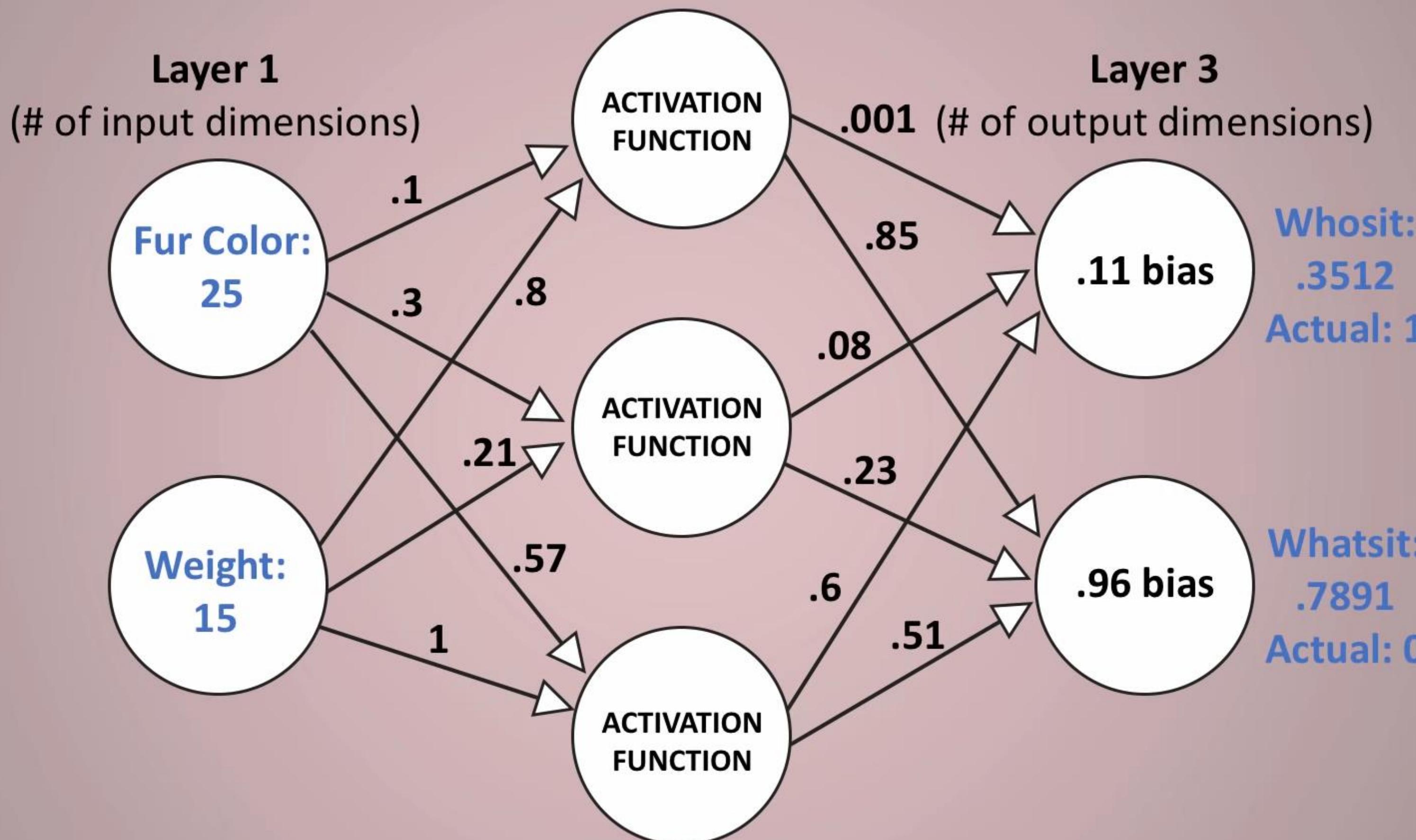


# Einführung in Data Science und maschinelles Lernen

**BEHANDLUNG VON  
FEHLENDEN WERTEN**

- Wiederholung wichtiger Konzepte des maschinellen Lernens
- Implementierung eines NN mit TensorFlow und Python
- Behandlung fehlender Werte
- (Support-Vektor-Maschinen)

