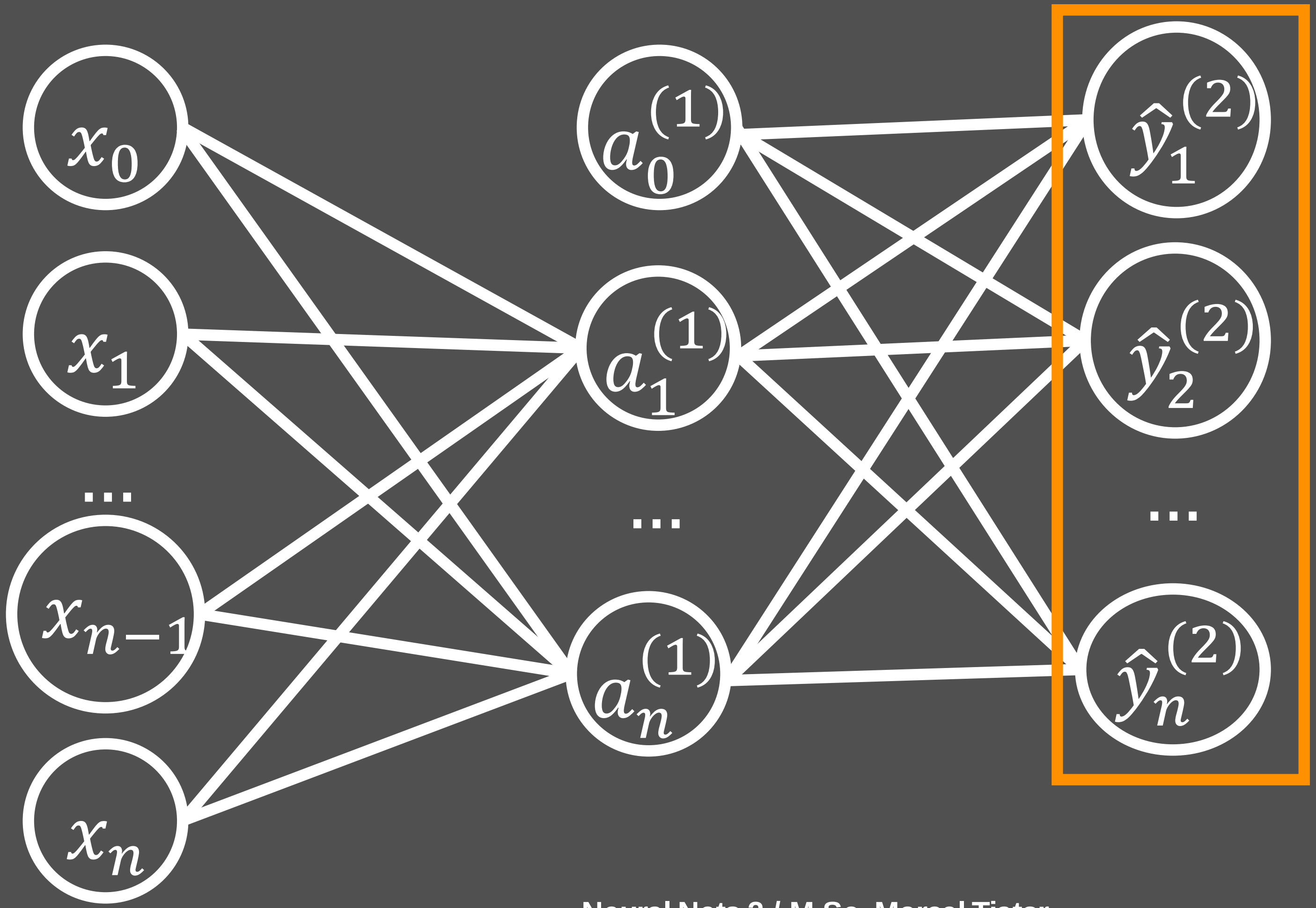
Neural Nets 2

Training & Tuning

Overview:

- Loss Functions
- Backpropagation
- Cross Validation
- Over- and Underfitting
- Tipps & Tricks (e.g. Learning Rate Schedule)
- CNN

Compute forward propagation (Prediction)



Compute error/loss $E(y, \hat{y})$

$\hat{y}_1^{(2)}$ $y_1^{(2)}$

$\hat{y}_2^{(2)}$ $y_2^{(2)}$

... ...

$\hat{y}_n^{(2)}$ $y_n^{(2)}$

1D Output:

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

nD Output:

$$E = \frac{1}{2m} \sum_{i=1}^{m} \frac{1}{n} \sum_{j=1}^{n} (\hat{y}_{ij} - y_{ij})^2$$

$$E = \frac{1}{2mn} \sum_{i=1}^{m} \sum_{j=1}^{n} (\hat{y}_{ij} - y_{ij})^2$$

Compute error/loss $E(y, \hat{y})$

$\hat{y}_1^{(2)}$ $y_1^{(2)}$

$\hat{y}_2^{(2)}$ $y_2^{(2)}$

… …

$\hat{y}_n^{(2)}$ $y_n^{(2)}$

1D Output (m: Batch Size):

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

nD Output:

$$E = \frac{1}{2m} \sum_{i=1}^{m} \frac{1}{n} \sum_{j=1}^{n} (\hat{y}_{ij} - y_{ij})^2$$

$$E = \frac{1}{2mn} \sum_{i=1}^{m} \sum_{j=1}^{n} (\hat{y}_{ij} - y_{ij})^2$$

$\hat{y}$	y
0	1
1	0
0	0

$$E = \frac{1}{2mn}\sum_{i=1}^{m}\sum_{j=1}^{n}(\hat{y}_{ij} - y_{ij})^2$$

$\hat{y}$	y
0	1
1	0
0	0

$$E = \frac{1}{2mn}\sum_{i=1}^{m}\sum_{j=1}^{n}(\hat{y}_{ij} - y_{ij})^2$$

$$m = 1, n = 3$$

$\hat{y}$	y
0	1
1	0
0	0

$$E = \frac{1}{2mn} \sum_{i=1}^{m} \sum_{j=1}^{n} (\hat{y}_{ij} - y_{ij})^2$$

$\hat{y}$	y
0	1
1	0
0	0

$$m = 1, n = 3$$

$$E = \frac{1}{2 * 1 * 3} \sum_{i=1}^{1} \sum_{j=1}^{3} (\hat{y}_{ij} - y_{ij})^2$$

$$E = \frac{1}{2mn}\sum_{i=1}^{m}\sum_{j=1}^{n}(\hat{y}_{ij} - y_{ij})^2$$

$\hat{y}$	y
0	1
1	0
0	0

$$m = 1, n = 3$$

$$E = \frac{1}{2 * 1 * 3}\sum_{i=1}^{1}\sum_{j=1}^{3}(\hat{y}_{ij} - y_{ij})^2$$

$$E = \frac{1}{6}\sum_{j=1}^{3}(\hat{y}_{j} - y_{j})^2$$

$$E = \frac{1}{2mn} \sum_{i=1}^{m} \sum_{j=1}^{n} (\hat{y}_{ij} - y_{ij})^2$$

$\hat{y}$	y
0	1
1	0
0	0

$$m = 1, n = 3$$

$$E = \frac{1}{2 * 1 * 3} \sum_{i=1}^{1} \sum_{j=1}^{3} (\hat{y}_{ij} - y_{ij})^2$$

$$E = \frac{1}{6} \sum_{j=1}^{3} (\hat{y}_j - y_j)^2$$

$$E = \frac{1}{6}\left((0-1)^2 + (1-0)^2 + (0-0)^2\right) = \frac{1}{3}$$

Logarithmic Loss (Cross Entropy)

- Captures the intuition of classification
- Can be used for output probabilities (e.g. Softmax)
- p: Predicted probability of a class (confidence)
- Binary: $E = -(y * \log(p) + (1 - y) * \log(1 - p))$
- Multiclass: $E = -\sum_{j=1}^{n} y_j * \log(p_j)$

$$E = -(y * \log(p) + (1 - y) * \log(1 - p))$$

Assume $y = 1$**:**

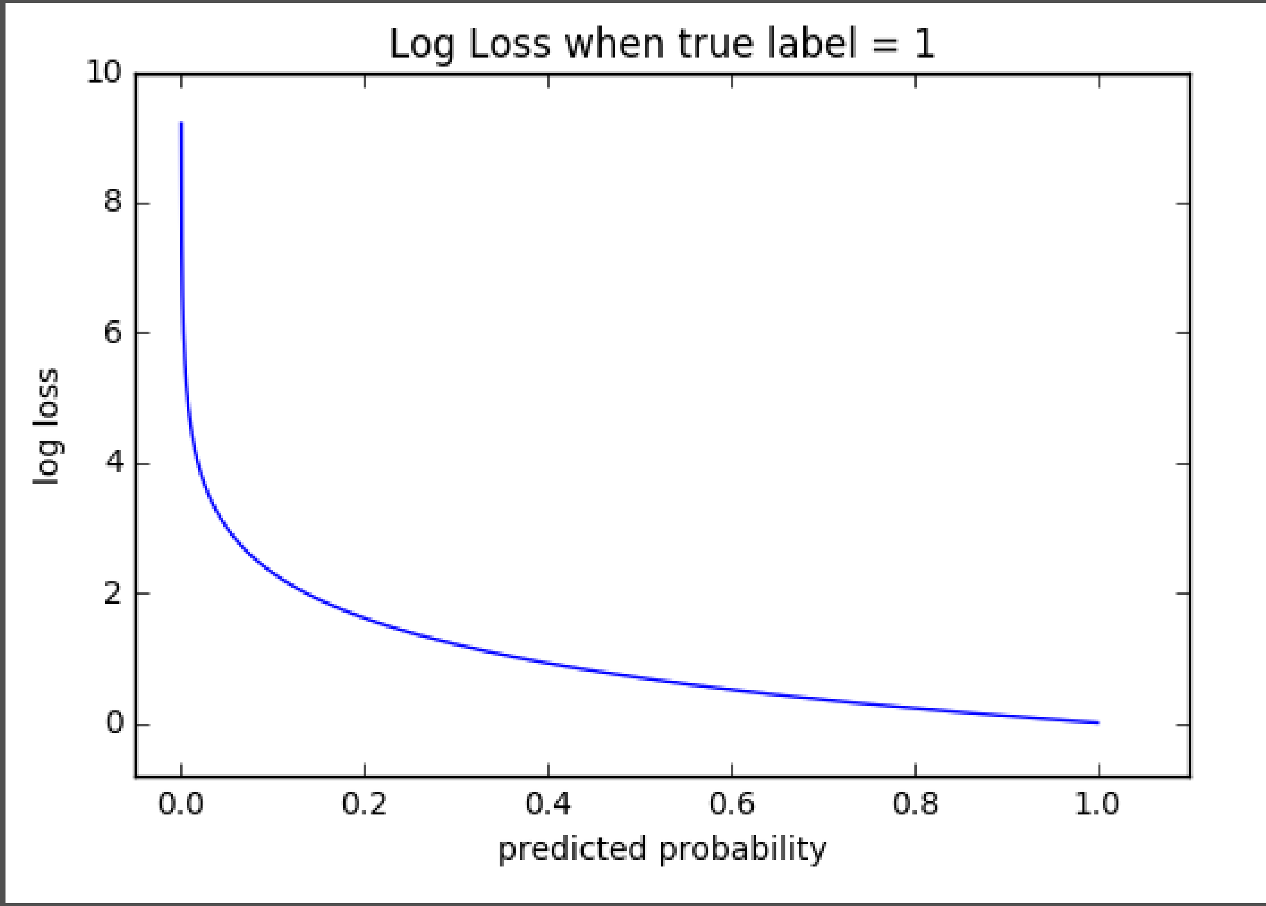
$$E(y = 1, p) = -\big(1 * \log(p) + (1 - 1) * \log(1 - p)\big)$$

