

Neural Networks 2

Applied Deep Learning

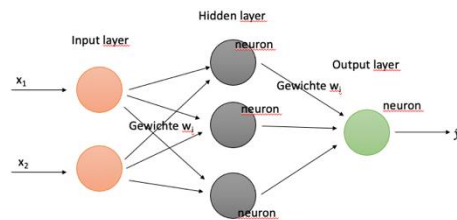
Goal for today

- Know internal mechanisms of neural networks
- Preparation of data
- Best practices for hyperparameter selection



Neurons in living beings and computers

Our
network: 9
neurons



Earthworm:
~300 neurons



Current networks:
~ n million neurons

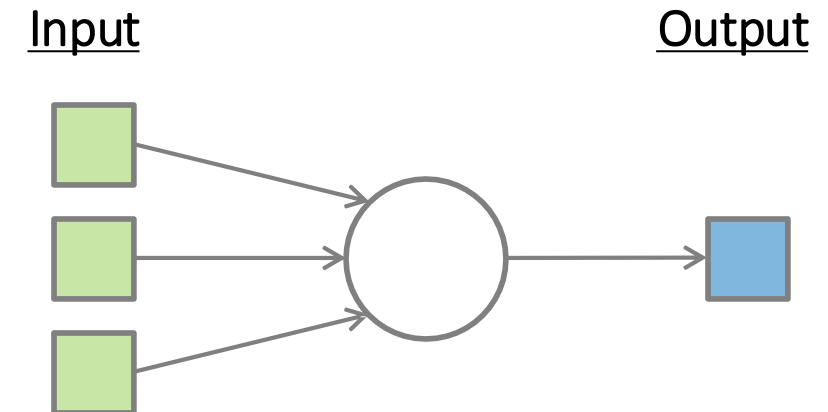


Man:
~100 billion neurons



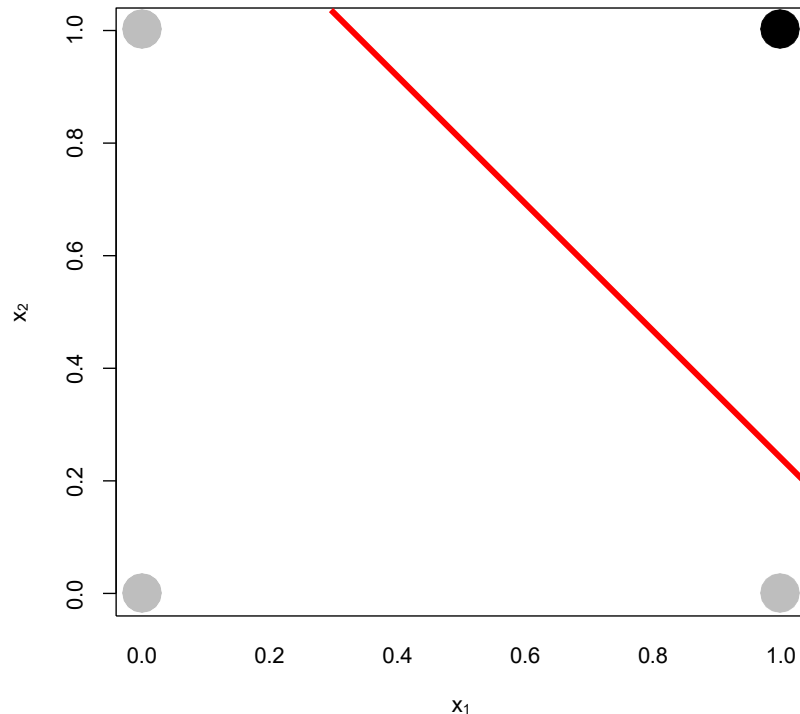
Performance of a neuron

- Single-Layer Neural Network
- Simplest design
- Which functions can be realized?



AND and OR as neural network

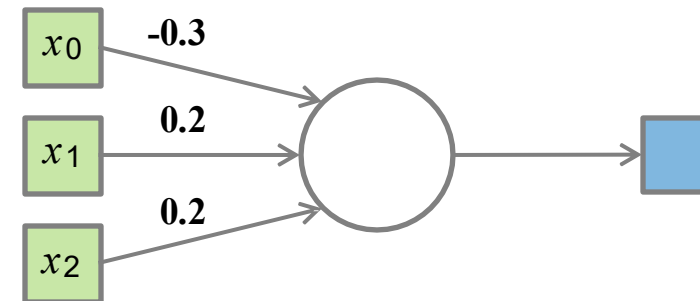
- Perceptron describes **separating hyperplane** that divides input into two classes



AND

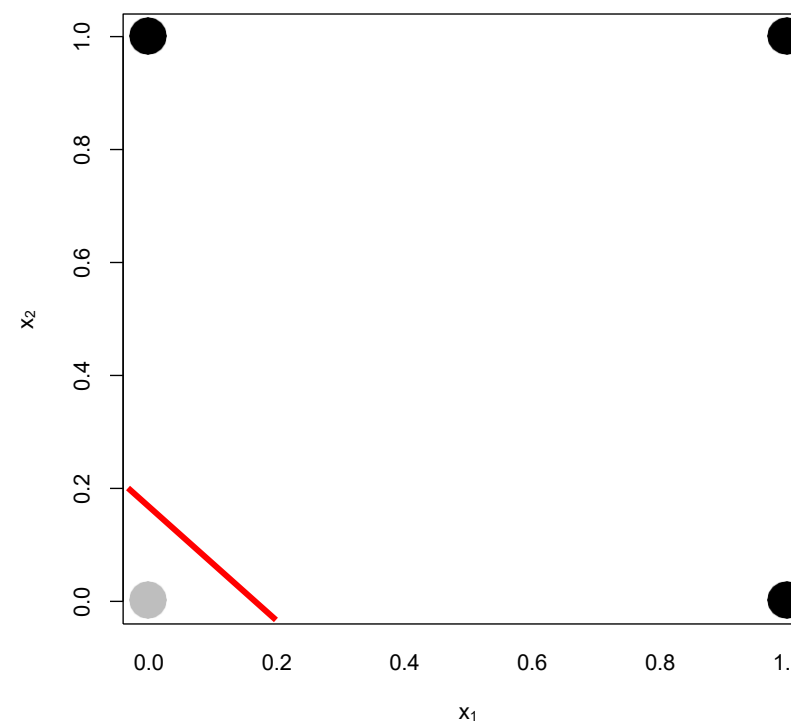
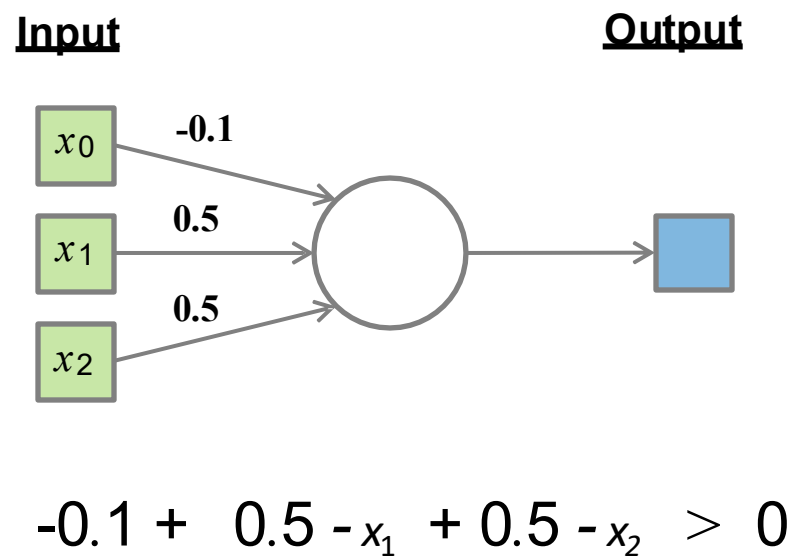
Input

