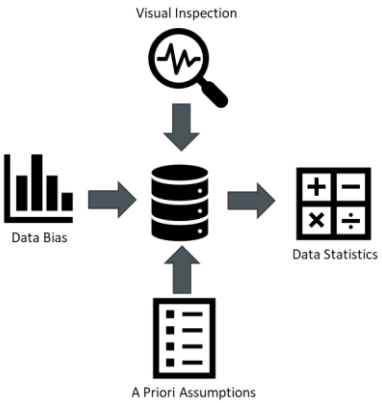
DEEP LEARNING

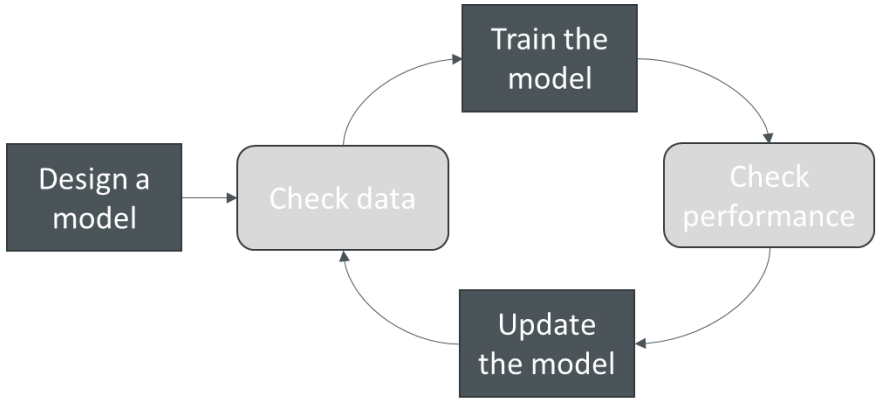
HERZLICH WILLKOMMEN ZUR 2. VORLESUNG „DEEP LEARNING“

Prof. Dr.-ing. Niklas Beuter, TH Lübeck

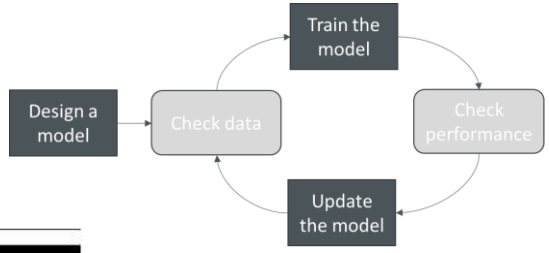
Repetition

- We learned that deep learning is a powerful tool to enhance a lot of applications in diverse areas
- Deep learning is based on data, which needs to represent the problem
  - You need to inspect the data by calculating statistics and visualize it
- The model performs a suitable transformation of the data into an embedding space to use downstream tasks on it.

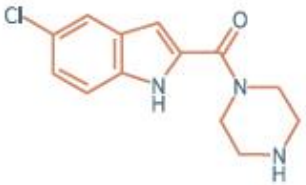




How does the data look like?

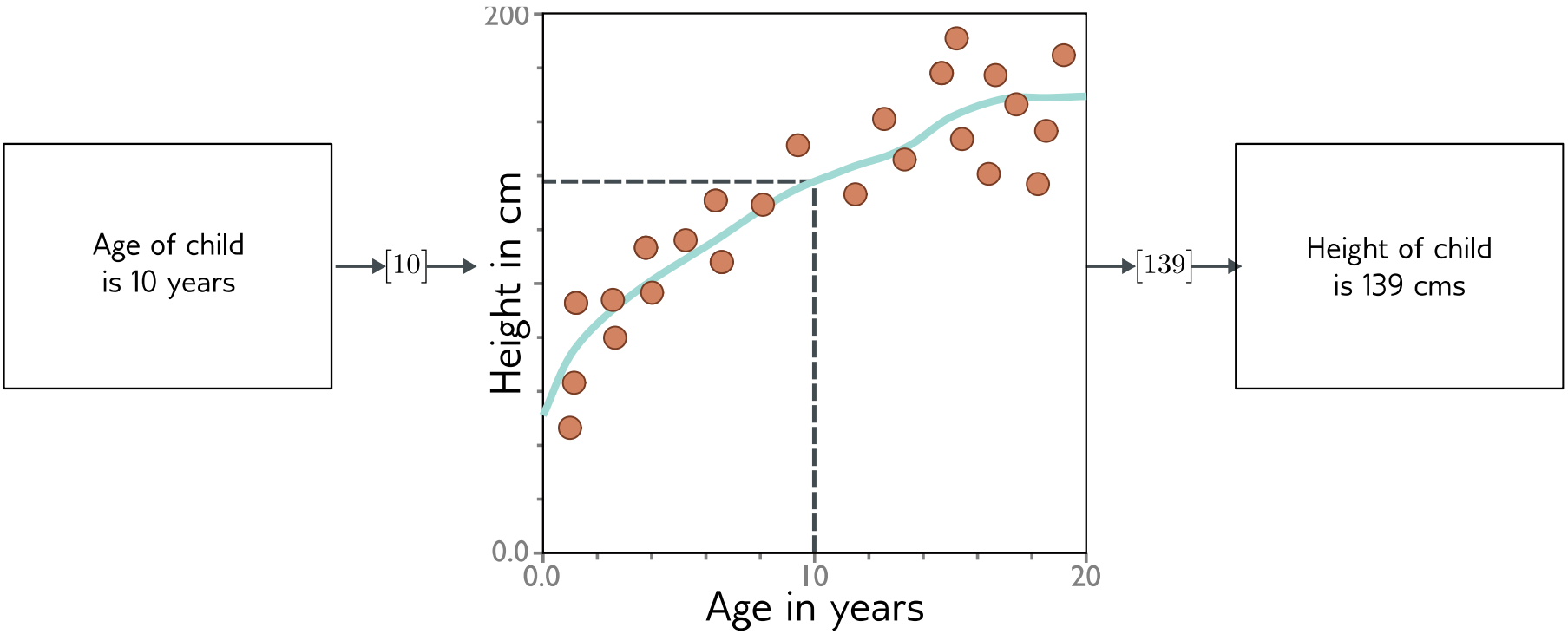


6000 square feet,
4 bedrooms,
previously sold for
$235K in 2005,
1 parking spot.



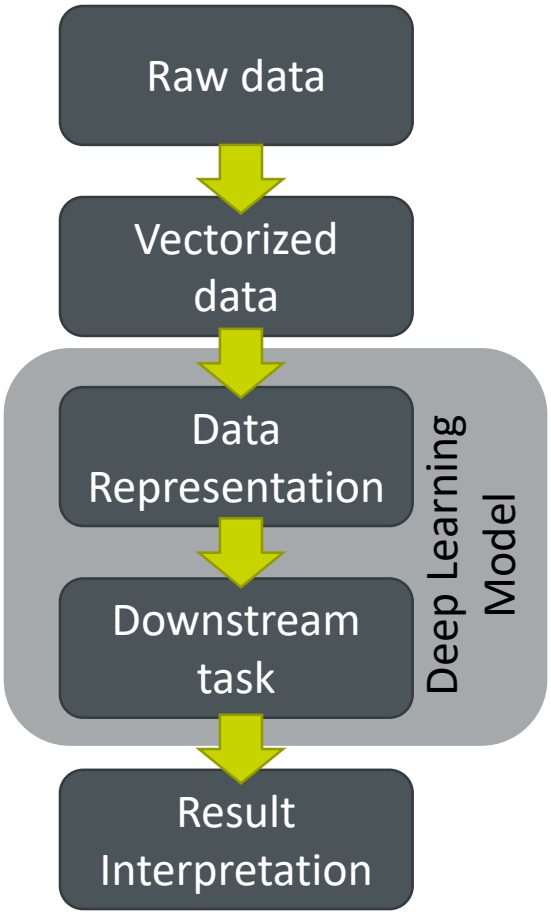
"The steak was terrible,
the salad was rotten, and
the soup tasted like socks"