$$E(y = 1, p) = -\big(1 * \log(p)\big)$$

$$E(y = 1, p) = -\log(p)$$

$$E = -(y * \log(p) + (1-y) * \log(1-p))$$

Assume $y = 1$: $E(y = 1, p) = -\log(p)$

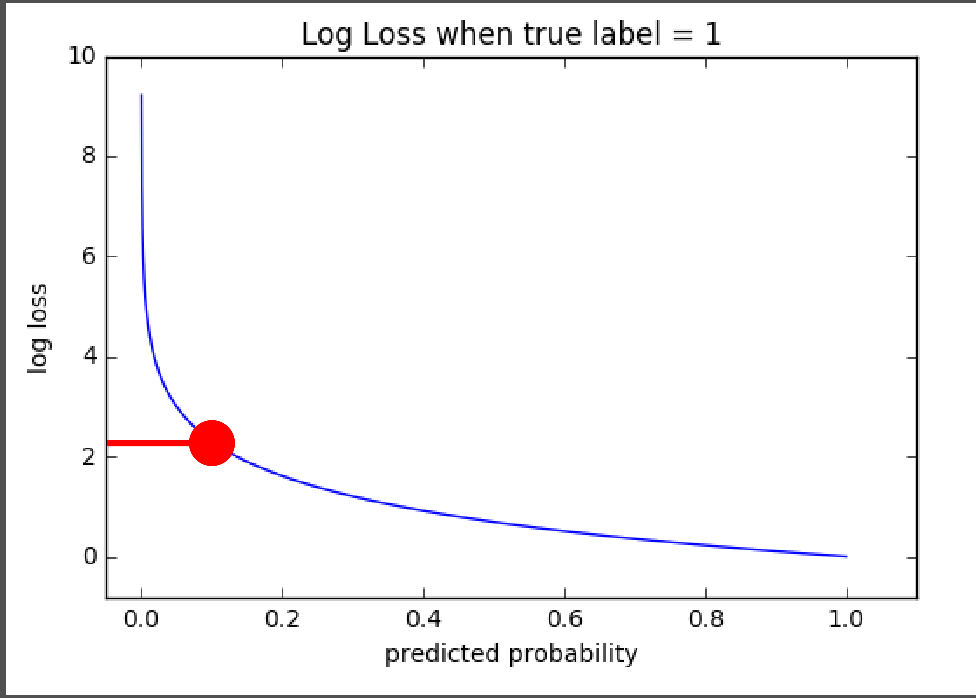


$$E = -\sum_{j=1}^{n} y_j * \log(p_j)$$

$$E = -(1 * \log(0.1) + 0 * \log(0.8) + 0 * \log(0.1))$$

$$E = -\log(0.1)$$

$\hat{y}$	y
0.1	1
0.8	0
0.1	0

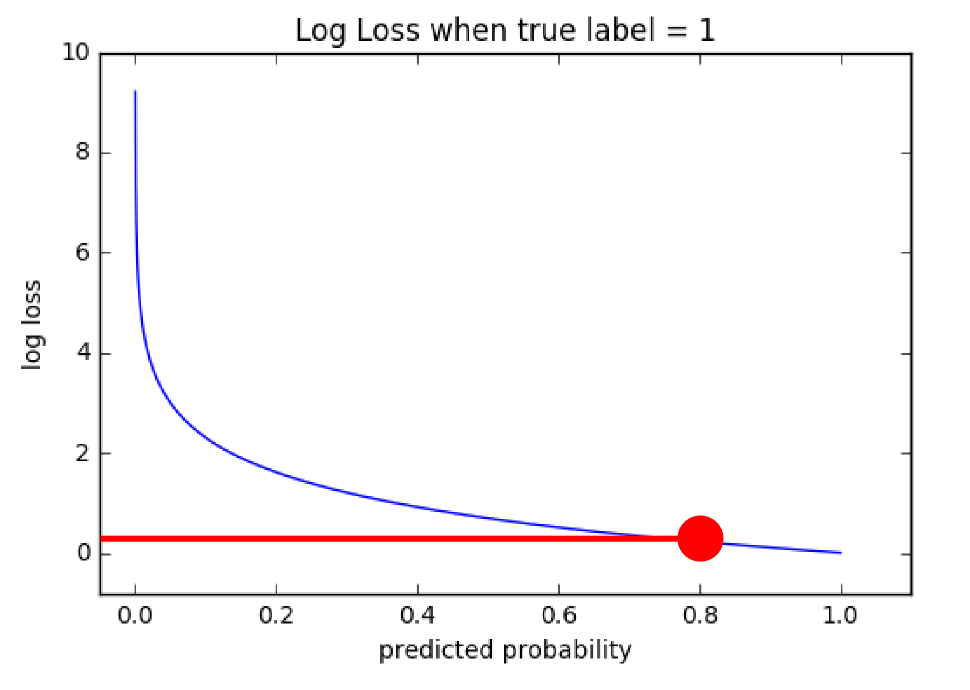


$$E = -\sum_{j=1}^{n} y_j * \log(p_j)$$

$$E = -(1 * \log(0.8) + 0 * \log(0.1) + 0 * \log(0.1))$$

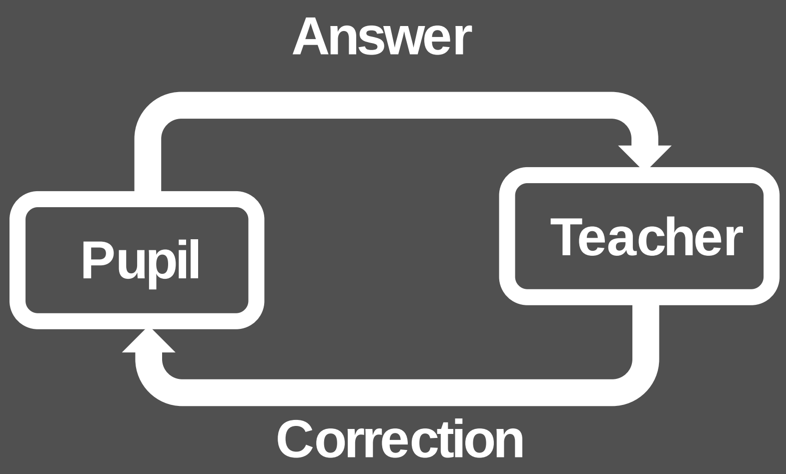
$$E = -\log(0.8)$$

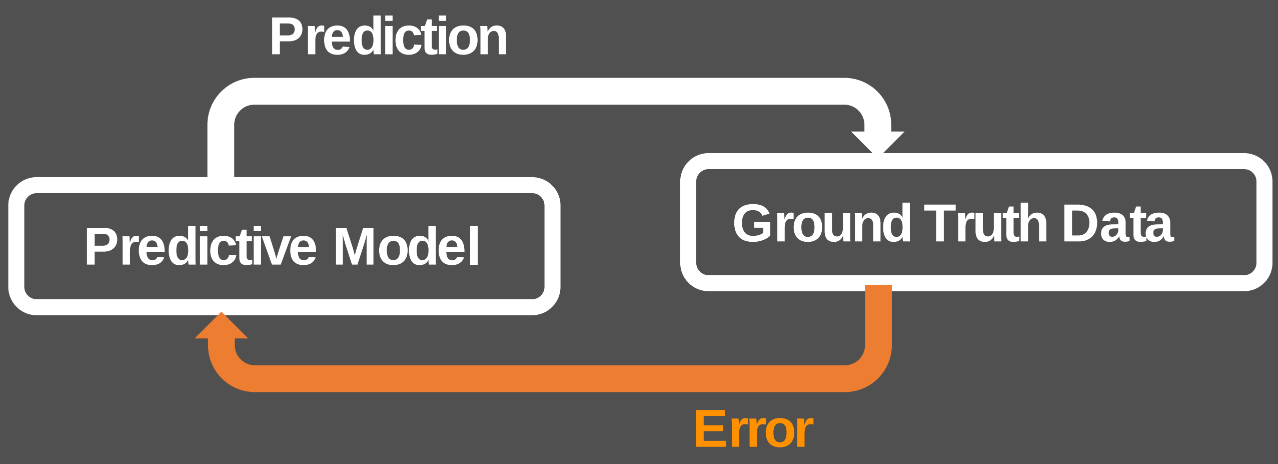
$\hat{y}$	y
0.8	1
0.1	0
0.1	0



Two Processes:

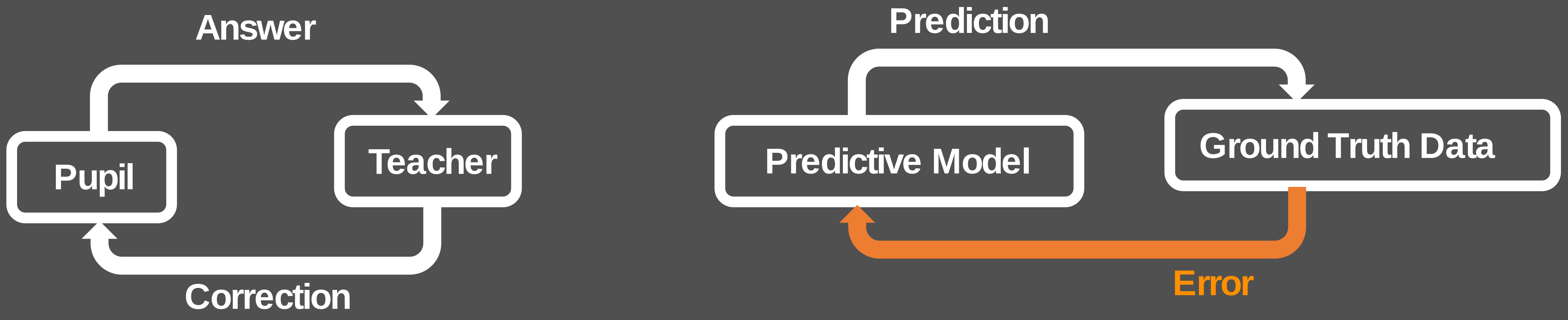
- Forward Propagation (Prediction)
- Backward Propagation (Weight Tuning)





Backpropagation

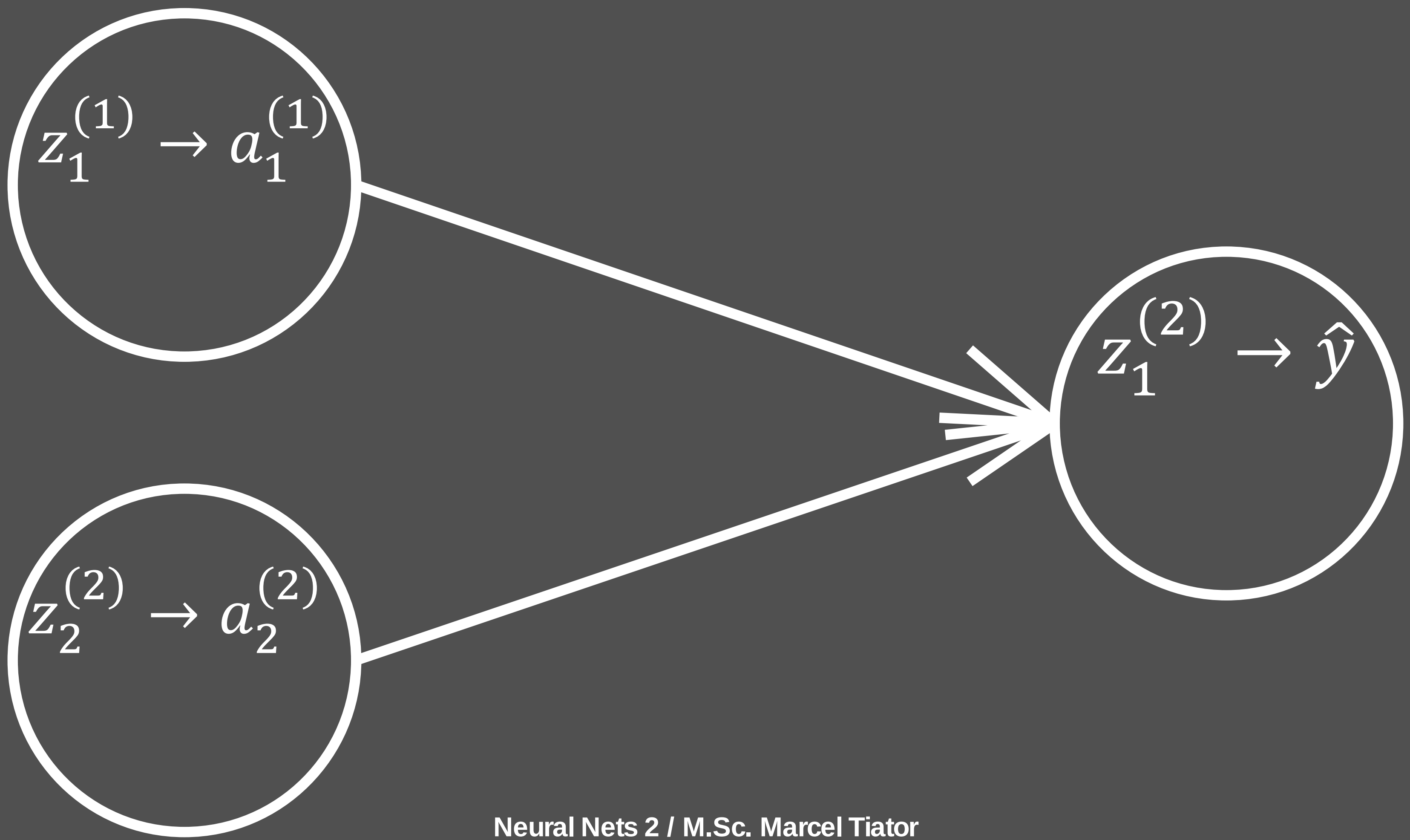
- Compare to more complex Gradient Descent
- Training of Neural Net



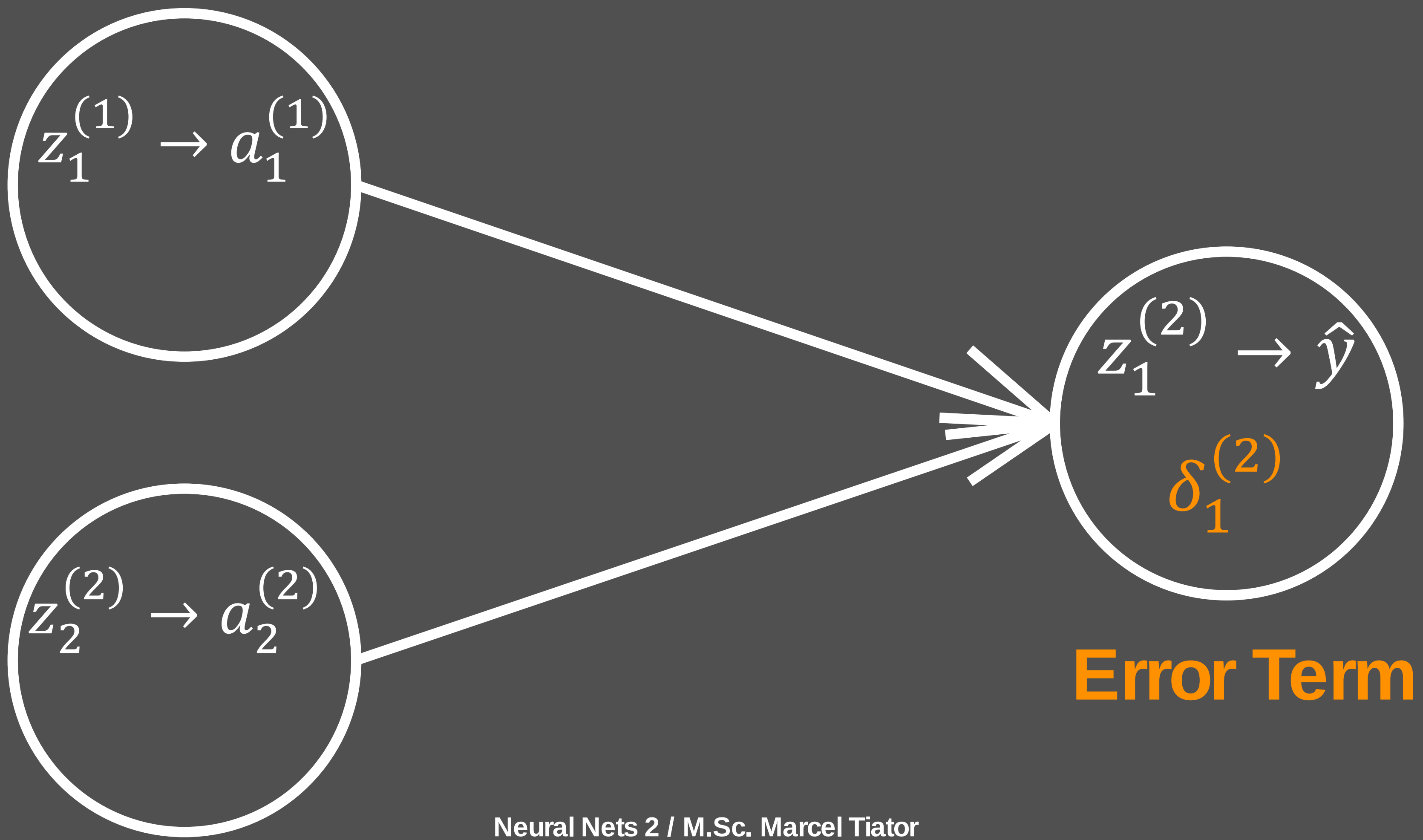
Algorithm (Given input x)