Output



$$-0.3 + 0.2 - x_1 + 0.2 - x_2 > 0$$

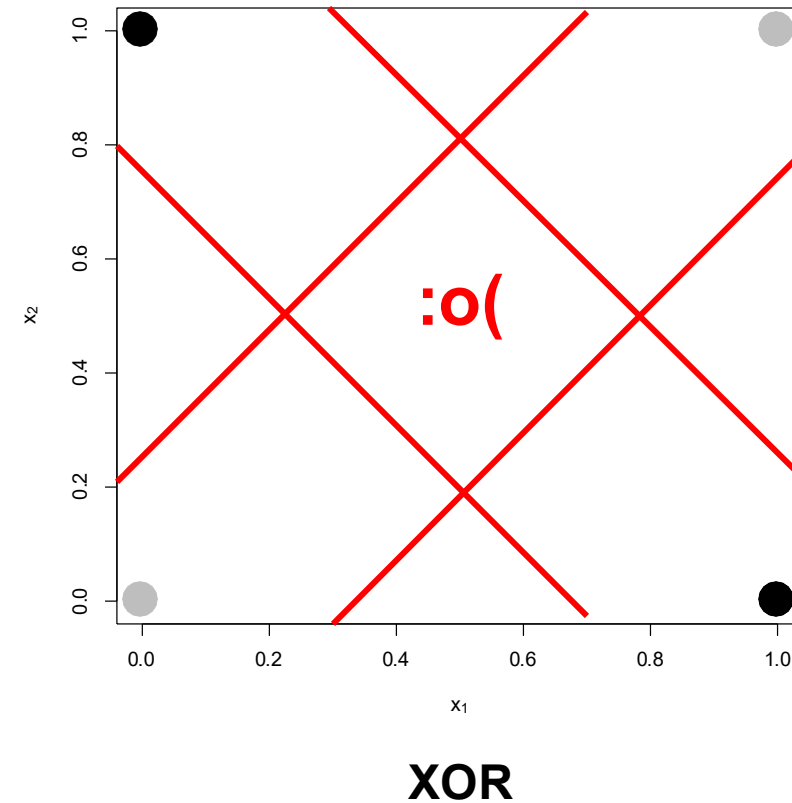
AND and OR as neural network



OR

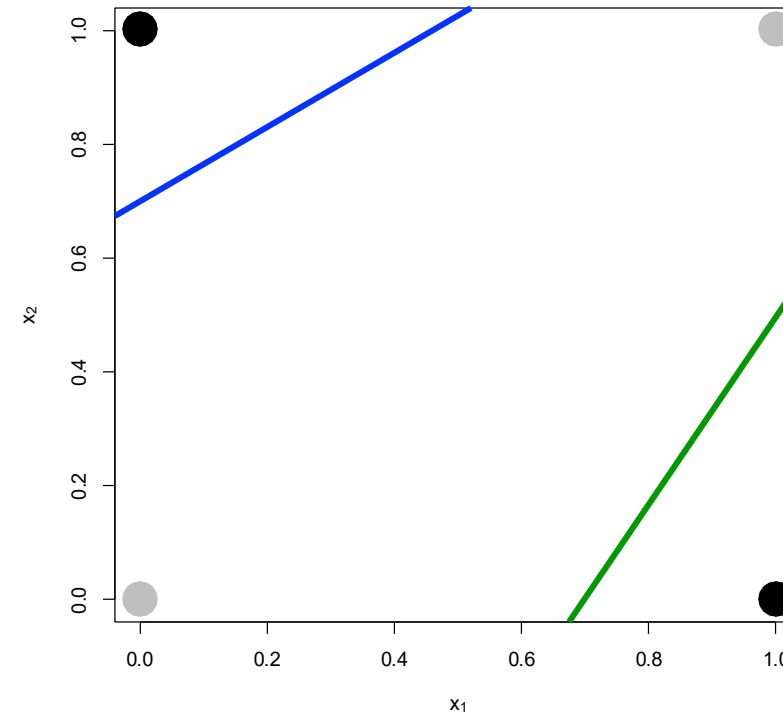
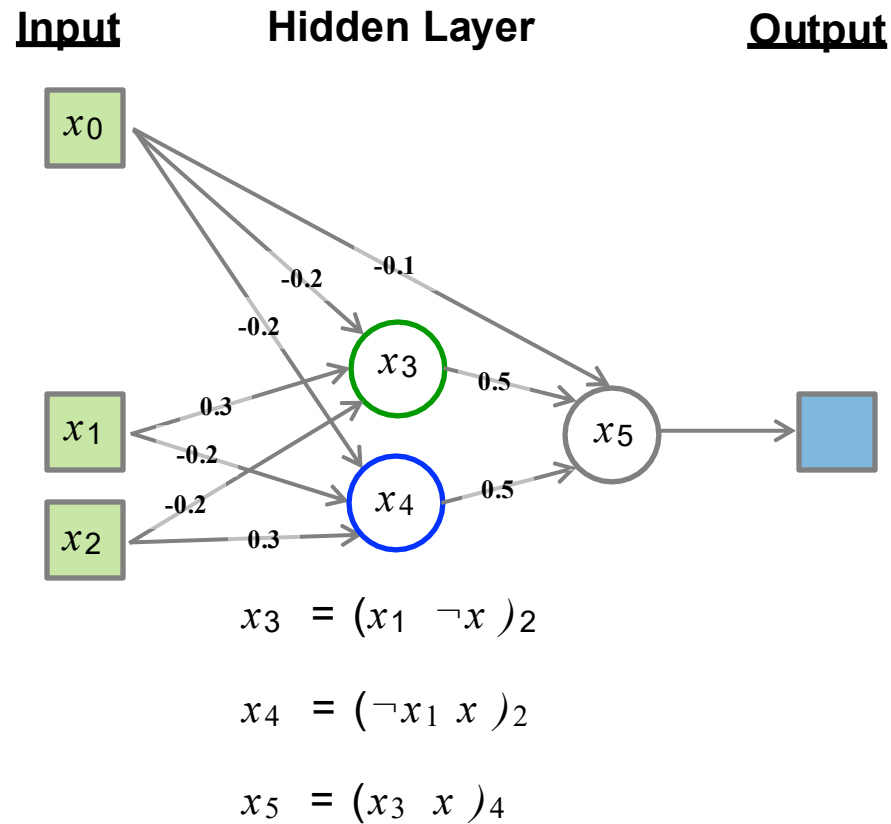
XOR as a neural network

- Cannot be modeled as a single perceptron



XOR as a neural network

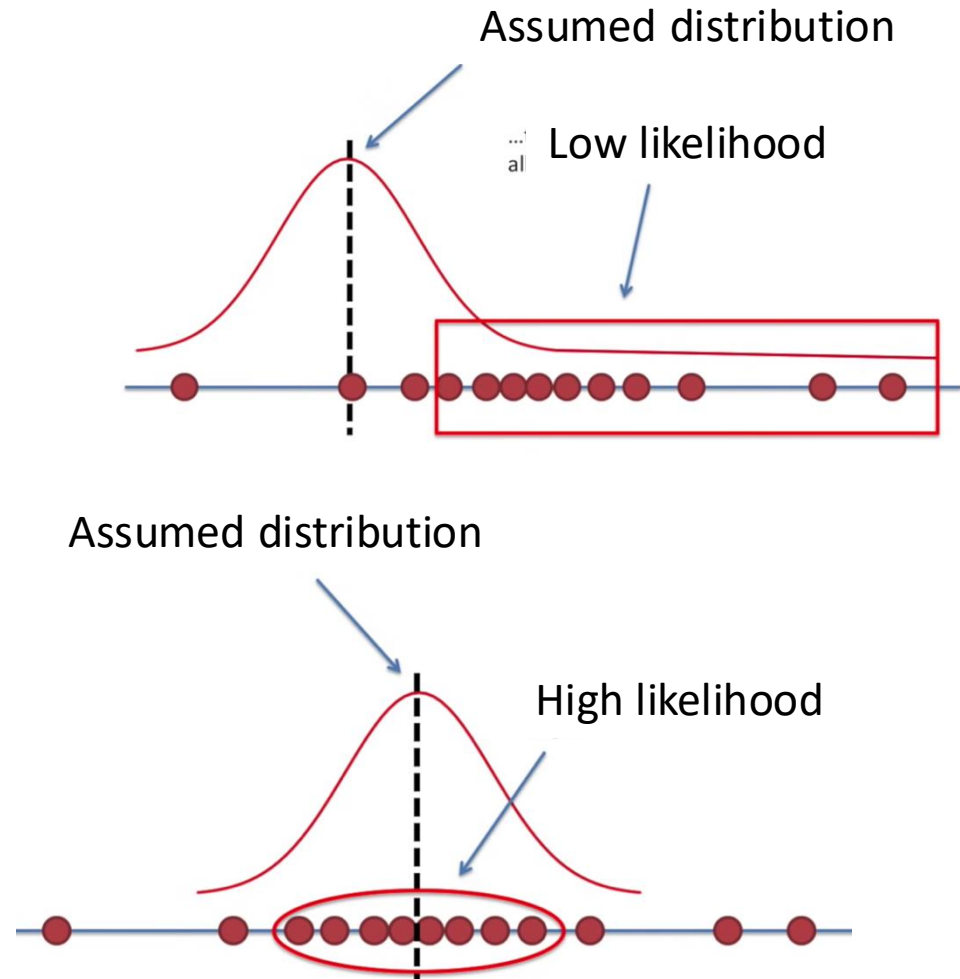
- But as a multilayer network



XOR

Loss functions

- Training of a NN corresponds to minimization of a cost function
= loss function / cost function / error function
- Maximum Likelihood Estimation is used to find the statistically best parameters from training data
- To do this, it compares the distributions of the data and the predictions (should be as similar as possible)
- Entropy corresponds to the degree of uncertainty of a random variable
- Cross-entropy: minimizes difference between class prediction and actual class membership in the context of the MLE.



- Binary classification
 - Loss function: Binary Cross Entropy (Log Loss)
 - 1 output neuron
 - usually by sigmoid activation function
- Multiclass classification
 - Loss function: Cross Entropy
 - One output neuron per class
 - usually Softmax activation function
- Regression
 - Loss function: Mean Squared Error (MSE, corresponds to Cross Entropy for Gaussian distribution)
 - 1 output neuron with continuous value

Optimizer

- SGD / Minibatch GD
- Momentum-based
- Adagrad
- RMS Prop
- Adam



Gradient descent method - algorithm

Start with a random parameter set w_{init}

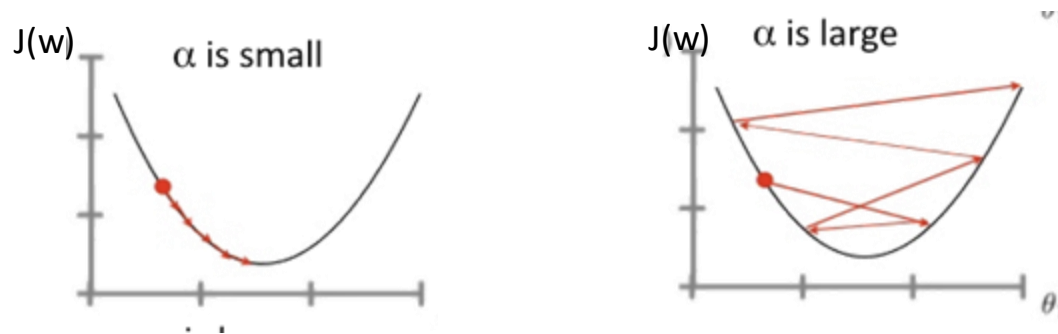
iterative descent in the direction with the largest (negative) change of the gradient

Repeat until convergence or after a specified number of steps.

$$w := w - \alpha \nabla J(w)$$

Learning rate α Gradient

Effect of the learning rate



Gradient descent method - implementations

Batch Gradient Descent (BGD)

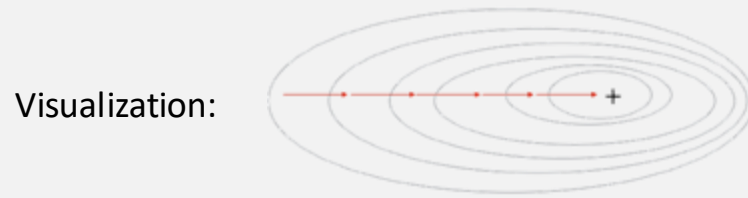
Model: $h_{\theta}(x) = \theta_0 + \theta_1 x$

Cost function
(m training samples) $J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$

Part. Derivatives:
(Math see e.g. [7]) $\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})$

$\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) \cdot x^{(i)}$

Update: $\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$



➔ In Batch Gradient Descent the complete dataset is scanned in each iteration/update step !