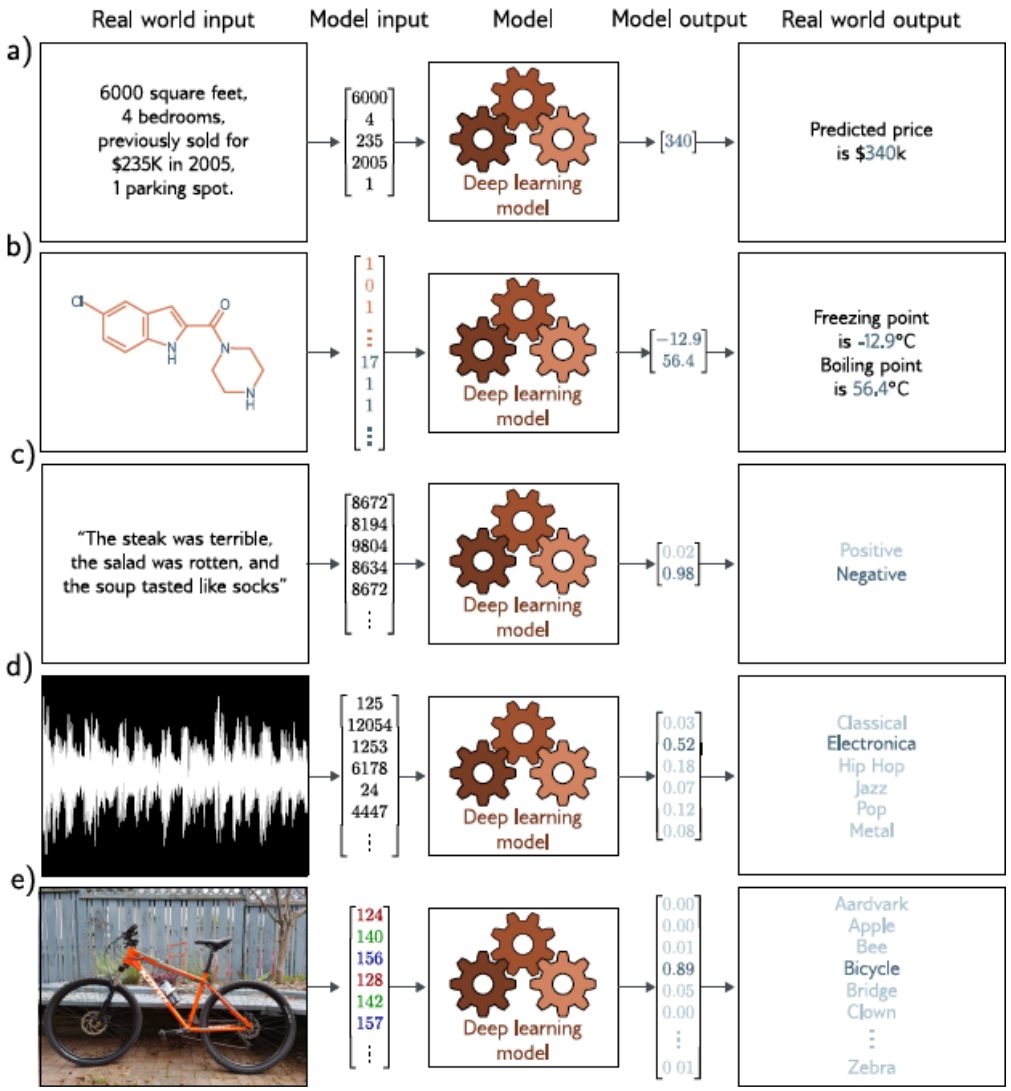
Deep Learning Model



- An equation relating input (age) to output (height)
- Search through family of possible equations to find one that fits training data well
- Deep learning is a high dimensional family of equations

Deep Learning - Model

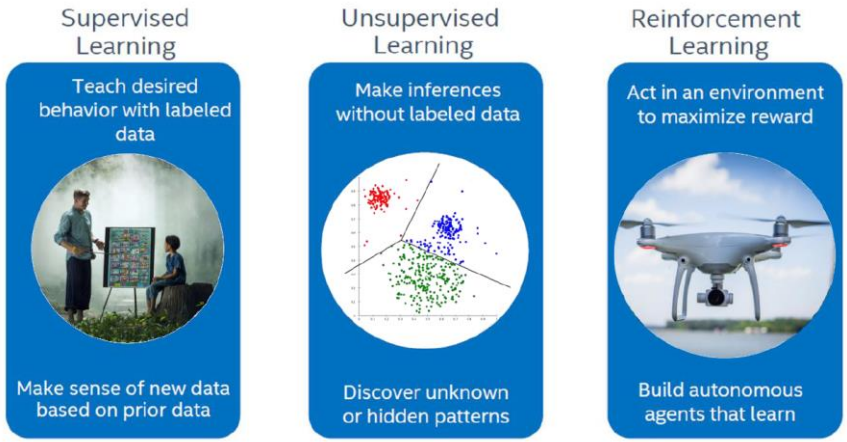




Definitions

Definition

- Supervised learning = Learn from input/output examples
- Unsupervised learning = Learn hidden structures from unlabeled data
- Reinforcement learning = Learn from actions and their rewards

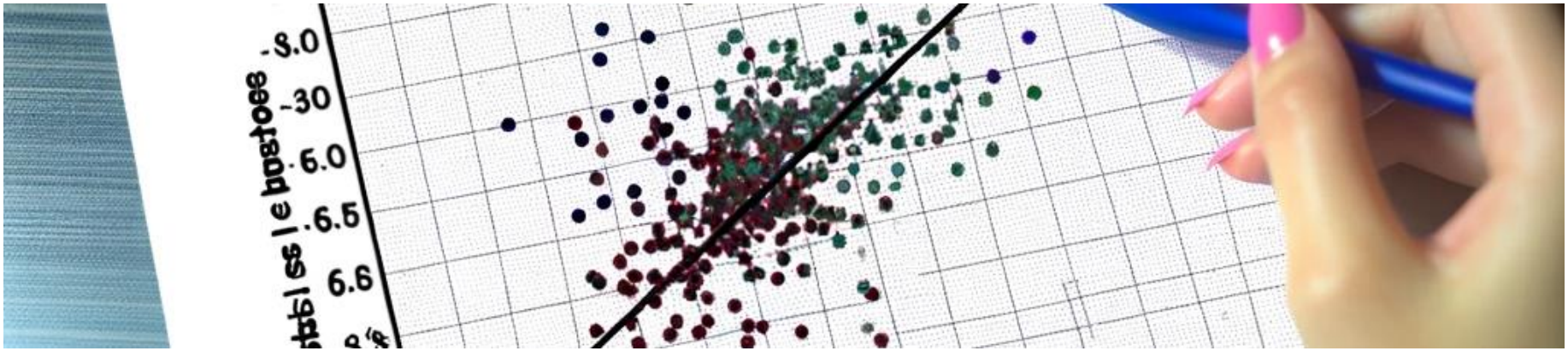


Definitions

Definition

- Classification = discrete classes as output
- Regression = continuous numbers as output
- Two class and multiclass classification treated differently
- Univariate = one output
- Multivariate = more than one output

TECHNISCHE HOCHSCHULE LÜBECK



LINEAR REGRESSION

First steps into deep learning

TECHNISCHE HOCHSCHULE LÜBECK

Price Prediction

Question

- You would like to sell your car.
- What would you do to set the price?

