TECHNISCHE UNIVERSITÄT DRESDEN
Fakultät für Elektro- und Informationstechnik, Professur für Grundlagen der Elektrotechnik und Elektronik

FliK Modul 2020

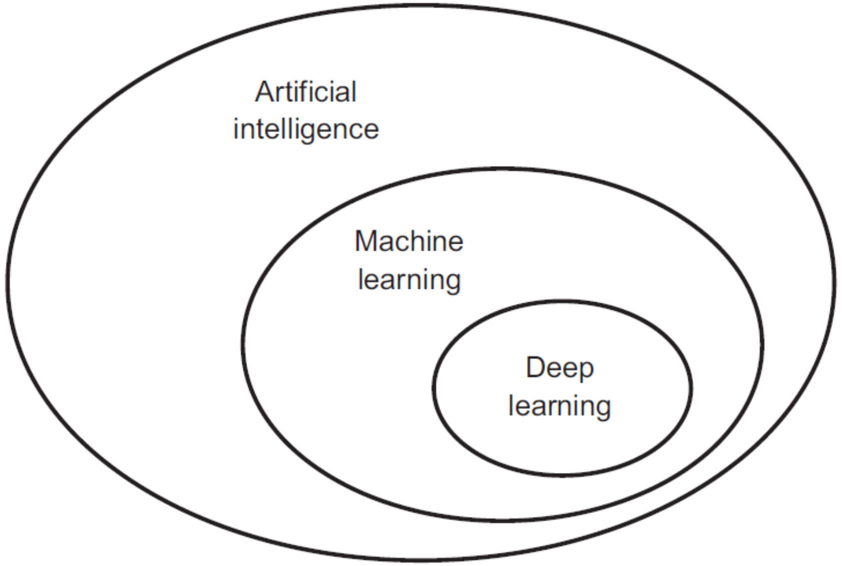
Deep Learning mit Keras
Neural Networks

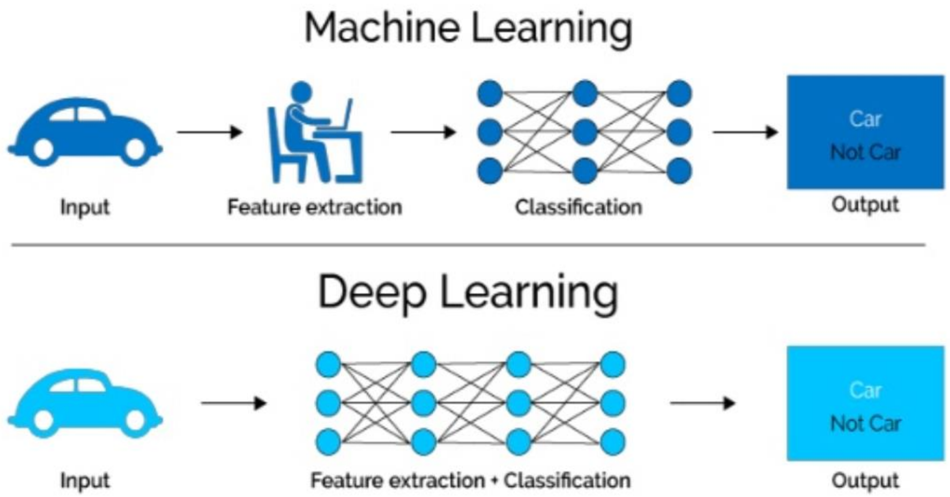
Steffen Seitz & Marvin Arnold

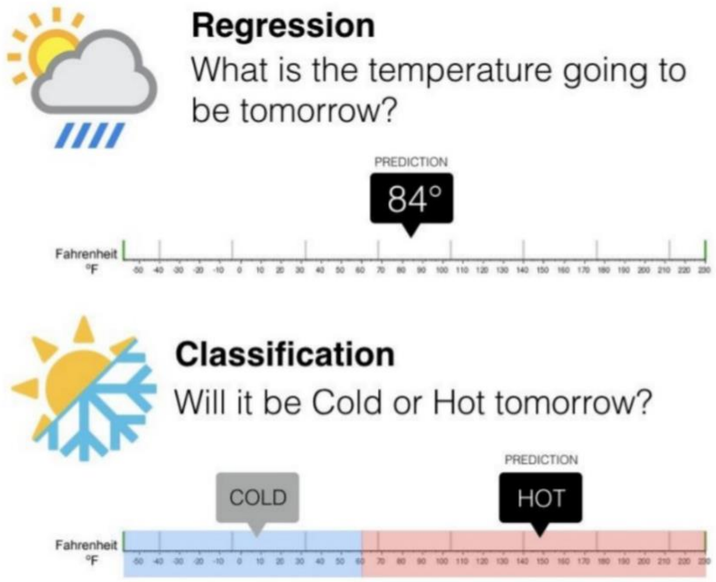
Prof. Ronald Tetzlaff

Dresden, 19-23.10.