Stochastic Gradient Descent (SGD)

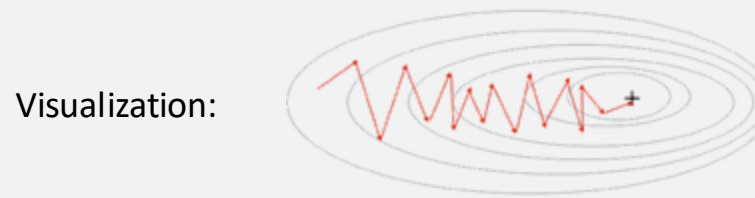
Model: $h_{\theta}(x) = \theta_0 + \theta_1 x$

Cost function
(m training samples) $J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$

Part. Derivatives:
(Math see e.g. [7]) $\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})$

$\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) \cdot x^{(i)}$

Update: $\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$



➔ In each iteration / update step, a random data point i is used for the calculation

Mini-Batch Gradient Descent (MBGD)

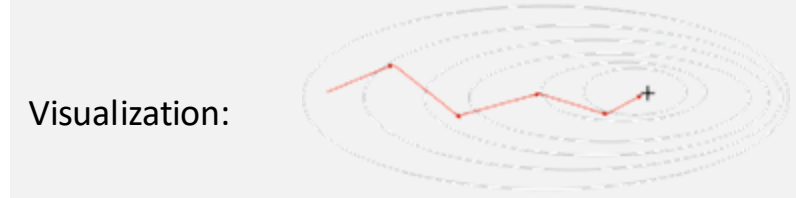
Model: $h_{\theta}(x) = \theta_0 + \theta_1 x$

Cost function
(m training samples) $J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$

Part. Derivatives:
(Math see e.g. [7]) $\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{1}{k} \sum_{i=1}^k (h_{\theta}(x^{(i)}) - y^{(i)})$

$\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \frac{1}{k} \sum_{i=1}^k (h_{\theta}(x^{(i)}) - y^{(i)}) \cdot x^{(i)}$

Update: $\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$

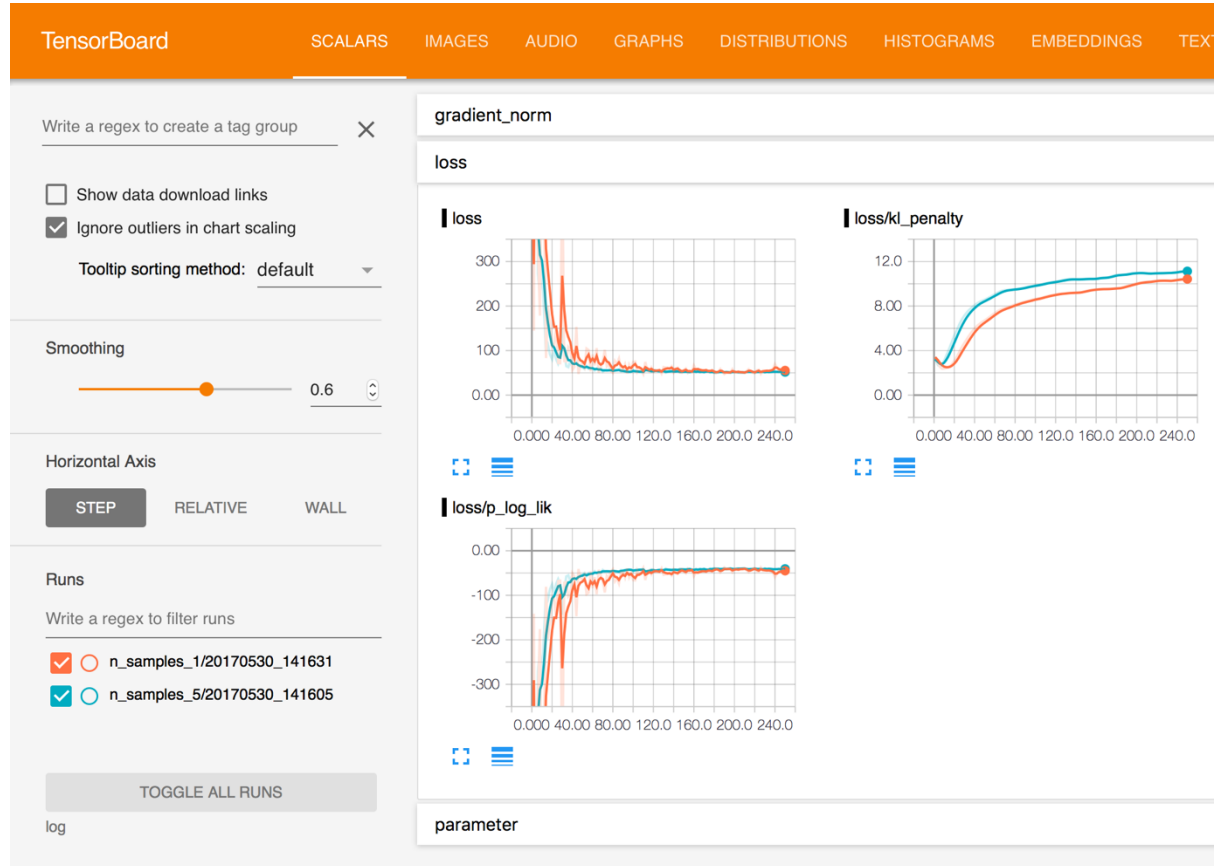


➔ In each iteration / update step, a subset of k points of the dataset is used for calculation

Visualization

- Tensorboard is the default logger for pytorch lightning
- Start it:
%reload_ext tensorboard
%tensorboard --logdir=lightning_logs/
- Make your own plots (Batch/Epoche)

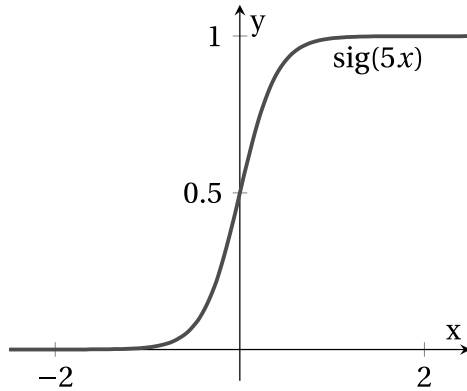
```
def training_step(self, batch, batch_idx):  
    self.log("my_metric", x)  
  
# or a dict to get multiple metrics on the same plot if the logger supports it  
def training_step(self, batch, batch_idx):  
    self.log("performance", {"acc": acc, "recall": recall})  
  
# or a dict to log all metrics at once with individual plots  
def training_step(self, batch, batch_idx):  
    self.log_dict({"acc": acc, "recall": recall})
```



<https://pytorch-lightning.readthedocs.io/en/stable/extensions/logging.html>

Activation functions

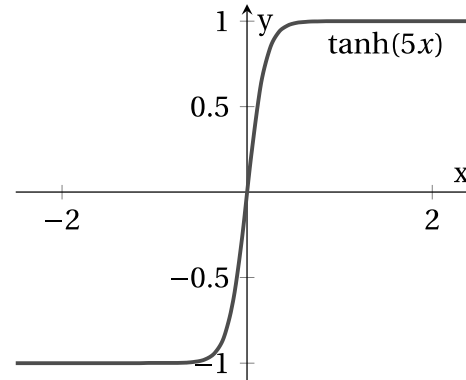
$$\varphi = \text{sig}(ax) = \frac{1}{1 + \exp(-ax)}$$



Sigmoid function

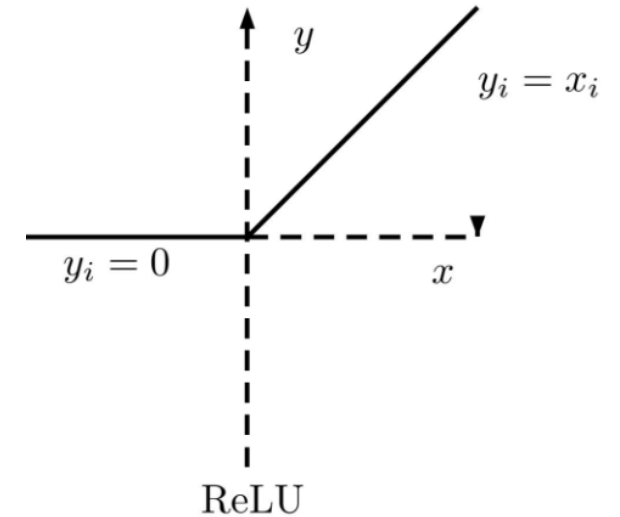
- For probabilities
- Monotonic and differentiable
- Can create "stuck" during training

$$\varphi = \tanh(a \cdot x)$$



Tangent Hyperbolic

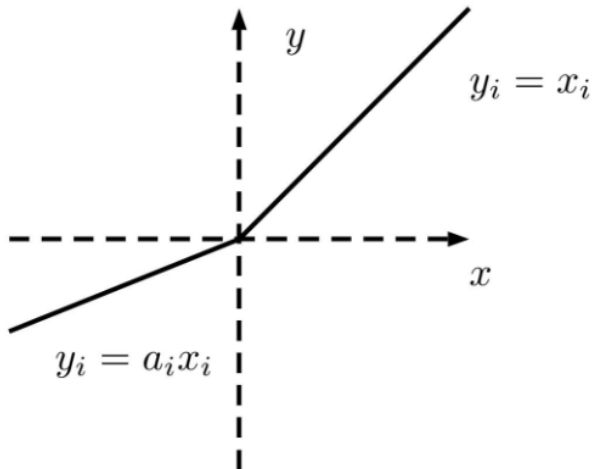
- Negative values generate negative outputs
- Monotonic and differentiable
- For classification with two classes



ReLU/Rectified Linear Unit

- Most widespread
- Monotone
- Neg. Values are mapped to zero

Activation functions

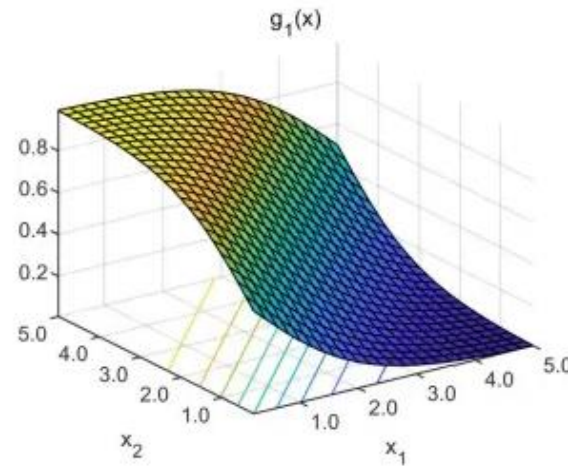


L_{ϵ} Leaky ReLU/PReLU

PReLU

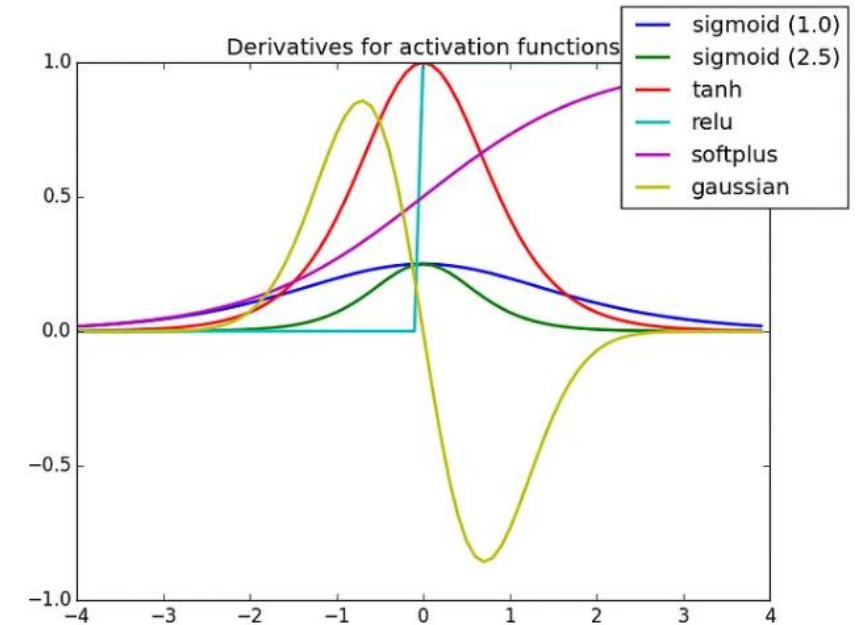
Parametric ReLU

- a mostly selected to 0.01
- Or randomized (PReLU)
- Monotone



Softmax

- Special form of the sigmoid function
- Input = vector instead of scalar
- For multiclass classification
- Output between 0 and 1
- Sum of expenses is 1