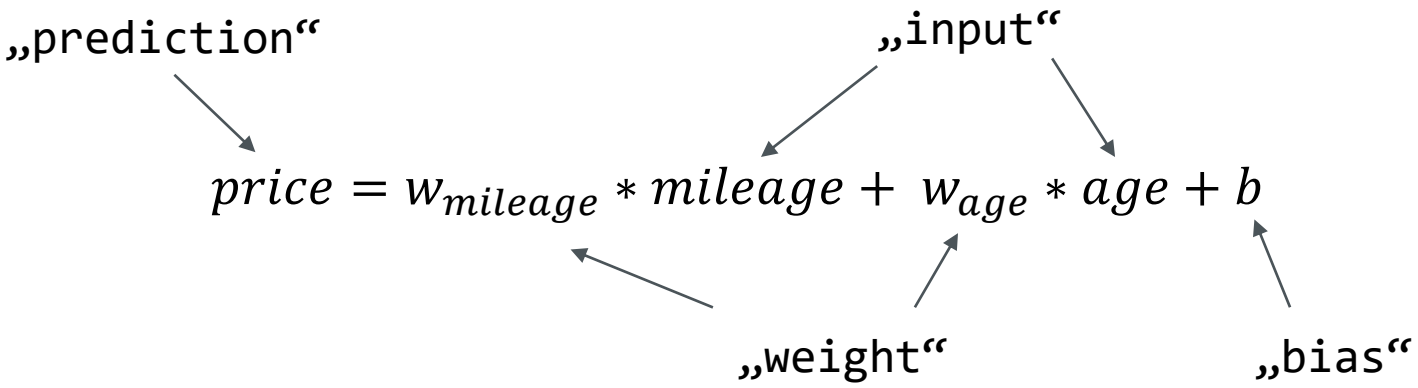
Linear Regression

Linear regression is a simple machine learning model, but helpful to understand the design of deep learning models (dates back to 19th century)

- **Basic Principle**: Linear regression is a statistical method used to *model the relationship* between a *dependent variable* (e.g., car sale price) and one or more *independent variables* (e.g., mileage, age) *by finding a straight line* (the regression line) that best describes the data points.
- **Objective**: The aim of linear regression is to *create an equation* that can predict the dependent variable as a function of the independent variables, allowing *for future values of the dependent variable* to be estimated *based on new values of the independent variables*.

$$price = y = f(x)$$

Linear Regression



- In easy cases with few inputs, use a weighted sum + bias. This calculation is called *„inference“*
- In order to *train* the model, we adapt the *weights*

$$\hat{y} = w_1 * x_1 + w_2 * x_2 + \cdots + w_d * x_d + b$$

- With $x \in \mathcal{R}^d, w \in \mathcal{R}^d$

Linear Regression

With $\boldsymbol{x} \in \mathcal{R}^d, \boldsymbol{w} \in \mathcal{R}^d$ *as vectors*

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_d \end{bmatrix}, \boldsymbol{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_d \end{bmatrix}$$

We can write it simply as:

$$\hat{y} = \boldsymbol{w}^T \boldsymbol{x} + b$$

Scalar / Vector / Matrix

- In machine / deep learning we handle a lot of data and network parameters
- In order to have an efficient representation, the calculations are based on vector / matrix computations
  - Scalar = 1,32
  - Vector = $\begin{bmatrix} 3 & 4 & -1 \end{bmatrix}$
  - Matrix = $\begin{bmatrix} 3 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 4 & \cdots & 2 \end{bmatrix}$
- Computation:
  - Sum, Sub, Mul, Diff
  - Dot-Product (Hadamard Product)

Vector / Matrix - Transpose

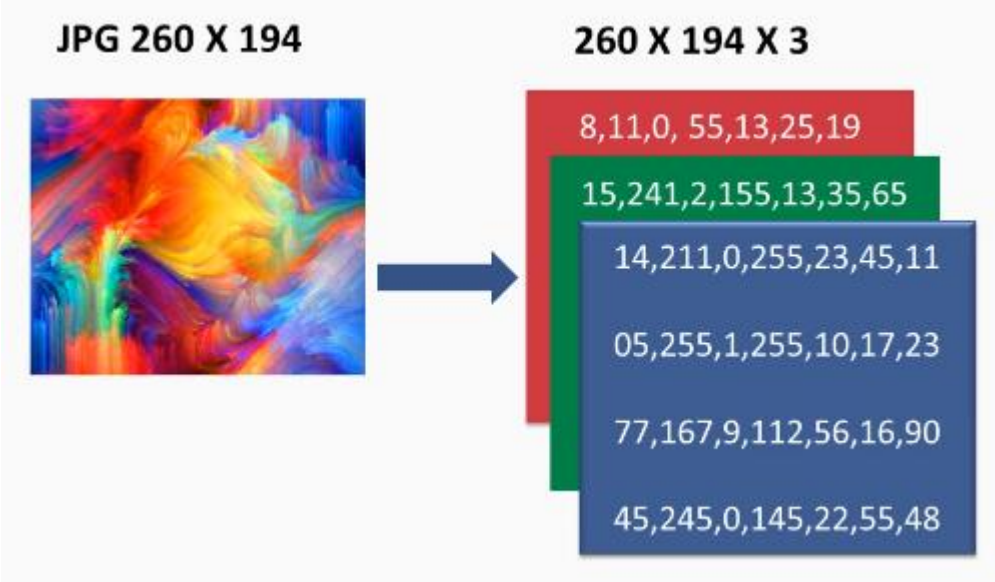
- For vector / matrix computation we need to keep care about the correct representation (row or column vectors)
- In order to change from column to row, we use *transpose*
  - $\text{Vector} = \begin{bmatrix} 3 & 4 & -1 \end{bmatrix}$ oder $Vector^T = \begin{bmatrix} 3 \\ 4 \\ -1 \end{bmatrix}$
  - $\text{Matrix} = \begin{bmatrix} 3 & 1 & 0 \\ 5 & 1 & 2 \end{bmatrix}$ oder $\text{Matrix}^T = \begin{bmatrix} 3 & 5 \\ 1 & 1 \\ 0 & 2 \end{bmatrix}$

Tensors

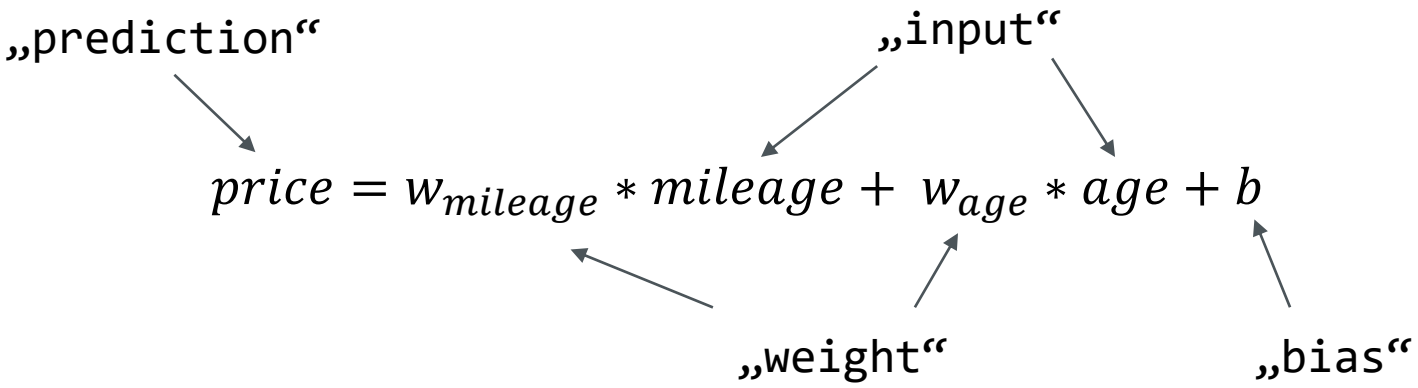
- For machine learning all forms of scalar, vector or matrix are represented in *tensors*
- A tensor could be a single value (scalar), a vector or a matrix
  - $x = torch.tensor(3.0)$
  - $\boldsymbol{v} = torch.tensor([1, 2, 3])$
  - $\boldsymbol{A} = torch.tensor([[1, 2, 3], [2, 0, 4], [3, 4, 5]])$
- All calculations of the linear Algebra are possible with a tensor
  - We are also allowed to directly compute following values of the tensor
    - Sum, mean, norm, abs, …
- Tensors can have arbitrary numbers of dimensions