- Compute forward propagation
  - Get a prediction $\hat{y}$
- Compute error/loss $E(y, \hat{y})$
- Propagate the error back through the network
  - Tune weights

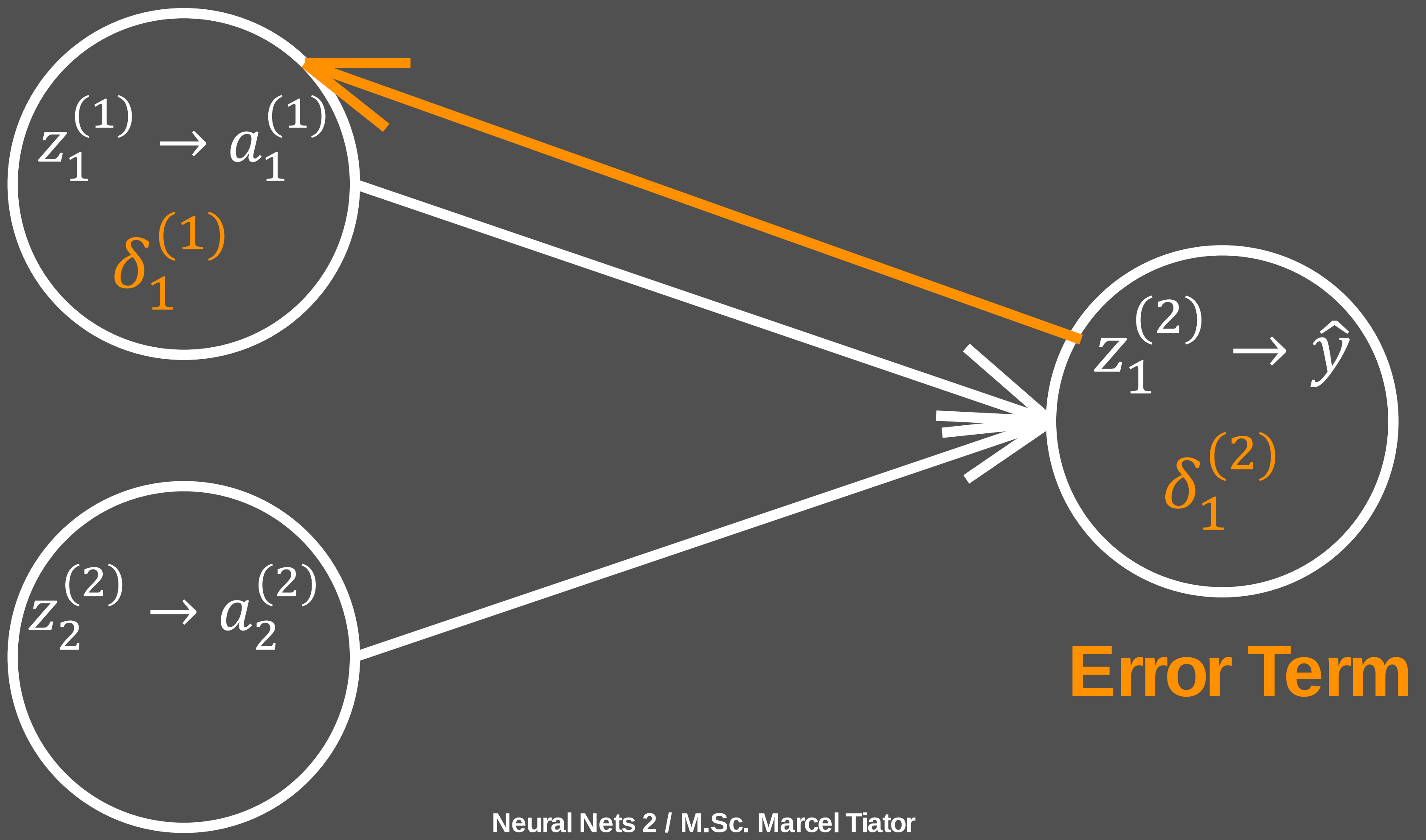
Propagate the error back through the network



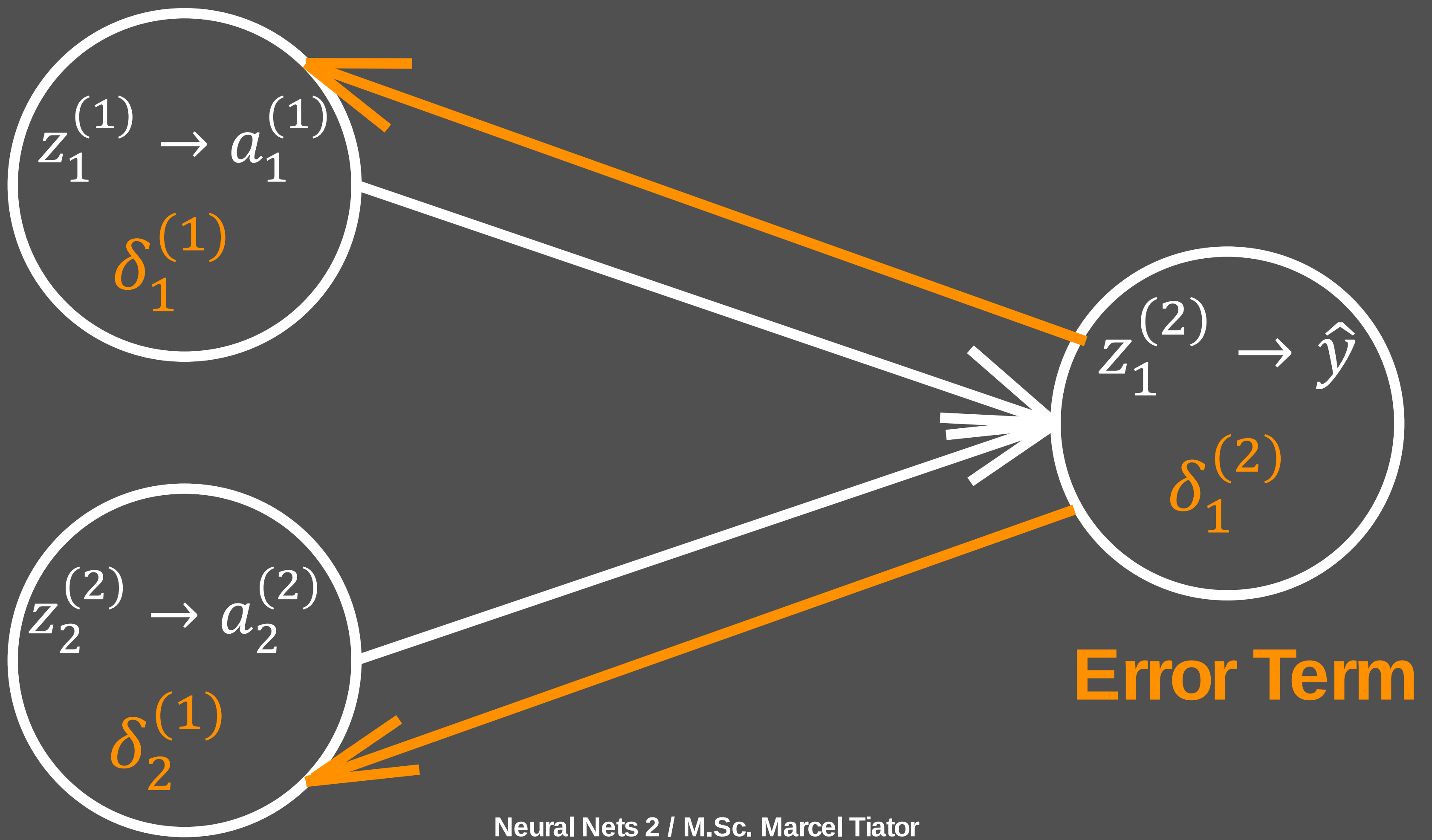
Propagate the error back through the network