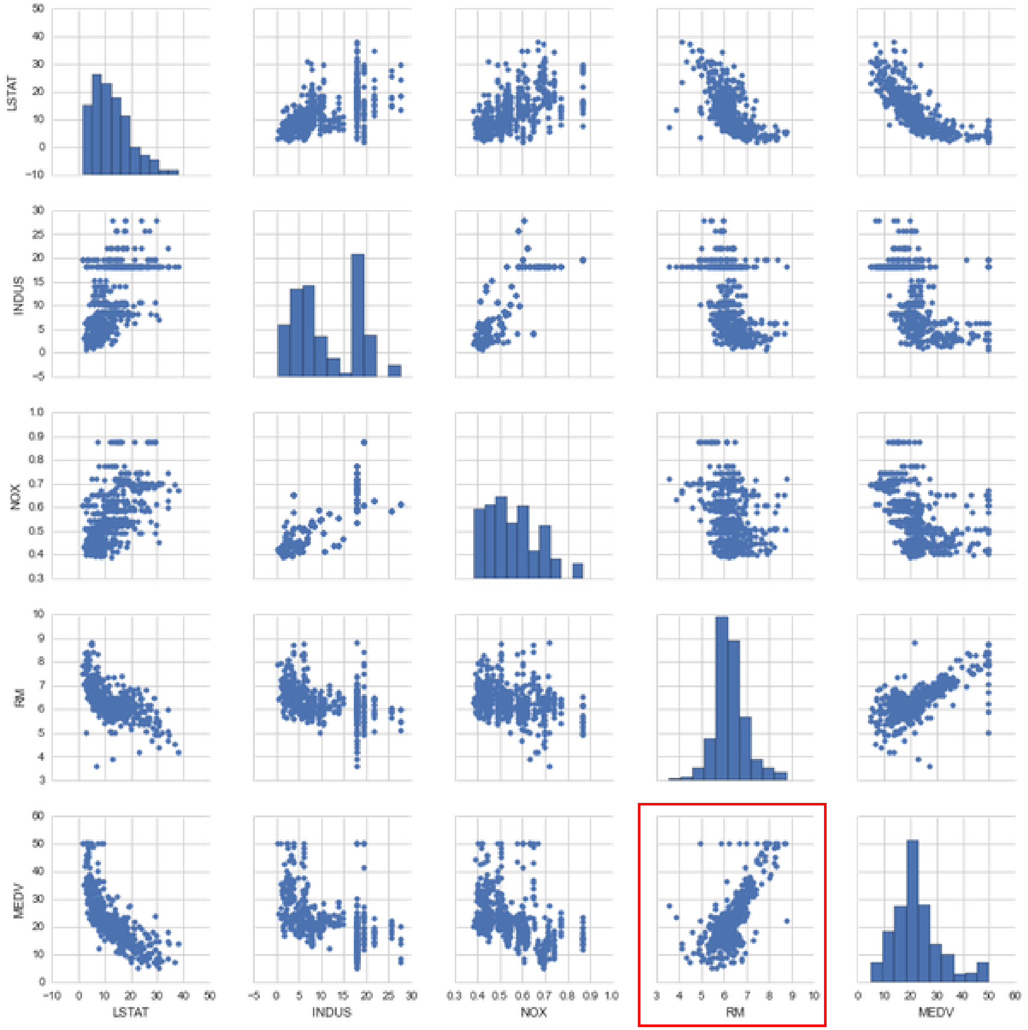
Neural Networks & Deep Learning



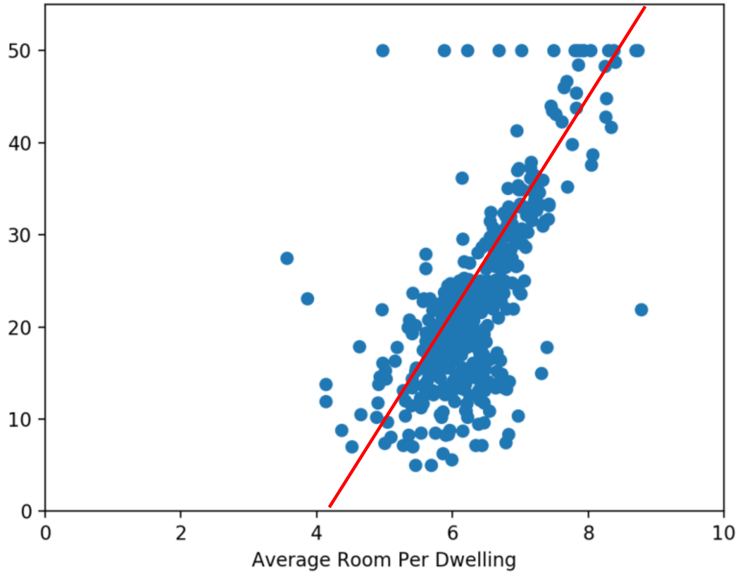




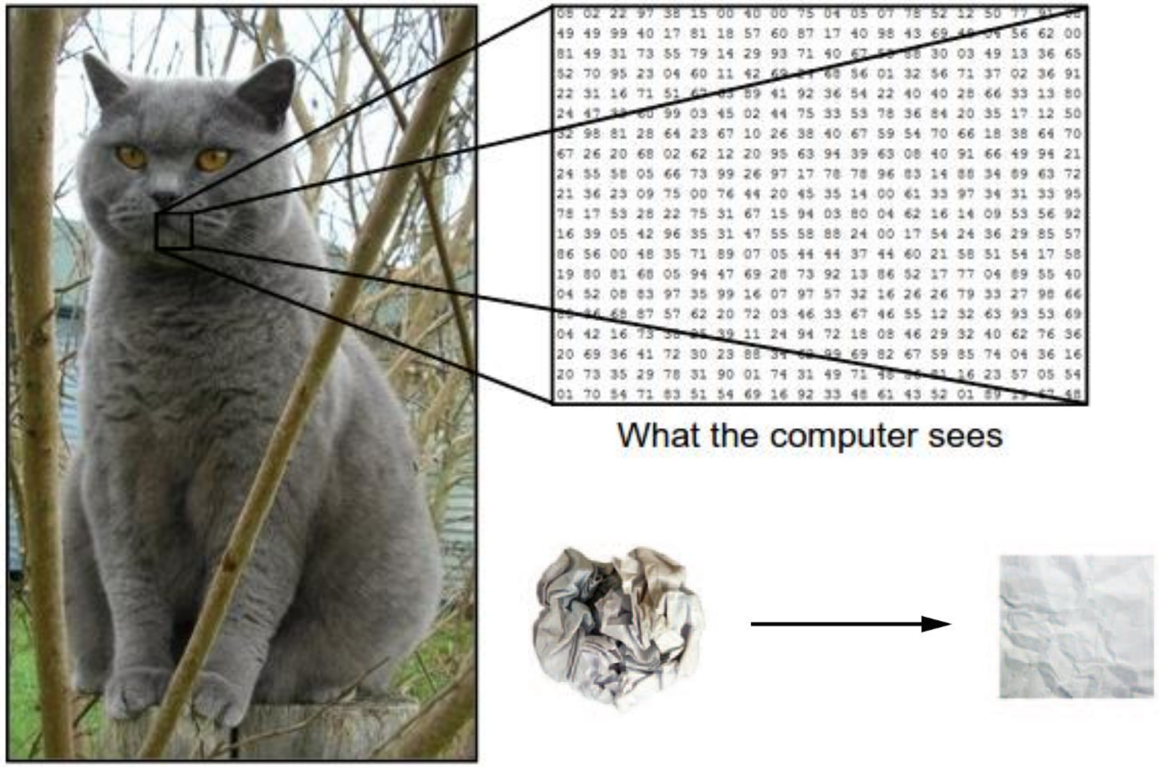
Regression: Boston Housing Dataset



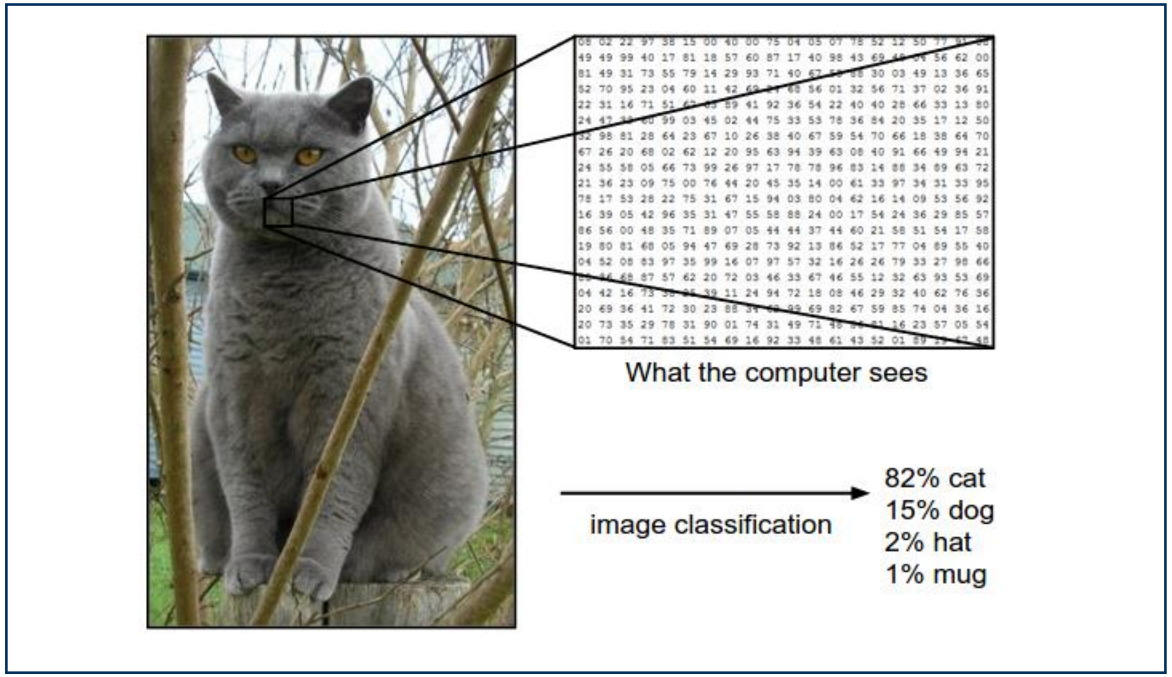
CRIM	Per capita crime rate by town
ZN	Proportion of residential land zoned for lots over 25,000 ft^2
INDUS	Proportion of nonretail business acres per town
CHAS	Charles River dummy variable (= 1 if tract bounds river; = 0 otherwise)
NOX	Nitric oxide concentration (parts per 10 million)