Tensors (for images)

- Tensors can have arbitrary numbers of dimensions
- A single image is a tensor of 3^{rd} order (width, height, channel [r,g,b])
- A collection of images is a tensor of 4^{th} order (imageNr, width, height, channel)



Linear Regression



- In easy cases with few inputs, use a weighted sum + bias. This calculation is called *„inference“*
- In order to *train* the model, we adapt the *weights*

$$\hat{y} = w_1 * x_1 + w_2 * x_2 + \cdots + w_d * x_d + b$$

- With $x \in \mathcal{R}^d, w \in \mathcal{R}^d$

Linear Regression

Definition

- Linear regression takes an input vector x (data describing the input) and produces an output scalar $\hat{y}$ based on a weighted sum using a weight vector w^T. The bias b determines the value, if all features are 0.

$$\hat{y} = \boldsymbol{w}^T \boldsymbol{x} + b$$

- The weights are trained on previously seen data

- Usually, we also want to provide multiple input data simultaneously
- Hence, stacking all examples n into an input matrix $\boldsymbol{X} \in \mathcal{R}^{nxd}$ leads to (one row per example, features per column)

$$\hat{y} = \boldsymbol{w}^T \boldsymbol{X} + b \quad \text{or} \quad \hat{y} = \boldsymbol{X}\boldsymbol{w} + b$$

Notation throughout the course

TECHNISCHE HOCHSCHULE LÜBECK

Definition

- We will notate individual samples by an upperscript for the $(i)^{th}$ example

$$\hat{y}^{(i)} = \boldsymbol{w}^T \boldsymbol{x}^{(i)} + b$$

- We will notate individual features of a vector or matrix with lowerscript

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \end{bmatrix}$$

Predict new prices?

Question

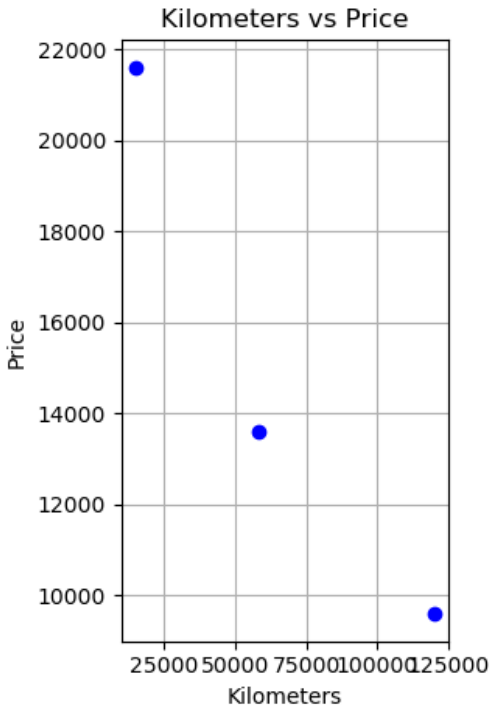
- Now we have a function for our model:

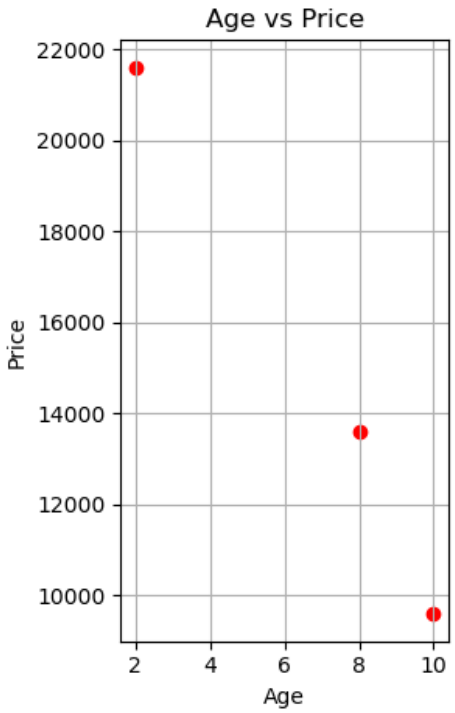
$$\hat{y} = \boldsymbol{w}^T \boldsymbol{x} + b$$

- What are the next steps to achieve a prediction on car prices?

How does the data look like?

- For supervised learning we need pairs of data $\mathrm{I} = \{y^{(i)}, \boldsymbol{x}^{(i)}\}$
- In our case:
  - Car1: $\boldsymbol{x}^{(1)} = [120.000, 10], y^{(1)} = 9.600$
  - Car2: $\boldsymbol{x}^{(2)} = [15.000, 2], y^{(2)} = 21.600$
  - Car3: $\boldsymbol{x}^{(3)} = [58.000, 8], y^{(3)} = 13.600$
  - …





Quality Measure – Loss Function

TECHNISCHE HOCHSCHULE LÜBECK

Definition

- A **loss function** determines the difference/distance between real value and estimated value (error of estimation)
  - Usually, the loss is a positive value, where closer to 0 means better
  - A simple loss function is the squared error

$$l^{(i)}(\boldsymbol{w}, b) = \frac{1}{2}(\hat{y}^{(i)} - y^{(i)})^2$$

- The entire loss is calculated as sum over the single losses:

$$L(\boldsymbol{w}, b) = \frac{1}{n}\sum_{i=1}^{n} l^{(i)}(\boldsymbol{w}, b) = \frac{1}{n}\sum_{i=1}^{n} \frac{1}{2}(\boldsymbol{w}^T x^{(i)} + b - y^{(i)})^2$$

Training using Loss function

Definition

- Training (finding parameters (w^*, b^*)) means minimizing the **Loss**:

$$(\boldsymbol{w}^*, b^*) = \underset{w,b}{\operatorname{argmin}} L(\boldsymbol{w}, b)$$

- For linear regression there are also analytic solutions
- We focus on the machine learning way of stochastic gradient descent

Stochastic gradient descent

TECHNISCHE HOCHSCHULE LÜBECK

Question

- How would you search for the lowest point in a park during dense fog?
  - Assume you have a map
  - Assume you do not have a map

Gradient descent

- Loss per input: $l^{(i)}(\boldsymbol{w}) = \frac{1}{2}(\hat{y}^{(i)} - y^{(i)})^2 \; with \; w_0 = b$
- Full loss: $L(\boldsymbol{w}) = \frac{1}{n}\sum_{i=1}^{n} l^{(i)}(\boldsymbol{w}) = \frac{1}{n}\sum_{i=1}^{n} \frac{1}{2}(\boldsymbol{w}^T x^{(i)} + w_0 - y^{(i)})^2$
- Optimization criterion for parameters $(\boldsymbol{w}^*) = \underset{w}{\operatorname{argmin}} L(\boldsymbol{w})$

- Idea:
  - $\frac{\partial_L}{\partial_{\boldsymbol{w}}} = \begin{bmatrix} \frac{\partial_L}{\partial_{w_0}} \\ \frac{\partial_L}{\partial_{w_1}} \\ \vdots \\ \frac{\partial_L}{\partial_{w_n}} \end{bmatrix} = \frac{\partial}{\partial_{\boldsymbol{w}}} \sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} l^{(i)} = \sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} \frac{\partial_{l^{(i)}}}{\partial_{\boldsymbol{w}}}$