Derivatives of the activation functions

Gradient Descent mit Momentum

- Problem: Komplexe Loss-Funktionen
- Lösung: Führe Trägheit ein (Momentum)

GD mit Momentum

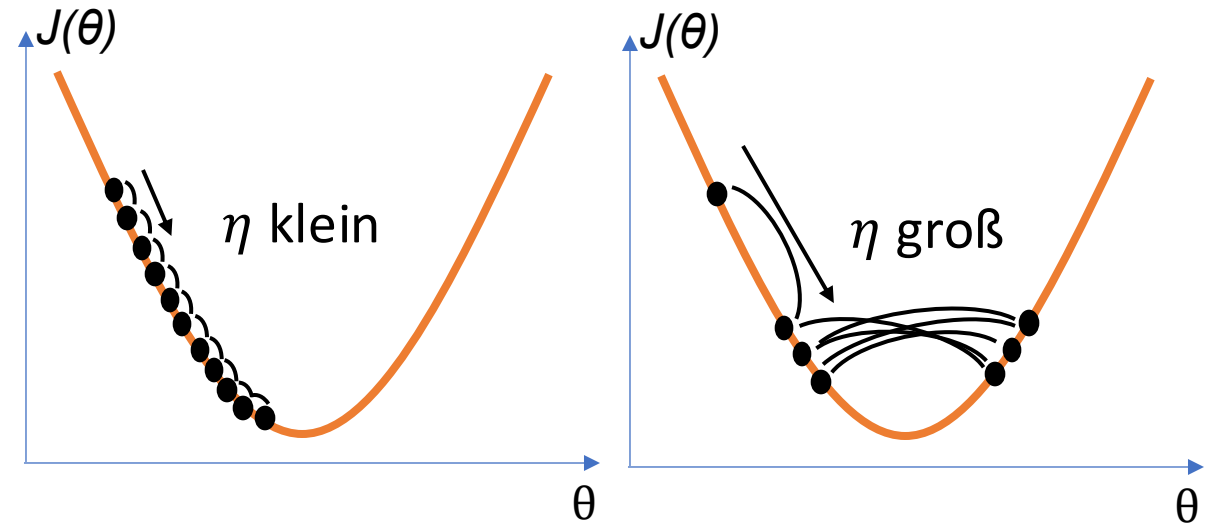
INIT $\eta, \gamma = 0.9$

WHILE (!end)

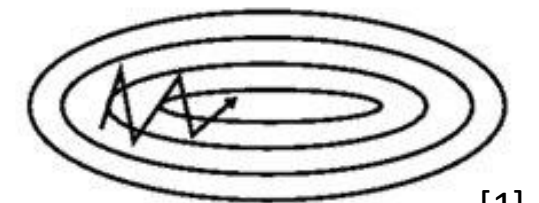
$$v_t = \gamma v_{t-1} + \eta \nabla_{\theta} J(\theta)$$

$$\theta_{t+1} = \theta_t - v_t$$

END



Ohne Moment



Mit Moment

[1]

Gradient Descent optimization AdaGrad

- Problem: Parameter density differs
- Solution: parameter-wise, adaptive learning rate

- Example for parameter i from GD

$$\text{Def.: } g_{t,i} = \nabla_{\theta_t} J(\theta_{t,i})$$

- Update step GD for parameter i

$$\theta_{t+1,i} = \theta_{t,i} - \eta \cdot g_{t,i}$$

- AdaGrad adaption

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \varepsilon}} \cdot g_{t,i}$$

$$G_{t,ii} = \sum_t g_t^2$$

Learning rate

- Vectorizing all features

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \varepsilon}} \cdot g_t$$

Gradient Descent optimization AdaDelta

- Problem AdaDelta: G_t grows steadily
→ Learning rate decreases steadily
- Solution: Limit the number of summed gradients in G
- Def.: $E[g^2]_t = \gamma E[g^2]_{t-1} + (1 - \gamma) g^2$
- Gradient Descent:
 - $\Delta\theta_t = -\eta \cdot g_t$
 - $\theta_{t+1} = \theta_t + \Delta\theta_t$
- Replace G by $E[g^2]$
 - $\Delta\theta = \frac{-\eta}{\sqrt{E[g^2]_{t+\varepsilon}}} \cdot g_t$
- *Denominator resembles RMS*
 - $\Delta\theta = \frac{-\eta}{RMS[g]_t} \cdot g_t$
- Eliminate η
 - $\Delta\theta = -\frac{RMS[\Delta\theta]_{t-1}}{RMS[g]_t} \cdot g_t$

```
Init:  $\eta,$   
       $\beta_1=0.9,$   
       $\beta_2=0.999,$   
       $\theta_0,$   
       $m_0=v_0=t=0$ 
```

```
while (not converge)
```

$$g_t = \nabla_{\theta_t} f(\theta_t)$$

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

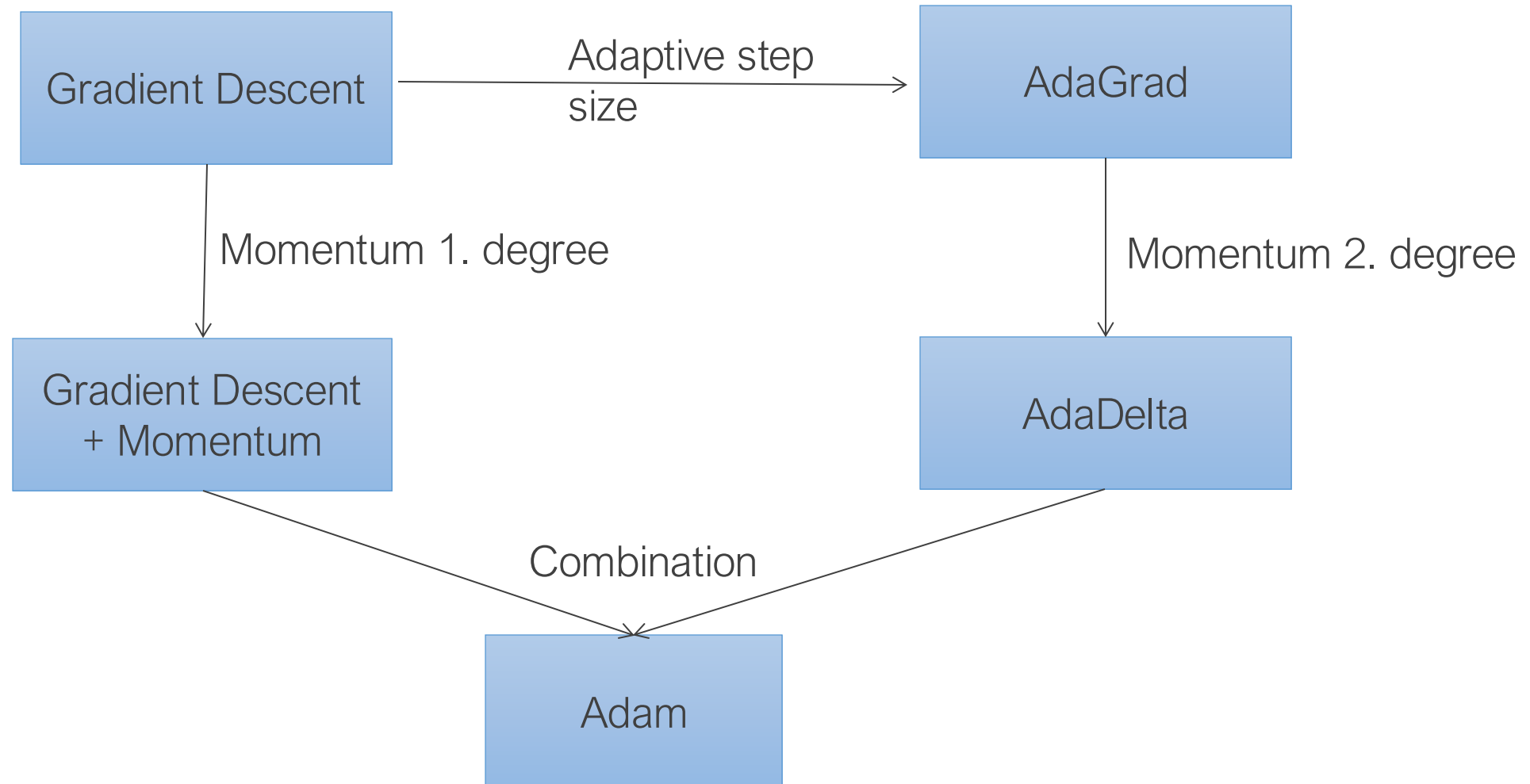
$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}} \cdot \hat{m}_t$$

```
end while
```

```
return  $\theta_{t+1}$ 
```

- Combination of Adadelta and GD Momentum
 - $m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$
 - $v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$
 - $\beta_1 = 0.9, \beta_2 = 0.999$
- initialize to 0
- strong 0-bias, therefore
 - $\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$
 - $\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$
- Update the parameters like with AdaDelta with the before mentioned changes
 - $\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}} \cdot \hat{m}_t$