Propagate the error back through the network

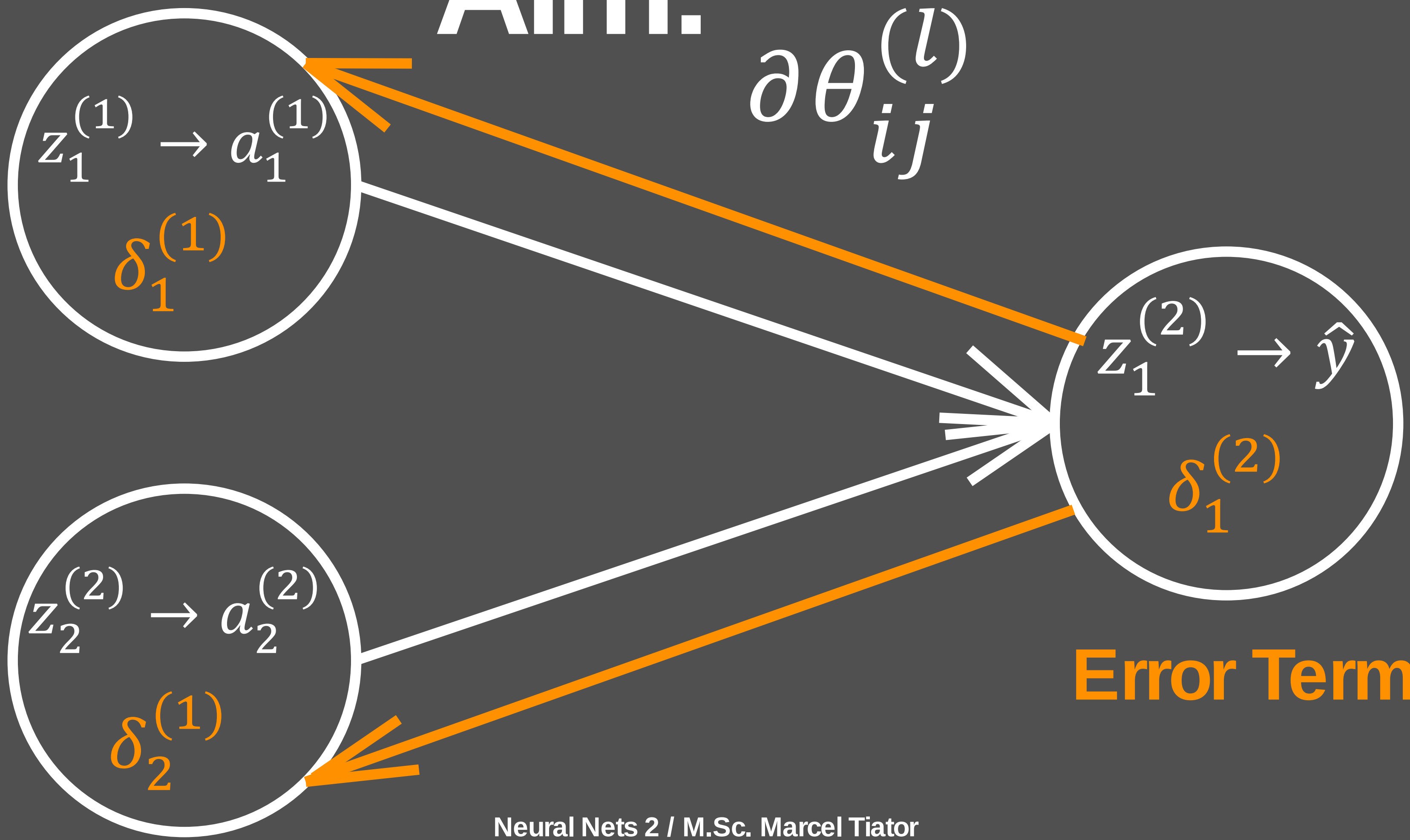


Propagate the error back through the network



Backpropagation

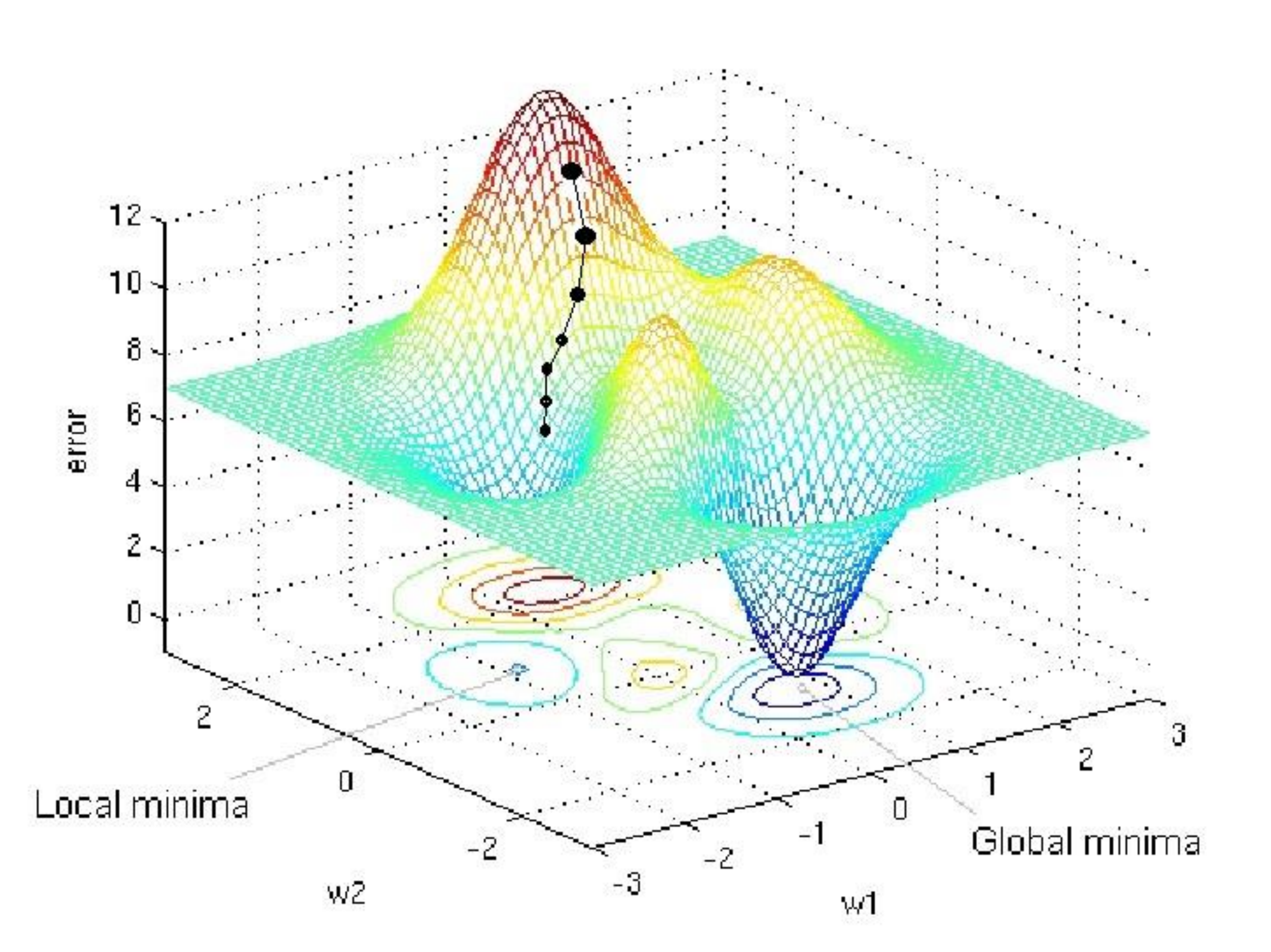
Aim: $\dfrac{\partial E}{\partial \theta_{ij}^{(l)}}$



Paramterupdate:

$$\theta_{ij}^{(l)} := \theta_{ij}^{(l)} - \alpha \frac{\partial E}{\partial \theta_{ij}^{(l)}}$$

(Remember Gradient Descent)



12
10
8
6
4
2
0
error
2
0
-2
w2
-3
-2
-1
0
1
2
3
w1
Local minima
Global minima

Number of weights in MLP

L: **Number of layer**

s_l: **Number of neuron in layer** l

$$\#weights = \sum_{l=1}^{L-1} s_{l+1} * (s_l + 1)$$

Bias-Unit (x_0, $a_0^{(l)}$)

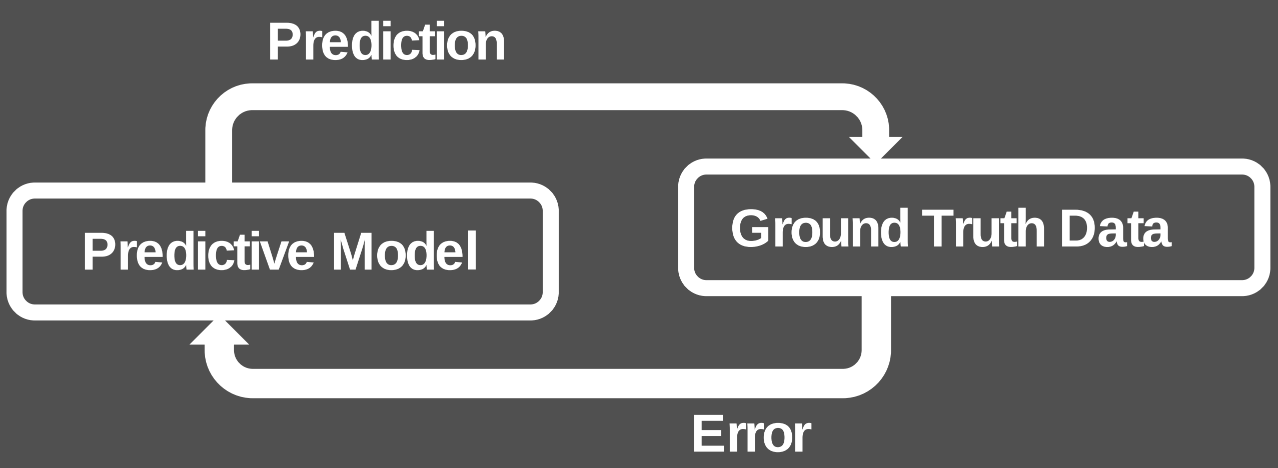
Given a MLP with following architecture: [3,4,4,3]

Calculate the $\#weights$ of the MLP?

$$\#weights = \sum_{l=1}^{L-1} s_{l+1} * (s_l + 1)$$

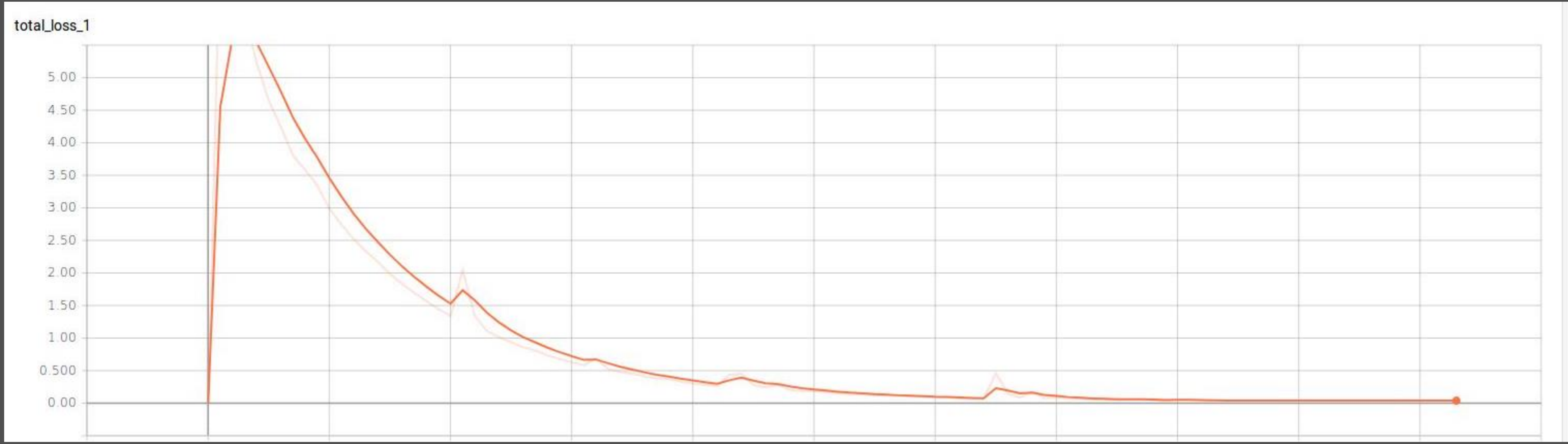
$$\#weights = \left(4 * (3 + 1)\right) + \left(4 * (4 + 1)\right) + \left(3 * (4 + 1)\right)$$

$$\#weights = 16 + 20 + 15 = 51$$



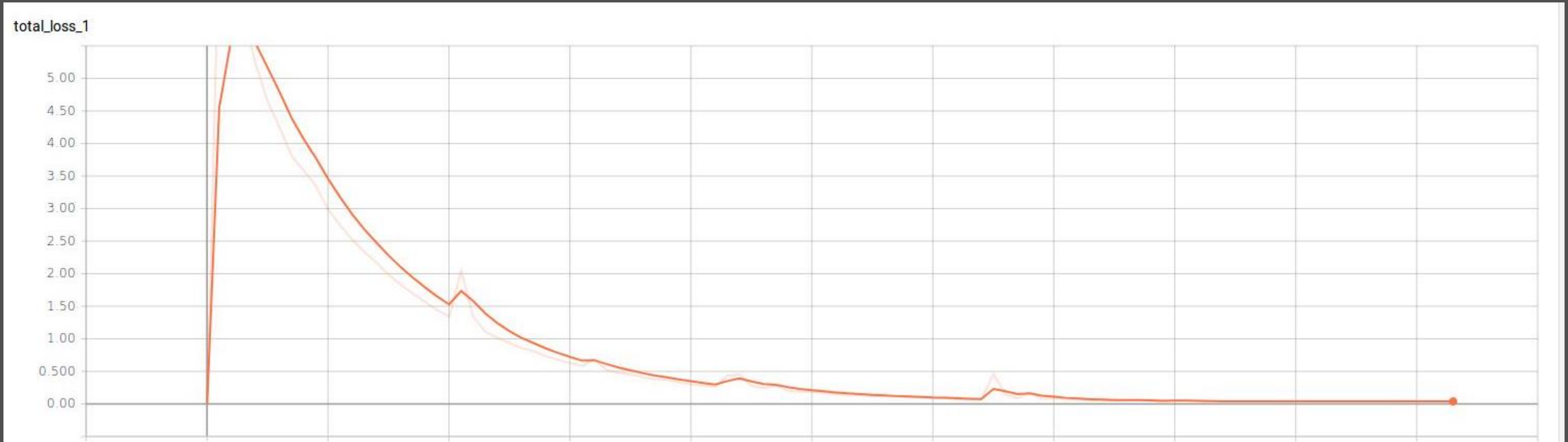
Learning Curves

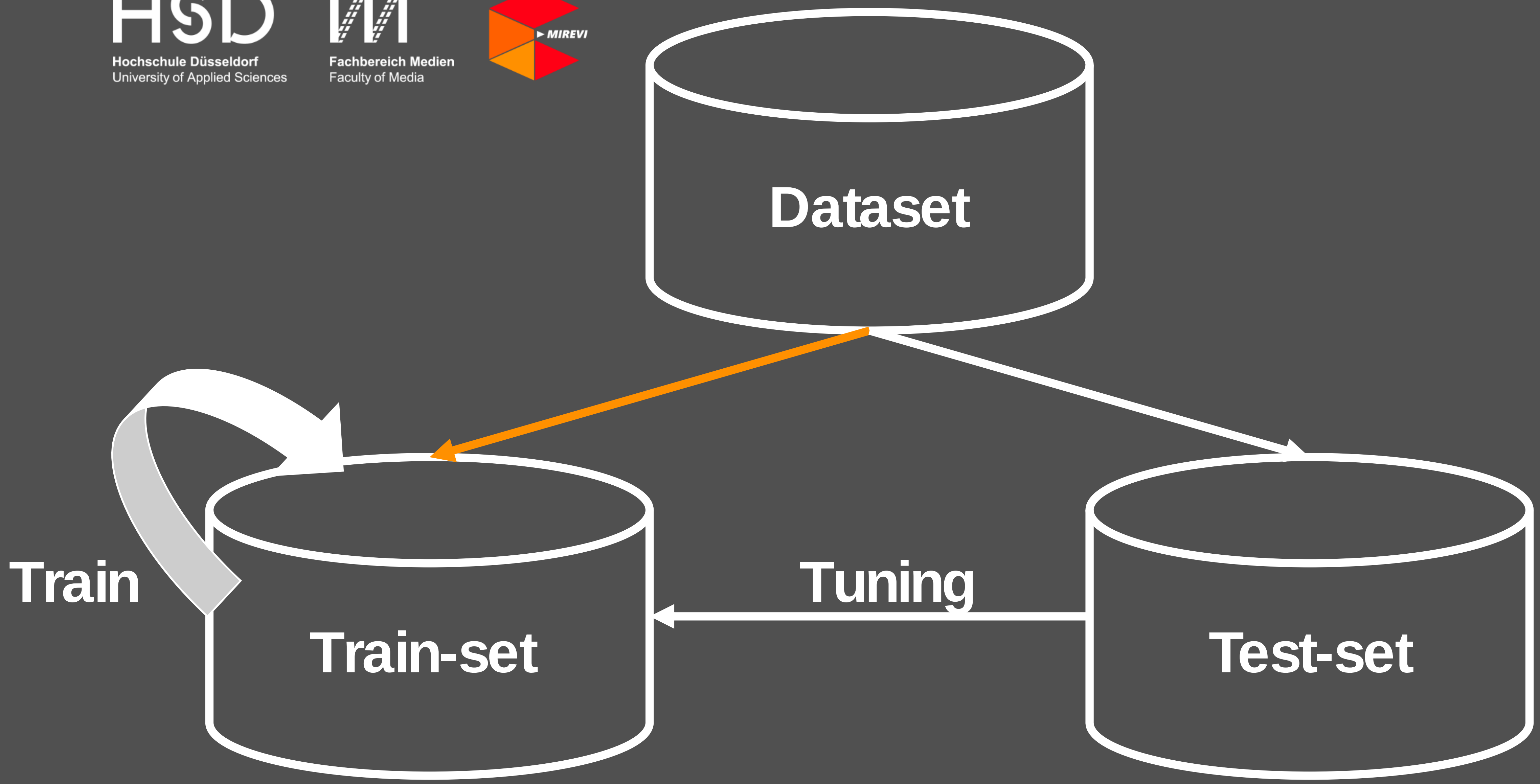
- x-Axis: Training Step
- y-Axis: Error/Loss