## HOW DID WE DO?



# NEURONALE NETZE

## Input Layer

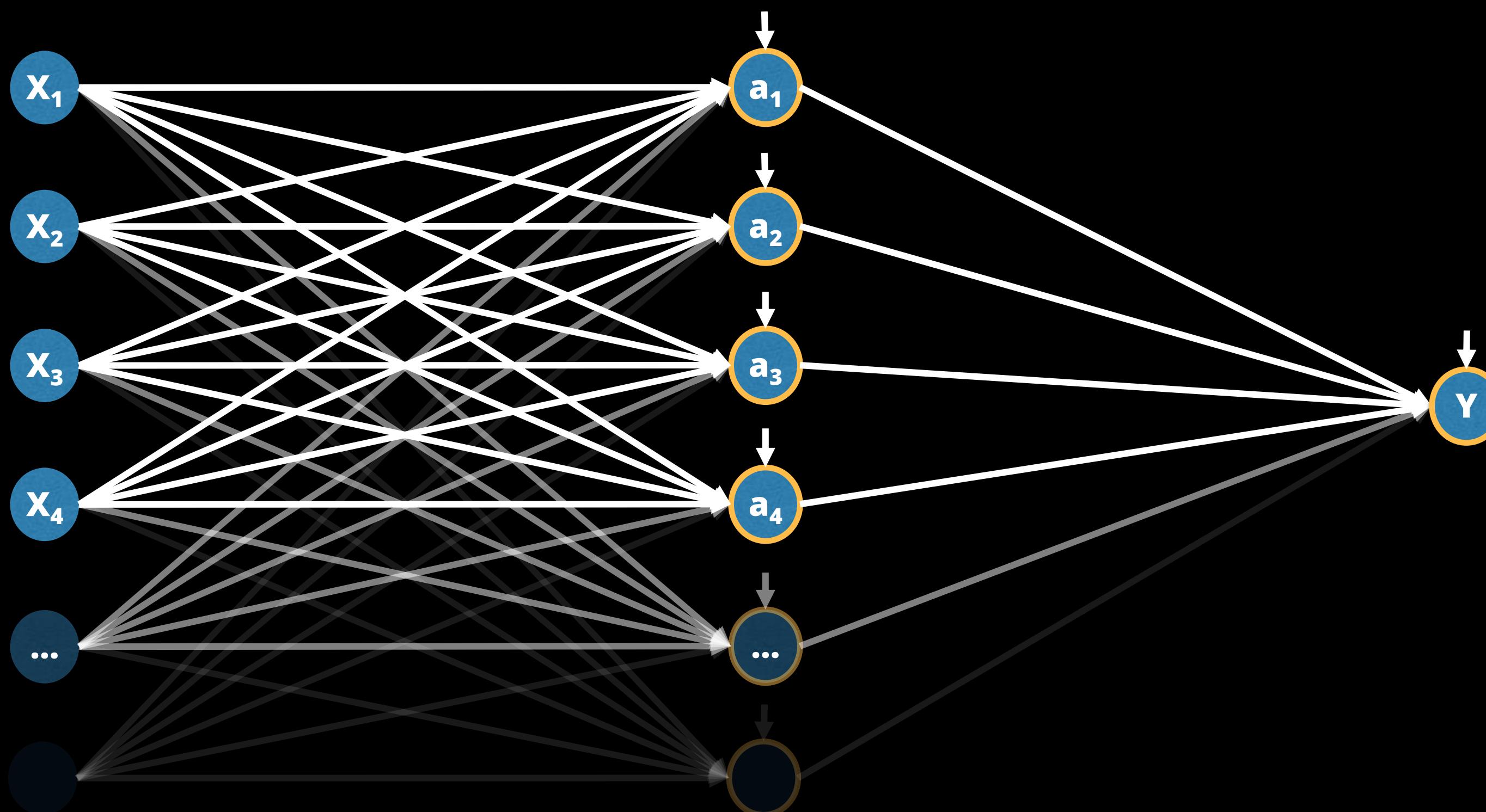
Besteht aus Input-Variablen/Features/Dimensionen

## Hidden Layer

Fasst mit Hilfe von Aktivierungsfunktionen und geschätzten Gewichten die Werte der vorherigen Schicht in jeweils einem Neuron zusammen.

## Output Layer

Fasst ebenfalls mit Hilfe von Aktivierungsfunktion und geschätzten Gewichten die Werte der vorherigen Schicht zusammen.



# WICHTIGE KONZEPTE

## Forward Propagation:

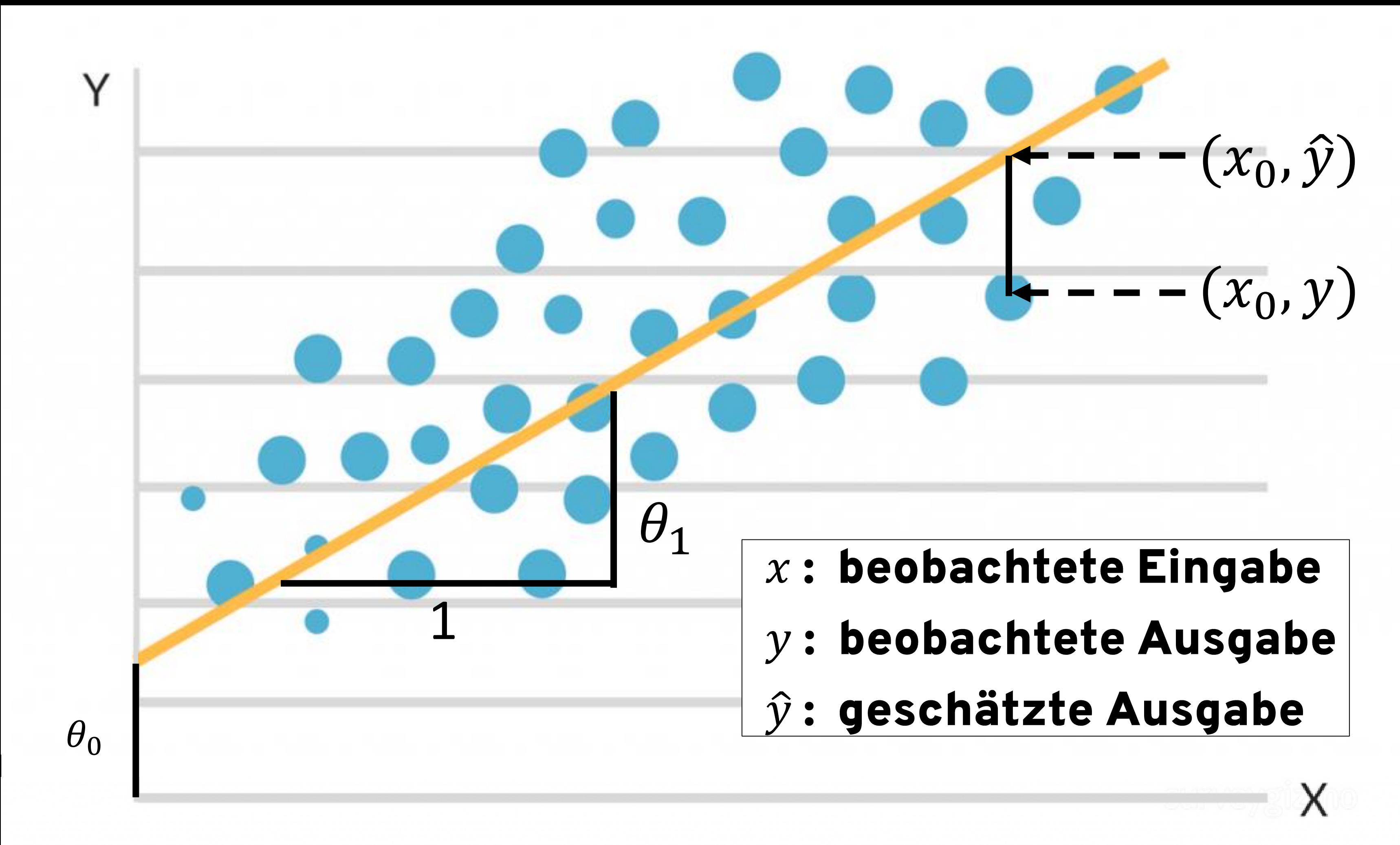
- **Berechnung der Vorhersage basierend auf den aktuellen Parametern und den definierten Aktivierungsfunktionen**

