

Deep Learning with Python & Keras



Tag 2

Marvin Göbel

22.6.2021



Tag 2: Training und Generalisierung von neuronalen Netzwerken



Recap

Was haben wir gestern gemacht?



Was ist ein Tensor?

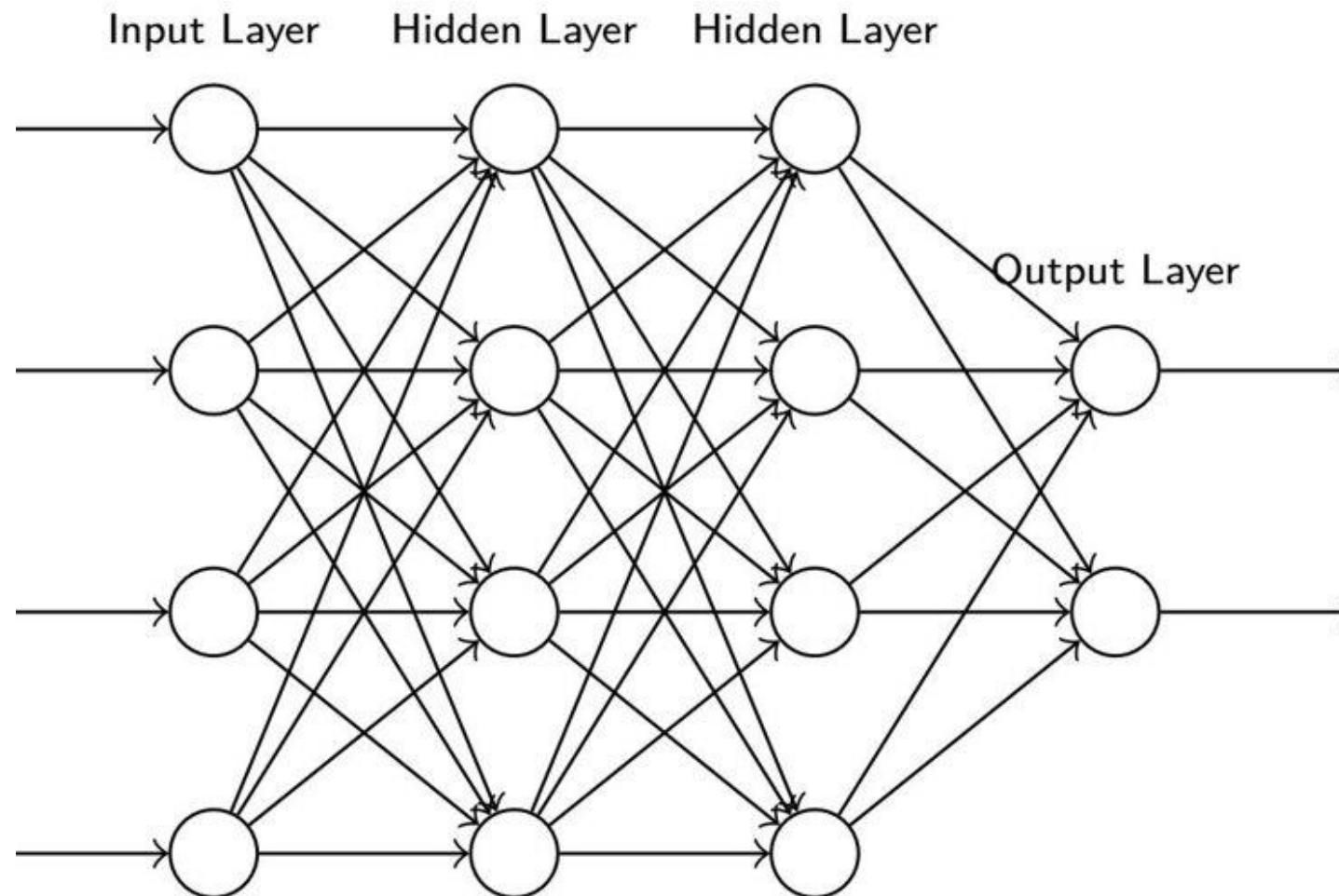
- Tensoren sind Verallgemeinerungen von Matrizen und Vektoren.
- Ein Skalar -1 ist ein Rang-0 Tensor
- Ein Array [0, -1, -2, -3] ist ein Rang-1 Tensor
- Eine Matrix ist ein Rang-2 Tensor

$$\mathbf{A} = \begin{bmatrix} 0 & -1 & -2 & -3 \\ 1 & 0 & -1 & -2 \\ 2 & 1 & 0 & -1 \end{bmatrix}$$