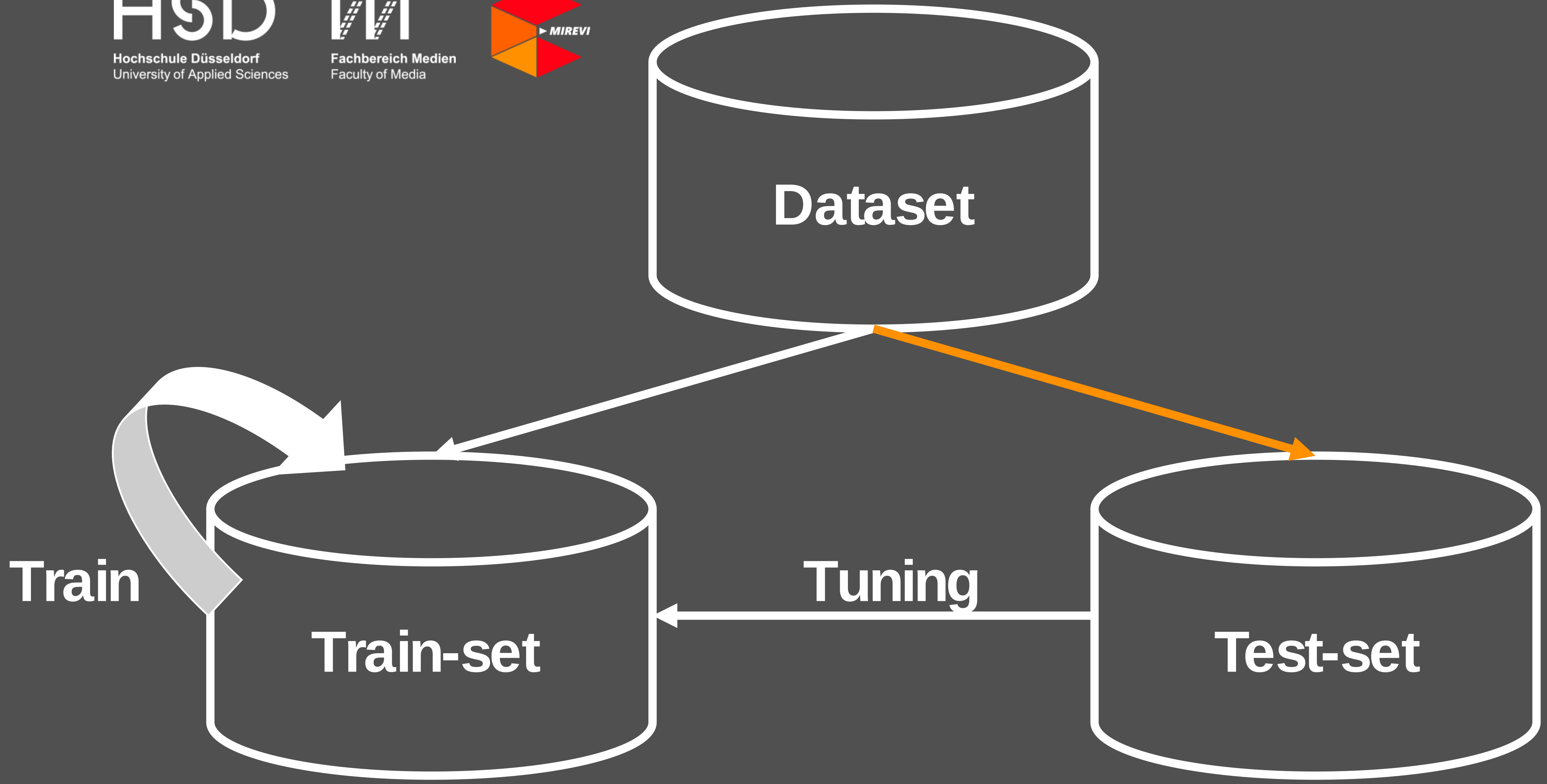


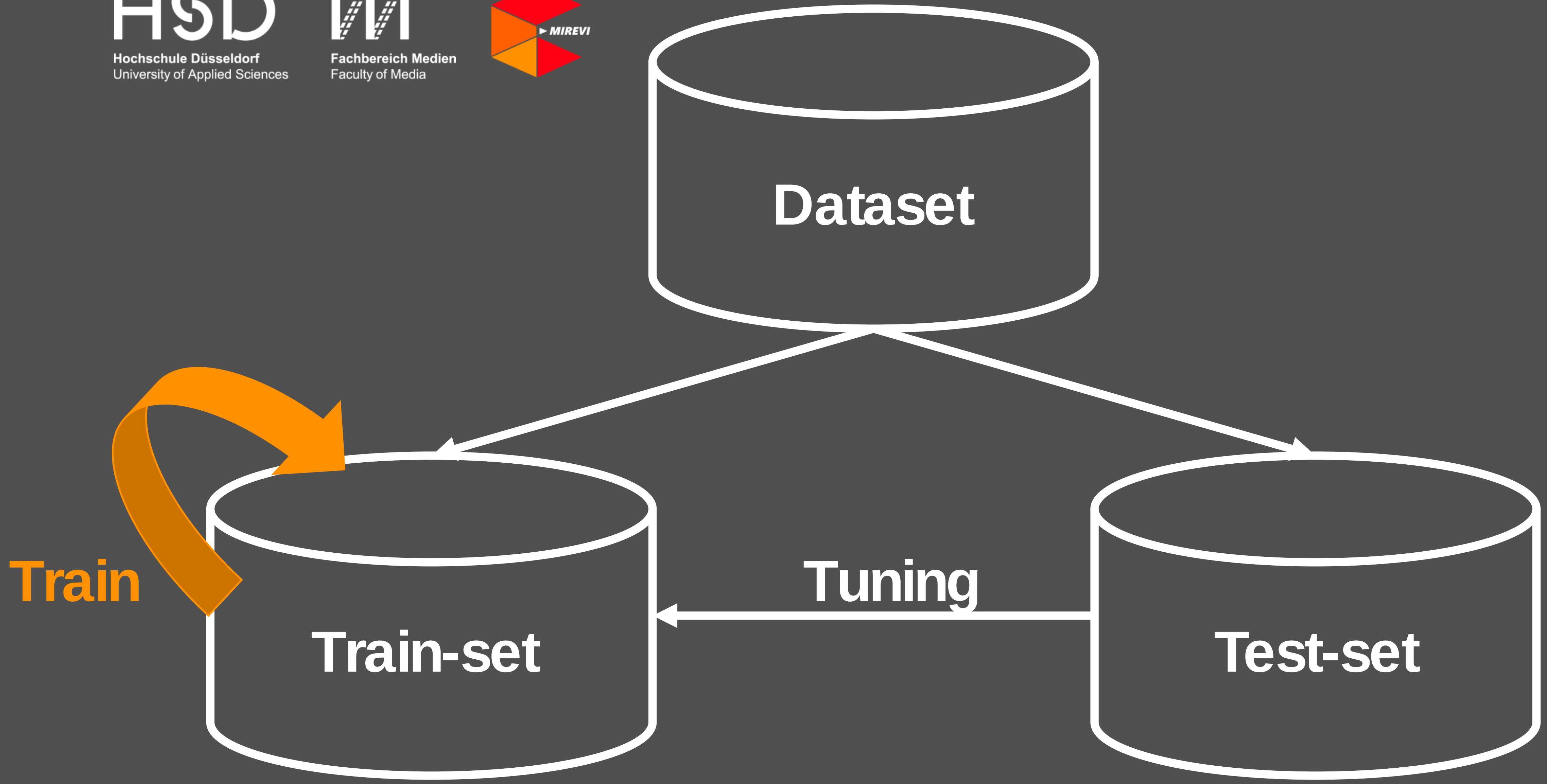
Rate performance of neural net

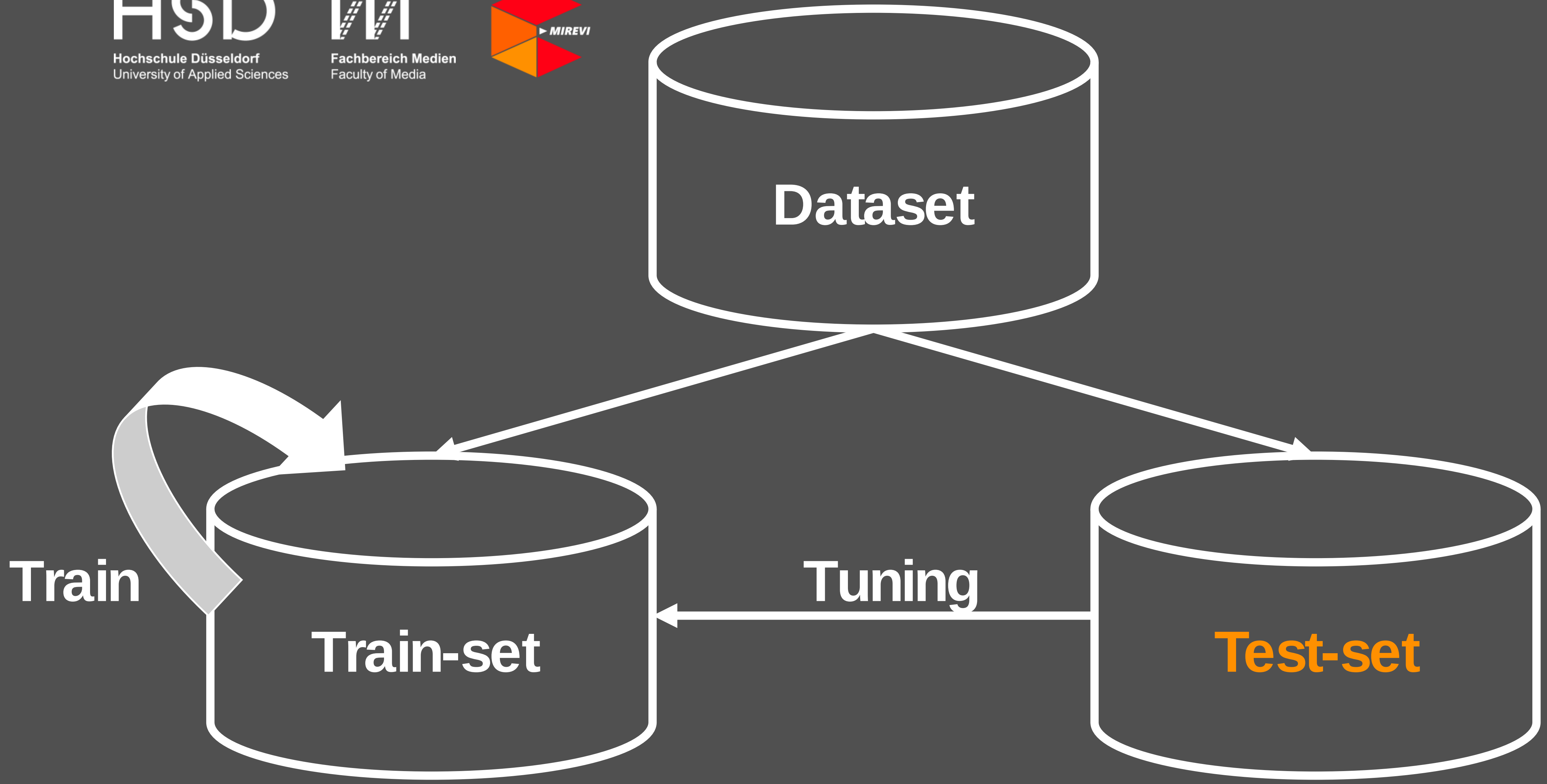
- High bias / underfitting
- High variance / overfitting