## Backward Propagation:

- **Berechnung des Schätzfehlers mit Hilfe der Kostenfunktion**
- **Berechnung neuer, verbesserter Parameter mit Hilfe der Optimierungsfunktion**

# LINEARES MODELL

$$\hat{y} = \theta_0 + \theta_1 x$$
$$= h_x(\theta_0, \theta_1)$$



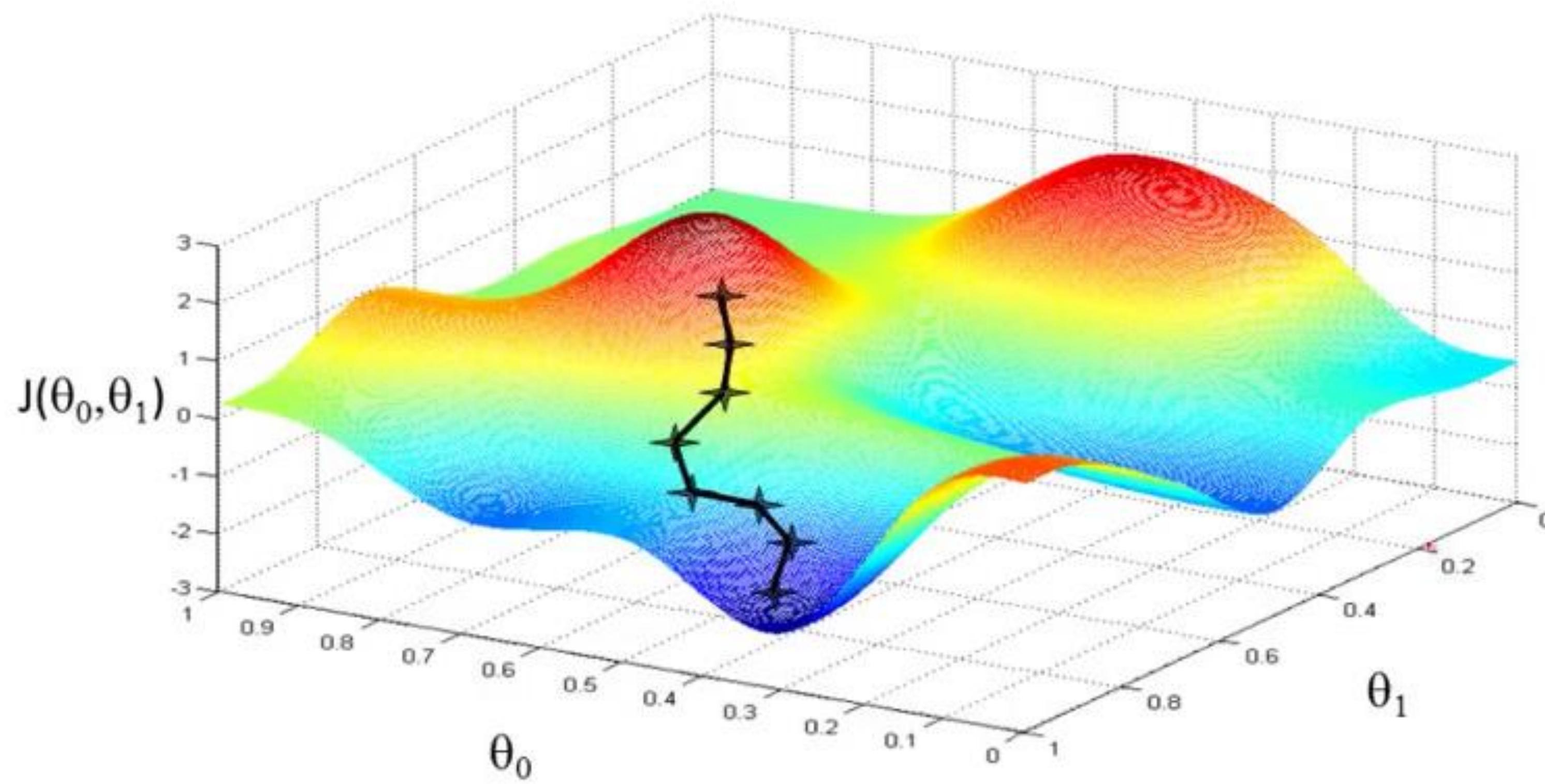
# KOSTENFUNKTION

Zur Berechnung der Funktion mit den optimalen Parametern  
 $\theta_0$  und  $\theta_1$ :