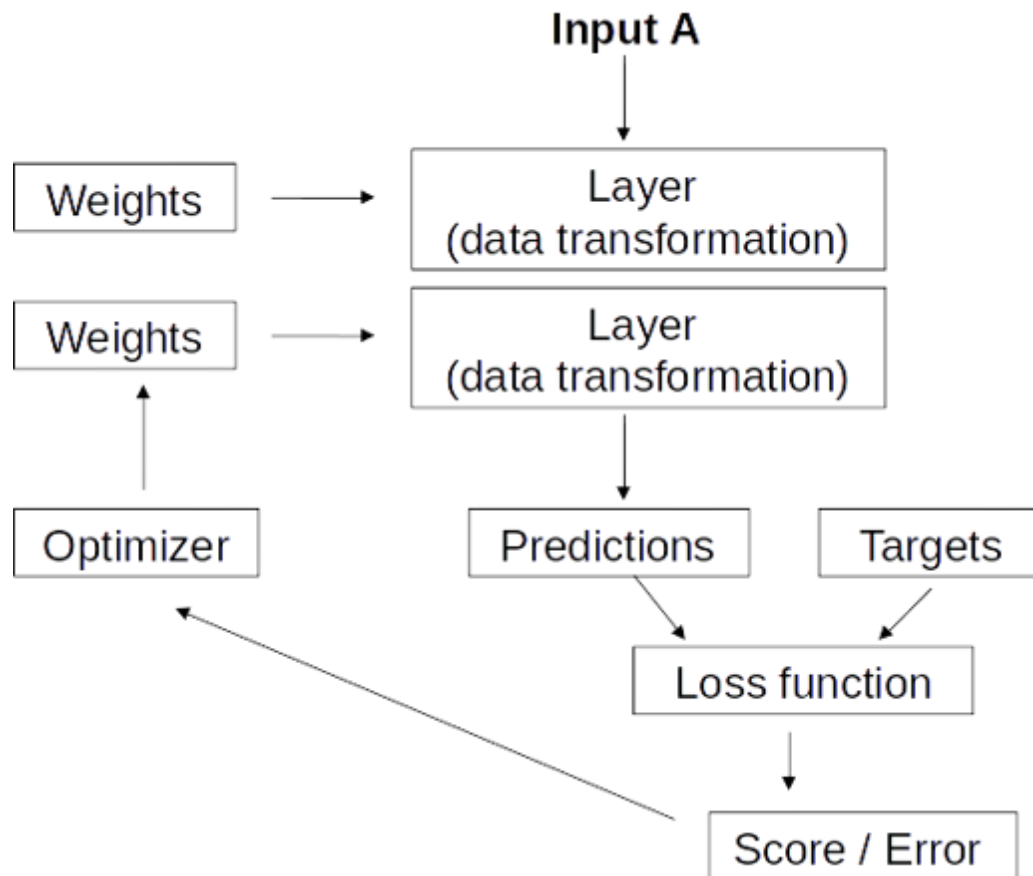
- Vorsicht: Mathematischer Rang einer Matrix \neq Tensor-Rang
- Rang wird auch Anzahl der „Axes“ genannt



Einführendes zum Training



Bausteine eines neuronalen Netzwerks



Adapted from 'Deep Learning with Python' by Francois Chollet.

- Modell • Gegeben ein Input soll ein Output erzeugt werden
- Loss • Wie formulieren wir mathematisch, was wir erreichen wollen?
- Optimierer • Welcher Algorithmus minimiert unseren Loss?
- Metrik • Welches menschlich verständliche Maß verwenden wir, um die Performance unseren Modell einzuschätzen?

Loss Funktionen

- Input der Loss Funktion:
 - target (Zielgröße)
 - prediction (Output)
- Output der Loss Funktion: Skalarer Wert
- Je kleiner der Loss Output desto besser entspricht der Output des Modells der Zielgröße

Training (Optimierungsproblem)

Minimiere Loss Funktion:

$$\min_{\theta} \mathcal{L}(\theta)$$

- Beispiele für Loss-Funktionen:
 - Regressionsprobleme Mean-Squared-Error:
 - Aktivierungsfunktion der Output Layer: Keine
 - Klassifikationsprobleme Cross Entropy:
 - Aktivierungsfunktion der Output Layer: Softmax

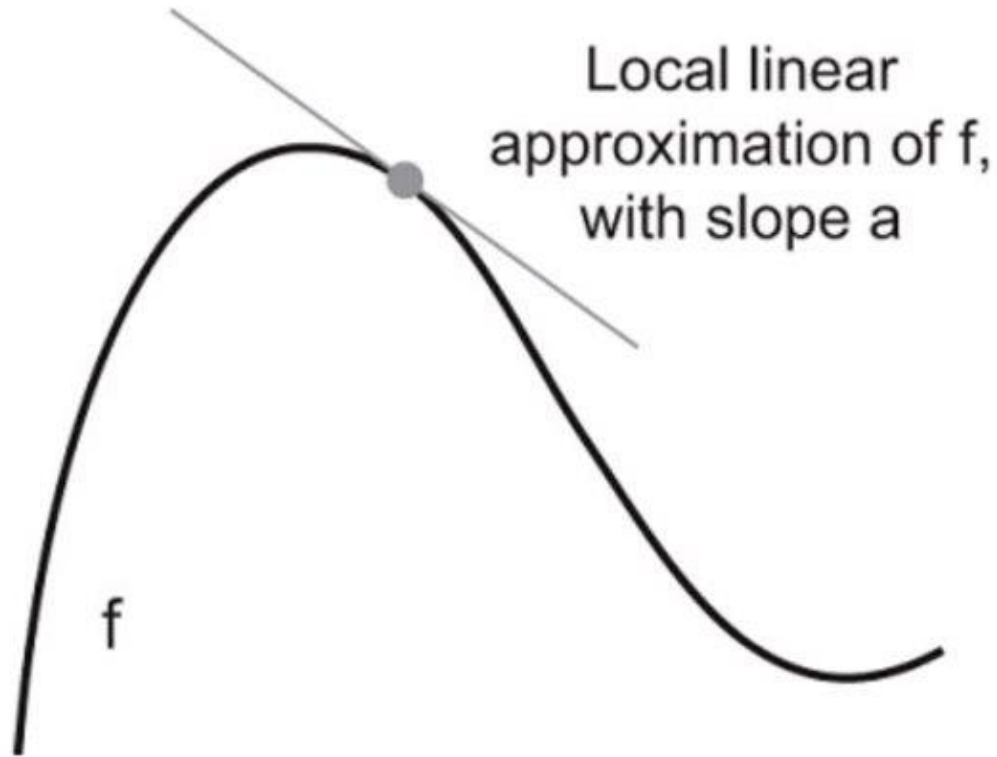
$$\mathcal{L} \propto \frac{1}{n} \sum_{i=1}^n (f(\mathbf{x}_i; \theta) - y_i)^2$$

$$\mathcal{L} \propto \sum_{i=1}^n p_i \log(q_i), \quad q_i = f(\mathbf{x}_i; \theta)$$



Gradient Descent

Die Ableitung einer Funktion sagt uns in welche Richtung sie lokal kleinere Werte hat.
Im mehrdimensionalen nennt man die Ableitung Gradient.
Der Gradient zeigt in Richtung der größten Steigung.