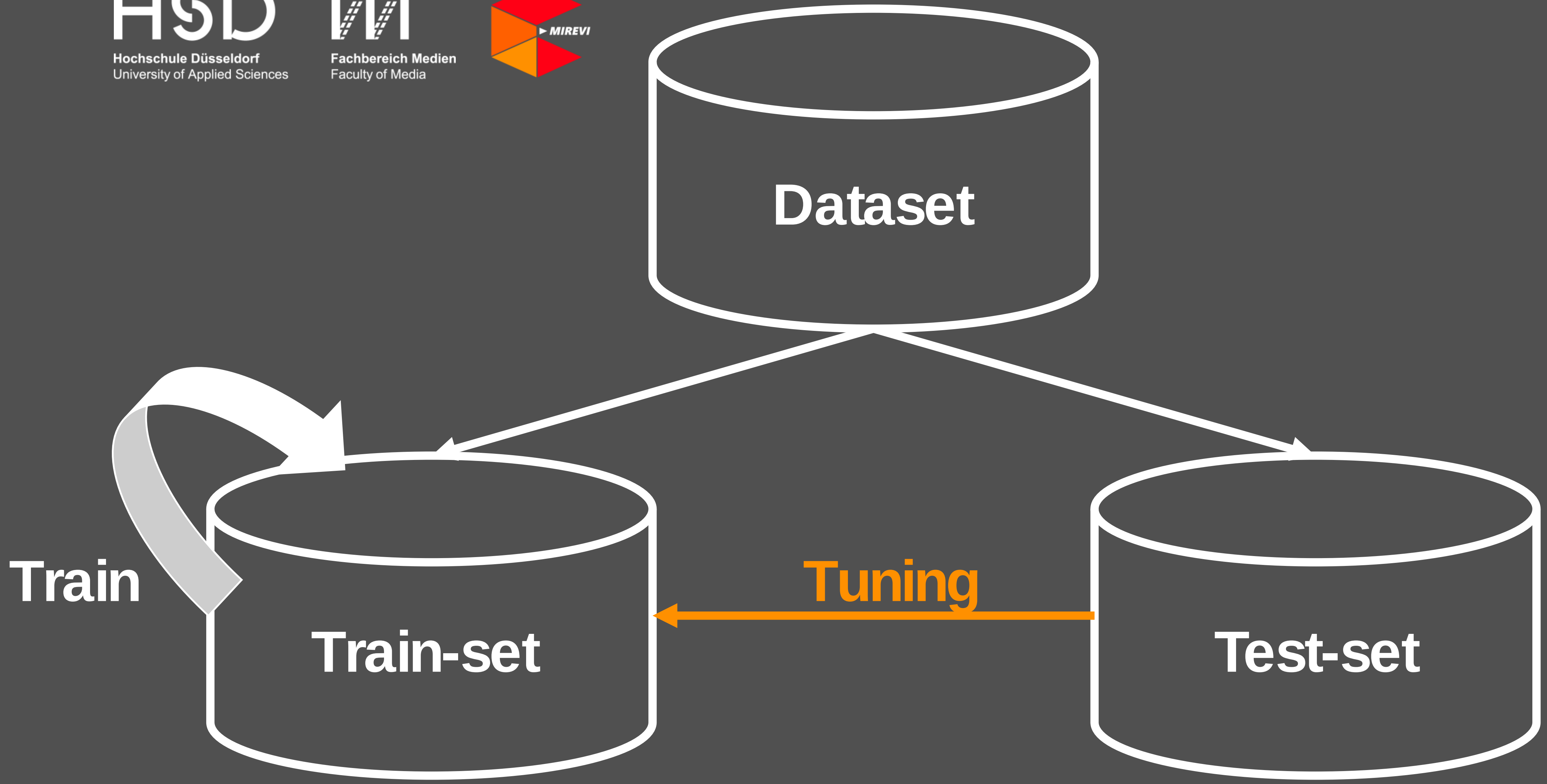



Dataset
Train
Train-set
Tuning
Test-set


Dataset
Train
Train-set
Tuning
Test-set

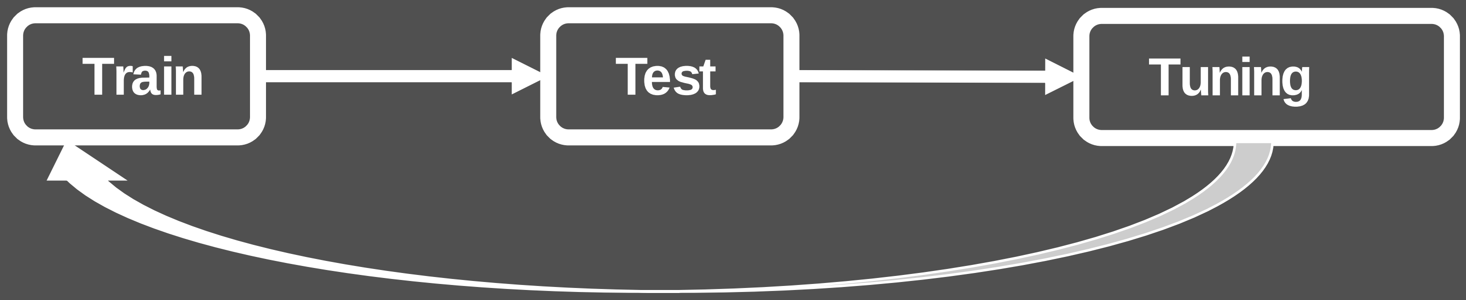

Dataset
Train
Train-set
Tuning
Test-set


Dataset
Train
Train-set
Tuning
Test-set


Dataset
Train
Train-set
Tuning
Test-set

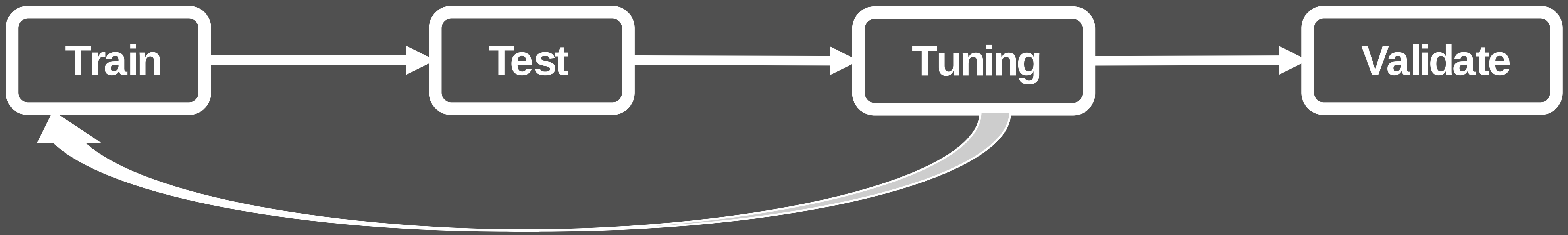
Cross Validation

- Tune your network with test-set performance (e.g. accuracy)
- Attention: Overfitting of test-set is possible
- Try 80% of data as train-set



Cross Validation

- Real Pro‘s use an additional validation-set
- Prevent overfitting
- Validation-set is a third seperate dataset
- Try 60% Train, 20% Test, 20% Validation



High Bias / Underfitting

- High Training Error & High Test Error
- Complexity of neural net is too low (Too less weights)
- Add layer or units (neurons in layer)
- If possible: add more features of data

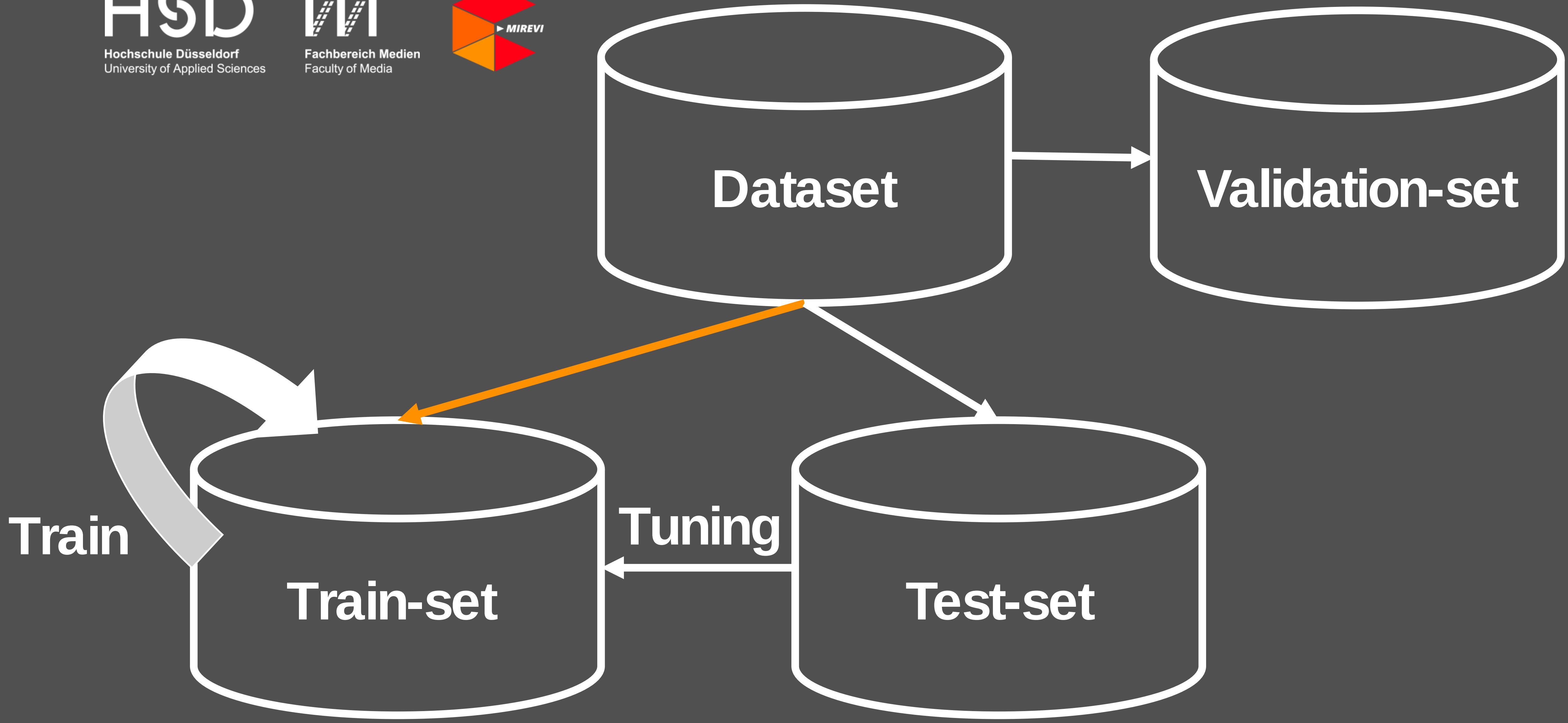
High Bias / Underfitting

- Create polynomial features
  - Add column with ${x_1}^2, {x_2}^2, x_1 x_2$, ...
- More trainings-examples will have no significant effect
- Reduce regularization

High Variance / Overfitting

- Neural net has learned the noise of the data
- Lower complexity
- More training examples can help
- Use regularization

Hochschule Düsseldorf
University of Applied Sciences
Fachbereich Medien
Faculty of Media



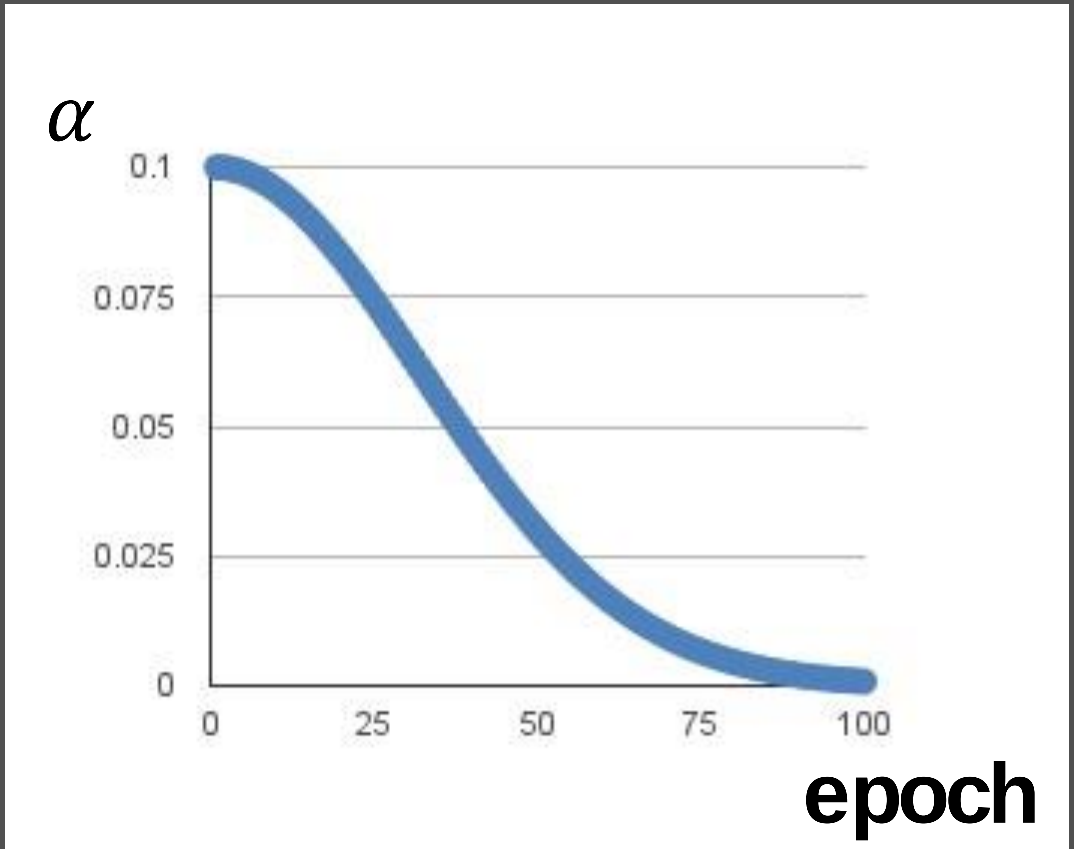
Regularization:

- Try to keep the values of the weights small
- λ: Regularization parameter
- L1-Regularization: $E = f(\hat{y}, y) + \frac{\lambda}{m}\sum|\theta|$
- L2-Regularization: $E = f(\hat{y}, y) + \frac{\lambda}{m}\sum\theta^2$