$$J_x(\theta_0, \theta_1) = (h_x(\theta_0, \theta_1) - y)$$

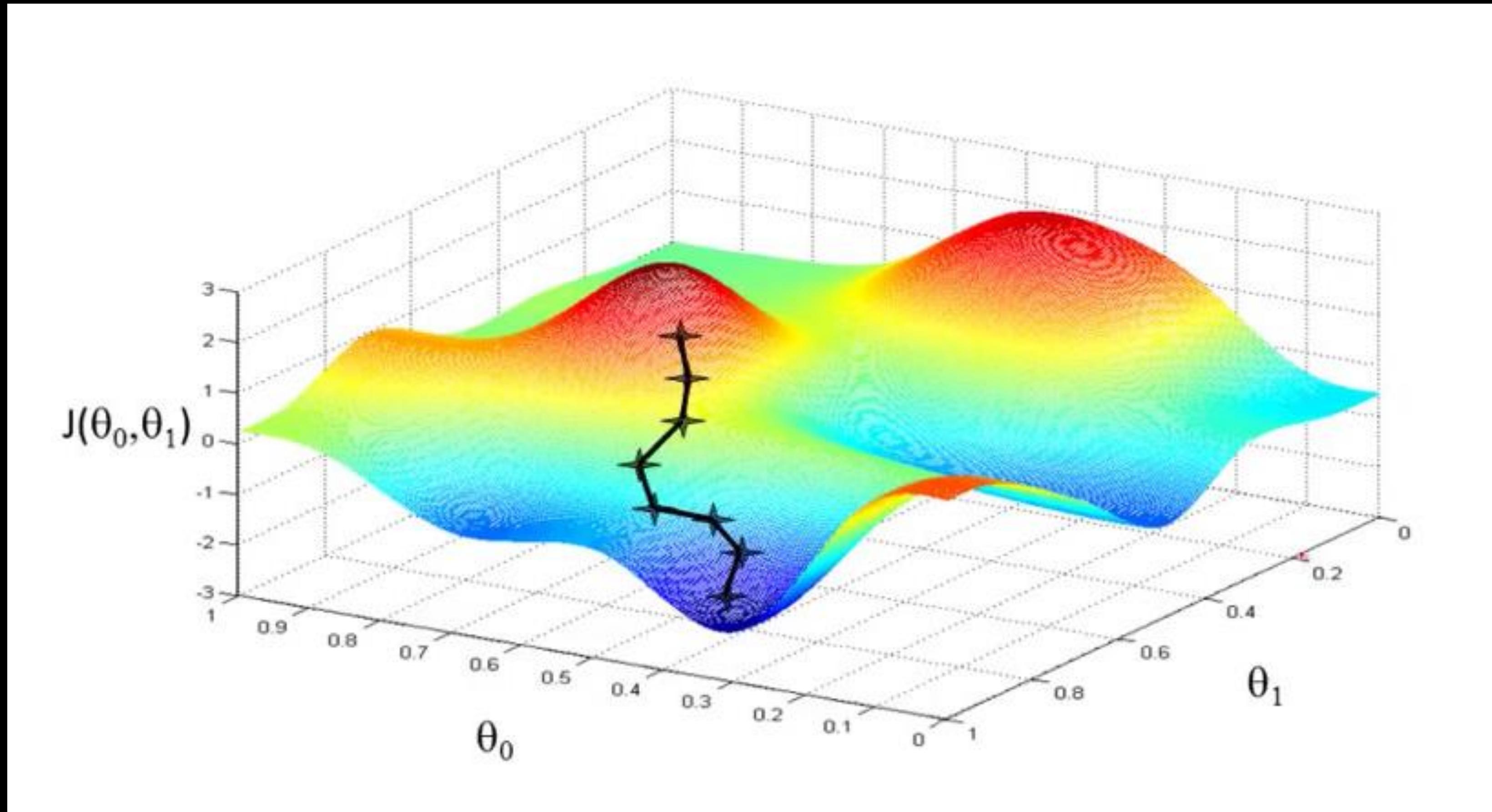
Mean Squared Error (MSE)

# MINIMIERUNG DER KOSTENFUNKTION ÜBER EINE OPTIMIERUNGSFUNKTION



- Minimierung der Kostenfunktion durch ein iteratives Verfahren (**Gradient Descent**), das sich dem Minimum annähert.
- Die Schrittgröße für die Annäherung kann durch die Lernrate („Learning Parameter“) kontrolliert werden.