RM	Average number of rooms per dwelling
AGE	Proportion of owner-occupied units built prior to 1940
DIS	Weighted distances to five Boston employment centers
RAD	Index of accessibility to radial highways
TAX	Full-value property-tax rate per $10,000
PTRATIO	Pupil/teacher ratio by town
B	$1000(Bk - 0.63)^2$ where Bk is the proportion of blacks by town
LSTAT	% Lower status of the population
MEDV	Median value of owner-occupied homes in $1000s



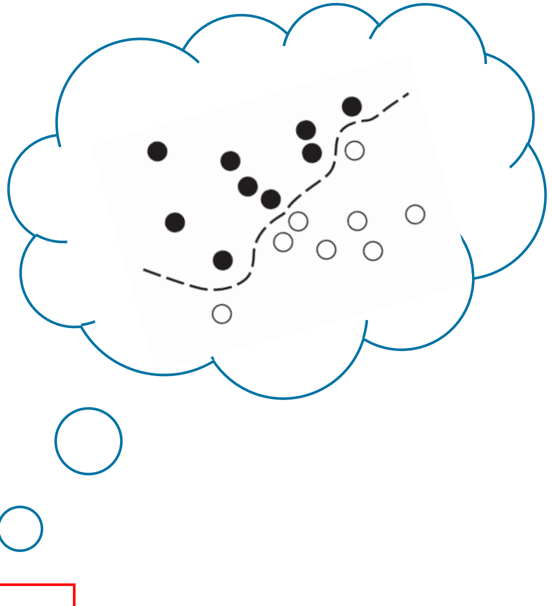
Classification:



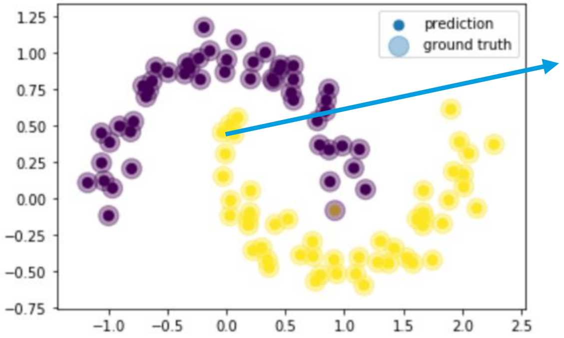
- Cat detection is a complex Task!
- Neural Networks try to unfold the complexity



Decision Boundery



Data



Vektor/Matrix

```
[ 0.0660451 ,  0.4392075 ],
[ 0.73663111, -0.39896339],
[-1.05692838,  0.2424558 ],
[-0.80216162,  0.20271838],
[-0.70641308,  0.77076218],
[ 0.2053884 ,  0.81875305],
[ 1.37804958, -0.44658032],
```

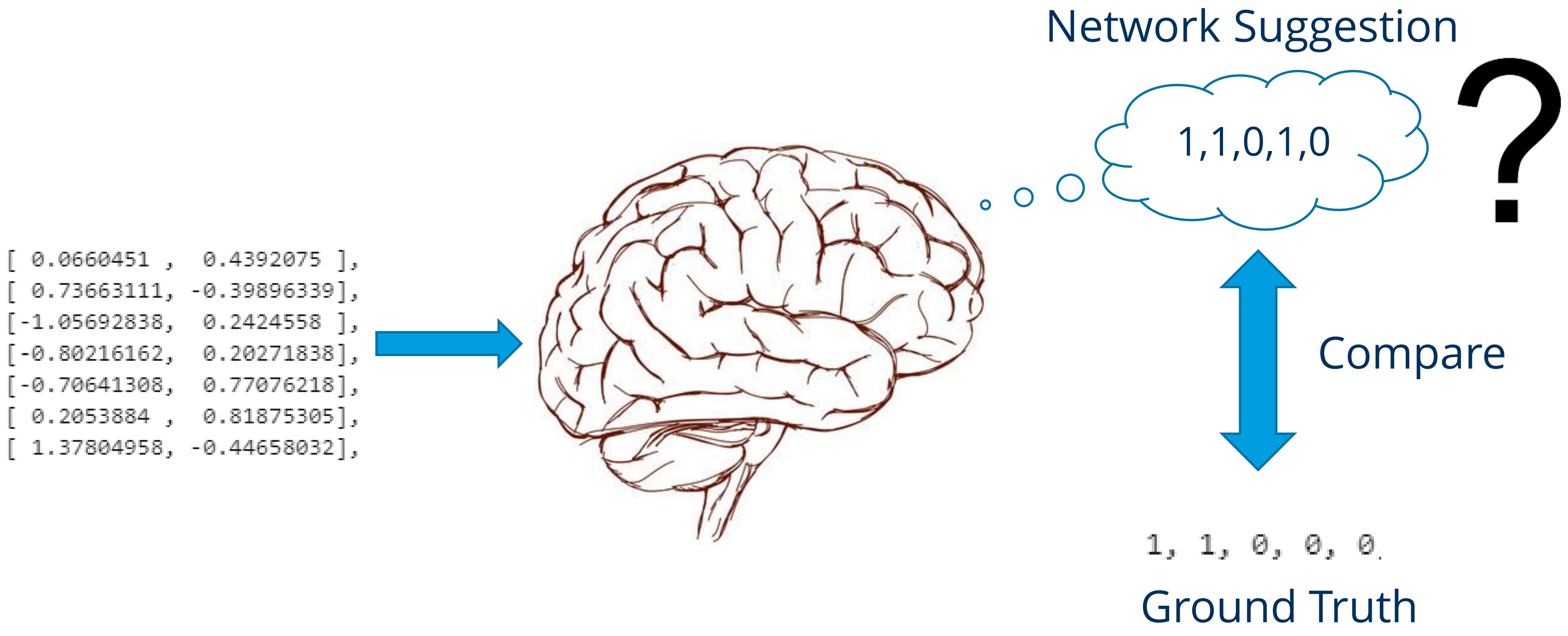
Network

Class decision

```
1, 1, 0, 0, 0, 0, 1,
```

Multilayer Perceptron
Loss functions

A loss functions is a grade **(Metric)** how good we have been doing our task.
Mathematicaly speaking, a metric is a **measure of „Distance“.**



Multilayer Perceptron
Loss functions

MSE (Naive approach)

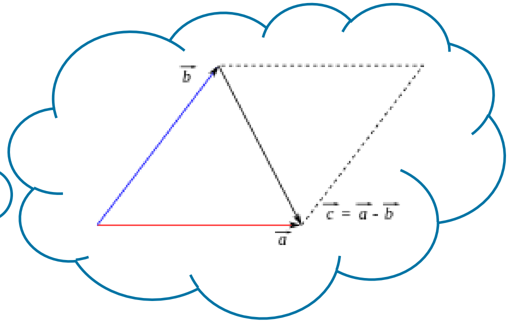
Computes the mean square Error which can defined as the **distance** of **two points** in a vector space.