$$\text{Gradient: } \mathbf{g} = \nabla f(\mathbf{x}) = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \dots \\ \frac{\partial f}{\partial x_n} \end{pmatrix}$$

$$\text{Abstieg: } \mathbf{x}_{\text{new}} = \mathbf{x} - \epsilon \mathbf{g}$$

ϵ : Learning rate (Parameter)

2. Tag: Deep Learning mit Python & Keras

1. Training einfacher Netzwerke
2. Klassifikationsprobleme
3. Backpropagation und Generalisierung



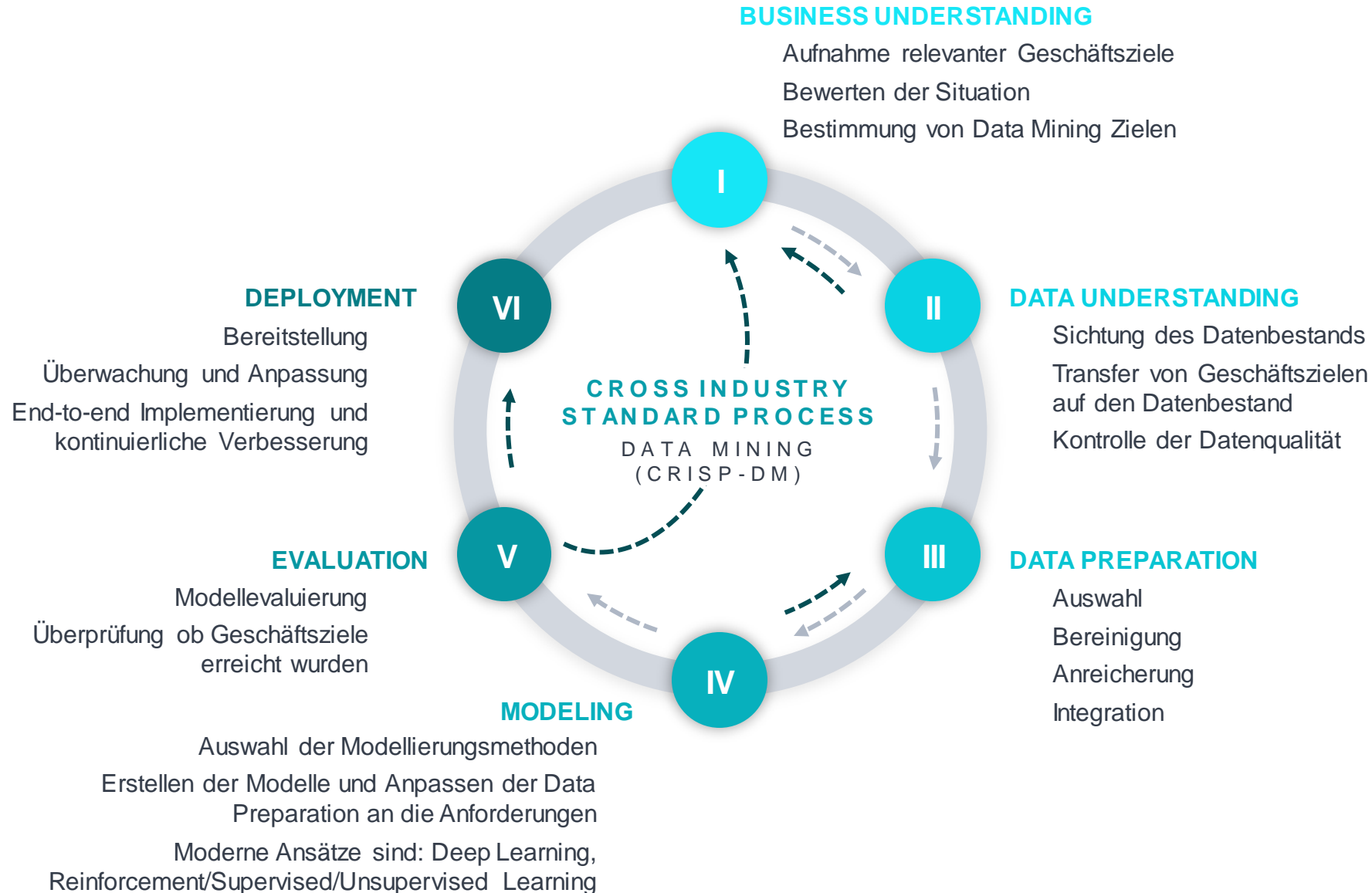
Machine Learning Workflow: Feature Engineering