# PARAMETER VON OPTIMIERUNGSFUNKTIONEN



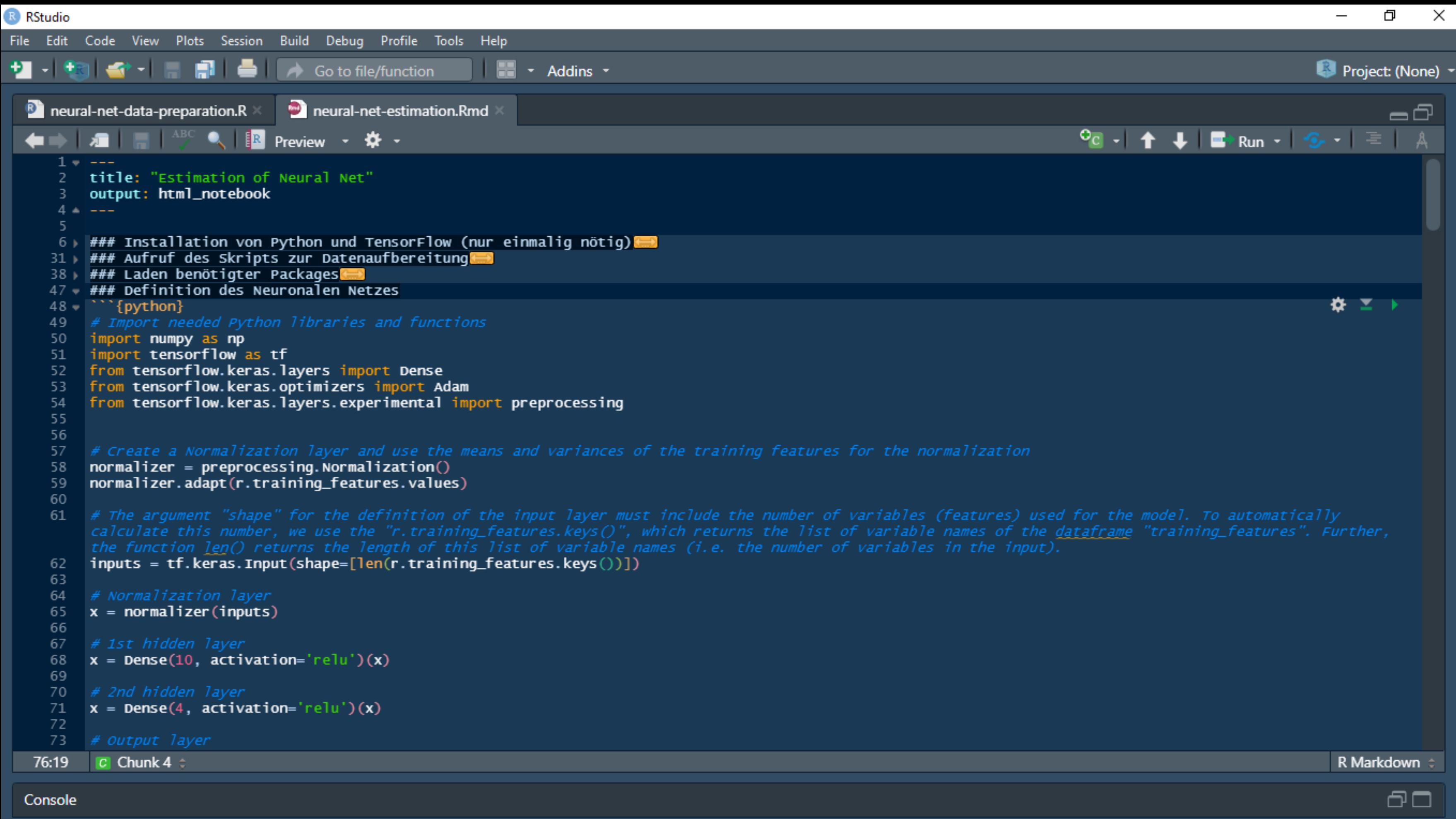
- **Schrittgröße für die Annäherung an das Kosten-Minimum („Learning Rate“)**
- **Trägheit bei Richtungsänderungen („Momentum“)**

Quelle: <https://www.coursera.org/learn/machine-learning>

# DATENAUFBEREITUNG FÜR TENSORFLOW

```
1
2 ######
3 ## Preparation of the Environment #####
4
5
6 ## Data Import #####
7
8 ## Data Preparation #####
9
10
11 # Recoding of the variables into one-hot encoded (dummy) variables
12 dummy_list <- c("view", "condition")
13 house_pricing_dummy = dummy_cols(house_pricing, dummy_list)
14
15
16 # Definition of lists for each one-hot encoded variable (just to make the handling easier)
17 condition_dummies = c('condition_1', 'condition_2', 'condition_3', 'condition_4', 'condition_5')
18 view_dummies = c('view_0', 'view_1', 'view_2', 'view_3', 'view_4')
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40 ## Selection of the Feature Variables and the Label variable #####
41
42
43
44
45
46
47
48
49
50 ## Selection of Training, validation and Test Data #####
51
52
53
54
55
56
57
```