Ground truth

$$\mathrm{MSE} = \boxed{\frac{1}{N}\sum_{i=1}^{N}}\boxed{(y_i} - \boxed{a})^2$$

Mean over all (possible) classes

Suggestion



Crossentropy

The (mean) cross entropy is a measure of **difference** between **two discrete** probability **distributions**. (created by Claude Shannon)

$$C = -\boxed{\frac{1}{N}\sum_{i=1}^{N}}[y_i \ln a + (1 - \boxed{y_i}) \ln(1 - \boxed{a})]$$

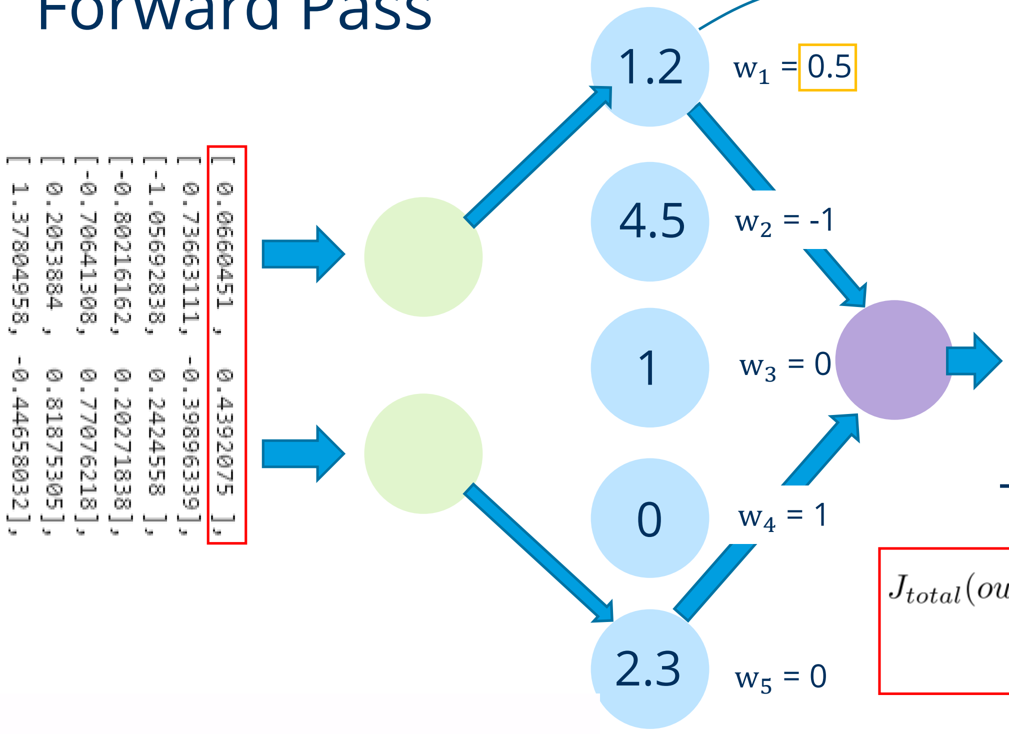
Mean over all (possible) classes

Ground truth

Suggestion

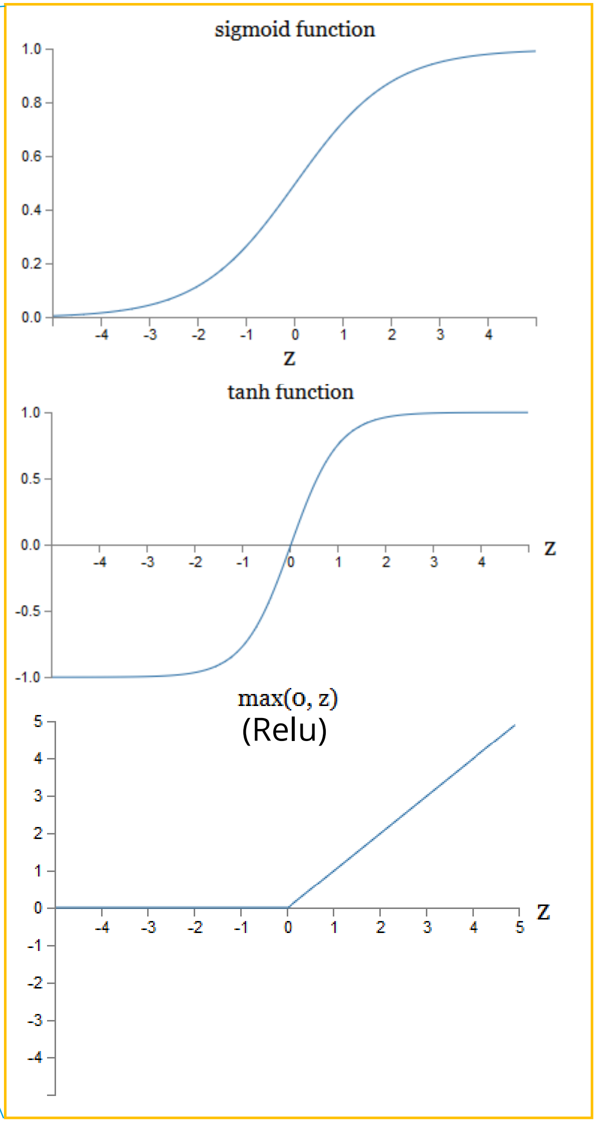
The naive approach of applying **MSE** for regression is ok. But (later) we will discuss a reason why classification **Crossentropy** is the way to go in **nearly every** Situation!