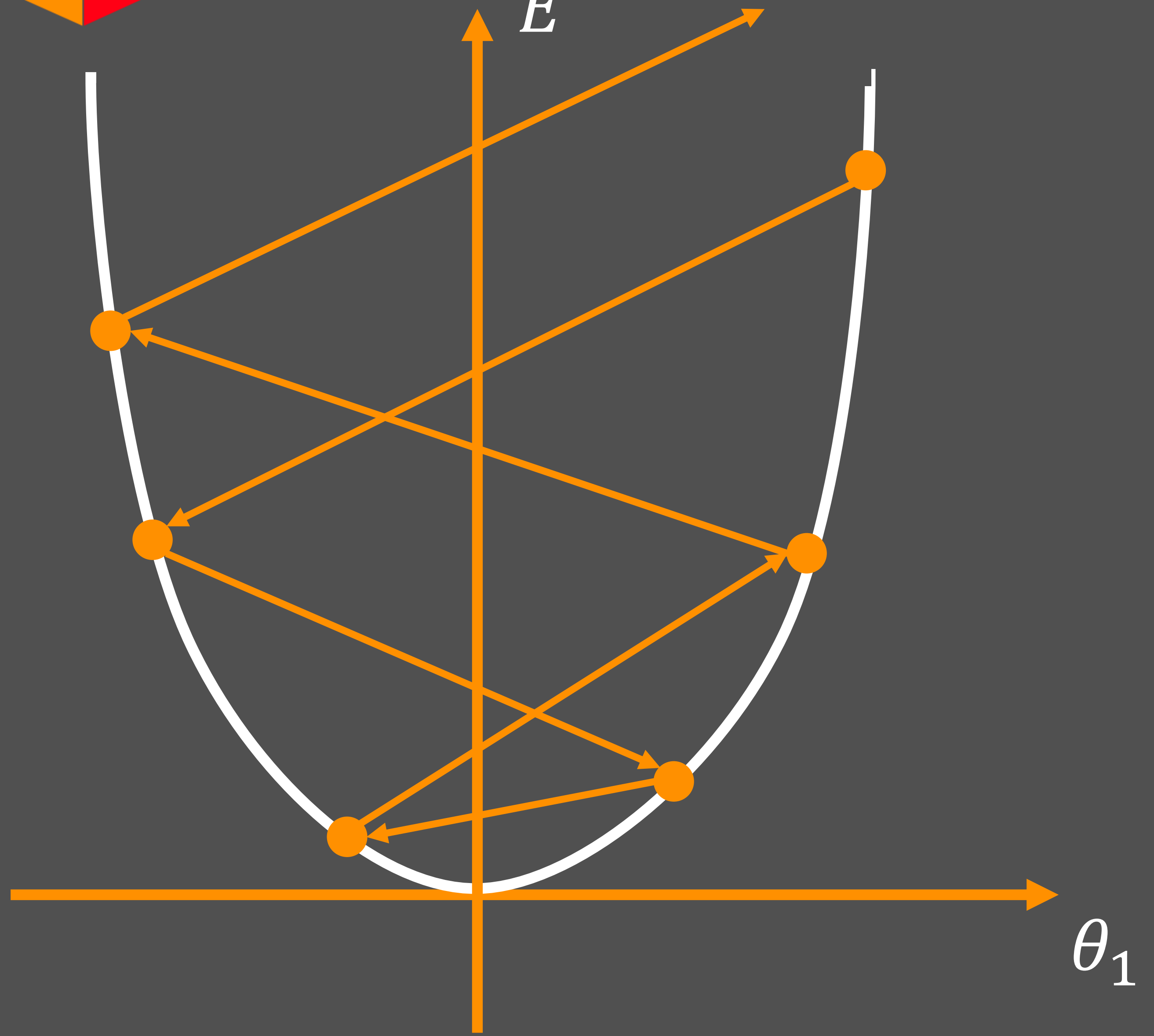
Please scale your data!

Learning Rate Decay (Learning Rate Schedule)

$$\alpha_n = \frac{\alpha}{1 + decay * epoch}$$

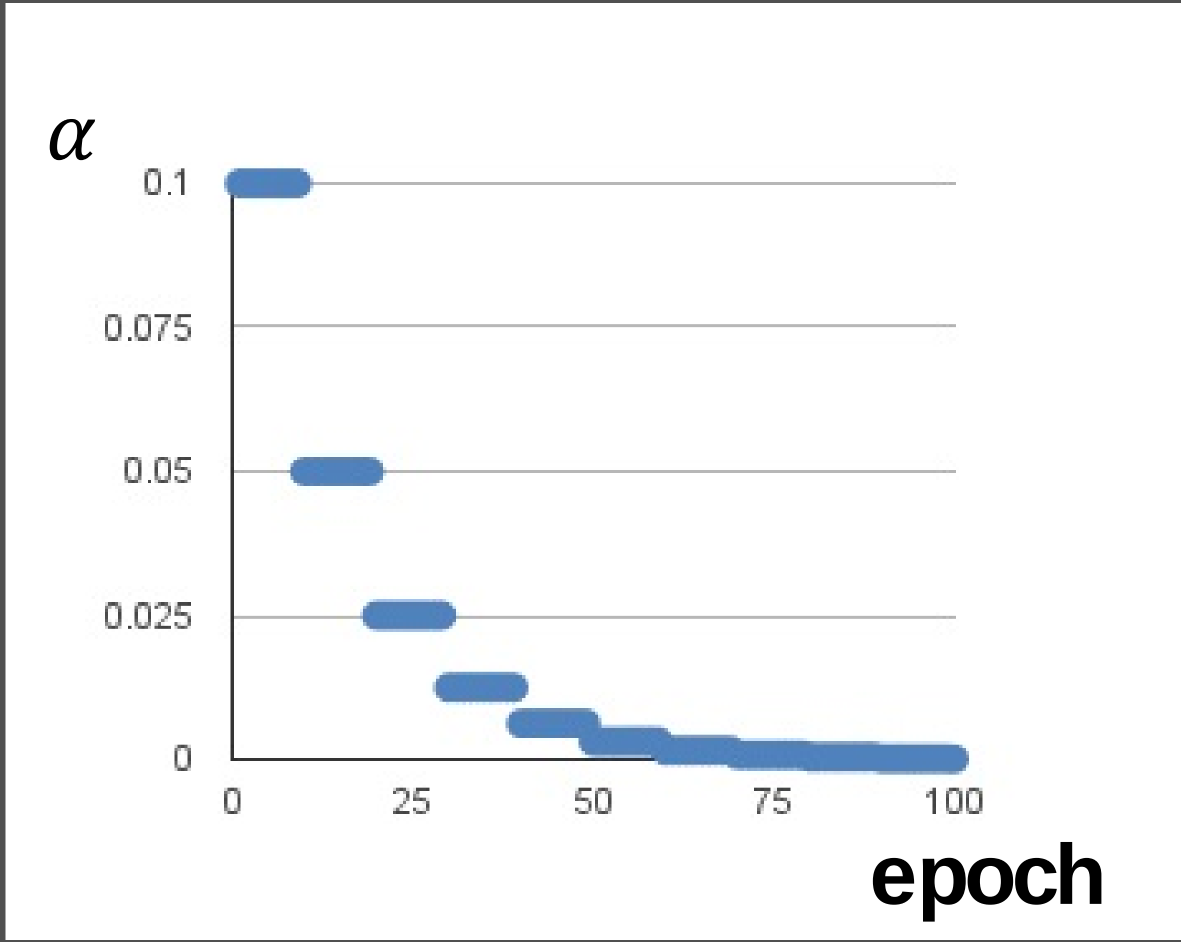


Prevent this with learning rate schedule



Drop Based Learning Rate Schedule

e.g. Half the learning rate every 10th epoch

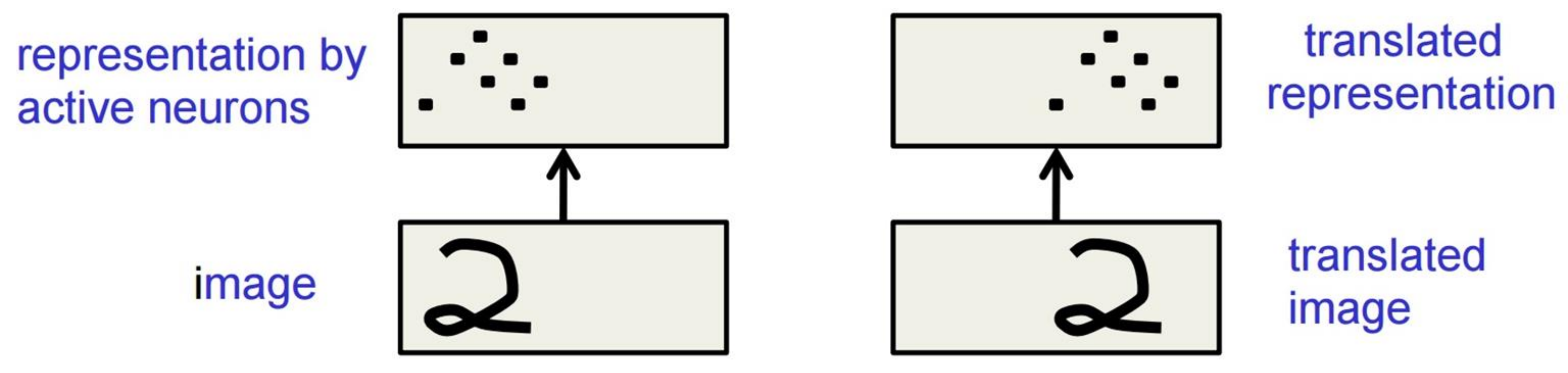


Hochschule Düsseldorf
University of Applied Sciences
Fachbereich Medien
Faculty of Media

Convolutional Neural Nets (CNN)

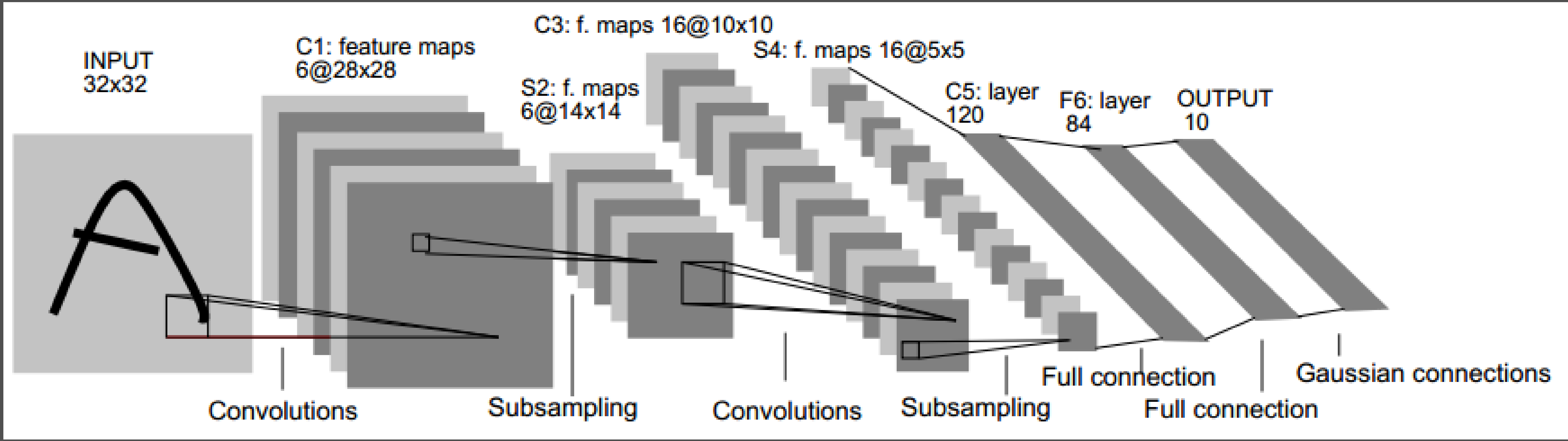
Disadvantage of using MLP in image recognition

- You can fool them by translating the object



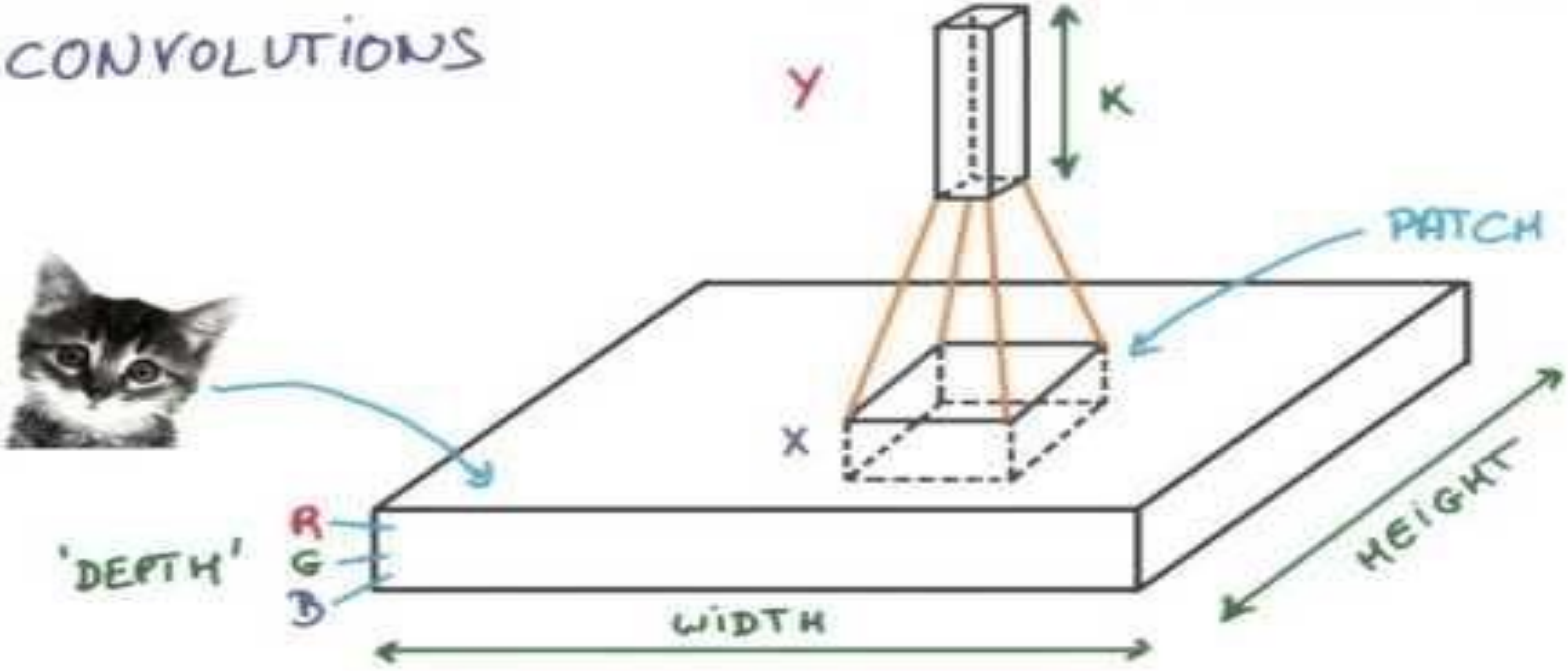
Convolutional Neural Nets (CNN‘s)