Gradient descent - Example

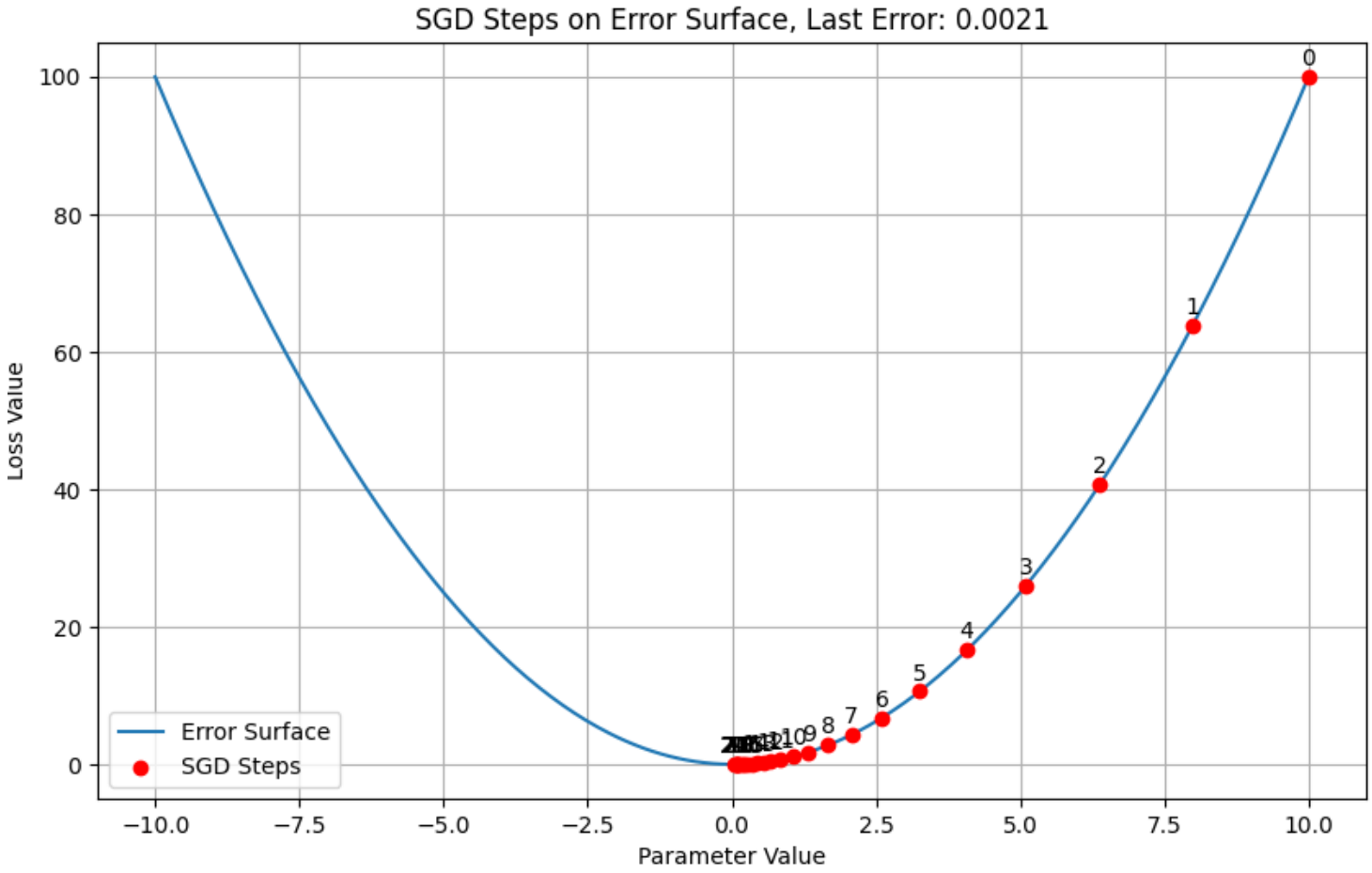
- Idea:
  - $\frac{\partial_L}{\partial_{\boldsymbol{w}}} = \begin{bmatrix} \frac{\partial_L}{\partial_{w_0}} \\ \frac{\partial_L}{\partial_{w_1}} \\ \vdots \\ \frac{\partial_L}{\partial_{w_n}} \end{bmatrix} = \frac{\partial}{\partial_{\boldsymbol{w}}} \sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} l^{(i)} = \sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} \frac{\partial_{l^{(i)}}}{\partial_{\boldsymbol{w}}}$

- Example with linear regression $(w_0 = b, w_1)$
  - $l^{(i)} = \frac{1}{2}(w_1 x^{(i)} + w_0 - y^{(i)})^2$
  - $\frac{\partial_{l^{(i)}}}{\partial_{w_0}} = \left(w_1 x^{(i)} + w_0 - y^{(i)}\right)$
  - $\frac{\partial_{l^{(i)}}}{\partial_{w_1}} = x^{(i)}(w_1 x^{(i)} + w_0 - y^{(i)})$

Gradient descent – Update and Epoch

- Example with linear regression $(w_0 = b, w_1)$
  - $l^{(i)} = \frac{1}{2}(w_1 x^{(i)} + w_0 - y^{(i)})^2$
  - $\frac{\partial_{l^{(i)}}}{\partial_{w_0}} = \left(w_1 x^{(i)} + w_0 - y^{(i)}\right)$
  - $\frac{\partial_{l^{(i)}}}{\partial_{w_1}} = x^{(i)}(w_1 x^{(i)} + w_0 - y^{(i)})$
- Update the parameter like following:
  - $w_0 = w_0 - \frac{\eta}{|n|}\sum_{i=1}^{n} \frac{\partial_{l^{(i)}}}{\partial_{w_0}}$
  - $w_1 = w_1 - \frac{\eta}{|n|}\sum_{i=1}^{n} \frac{\partial_{l^{(i)}}}{\partial_{w_1}}$
- Do this t-times, for t epochs. In each epoch present all training examples once

Gradient Descent – Illustration for x^2



(Stochastic) Gradient Descent

- Gradient Descent takes the full training set at once for training
  - Derivate of the loss function as average over all losses on all examples
  - Very efficient, but not good on e.g. redundant data

- Stochastic Gradient Descent (SGD) could take one example
  - Derivate of the loss function as over loss of one example
  - Adds noise to the update step due to smaller input set
  - Very good for large datasets and redundant data, but slower

- Stochastic Gradient Descent (SGD) takes a batch of examples
  - Derivate of the loss function as average over losses of B examples, with $B \in \mathbb{N}^+$
  - Very good for large datasets and redundant data, a good compromise

Minibatch Stochastic Gradient Descent

TECHNISCHE HOCHSCHULE LÜBECK

Definition

- We pick a minibatch B, compute the gradient of the average loss with respect to the model parameters
- We then normalize it with the size of the minibatch and multiply it with a learning rate η
- In each epoch t we update using the data as minibatches (/wo repetition)

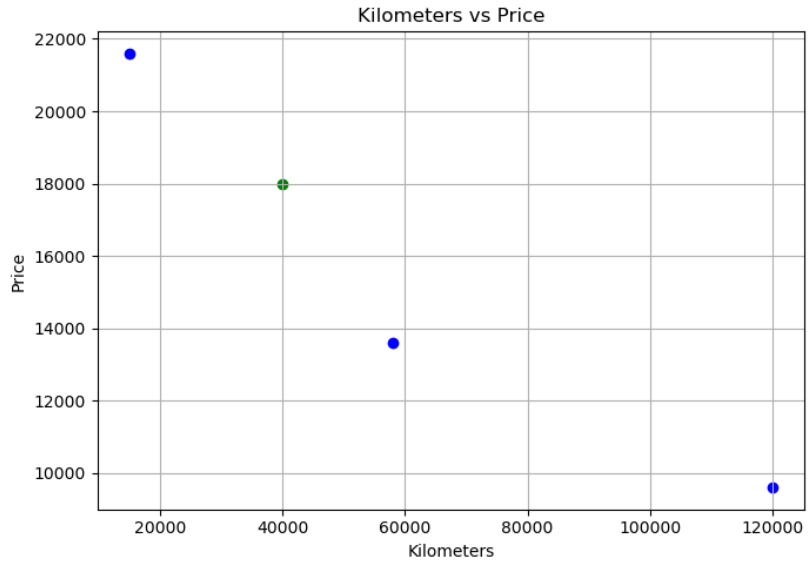
$$\boldsymbol{w} = \boldsymbol{w} - \frac{\eta}{|\boldsymbol{B}|}\sum_{i\epsilon B_t} \partial_{\boldsymbol{w}} l^i(\boldsymbol{w}, b) = \boldsymbol{w} - \frac{\eta}{|\boldsymbol{B}|}\sum_{i\epsilon B_t} \boldsymbol{x}_i\left(\boldsymbol{w}^T\boldsymbol{x}^{(i)} + b - y^{(i)}\right)$$