Data preparation

- Missing values
 - Errors in data acquisition
 - Intentionally released (survey)
- NaN
 - Can usually not be processed
 - Garbage in, garbage out

```
>>> import pandas as pd
>>> from io import StringIO
>>> csv_data = \
...     '''A,B,C,D
...     1.0,2.0,3.0,4.0
...     5.0,6.0,,8.0
...     10.0,11.0,12.0,'''
>>> # If you are using Python 2.7, you need
>>> # to convert the string to unicode:
>>> # csv_data = unicode(csv_data)
>>> df = pd.read_csv(StringIO(csv_data))
>>> df
```

	A	B	C	D
0	1.0	2.0	3.0	4.0
1	5.0	6.0	NaN	8.0
2	10.0	11.0	12.0	NaN

- Missing values
 - Errors in data acquisition
 - Intentionally released (survey)
- NaN
 - Can usually not be processed
 - Garbage in, garbage out
 - isnull() helps

```
>>> df.isnull().sum()
A      0
B      0
C      1
D      1
dtype: int64
```


Data preparation

- Different strategies for handling
- Elimination of entries
 - Simplest option
- Data imputation
 - E.g. by mean values
 - Or with the help of a transformer

```
>>> df.dropna(axis=0)
```

	A	B	C	D
0	1.0	2.0	3.0	4.0

```
>>> df.dropna(axis=1)
```

	A	B
0	1.0	2.0
1	5.0	6.0
2	10.0	11.0

```
>>> df.fillna(df.mean())
```

```
>>> imputed_data
```

```
array([[ 1.,  2.,  3.,  4.],
```


```
       [ 5.,  6.,  7.5,  8.],
```

```
       [10., 11., 12.,  6.]])
```

Data preparation

- How can nominal and ordinal features be encoded in a way that they can be used in neural networks?
- Nominal characteristics (e.g., origin) are encoded into a binary characteristic for each value present in the data (one-hot encoding)

Orgin	1	3	1	2
-------	---	---	---	---



Orgin 1	Orgin 2	Orgin 3
1	0	0
0	0	1
1	0	0
0	1	0

- Ordinal characteristics (e.g. energy efficiency class) are mapped to integers so that the order of the original values is preserved



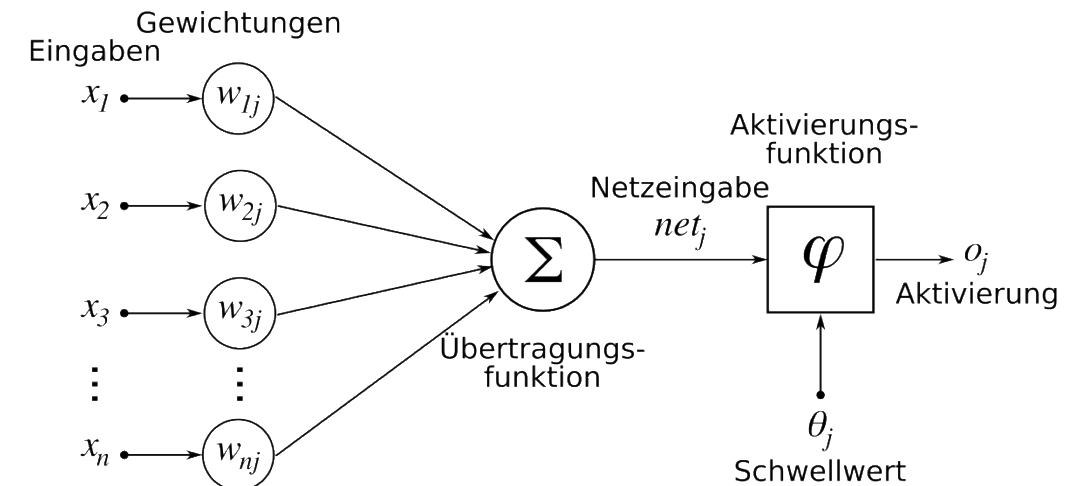
Energy efficiency class	Energy efficiency class
A	1
C	3
B	2

- Here it is implicitly assumed that the distances between neighboring values are equal, i.e. they are equidistant

Normalization and standardization

- Neural networks should receive values in the interval $[0,1]$ or $[-1,1]$.
- Reason: many learning methods are sensitive to very large or very small input values

- $f(x) = \max(0, x)$



Normalization

- Normalization is done per column (i)
- $x^{(i)}$ = vector of column i
- Maximum value of column i
- Minimum value of column i
- normalized column i
- $$\hat{x}^{(i)} = \frac{x^{(i)} - x_{min}^{(i)} * (1, 1, \dots, 1)^T}{x_{max}^{(i)} - x_{min}^{(i)}}$$
- ➔ Value ranges are guaranteed
- ➔ different structure regarding standard deviation remains (can be good, does not have to be)

Example vehicle table

- We will always use the middle columns kW, engine displacement, kg, l/100km and doors as features in the following. The columns price and class, on the other hand, occur as target values for classification (class) or regression (price). Class is an ordinal scale. We therefore code the classes in ascending order, starting at one for simplicity, as follows:
 1. smallest car
 2. small car
 3. lower middle class
 4. middle class
 5. upper middle class
 6. upper class

			1	2	3	4	5	
Nr.	Bezeichnung	Preis €	kW	Hubraum	kg	l/100km	Türen	Klasse
2	Bugatti Chiron	2856000	1103	7993	2070	22.5	2	6
338	Ford GT	500000	475	3497	1385	14.9	2	6
361	Lamborghini Urus	204000	478	3996	2200	12.7	5	6
389	Porsche 911	152416	368	3996	1488	12.9	2	6
126	BMW M3	77500	317	2979	1595	8.8	4	4
145	Alfa Romeo 4C	63500	177	1742	970	6.8	2	4
308	Porsche 718	52694	220	1988	1410	7.4	2	5
325	Mercedes E 200	43019	135	1991	1575	6.1	4	5
463	Audi S3	41450	228	1984	1480	7.0	3	3
465	Ford Focus RS	40675	257	2261	1560	7.7	5	3
471	Opel Astra OPC	36360	206	1998	1550	7.8	3	3
472	Honda Civic R	36050	235	1996	1380	7.7	5	3
207	Nissan 370Z	34130	241	3696	1496	10.6	3	4
213	BMW 318i	32850	100	1499	1475	5.1	4	4
217	Mercedes C 160	31868	95	1595	1395	5.2	4	4
218	Audi A4 1.4	31850	110	1395	1395	5.2	4	4
219	Skoda Octavia	31590	169	1984	1442	6.5	5	4
483	Volvo XC40 T3	31350	114	1498	1725	6.8	5	3

			1	2	3	4	5	
Nr.	Bezeichnung	Preis €	kW	Hubraum	kg	l/100km	Türen	Klasse
36	Toyota Yaris GRMN	30800	156	1798	1135	7.5	5	2
485	VW Golf GTI	30425	169	1984	1364	6.4	3	3
489	Hyundai i30 N	29700	184	1998	1475	7.0	5	3
498	Mercedes B 160	26638	75	1595	1395	5.5	5	3
40	Opel Corsa OPC	24930	152	1598	1293	7.5	3	2
251	Toyota Avensis	24740	97	1598	1430	6.1	4	4
512	BMW 116i	24700	80	1499	1375	5.3	3	3
41	Peugeot 208	23990	153	1598	1235	5.4	3	2
9	VW up! GTI	16975	85	999	1070	4.8	3	1
591	Toyota Auris 1.33	16490	73	1329	1225	5.5	5	3
67	MINI One First	16400	55	1198	1165	5.1	3	2
75	Hyundai ix20 1.4	15790	66	1396	1253	5.6	5	2
597	SEAT Leon 1.2	15490	63	1197	1188	5.1	5	3
93	Fiat Punto 1.2	12790	51	1242	1105	5.4	5	2
606	Lada Vesta 1.6	12740	78	1596	1250	6.1	4	3
17	Suzuki Ignis 1.2	12540	66	1242	885	4.6	5	1
97	SEAT Ibiza 1.0	12490	48	999	1091	4.9	5	2
19	Opel ADAM 1.2	12135	51	1229	1086	5.3	3	1
	Mittelwert (≈)	51459	161	1891	1358	6.9	3.9	-
	σ (≈)	85702	110.37	788.35	236.97	2.4	1.06	-

Results after normalization

	Preis €	kW	Hubraum	kg	l/100km	Türen	Klasse
Standardabweichung (\approx)	0.1757	0.2567	0.2630	0.1802	0.2325	0.3543	<i>0.2622</i>
Mittelwert (\approx)	0.0806	0.2623	0.2977	0.3599	0.2255	0.6286	<i>0.4457</i>
Median (\approx)	0.0386	0.2023	0.1999	0.3802	0.1456	0.6667	0.400

Centering (statistics)

- In statistics, centering is understood as a transformation with shifting of the values of a variable by the value of the mean of this variable. A typical example in the social sciences is the shifting of age data of persons by the mean value of the age data of a population. The advantage is that below-average values thereby receive negative signs. (Wikipedia: 2019-03-23)

Standardization

- Also referred to as studentization:
- Studentization or Studentizing in mathematical statistics is a **transformation of the realizations of a random variable** so that **the resulting values** have an **arithmetic mean of zero** and an empirical **variance of one**. Since the empirical standard deviation is equal to the root of the sample variance, it is therefore also equal to one.
Studentize is necessary, for example, to compare differently distributed random variables. (Wikipedia: 2019-03-23)