- 2D CNN‘s assume images as input
- They can learn transformational invariance
- Different architecture than MLP
- Trained with Backpropagation


INPUT
32x32
C1: feature maps
6@28x28
S2: f. maps
6@14x14
C3: f. maps 16@10x10
S4: f. maps 16@5x5
C5: layer
120
F6: layer
84
OUTPUT
10
Convolutions
Subsampling
Convolutions
Subsampling
Full connection
Full connection
Gaussian connections

Architectural elements of a CNN

- Convolutional Layer
- Max-Pooling Layer
- Fully Connected Layer


CONVOLUTIONS
Y
K
PATCH
X
R
G
B
'DEPTH'
WIDTH
HEIGHT

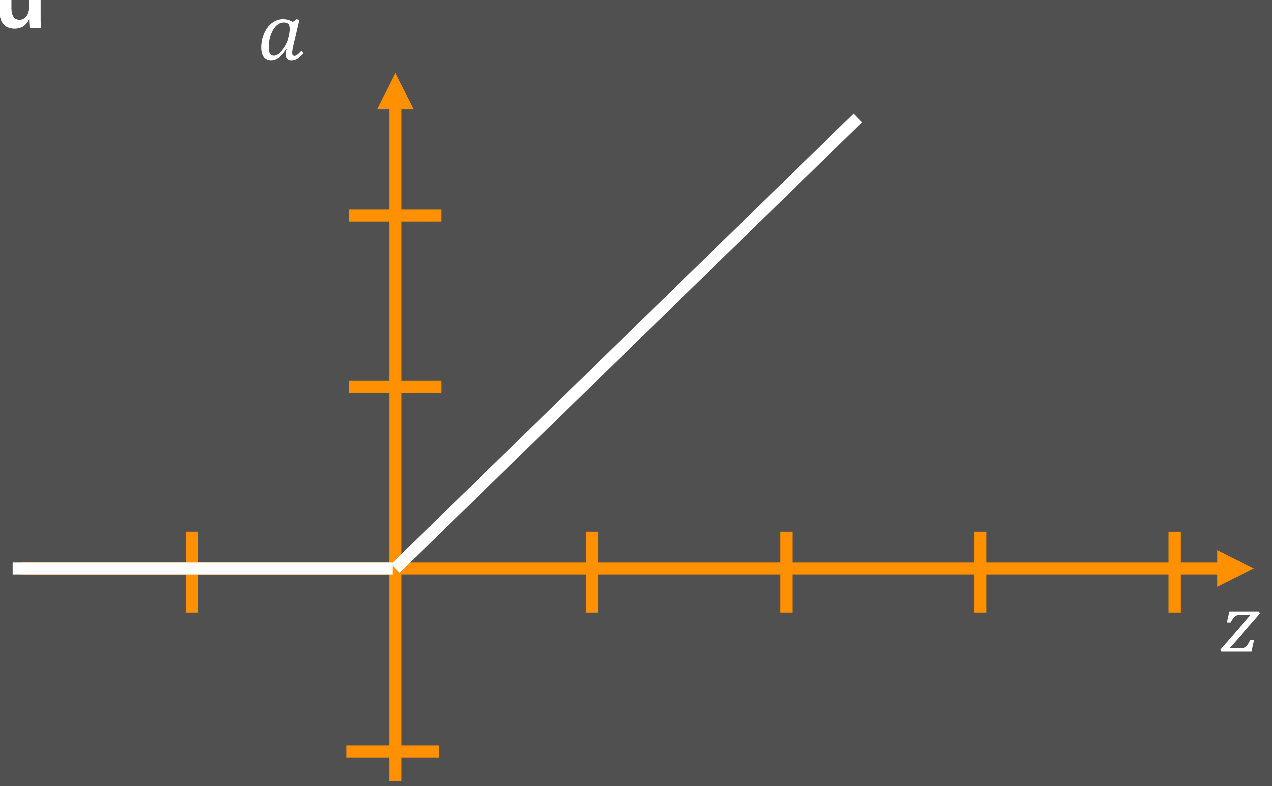
Convolutional Layer

- Apply image filters
- Feature detectors in combination with activation
- Special parameters:
  - Stride
  - Padding

Hochschule Düsseldorf
University of Applied Sciences
Fachbereich Medien
Faculty of Media

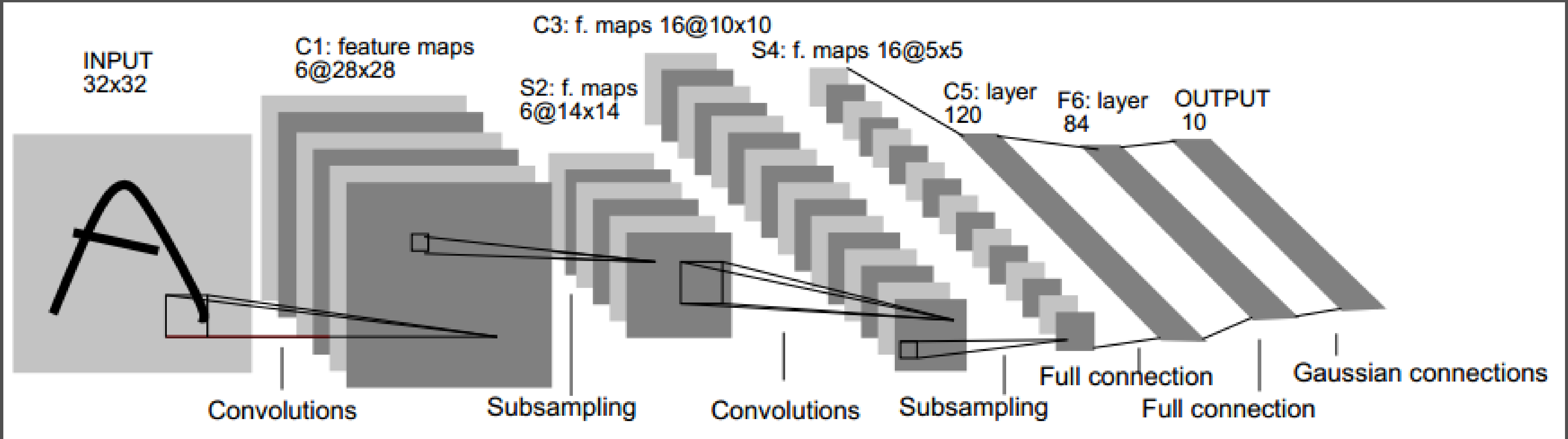
Activation after convolution

- We will apply an activation function after convolution
- Modern CNN‘s use ReLu



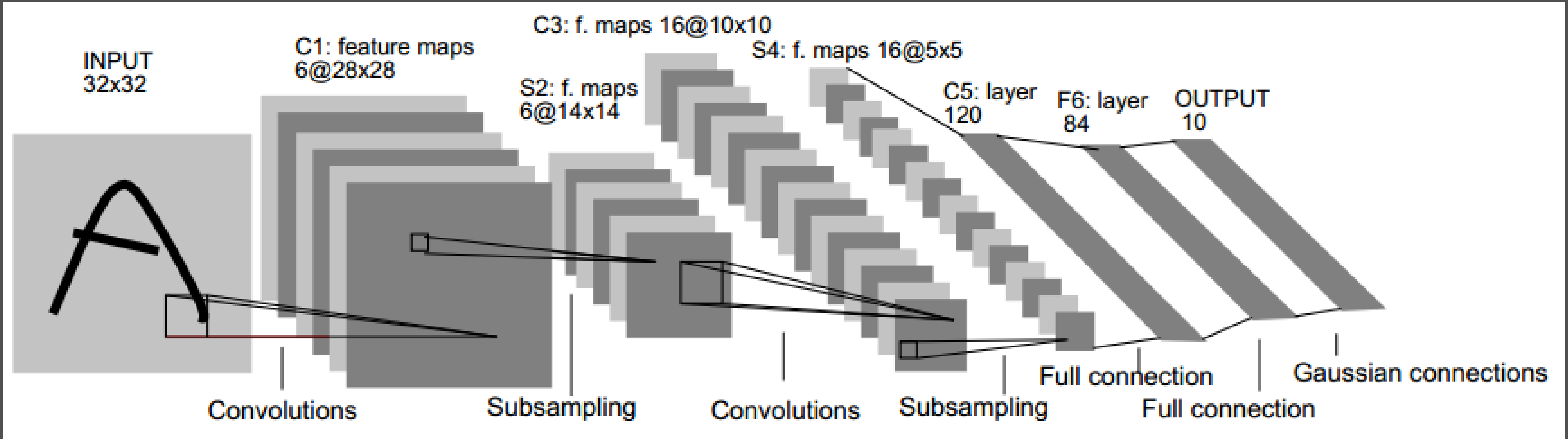
Max-Pooling Layer

- Dimensionalty reduction
- Pass the greatest value of a field to the next layer



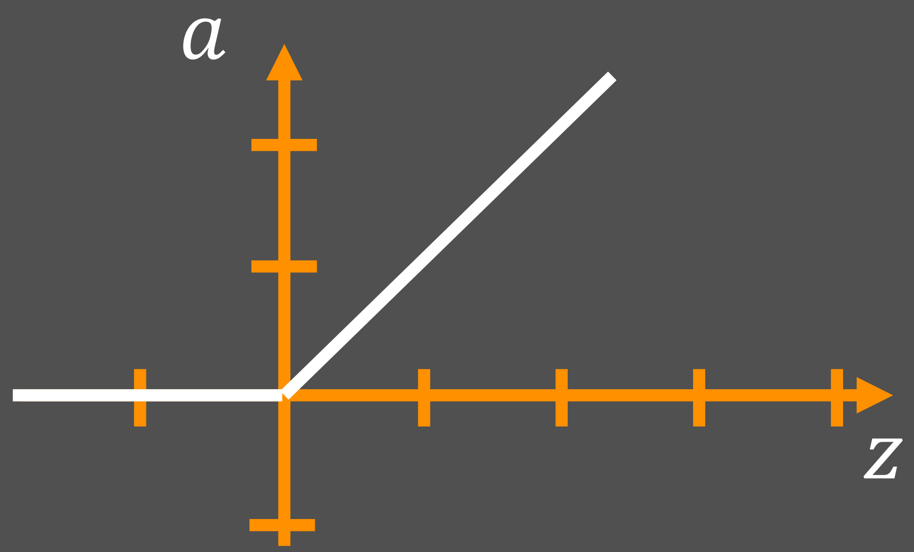
Fully Connected Layer

- Flatten the 2D (in case of image) to 1D layer
- Use fully connected layer up to the output



Hochschule Düsseldorf
University of Applied Sciences
Fachbereich Medien
Faculty of Media
MIREVI

Sometimes useful: Normalization Layer



Star Recognition





Neural Nets 2 / M.Sc. Marcel Tiator

Hochschule Düsseldorf
University of Applied Sciences
Fachbereich Medien
Faculty of Media

Todays exercise:

- Complete the code to train a CNN
- Real world image classification problem
- Given a face of a person: classify star

Sources:

- cnn.py
- cnn_train.py
- cnn_eval.py
- cnn_predict_cv.py
- download_image_data.py
- write_tfrecord.py
- read_tfrecord.py
- utils.py

Hochschule Düsseldorf
University of Applied Sciences

Fachbereich Medien
Faculty of Media

MIREVI

How to get image data?

- Execute the steps in the README.md of this project
- Execute download_data.py
- Faces will be extracted through haarcascade detection (OpenCV)
- Already done, if you download the data from the Nextcloud

utils.py (First Step):

- Mainly image processing
- Face Extraction
- Grayscale
- Scaling to quadratic size
- Normalization

Write .tfrecord file (will be extended soon, Second step)

- Execute the write_tfrecord.py script
- Special binary-dataset will be written to disk
- We will directly apply image preprocessing
  - Execute read_tfrecord.py to validate the contents of the .tfrecord file (Third Step)
- Execute read_tfrecord.py to validate the contents of the .tfrecord file

cnn.py – Convolutional Neural Network

- Definition of the architecture
- Loss function
- Definition of training and optimizer
- Script not executable

cnn_train.py – training of the CNN (Fourth Step)

- Executable script
- Tuning of hyperparameters (e.g. learning rate, etc.) possible
- Read .tfrecord file
- Manages the training process

cnn_eval.py (Sixth Step)

- Executeable
- Possibility to validate the model with a test-set

cnn_predict.py (Seventh Step)

- Executeable
- Prediction of star with one image

Tasks:

- Image Processing in utils.py
- Write .tfrecord file with write_tfrecord.py
- Validate .tfrecord with read_tfrecord.py
  - Tipp: If you leave out preprocessing (without float conversion and normalization) of image you should see a colored image of the face of a star
- Prepare training of CNN in cnn_train.py
- Evaluate the result with cnn_eval.py
- Classify one image with cnn_predict_cv.py
- Add train, test, validation split functionality

- **Data (if not yet downloaded):** https://nextcloud.mirevi.medien.hs-duesseldorf.de/index.php/s/kPXwJiac7vTQVeu
- **Pretrained Weights:** https://nextcloud.mirevi.medien.hs-duesseldorf.de/index.php/s/L6Y6tnD3PpANKmr
- **Repository:** https://github.com/mati3230/modalg181
- **Read:** https://www.tensorflow.org/tutorials/deep_cnn