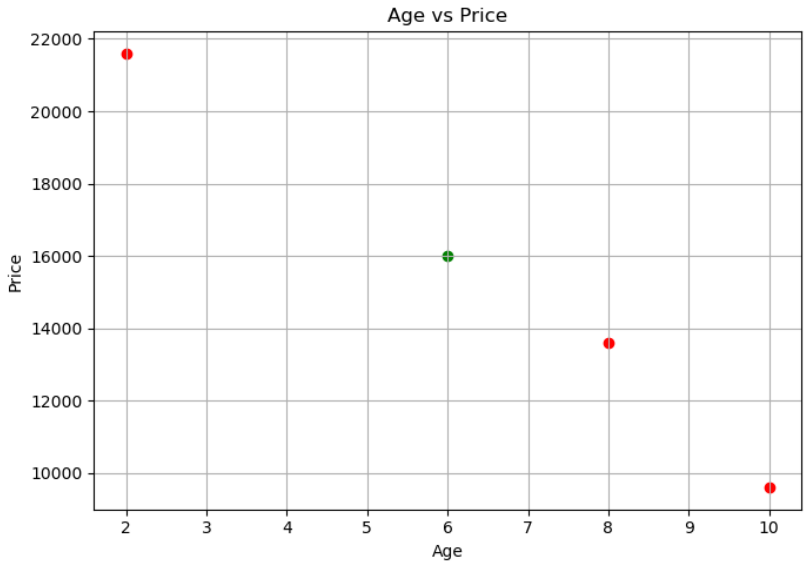
$$b = b - \frac{\eta}{|\boldsymbol{B}|}\sum_{i\epsilon B_t} \partial_b l^i(\boldsymbol{w}, b) = b - \frac{\eta}{|\boldsymbol{B}|}\sum_{i\epsilon B_t} \left(\boldsymbol{w}^T\boldsymbol{x}^{(i)} + b - y^{(i)}\right)$$

- Train till criterion is met or number of iterations t is finished

Prediction / Inference

- After training the parameters on real data, we can start the *prediction*. In machine learning terminology we call this mostly *inference* (although it is not absolutely correct for statistics)
- Take a new input vector with car features and predict a price for it





Demo linear regression

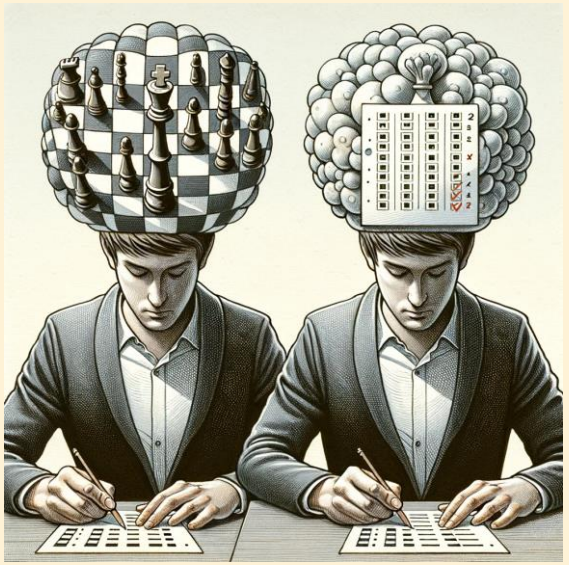
- Predict car prices

Generalization

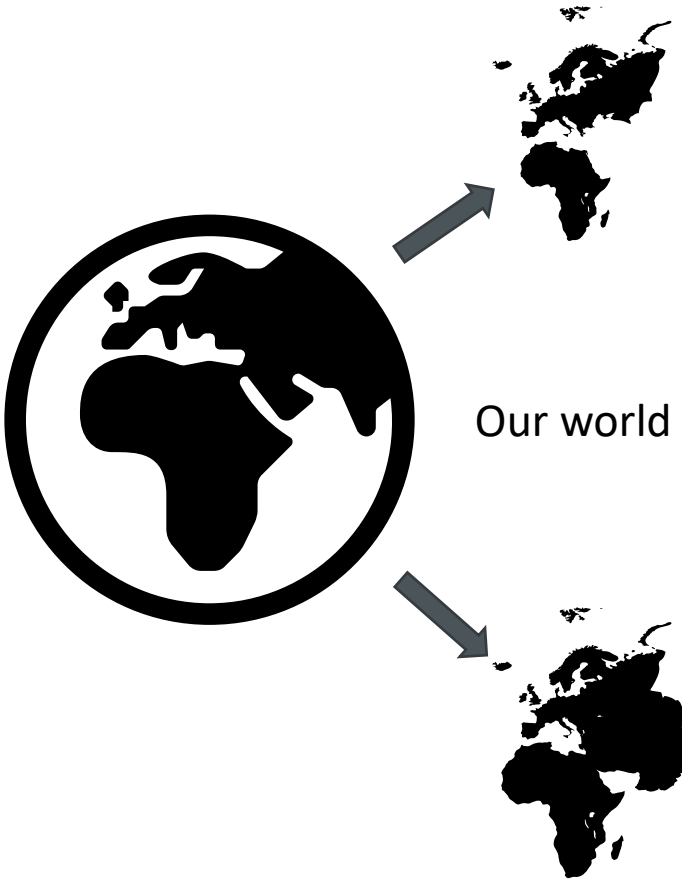
Question

- Who do you think would perform better in a test?
  - A person who remembers everything (and learns from the old available exams), but can't answer any other new question
  - A person who remembers nearly nothing, but is very good in recognizing and understanding patterns

- It depends on the kind of test:
  - Who is better in a complete recycled exam?
  - Who is better in a complete new exam with fresh questions?



Generalization



We pick examples for training = Distribution Q

Our world of data = Distribution T

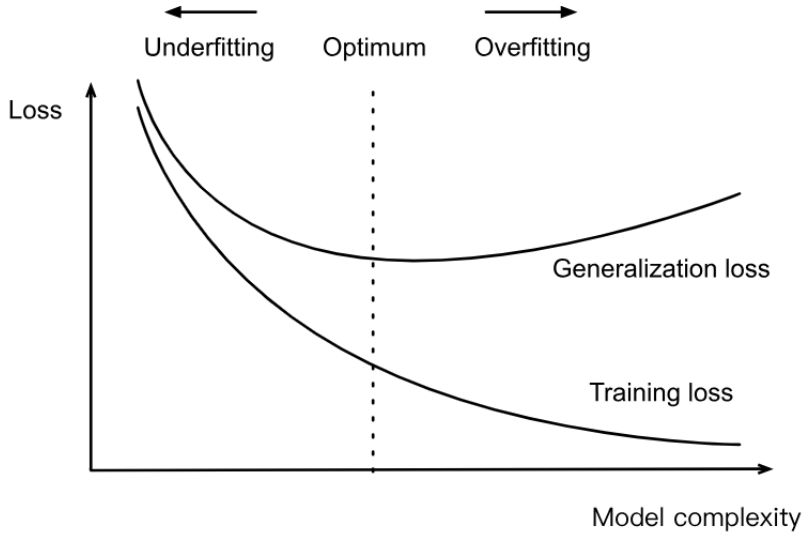
We pick examples for testing = Distribution P

Training and Test data should be drawn Independently from Identical Distributions (IID)

Goal: Q = P (=T)

Overfitting / Underfitting

- We use a training set with training error R_Q and a validation set with validation error R_P
- The difference between R_Q and R_P describes the generalization error which shall be as small as possible to enable good results on new data
- Model and data need to fit to each other