Daten für Modelle vorbereiten

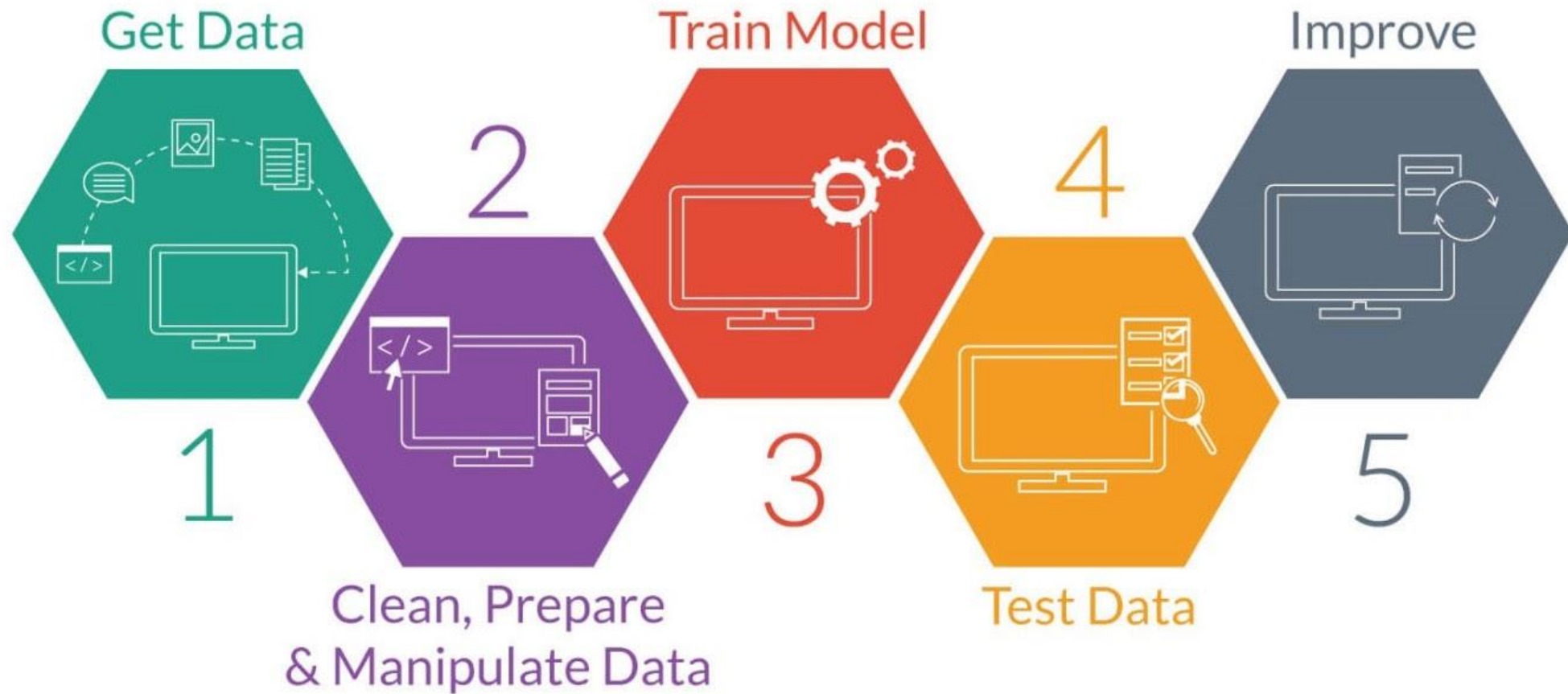


Methodisches Vorgehen bei Data Science Projekten

Cross industry standard process for data mining (CRISP-DM)



Machine Learning Workflow - vereinfacht



1. Get Data

Machine Learning Terminologie

Header															
	pclass	survived	name	sex	age	sibsp	parch	ticket	fare	cabin	embarked	boat	body	home.dest	
Rows/ Examples/ Instances	0	1	1	Allen, Miss. Elisabeth Walton	female	29.0000	0	0	24160	211.3375	B5	S	2	NaN	St Louis, MO
	1	1	1	Allison, Master. Hudson Trevor	male	0.9167	1	2	113781	151.5500	C22 C26	S	11	NaN	Montreal, PQ / Chesterville, ON
	2	1	0	Allison, Miss. Helen Loraine	female	2.0000	1	2	113781	151.5500	C22 C26	S	NaN	NaN	Montreal, PQ / Chesterville, ON
	3	1	0	Allison, Mr. Hudson Joshua Creighton	male	30.0000	1	2	113781	151.5500	C22 C26	S	NaN	135.0	Montreal, PQ / Chesterville, ON
	4	1	0	Allison, Mrs. Hudson J C (Bessie Waldo Daniels)	female	25.0000	1	2	113781	151.5500	C22 C26	S	NaN	NaN	Montreal, PQ / Chesterville, ON
	5	1	1	Anderson, Mr. Harry	male	48.0000	0	0	19952	26.5500	E12	S	3	NaN	New York, NY
	6	1	1	Andrews, Miss. Kornelia Theodosia	female	63.0000	1	0	13502	77.9583	D7	S	10	NaN	Hudson, NY
	7	1	0	Andrews, Mr. Thomas Jr	male	39.0000	0	0	112050	0.0000	A36	S	NaN	NaN	Belfast, NI
	8	1	1	Appleton, Mrs. Edward Dale (Charlotte Lamson)	female	53.0000	2	0	11769	51.4792	C101	S	D	NaN	Bayside, Queens, NY
	9	1	0	Artagaveytia, Mr. Ramon	male	71.0000	0	0	PC 17609	49.5042	NaN	C	NaN	22.0	Montevideo, Uruguay
Index	Features/Attributes/Dimensions														



2. Clean and Prepare Data

PassengerId	Survived	Pclass	Gender	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	0	3	Male	22	1	0	A/5 21171	7.25		S
2	1	1	Female	38	1	0	PC 17599	71.2833	C85	C
3	1	3	Female	26	0	0	STON/O2. 3101282	7.925		S
4	1	1	Female	35	1	0	113803	53.1	C123	S
5	0	3	Male	35	0	0	373450	8.05		S
6	0	3	Male		0	0	330877	8.4583		Q



Missing values

Figure 2.7 The Titanic Passengers dataset has missing values in the Age and Cabin columns. The passenger information has been extracted from various historical sources, so in this case the missing values stem from information that couldn't be found in the sources.