Standardization

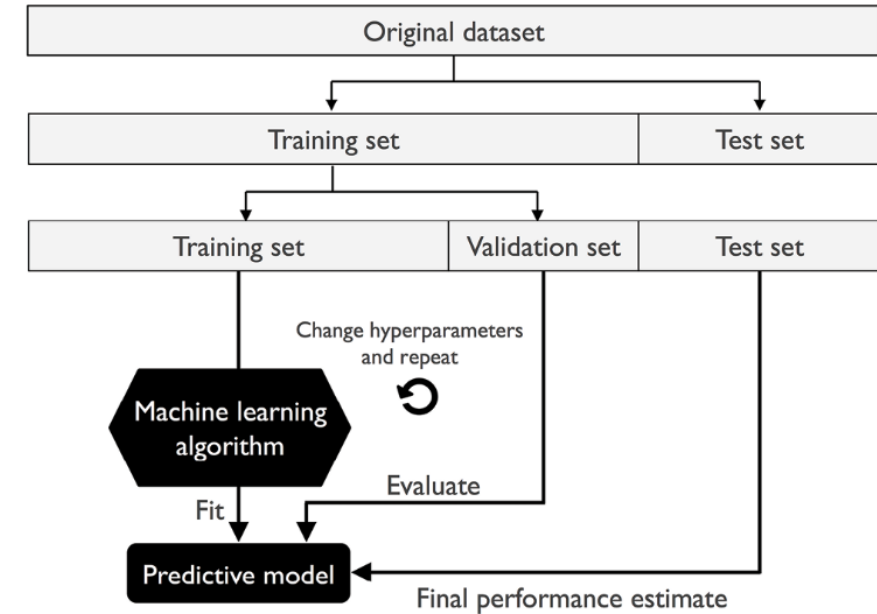
- Mean value of the i-th column
- Standard deviation of the i-th column
- $\sigma_{x^{(i)}} = \sqrt{\frac{1}{n} \sum_k (x_k^{(i)} - \bar{x}^{(i)})^2}$
- Standardization of the i-th column
- $\tilde{x}^{(i)} = \frac{x^{(i)} - \bar{x}^{(i)} * (1, 1, \dots, 1)^T}{\sigma_{x^{(i)}}}$

	x		\hat{x}		\tilde{x}	
Bezeichnung	Preis €	kg	Preis €	kg	Preis €	kg
Ford GT	500000	1385	1.0000	0.3802	5.2337	0.1126
Lamborghini Urus	204000	2200	0.3933	1.0000	1.7799	3.5518
⋮	⋮	⋮	⋮	⋮	⋮	⋮
Toyota Yaris	30800	1135	0.0383	0.1901	-0.2410	-0.9423
VW Golf GTI	30425	1364	0.0375	0.3643	-0.2454	0.0239
⋮	⋮	⋮	⋮	⋮	⋮	⋮
SEAT Ibiza 1.0	12490	1091	0.0007	0.1566	-0.4547	-1.1280
Opel ADAM 1.2	12135	1086	0.0000	0.1528	-0.4588	-1.1491

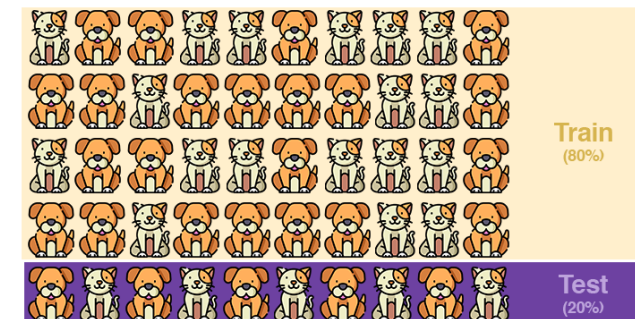
- Model Selection
 - Selection of a model from possible candidates (e.g. SVM or KNN)
 - Choice of suitable hyperparameters (e.g. choice of a kernel)
- Criteria
 - Idea of a "best" model rarely helpful
 - Model that meets the requirements of the stakeholders
 - Taking into account the possible resources
 - Choose state-of-the-art models
- Selection techniques
 - Probabilistic measures
 - based on Bayes Information Criterion (BIC) or Structural Risk Minimization (SRM)
 - Good for simple models, like linear regression
 - Resampling measures
 - Holdout/Cross-Validation/Bootstrap
 - For more complex models

Model Selection

- Holdout method
 - Classical division in training/testing data sets (80/20)
 - Extended division into Train/Eval/Test
 - Dependend on data distribution
 - "Lucky sample" possible
 - Good for "quick" model selection



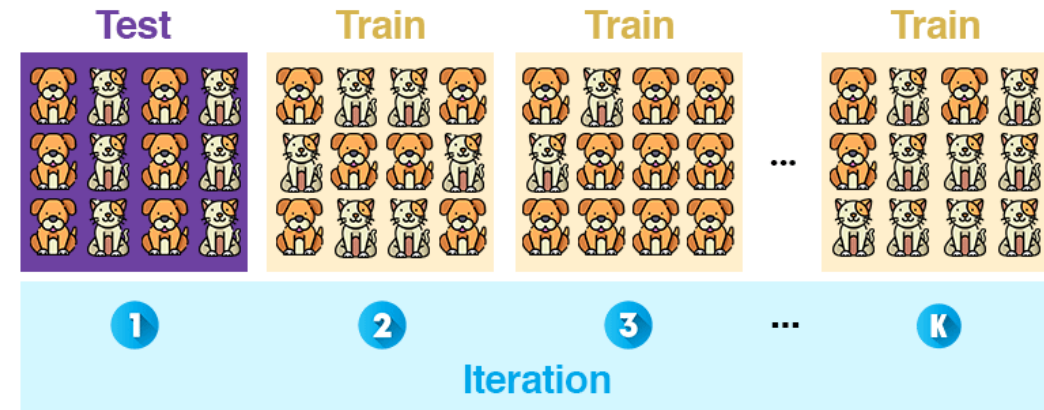
Holdout Method



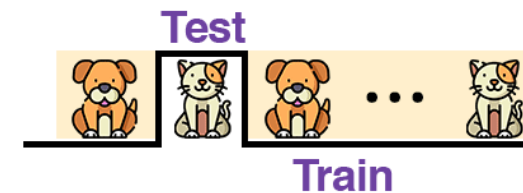
Model Selection

- K-fold Cross Validation
 - Division into k folds of equal size
 - One fold for testing, the rest for training
 - K repetitions (parallelizable)
 - Average of the performance as a result
 - Minimize risk
- Leave-one-out
 - K-fold Cross Validation with $k = \text{number of data}$
 - Only 1 test sample
 - Good with little data
 - Computationally intensive

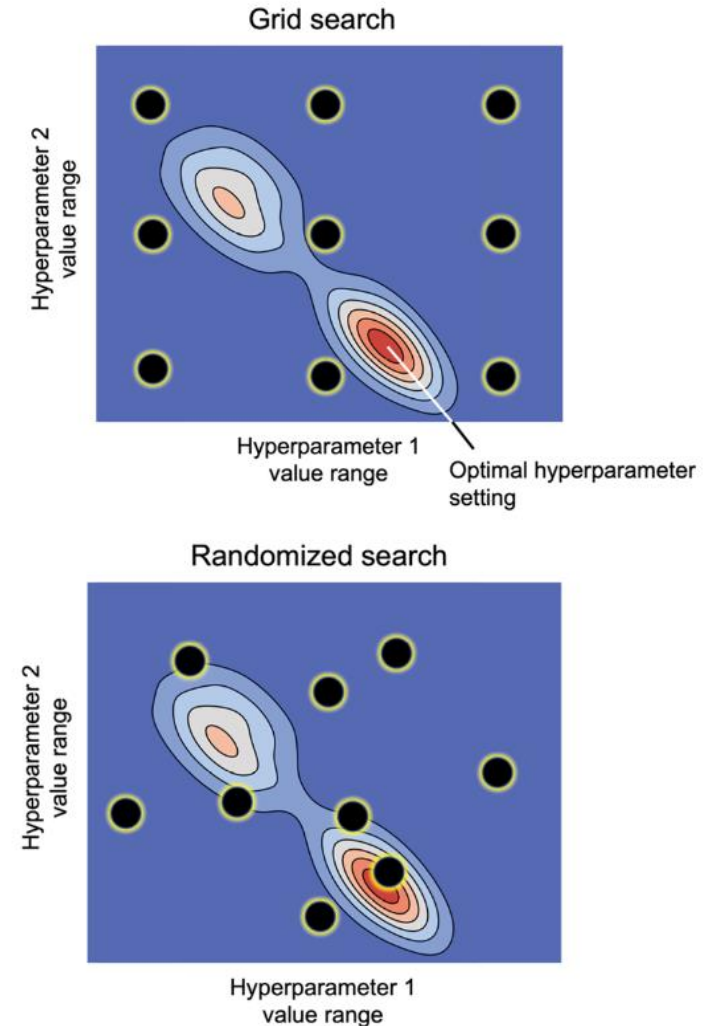
k-Fold Cross Validation



Data: Leave-one-out

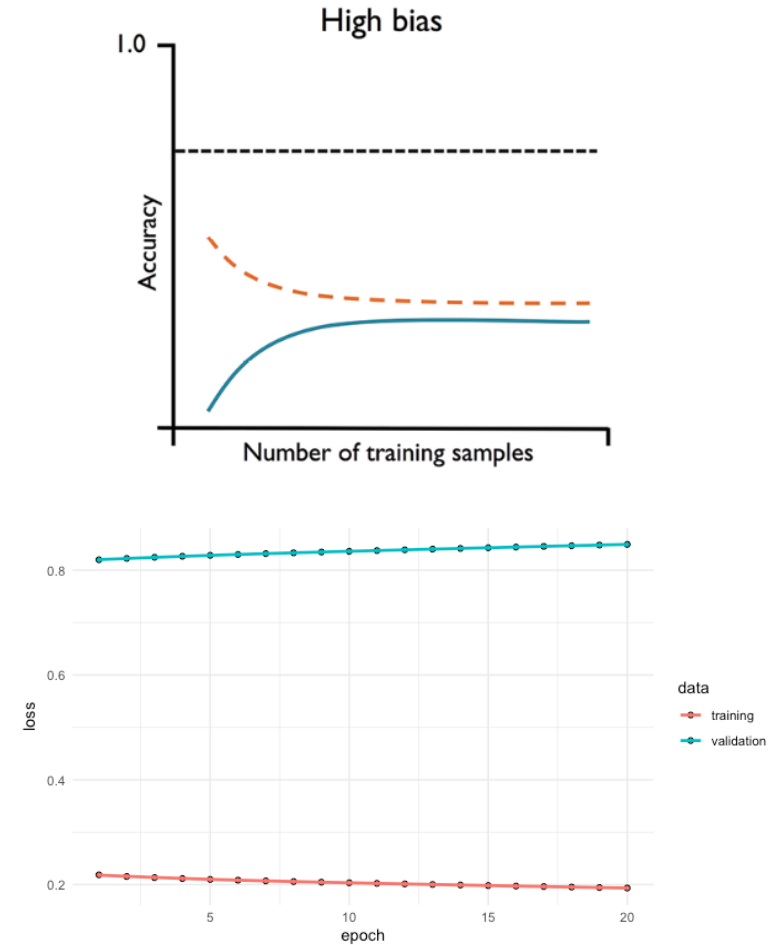


- Grid Search
 - Brute force approach
 - List of hyperparameter combinations is specifically evaluated
- Randomized Search
 - When GridSearch is too time-consuming
 - Selection of random expressions
 - Can be optimized (e.g. by successive halving)
 - Often used in combination with k-fold cross validation