3.6. Generalization — Dive into Deep Learning 1.0.3 documentation (d2l.ai)

Model selection / Data set size

- More data is always better. With less data simpler models outperform deep learning models. Be aware of over- and underfitting!
- In best cases we use a 3-split of data for model selection
  - Training, test and validation data
  - Helps to select a good model but avoids to just purely select the data based on train and test results
- In case data size is scarce K-fold cross-validation is a good choice
  - Split data is K non-overlapping subsets
  - Execute model training and validation K-times (K-1 for training, 1 for validation)
  - Average training and validation error over all K experiments

Generalization – Weight Decay

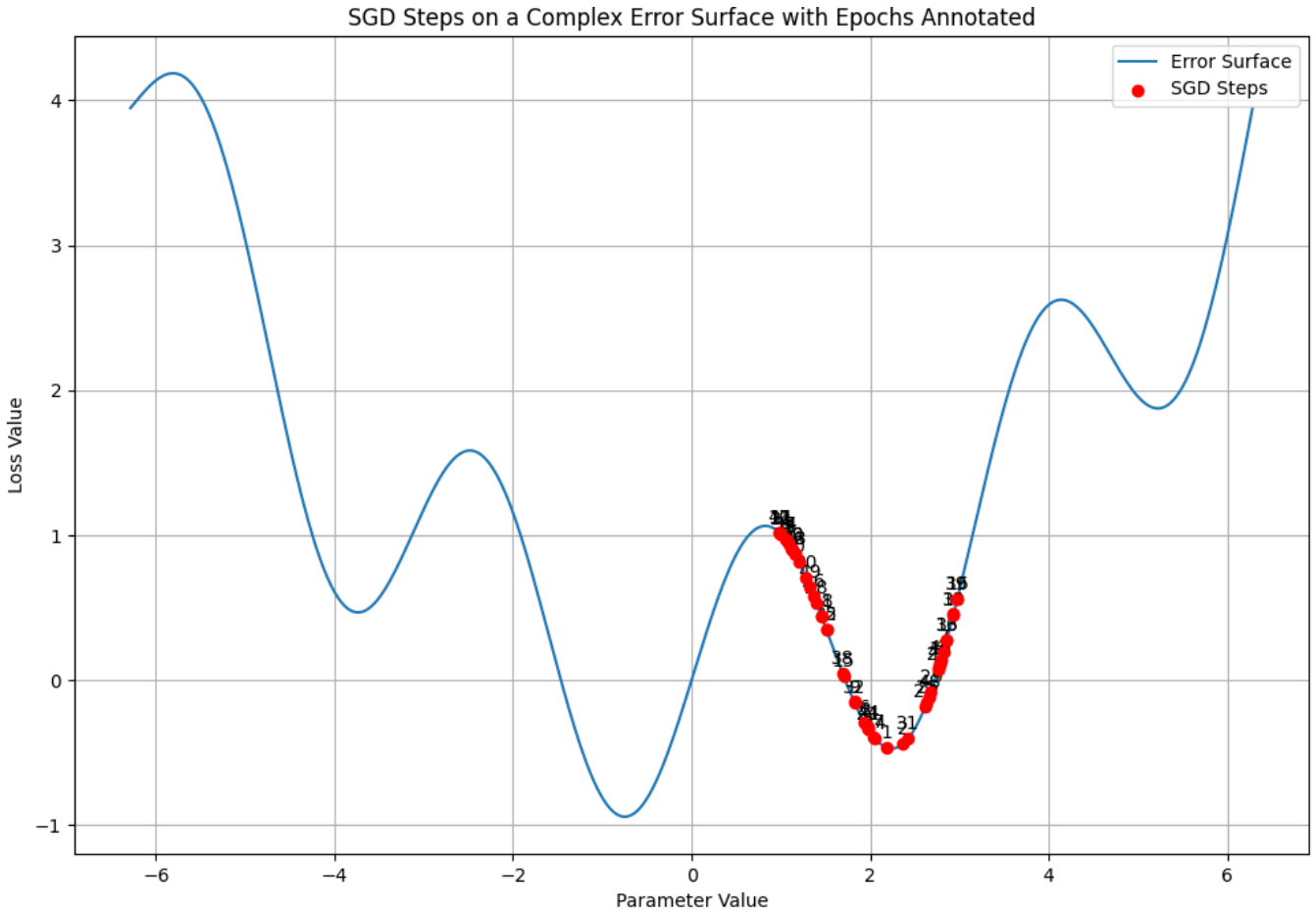
- In order to avoid overfitting there are more generalization techniques than adding more data

- Weight decay adds a penalty term to the loss
  - Forces the optimizer to create small weights, which creates more smooth models, which generalize better
  - Use hyperparameter λ to control the weight decay

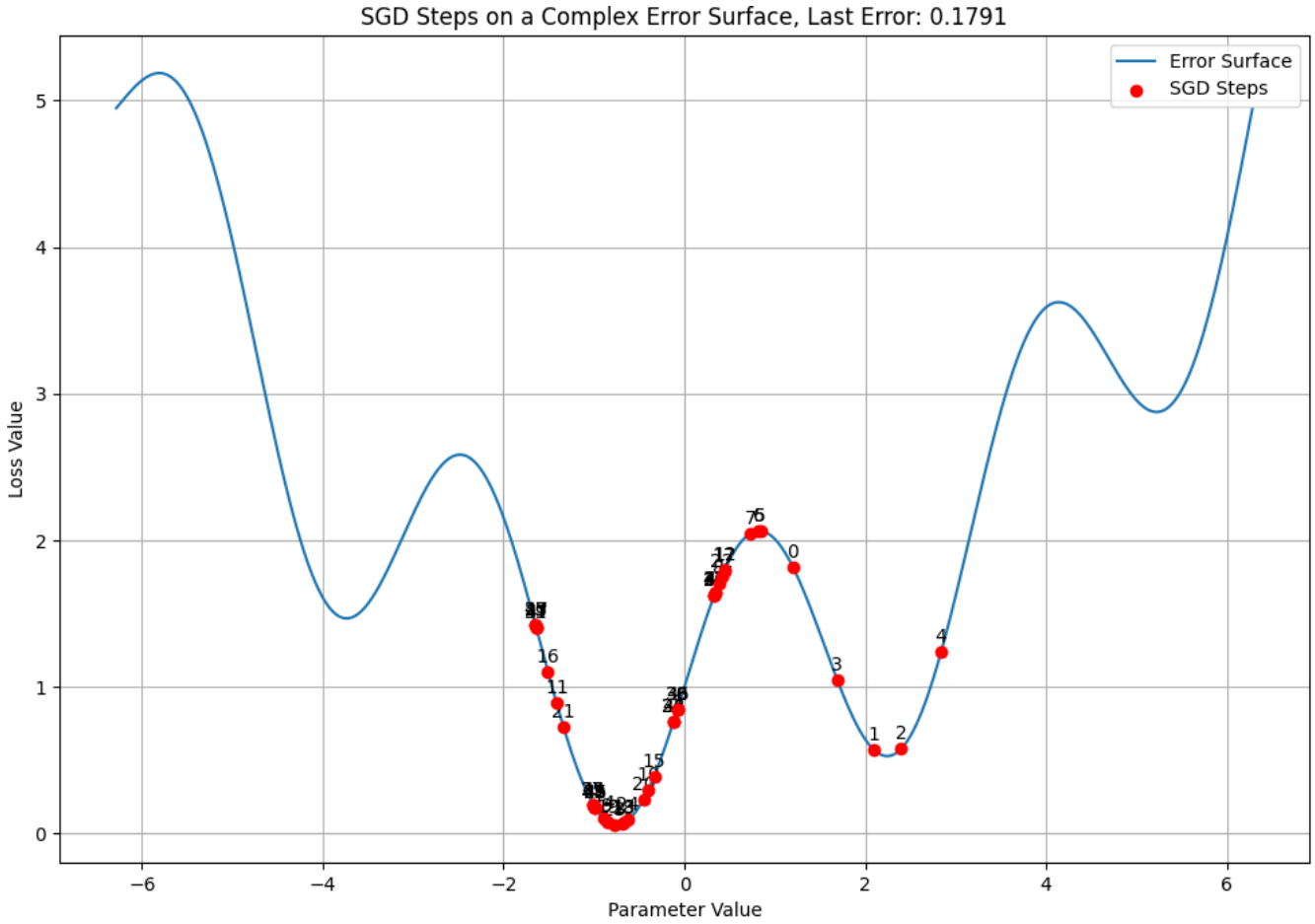
  - $Loss_{total} = Loss_{original} + \frac{\lambda}{2}\sum_i w_i^2$