Einführendes zum Training



Überblick über den Trainingsprozess – In Worten

Der Trainings-Loop:

- Nehme die Trainingsbeispiele
- Steck sie in das Model und berechne die Vorhersagen (Forward-Propagation)
- Vergleiche Vorhersagen mit dem Target also der Groundtruth (Berechnung des Losses)
- Berechne in welche Richtung sich die Parameter ändern müssen, sodass sich der Loss verringert (Backpropagation)
- Verändere die Parameter des Modells, so dass der Loss sich verringert. (Gradientenabstieg)

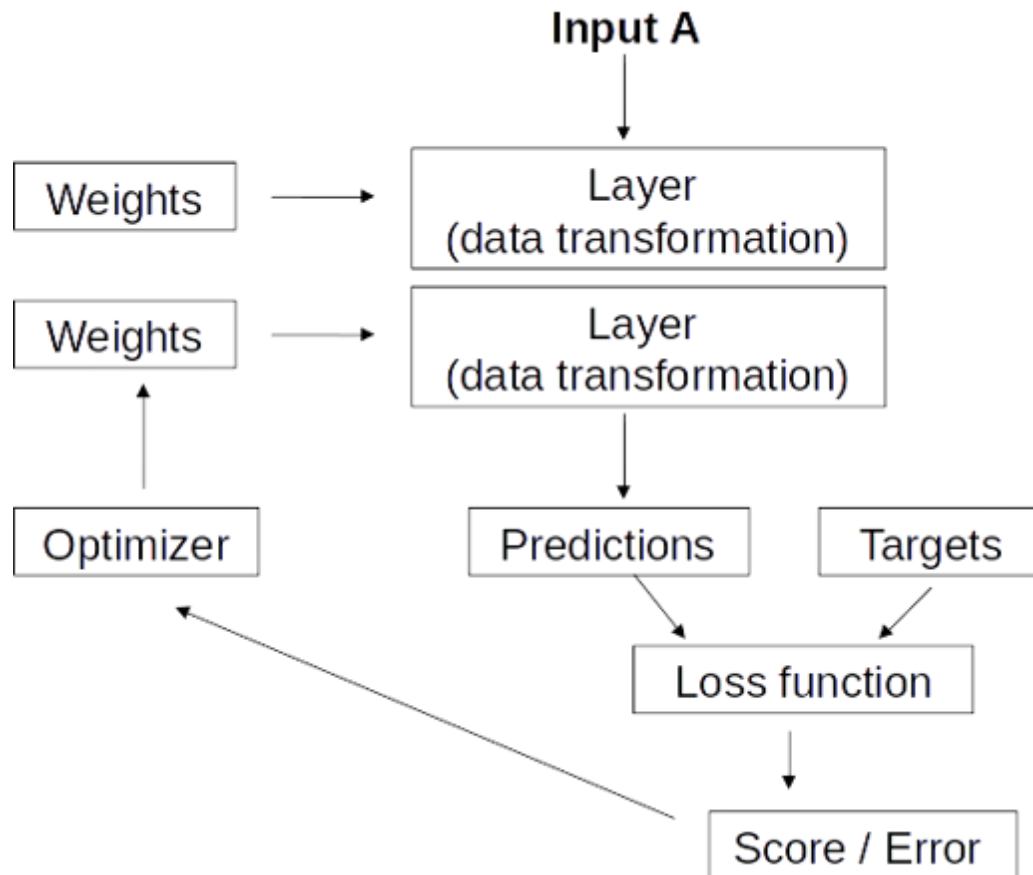


Überblick über den Trainingsprozess – In Formeln

- Netzwerk erzeugt Output $\hat{y}_i = f(\mathbf{x}_i; \theta)$
- Berechne den Loss $\mathcal{L}(\theta) = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2 = \frac{1}{n} \sum_{i=1}^n (y_i - f(\mathbf{x}_i; \theta))^2$
- Berechne Gradient (Backpropagation) $\mathbf{g} = \nabla_{\theta} \mathcal{L}(\theta)$
- Gradientenabstieg zum Parametervektor θ^* (Minima des Losses) $\theta_{\text{new}} = \theta - \epsilon \mathbf{g}$



Bausteine eines neuronalen Netzwerks



Adapted from 'Deep Learning with Python' by Francois Chollet.

- Modell • Gegeben ein Input soll ein Output erzeugt werden
- Loss • Wie formulieren wir mathematisch, was wir erreichen wollen?
- Optimierer • Welcher Algorithmus minimiert unseren Loss?
- Metrik • Welches menschlich verständliche Maß verwenden wir, um die Performance unseres Modell einzuschätzen?

Unterschied - Loss und Metrik

Entscheidend für die Güte unseres Modells ist letzten Endes die Performance Metrik und nicht der Loss.

- Also warum optimieren wir nicht nach der Metrik?
- Metrik nicht unbedingt ableitbar
- Metrik ändert sich oft initial wenig
- Metrik kann natürliche Plateaus erreichen
- Gute Lossfunktionen weisen diese Eigenschaften nicht auf!



Batch Training

Typischerweise wird nicht der ganze Trainingssatz zur Berechnung des Gradienten eingesetzt

- Stattdessen: Wähle einen zufälligen Batch (Teilmenge) von Trainingsbeispielen

In Formeln:

$$\mathbf{g} = \begin{cases} \frac{1}{|I|} \sum_{i \in I} \nabla_{\theta} \mathcal{L} & \text{mini-batch Gradient Descent, } I \subseteq \{1, 2, \dots, n\} \\ \frac{1}{n} \sum_{i=1}^n \nabla_{\theta} \mathcal{L} & \text{batch Gradient Descent} \end{cases}$$