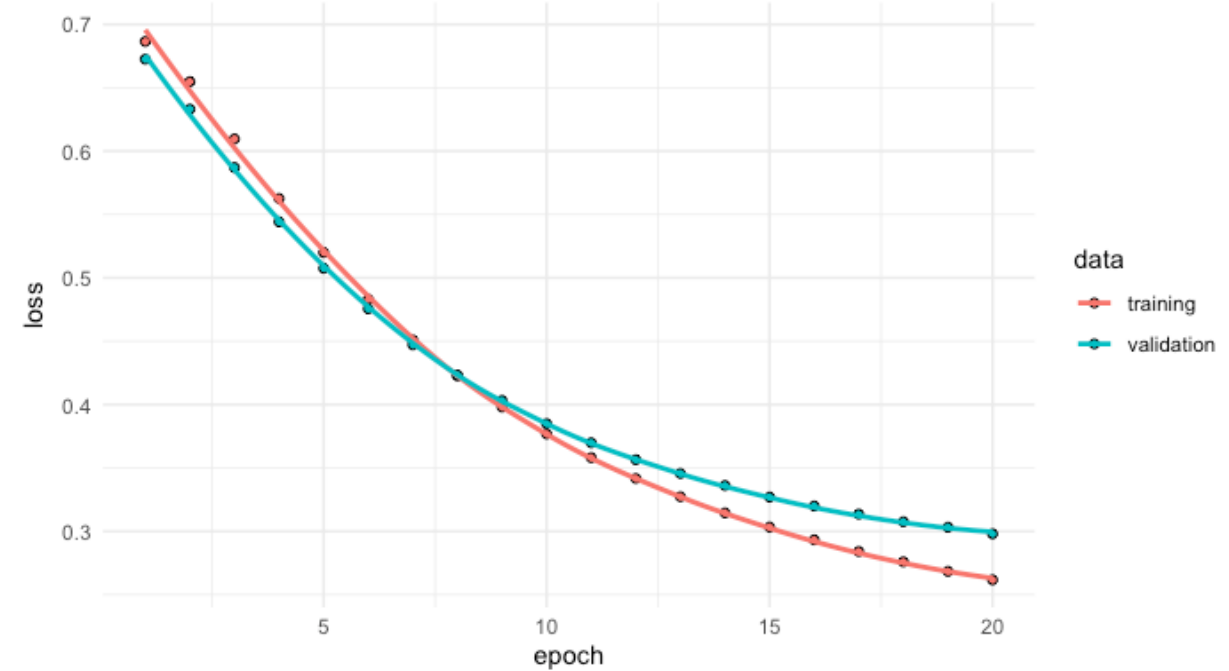
Model debugging

- Selection of a suitable metric
- Define a desired target value
- Viewing the curves
- Underfitting
 - Model cannot represent underlying complexity
 - More data ← often does not work
 - Reduce/remove regularization
 - Increase model complexity



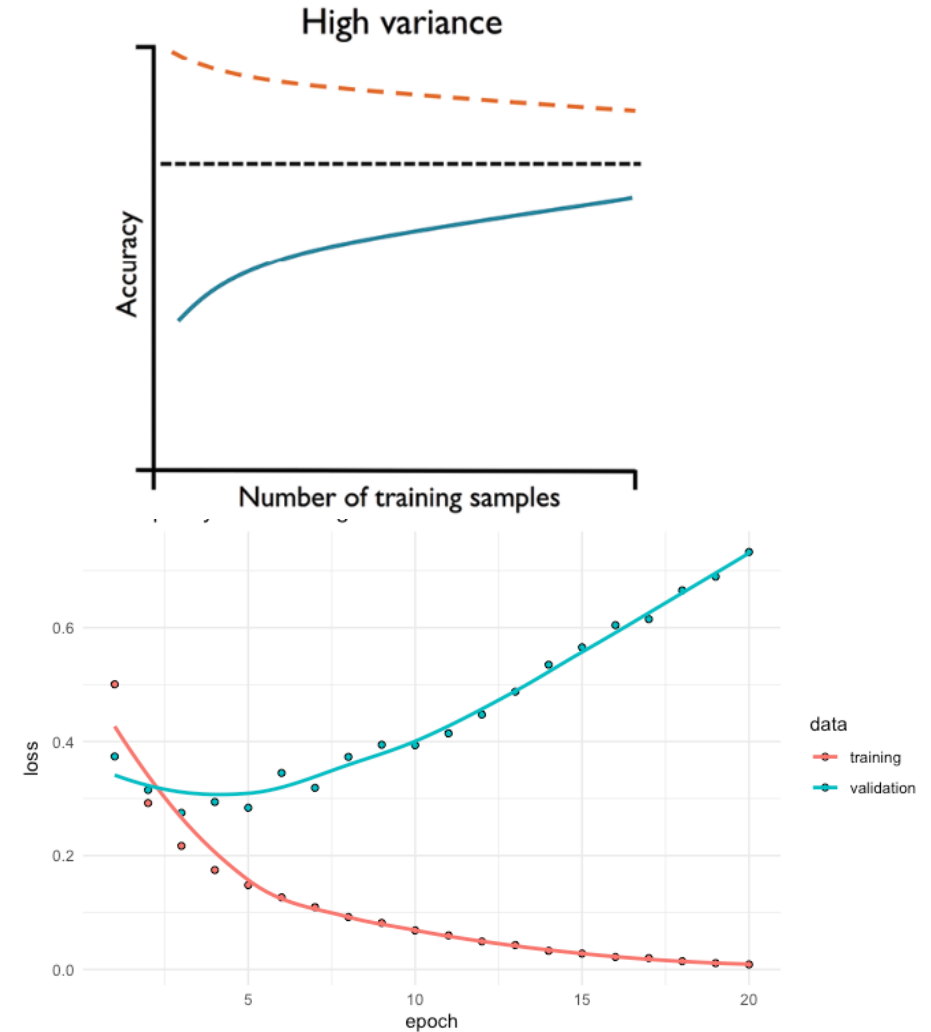
Model debugging

- Underfitting
 - Further iterations as long as loss decreases
 - Alternative: Increase learning rate



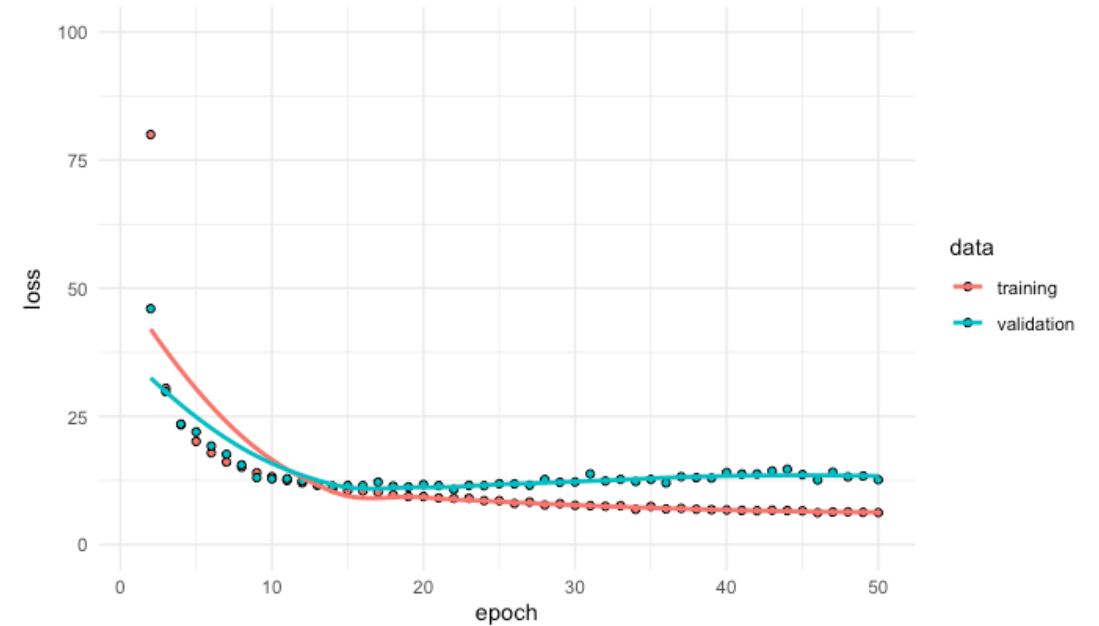
Model debugging

- Overfitting
 - The model has learned the training data too accurately (including noise and random fluctuation)
- Regularize
- Reduce model complexity (fewer layers)
- Dropout



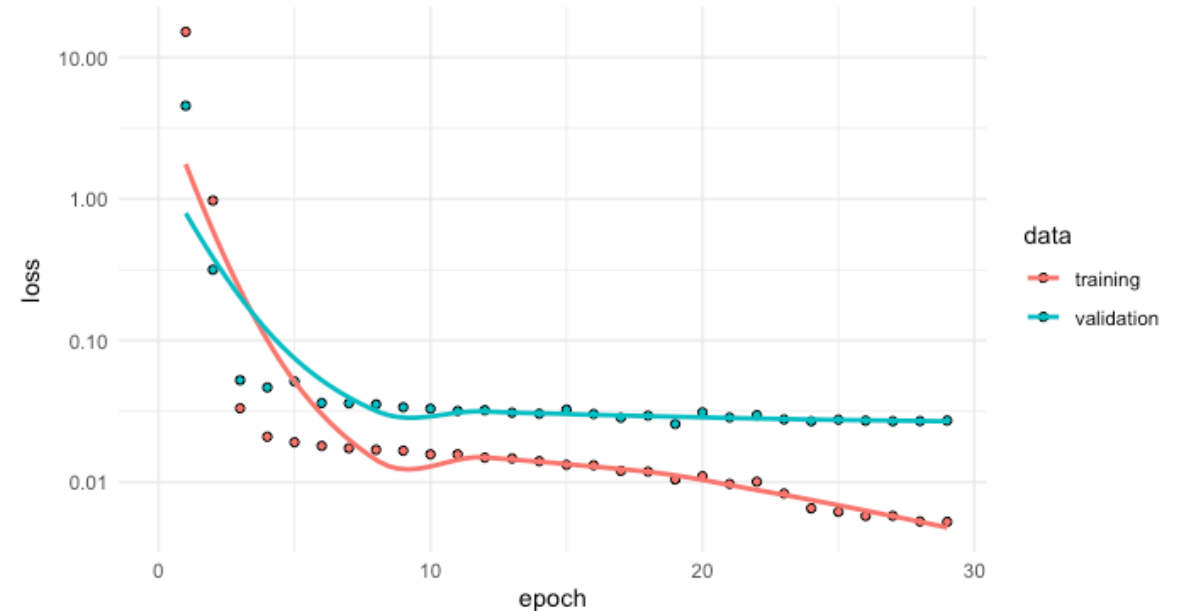
Model debugging

- Minimal overfitting
 - Early stopping as soon as the validation curve does not improve further