  - This weight decay is equivalent to L2 regularization

SGD without weight decay -



SGD with weight decay



Noise in SGD – Dynamic learning rate

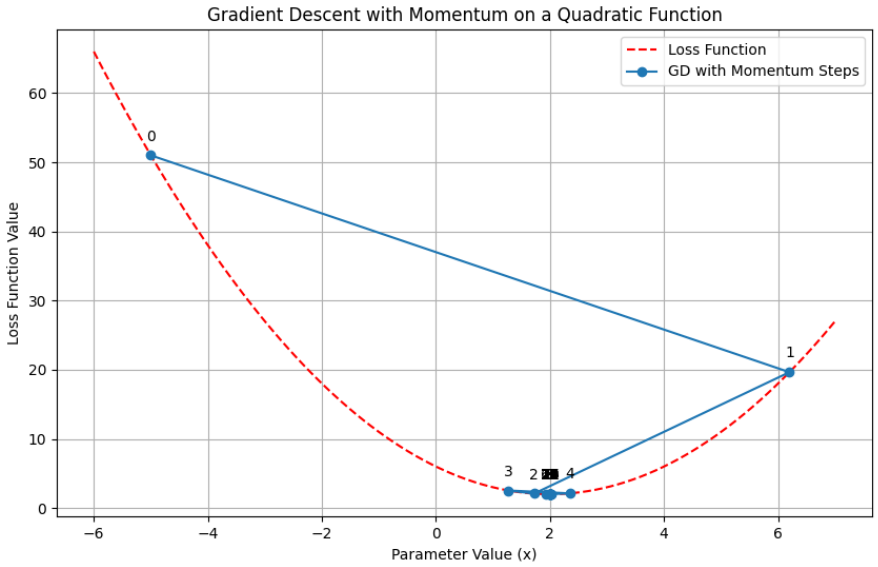
- Training with SGD is more noisy than using GD
- Adapting the learning rate leads to more smooth or more noisy jumps
  - But, for deep learning both is needed
- One solution is to adapt the learning rate dynamically
  - One possibility:
    - Decrease learning rate by time
  - Another option:
    - Decrease learning rate, if optimization stalls
  - Many other options are possible

SGD - Momentum

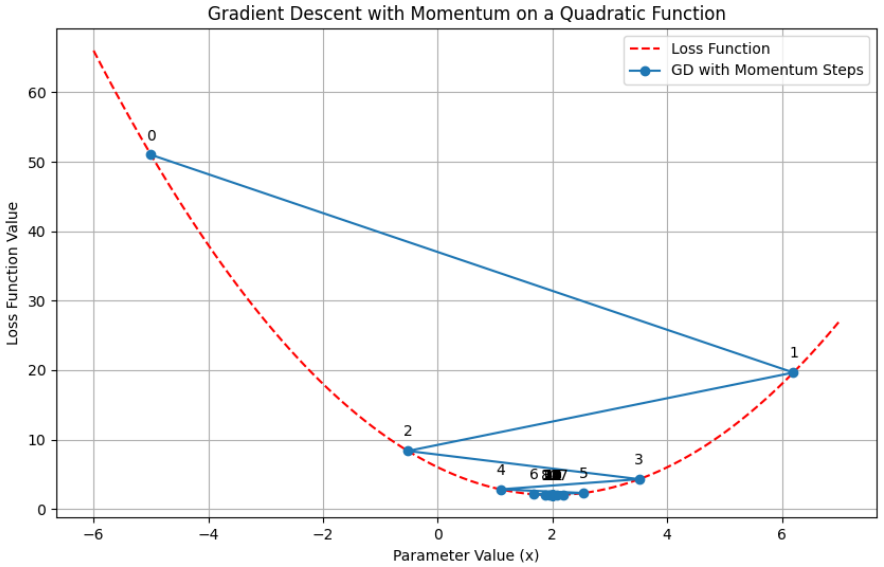
- Adding a momentum term helps to smooth the gradient over time
- $m_{t+1} \leftarrow \beta * m_t + (1-\beta) \sum_{i \epsilon B_t} \frac{\partial_{l^{(i)}}}{\partial_{\boldsymbol{w}}}$
- $\boldsymbol{w}_{t+1} \leftarrow \boldsymbol{w}_t - \eta * m_{t+1}$
    - With m_t as Momentum, $\beta \epsilon [0,1)$
- The momentum m_t accumulates past gradients and averages them weighted into a smoothed update. $m_t \; with \; t = 0$ initiates with 0.

SGD - Momentum

SGD with momentum



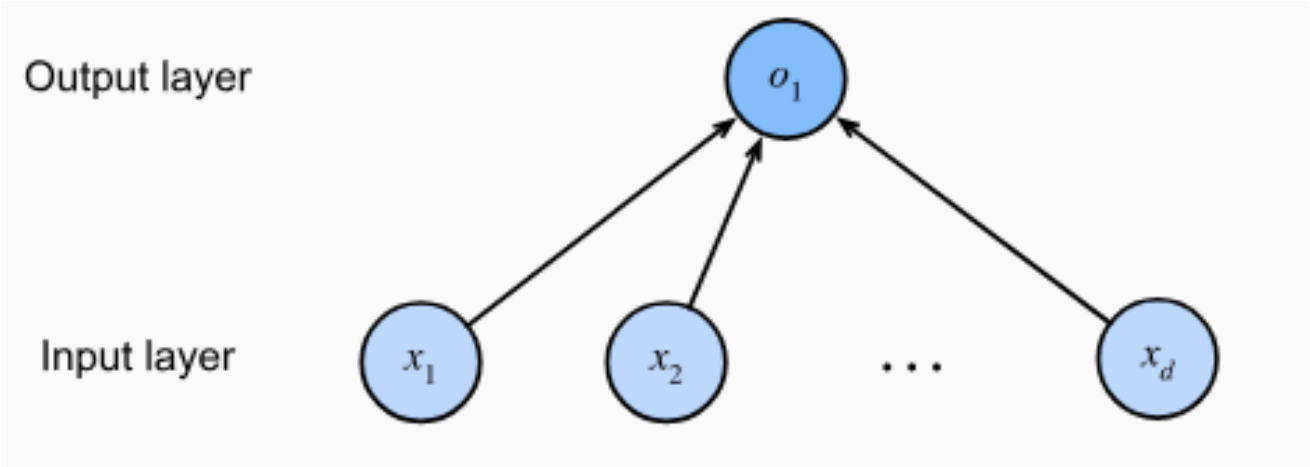
SGD without momentum



Adagrad, RMSProp, Adadelta, Adam

- There are many more optimizations of the learning rate
  - Will be handled in detail in the “Praktikum”

Linear regression as Neural Network



Summary

- **Loss Function:** A loss function measures the discrepancy between the predictions of the neural network and the actual target values. The goal of training is to minimize this loss.
- **Gradient:** The gradient of the loss function with respect to a weight in the network represents how much a small change in that weight would change the loss. Essentially, it tells us the direction to adjust our weights to minimize the loss.
- **SGD:** Stochastic Gradient Descent (SGD) is a fundamental optimization algorithm used in training deep neural networks. The derivative calculation in SGD, especially for deep neural networks, involves the backpropagation algorithm, which efficiently computes gradients of the loss function with respect to the weights of the network.