Multilayer Perceptron
Forward Pass



$$output = \boxed{\sigma}\left(\underbrace{\sum_{j=1} \omega_j x_j + b}_{z}\right)$$

$b = 0$

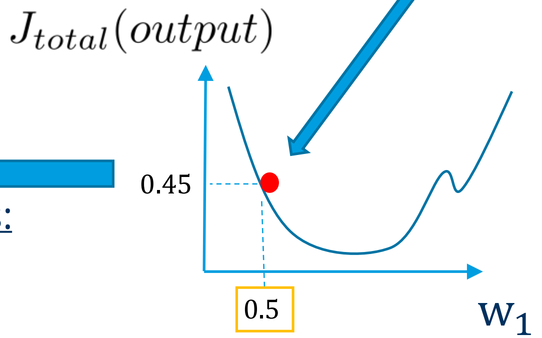


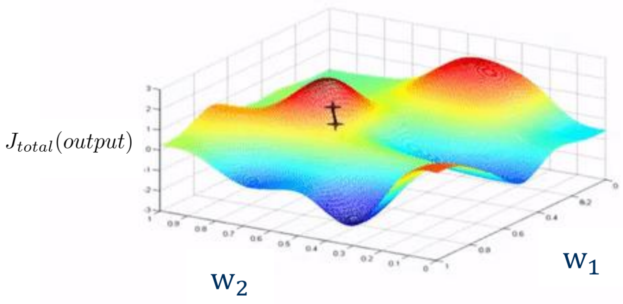
$output = \sigma(1.2 \times 0.5 - 4.5 \times 1)$
$= \mathbf{0.05}$

Target: $\boxed{1,}$ 1, 0, 0

$$J_{total}(output) = \frac{1}{2}(target - output)^2$$
$$= \frac{1}{2}(1 - 0.05)^2 = \mathbf{0.45125}$$

(MSE loss)





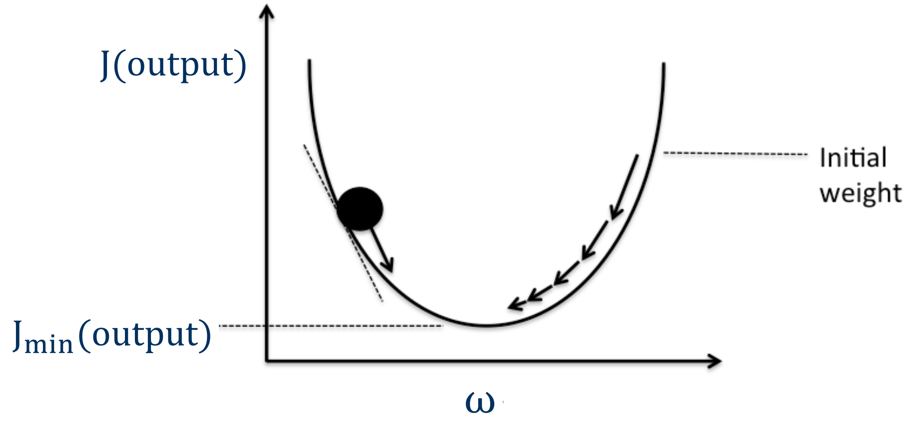
For all weights:
Sum over all training data

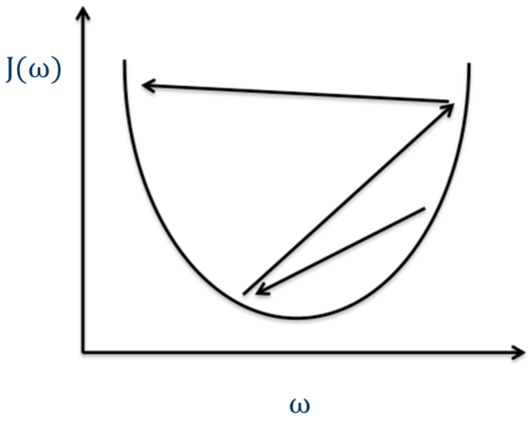
Multilayer Perceptron
Backward Pass

Gradien Descent update

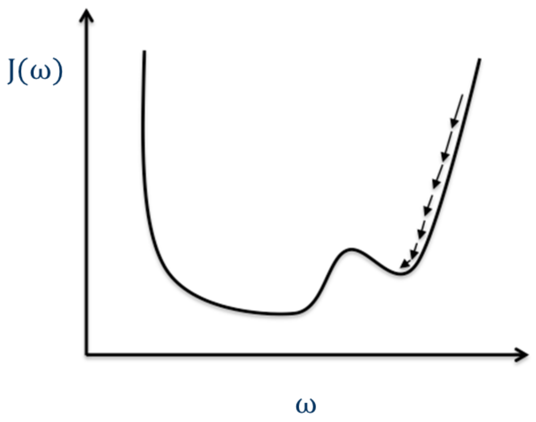
Update rule: $\boxed{w_1(new) = w_1(old) - \alpha \cdot \frac{\partial J_{total}(output)}{\partial w_1}}$

Learning rate





Large learning rate: Overshooting



Small learning rate: Many iterations until convergence and trapping in local minima

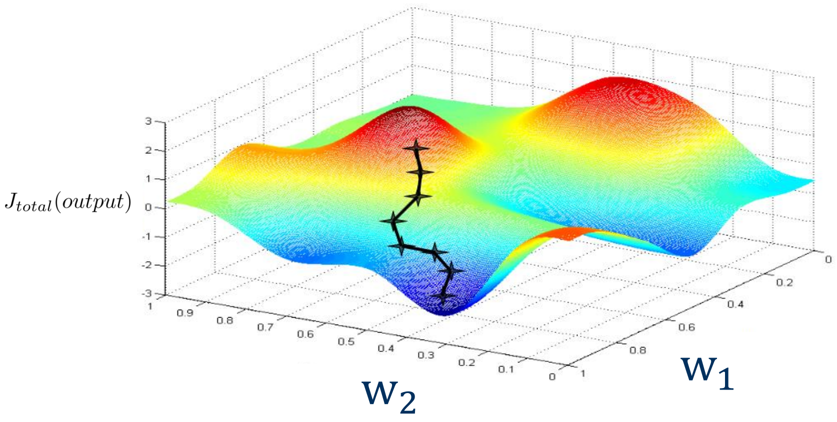
By chain rule we know for the weights that:

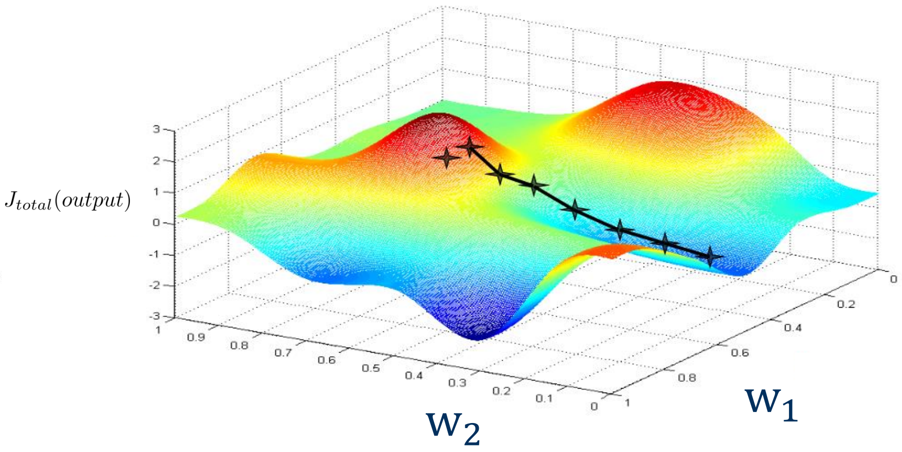
$$\frac{\partial input_{o1,1}}{\partial w_1} \cdot \frac{\partial output}{\partial input_{o1,1}} \cdot \frac{\partial J_{total}(output)}{\partial output} = \frac{\partial J_{total}(output)}{\partial w_1}$$

$$J_{total}(output) = \frac{1}{2}(target - output)^2$$ (MMSE Loss)

$$output = \frac{1}{1+e^{-input_{o1,1}}}$$ (Sigmoid)

$$input_{o1,1} = w_1 \cdot output_{h1,1} + w_2 \cdot output_{h1,2} + \; ...$$





Initialisation matters!

"Glorot" init:

$$\sigma(w) = 1/n$$
$$\mu(w) = 0$$

"He" init (Relu):

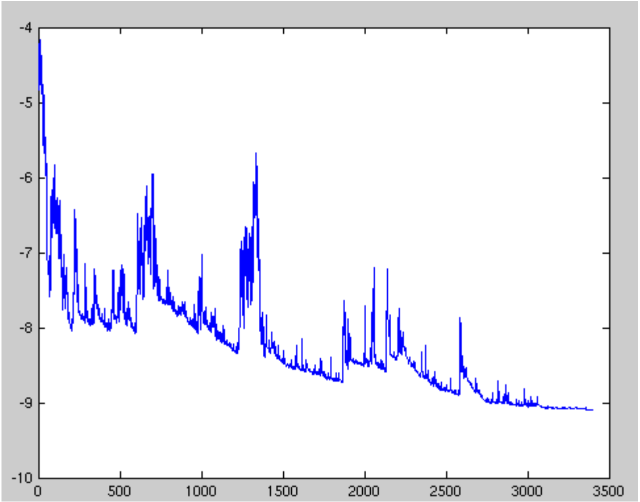
$$\sigma(w) = 2/n$$
$$\mu(w) = 0$$

Bias is set to zero.

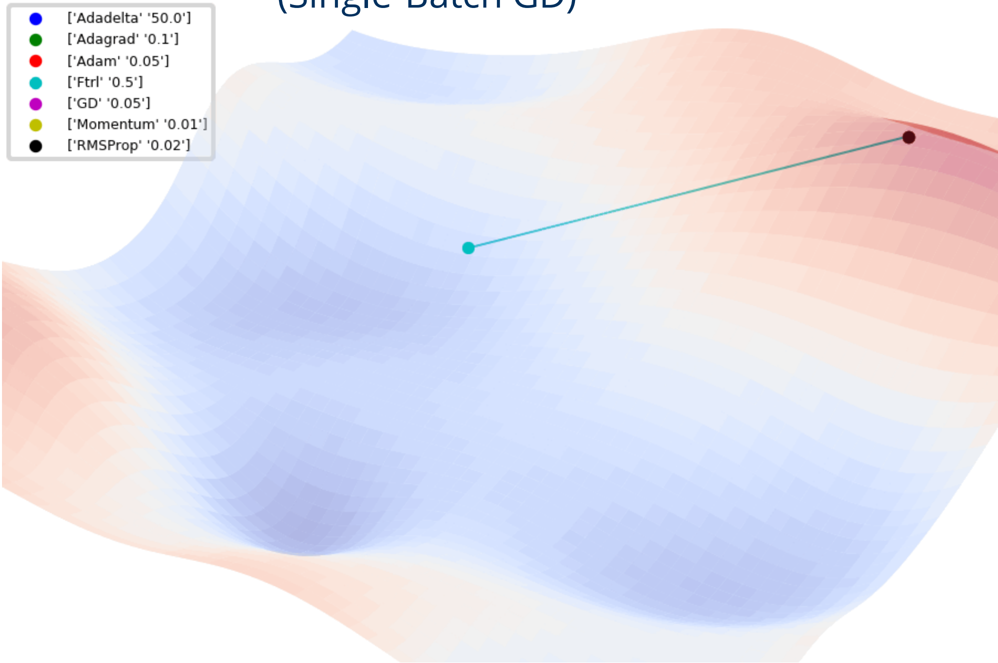
n … Number of Neurons

Optimiser – The "learning" Backbone
Update rules

$$\boxed{w_1(new) = w_1(old) - \alpha \cdot \frac{\partial J_{total}(output)}{\partial w_1}}$$