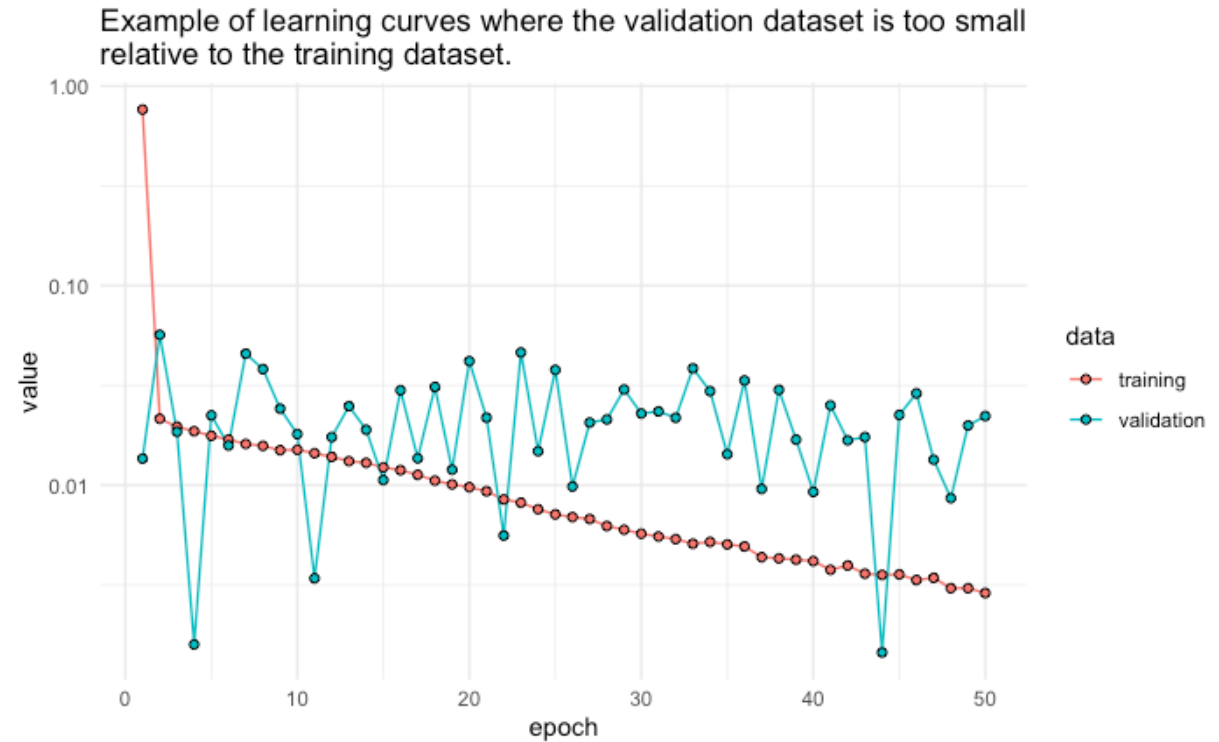
Model debugging

- Non representative training data set
 - Training data set has too few samples compared to the evaluation set
 - Features have less variance in the training set than in the eval set
 - Set could be ordered
 - for CNN's: Data Augmentation
 - Random Sampling Train/Test Data
 - Cross Validation



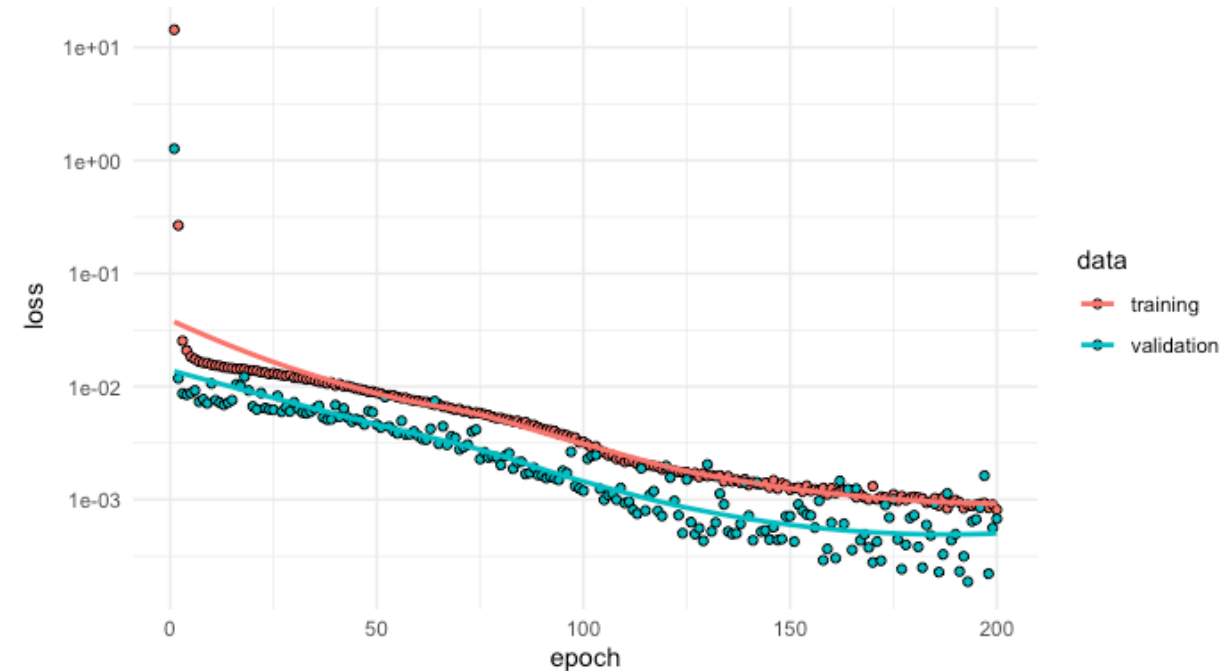
Model debugging

- Non representative validation set
 - Validation set is too small
 - Validation Set
 - Change division train/eval
 - Cross Validation



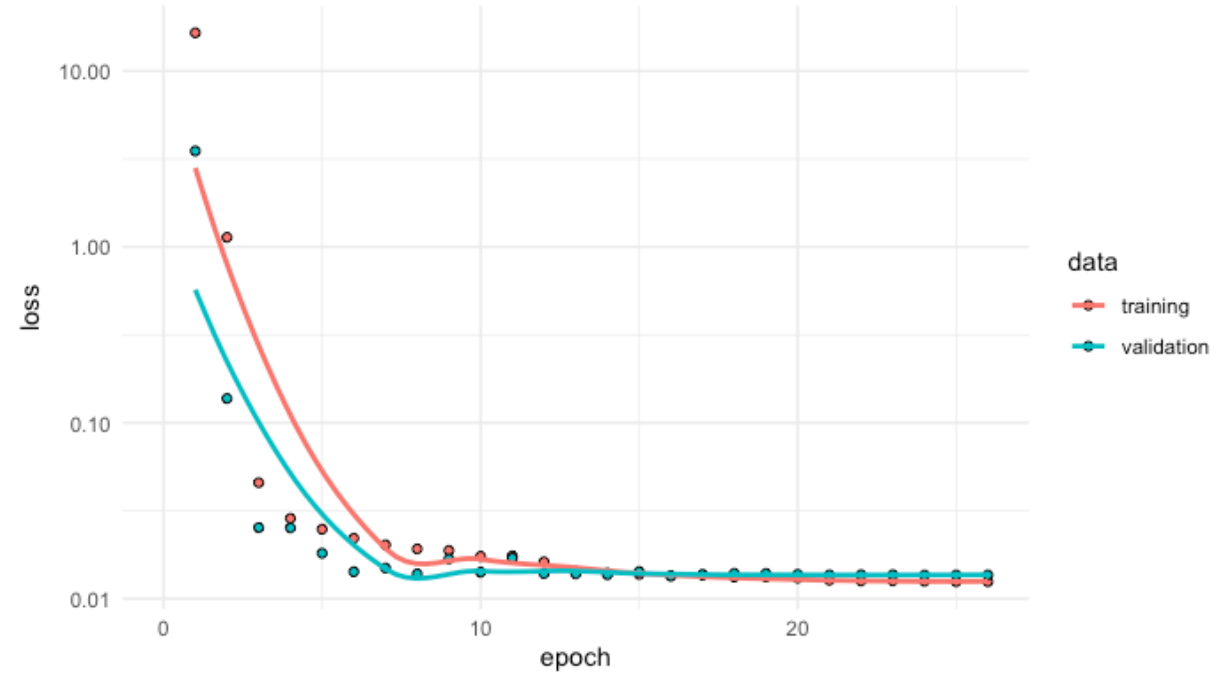
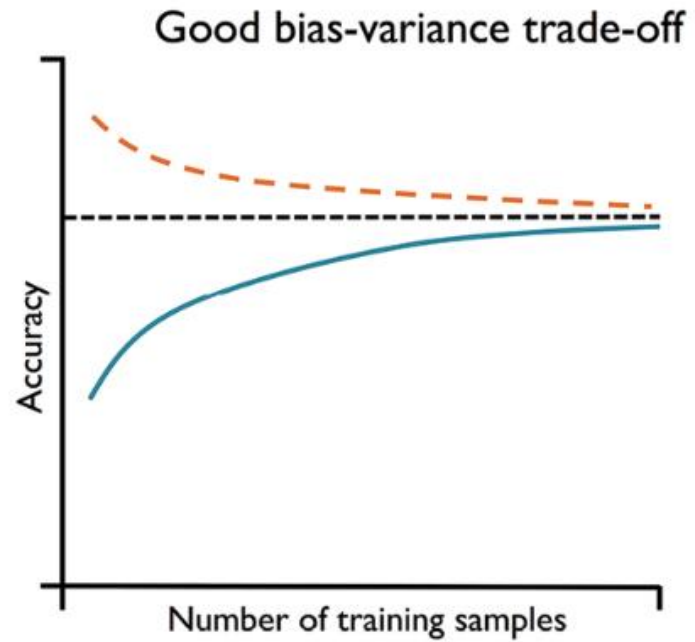
Model debugging

- Information leakage
 - Validation loss is consistently lower than training loss
 - Data from the training set has seeped into the validation set
 - E.g. due to duplicate entries
 - E.g. due to poor sampling strategy
- Find information leak
- Use cross validation



Model debugging

- Optimal Fit



Summary

- Data must be prepared for processing in NN
- There are several activation functions in the NN
- In training, the loss function is minimized
- Debugging can be performed on the basis of the evaluation curves