Iteration : Einen (mini-) batch durchlaufen

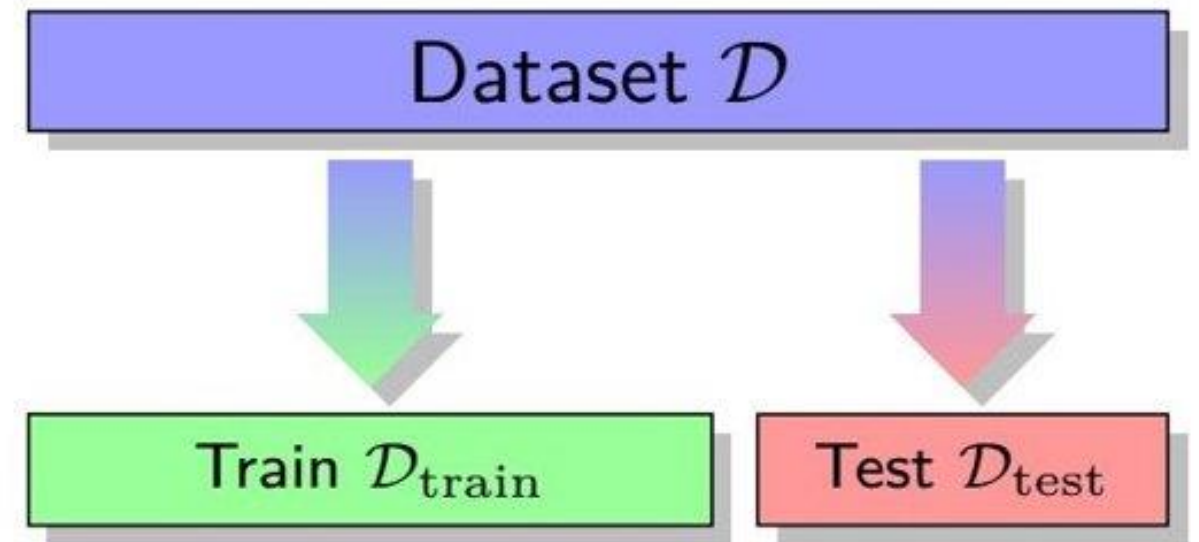
Epoch : Einmal den ganzen Trainingsdatensatz durchlaufen



Training-Test Split

Der Trainingsdatensatz wird aufgeteilt in einen Trainingsdatensatz und einen Testdatensatz

- Auf dem Trainingsdatensatz wird trainiert
- Nach dem Training wird auf dem Testdatensatz getestet z.B. unter Verwendung der Performance Metrik
- Durch Verwendung des Testdatensatzes wird festgestellt ob das Modell auf neue Beispiele generalisiert



Zurück zu Neural Nets

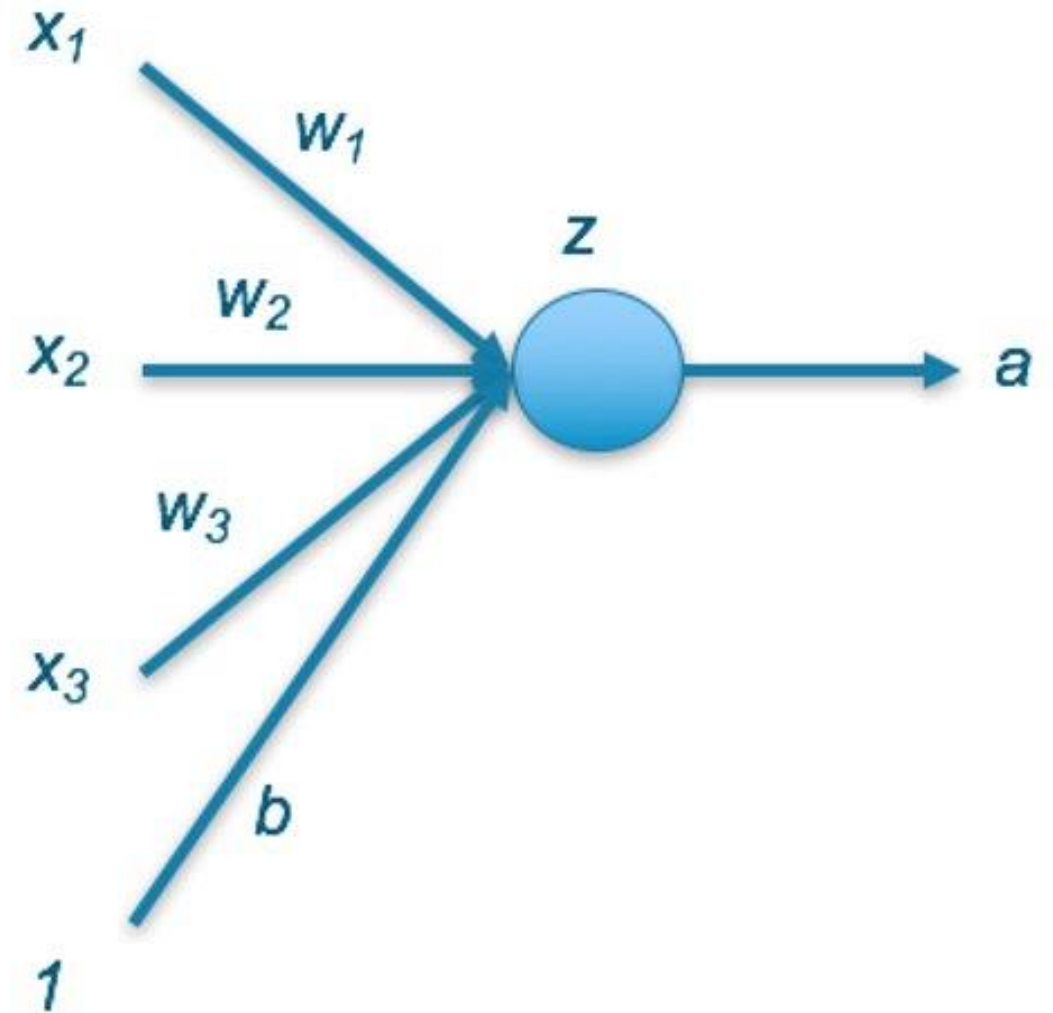
Neurone:

- Linear-affine Abbildung:

$$z = \sum_{i=1}^3 w_i x_i + b$$

- Nicht-lineare Aktivierungsfunktion:

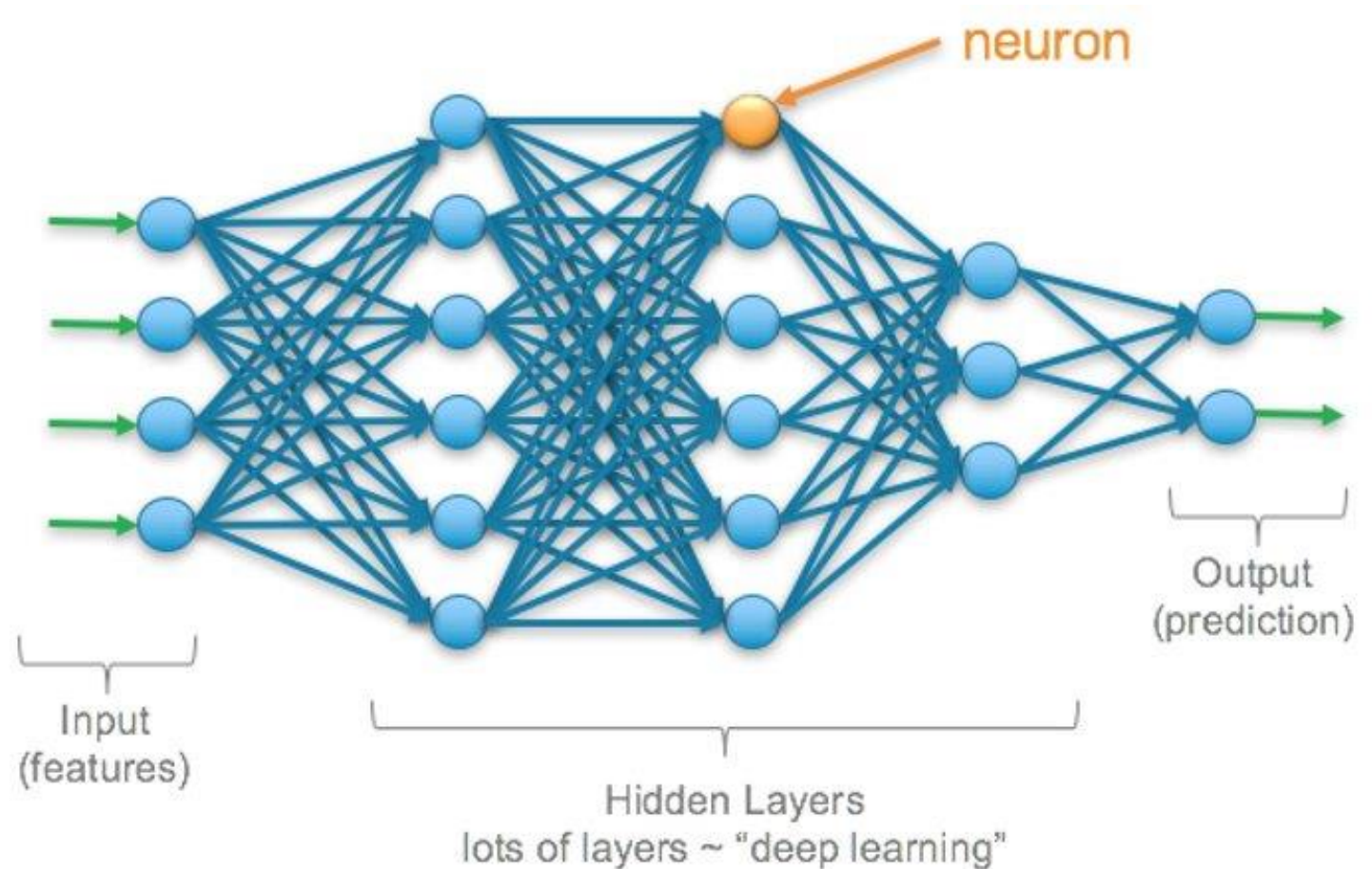
$$a = \varphi(z) = \varphi \left(\sum_{i=1}^3 w_i x_i + b \right)$$



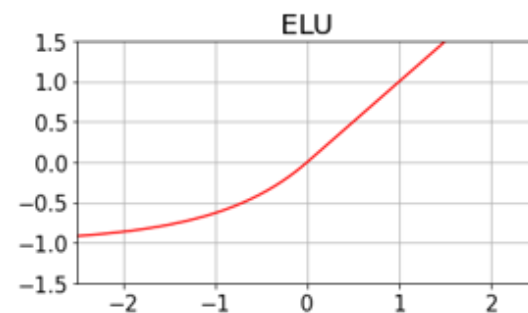
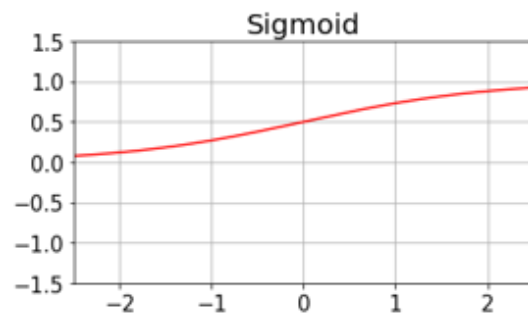
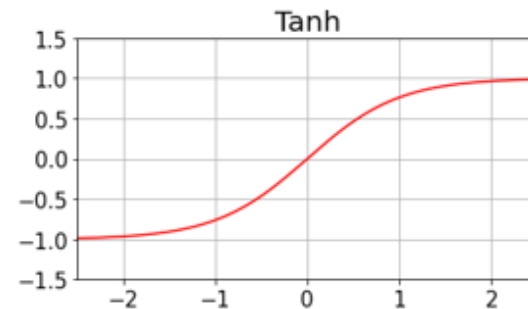
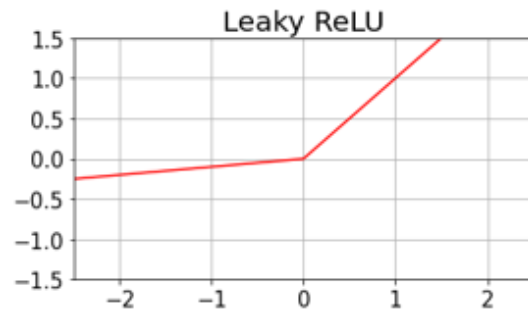
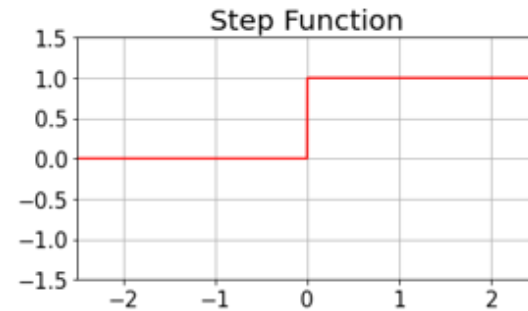
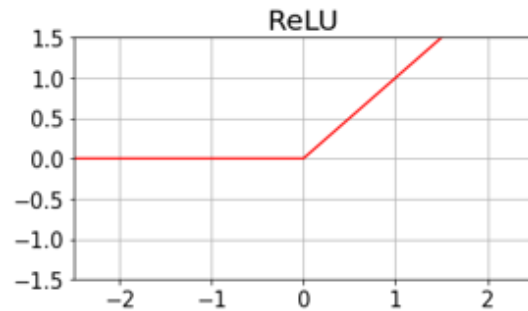
Source: <https://medium.com/@srngn/deep-learning-overview-of-neurons-and-activation-functions-1d98286cf1e4>