# DEFINITION EINES NEURONALEN NETZES



The screenshot shows the RStudio interface with two files open: "neural-net-data-preparation.R" and "neural-net-estimation.Rmd". The "neural-net-estimation.Rmd" file is the active tab, displaying R Markdown code. The code defines a neural network for estimation, including imports for TensorFlow and Keras, normalization of training features, and the construction of multiple hidden layers using the Dense layer type with ReLU activation.

```
1 ---  
2 title: "Estimation of Neural Net"  
3 output: html_notebook  
4 ---  
5  
6 ### Installation von Python und TensorFlow (nur einmalig nötig)  
31 ### Aufruf des Skripts zur Datenaufbereitung  
38 ### Laden benötigter Packages  
47 ### Definition des Neuronalen Netzes  
48 ^{python}  
49 # Import needed Python libraries and functions  
50 import numpy as np  
51 import tensorflow as tf  
52 from tensorflow.keras.layers import Dense  
53 from tensorflow.keras.optimizers import Adam  
54 from tensorflow.keras.layers.experimental import preprocessing  
55  
56  
57 # Create a Normalization layer and use the means and variances of the training features for the normalization  
58 normalizer = preprocessing.Normalization()  
59 normalizer.adapt(r.training_features.values)  
60  
61 # The argument "shape" for the definition of the input layer must include the number of variables (features) used for the model. To automatically calculate this number, we use the "r.training_features.keys()", which returns the list of variable names of the dataframe "training_features". Further, the function len() returns the length of this list of variable names (i.e. the number of variables in the input).  
62 inputs = tf.keras.Input(shape=[len(r.training_features.keys())])  
63  
64 # Normalization layer  
65 x = normalizer(inputs)  
66  
67 # 1st hidden layer  
68 x = Dense(10, activation='relu')(x)  
69  
70 # 2nd hidden layer  
71 x = Dense(4, activation='relu')(x)  
72  
73 # output layer
```

# HYPERPARAMATER IN NEURONALEN NETZEN

- **Wahl der Architektur:**
  - Anzahl der Hidden Layer des Netzes
  - Typen der Hidden Layer
  - Anzahl der Neuronen je Hidden Layer
  - Wahl der Aktivierungsfunktionen
- **Wahl der Kostenfunktion („Loss Function“)**
- **Wahl der Optimierungsfunktion („Optimizer“)**
- **Wahl der Parameter des Optimizers**