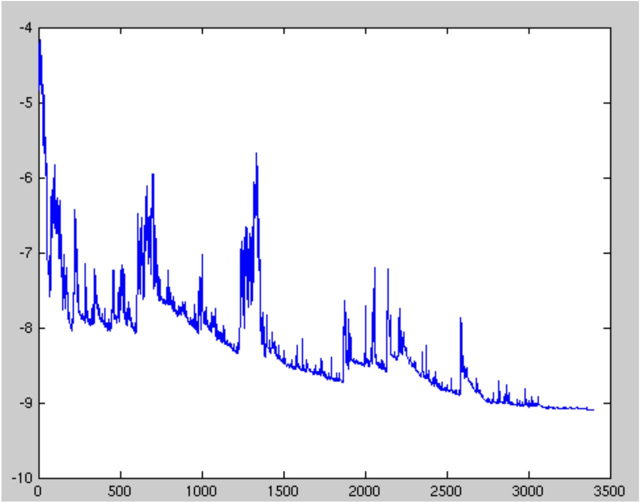


Stochastic Gradient Descent (Single-Batch GD)

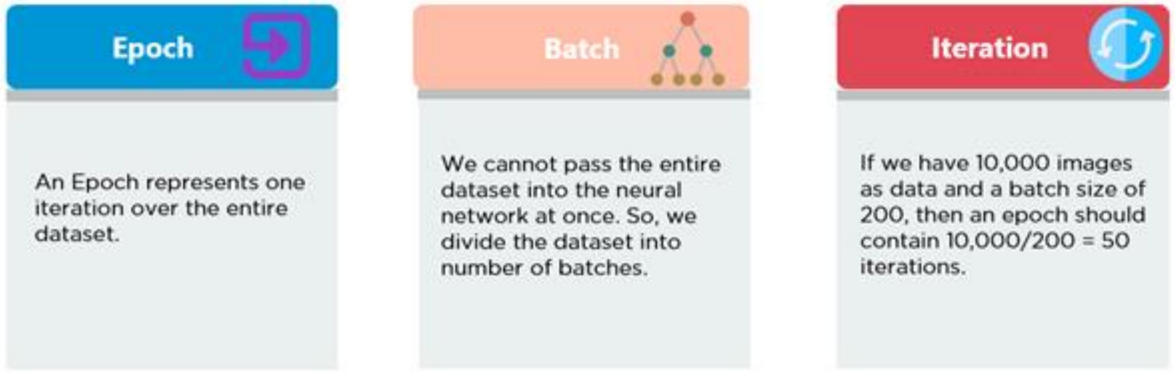


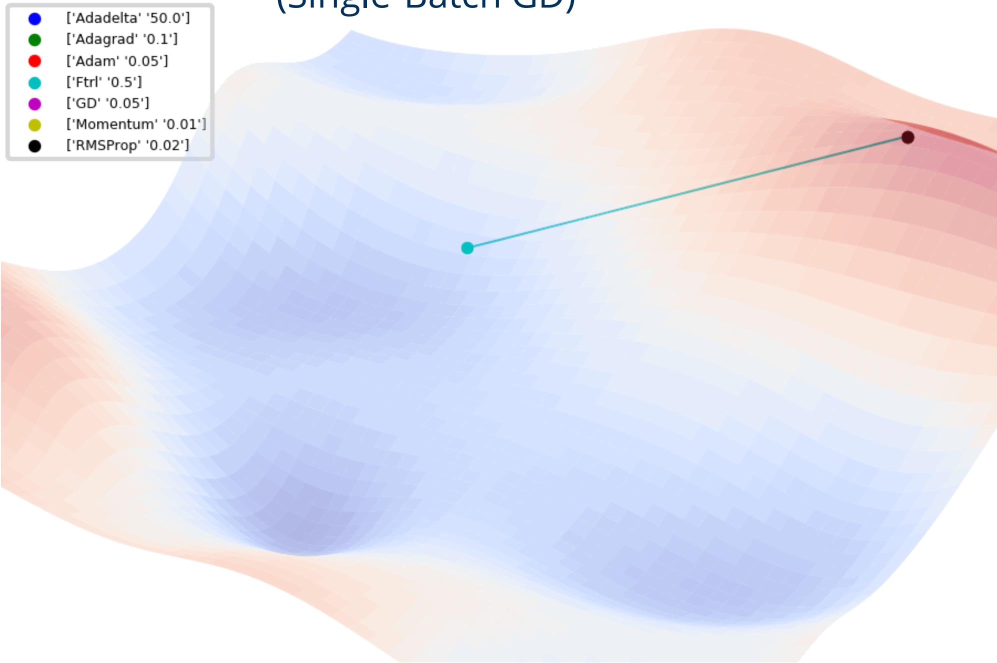
Optimizer – The “learning” Backbone
Update rules

$$\boxed{w_1(new) = w_1(old) - \alpha \cdot \frac{\partial J_{total}(output)}{\partial w_1}}$$



Stochastic Gradient Descent
(Single-Batch GD)





Multilayer Perceptron
Decision Layers

Data → Feature extraction → Decision Layer

How to decide which output to print?
There are two possible solutions here.

[0.0660451 , 0.4392075] →

Network output	Class
0.25	Cat
10	Horse
2	Car
4.02	House

→ Horse!

Argmax(x)

Network output x	Class	Argmax(x)
0.25	Cat	0
10	Horse	10
2	Car	0
4.02	House	0

Naive approach. Just take the maximum of the output vector. → Downside: **No interpretability**

Softmax(x)

Network output x	Class	Softmax(x)	Argmax(Softmax(x))
0.25	Cat	0.0105	
10	Horse	0.8	→ Horse!
2	Car	0.085	
4.02	House	0.1	

Better: Interpret output as „pseudo“ probability first. The output sums up to 1 now. Then argmax!

3. Exercise

Let‘s train our first classifier from scratch!