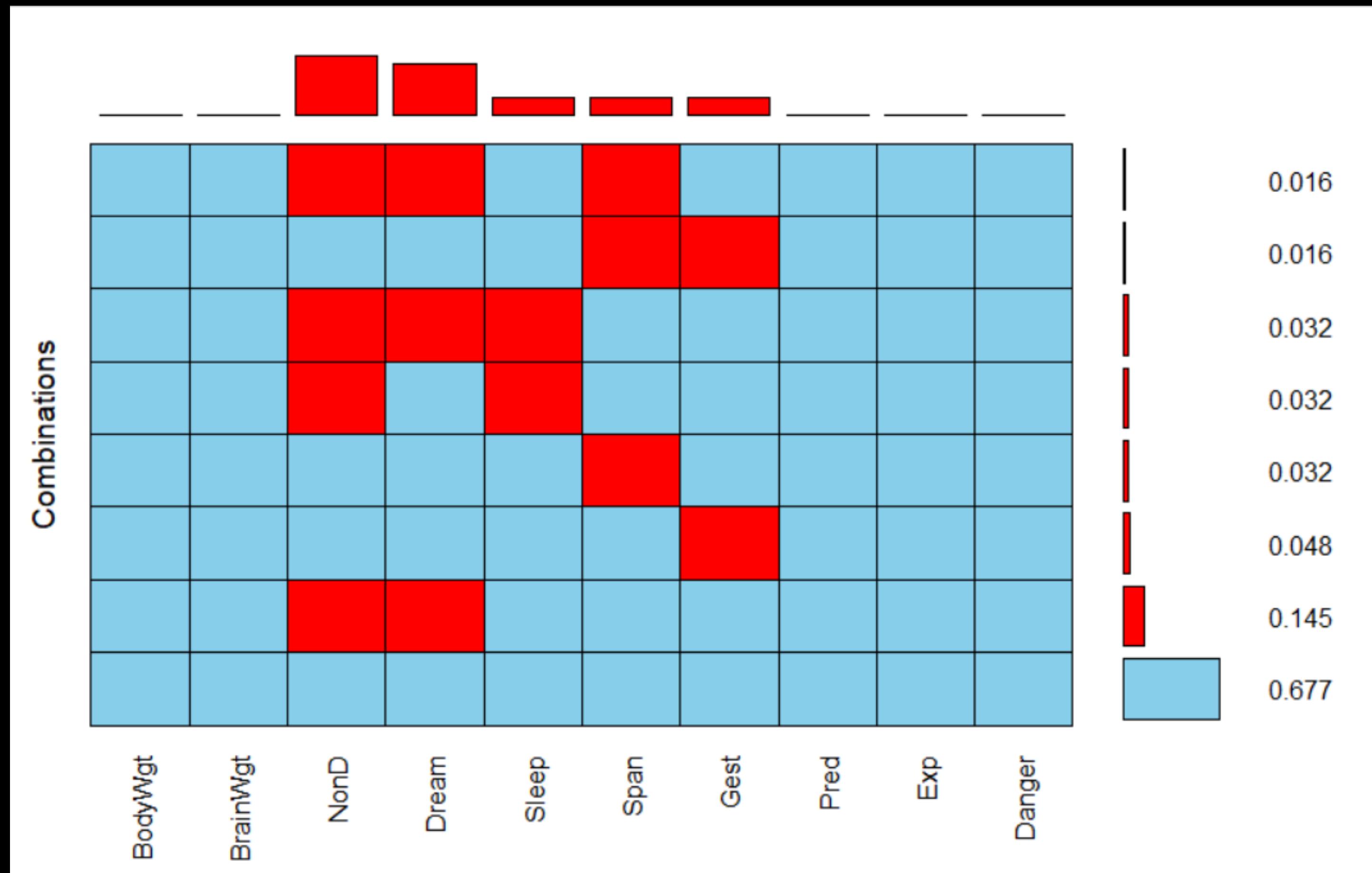
# **GRÜNDE FÜR FEHLENDE WERTE**

- **Nicht gegebene Antworten in Umfragen**
- **Zusammenführung von Daten aus verschiedenen Quellen mit unterschiedlichen Variablen**
- **Technische Probleme in der Datenerhebung oder Aufzeichnung**
- **...**

# TYPEN VON FEHLENDEN WERTEN

- Missing Completely at Random (MCAR)
- Missing at Random (MAR)
- Missing not at Random (MNAR)

# VISUALISIERUNG FEHLENDER WERTE



**VIM package**  
**(Visualization and  
Imputation of  
Missing Values)**

`aggr(dataset,  
combined=TRUE,  
numbers=TRUE)`

Kowarik, A., & Templ, M. (2016). Imputation with the R Package VIM. *Journal of Statistical Software*, 74, 1–16.

# BEHANDLUNG VON FEHLENDEN WERTE

- **Zugehörige Fälle löschen (listwise deletion)**
- **Einfache „Spender“-basierte Imputation (donor-based)**
  - **Mittelwertschätzung (bzw. Median oder Mode)**
  - **nach „Ähnlichkeit“ (Hot-Deck Imputation)**
  - **durch minimalen Abstand (k Nearest Neighbors)**
- **Einfache modellbasierte Imputation**
  - **Iterative Regression**
- **Multiple Imputation**

# HOT-DECK IMPUTATION

## Nach Domänen

The diagram illustrates the domain-based hot-deck imputation process for the 'PhysActive' dataset. It shows two versions of the dataset: the original with missing values and the imputed version.

**Original Data:**

PhysActive	Weight
TRUE	51
TRUE	63
FALSE	98
TRUE	NA
FALSE	81
FALSE	88

**Imputed Data:**

PhysActive	Weight
TRUE	51
TRUE	63
FALSE	98
TRUE	NA
FALSE	81
FALSE	88

A red arrow points from the original NA value to the imputed value in the second row, indicating that the missing value was replaced by the value from the same domain (TRUE).

## Nach Korrelation

The diagram illustrates the correlation-based hot-deck imputation process for the 'Height' dataset. It shows two versions of the dataset: the original with missing values and the imputed version.

**Original Data:**

Height	Weight
150	51
161	63
189	98
155	NA
182	81
184	88

**Imputed Data:**

Height	Weight
150	51
155	NA
161	63
182	81
184	88
189	98

A red arrow points from the original NA value to the imputed value in the second row, indicating that the missing value was replaced by the value from the same column (Height).

# K NEAREST NEIGHBORS (KNN)

- **Suche nach den k Fällen mit minimalem Abstand**
  - je nach Variablentyp unterschiedliche Abstandsmessung
  - Zusammenführung der Abstände über eine Summenfunktion
- **Zufällige Ziehung aus den k Fällen**

# ITERATIVE REGRESSION

A	B	C	D
	5	34	NA
	1	22	NA
NA		65	55
	4	87	27
NA		23	10

# ITERATIVE REGRESSION

## 1) Vorhersage fehlender Werte in A

A	B	C	D
5	34	NA	1
1	22	NA	4
5	65	55	2
4	87	27	2
2	23	10	1

# ITERATIVE REGRESSION

2) Vorhersage Fehlender Werte in C mit den imputierten  
werten von A

A	B	C	D
5	34	32	1
1	22	16	4
5	65	55	2
4	87	27	2
2	23	10	1

# ITERATIVE REGRESSION

3) Vorhersage fehlender Werte in A mit den imputierten Werten von C

A	B	C	D
5	34	32	1
1	22	16	4
NA	65	55	2
4	87	27	2
NA	23	10	1

→ Wiederholung bis keine Änderung mehr eintritt

# ITERATIVE REGRESSION

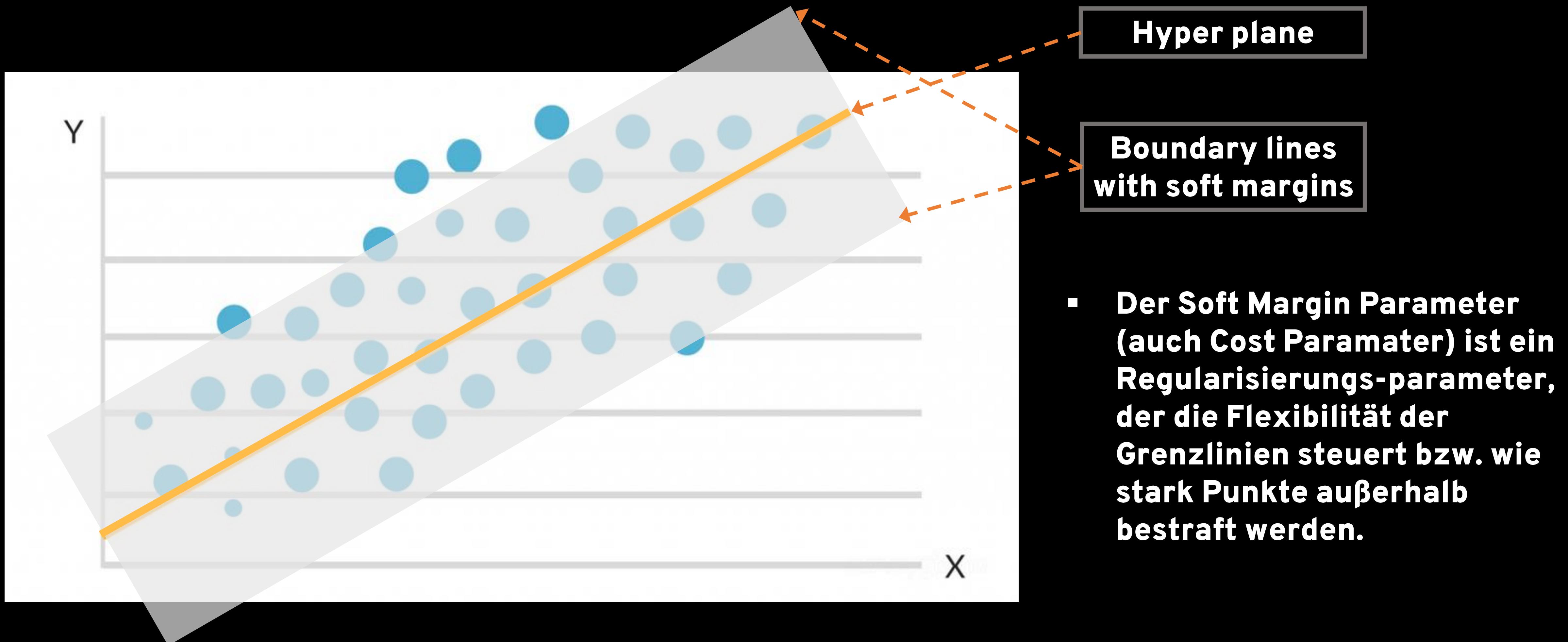
- 1) Gehe schrittweise durch alle Variablen des Datensatz
  - 2) Stelle dabei für jede Variable eine Regressionsmodell basierend auf allen anderen Variablen auf
  - 3) Berechne für alle fehlenden Werte eine Vorhersage
- 
- Jetzt wiederhole Schritt 1) bis 3) erneut und schätze die fehlenden Werte erneut - dieses Mal mit den bereits imputierten fehlenden Werten.
  - Wiederhole dies, bis sich die imputierten Werte nicht mehr ändern.

# AUFGABEN

- Wählt ein (bzw. verschiedene) Verfahren, um fehlenden Werte in Eurem Datensatz zu ersetzen.
- Schaut [dieses Video](#) (5 Minuten) zu Zeitreihenanalysen an.
- Teilt Euch die Aufgaben im Team gut auf:  
Wer arbeitet an der Datenoptimierung, wer an der Modelloptimierung?

# SUPPORT VECTOR MACHINE (SVM)

## [SUPPORT VECTOR REGRESSION (SVR)]



# SUPPORT VECTOR MACHINES

## Lernressourcen

- Blog:  
<https://towardsdatascience.com/an-introduction-to-support-vector-regression-svr-a3ebc1672c2>
- Video:  
<https://www.youtube.com/watch?v=efR1C6CvhmE>

# **OPTIMIERUNG EINER SVM IN R**

**R-Package „e1071“**

**svm()**      **Optimierung eines SVM Modells**

**tune()**      **Optimierung mehrerer SVM Modelle mit gleichzeitiger Optimierung des C- und Gamma-Parameters**

**predict()**      **Vorhersage auf Basis eines optimierten Modells**

# BEISPIELCODE IN R

```
1 ---  
2 title: "Support Vector Machine"  
3 output: html_notebook  
4 ---  
5  
6  
7 ## Imports   
22 ## Splitting Training and Test Data   
43 ## Data Preparation   
51 ## Training the SVM  
52  
53 ````{r}  
54 # Optimization of an SVM with standard hyper parameters  
55 # Typically NOT used; Instead, the function svm_tune() is used in order to also get a model with optimized hyper parameters  
56 model_svm <- svm(price ~ bathrooms, train_dataset)  
57 ````  
58  
59 ````{r}  
60 # Optimization of various SVM using systematically varied hyper parameters (typically called 'grid search' approach) and  
# cross validation  
61 # the resulting object includes the optimal model in the element named 'best.model'  
62 svm_tune <- tune(svm, price ~ bedrooms + bathrooms + sqft_living + zipcode, data=train_dataset,  
63 ranges = list(epsilon = seq(0.2,1,0.1), cost = 2^(2:3)))  
64 ````  
65  
66  
67 ## Checking the Prediction Quality 
```

# ZUSAMMENFASSUNG SVM

- Populärer, weil einfach zu optimierender Lernalgorithmus, der schnell gute Ergebnisse bringt.
- Geeignet zur Klassifikation und zur Regression.