Zurück zu Neural Nets

- Neurone bilden den Grundbaustein für Deep Learning Modelle
- Funktionsweise: Lineare Regression
- Aktivierungsfunktion:
 - Viele Arten
 - Immer nicht-linear



Aktivierungsfunktionen



Wichtige Aktivierungsfunktionen:

- ReLU (Rectified Linear Unit):

$$\max(0, x)$$

- Meistverwendete Aktivierungsfunktion für hidden layer

- Sigmoid: $\frac{1}{1 + e^{-x}}$

- Als Output layer für binäre Klassifikationsprobleme

- Tanh: $\tanh(x)$

- Findet Anwendung in rekurrenten Netzen z.B. LSTMs



Notebooks

- **01_RegressionNN.ipynb**

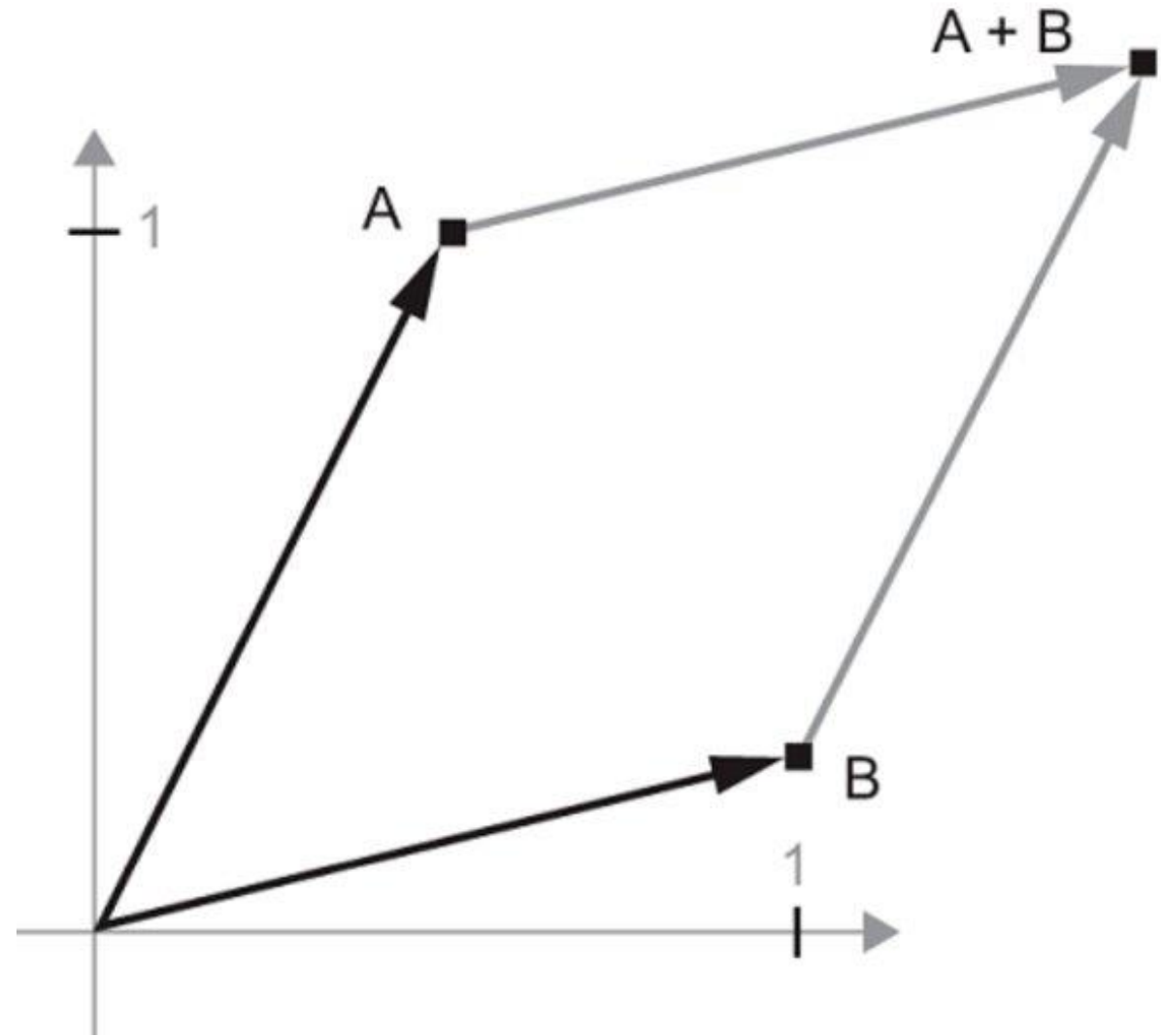


Geometrische Interpretation affin-linearer Transformationen



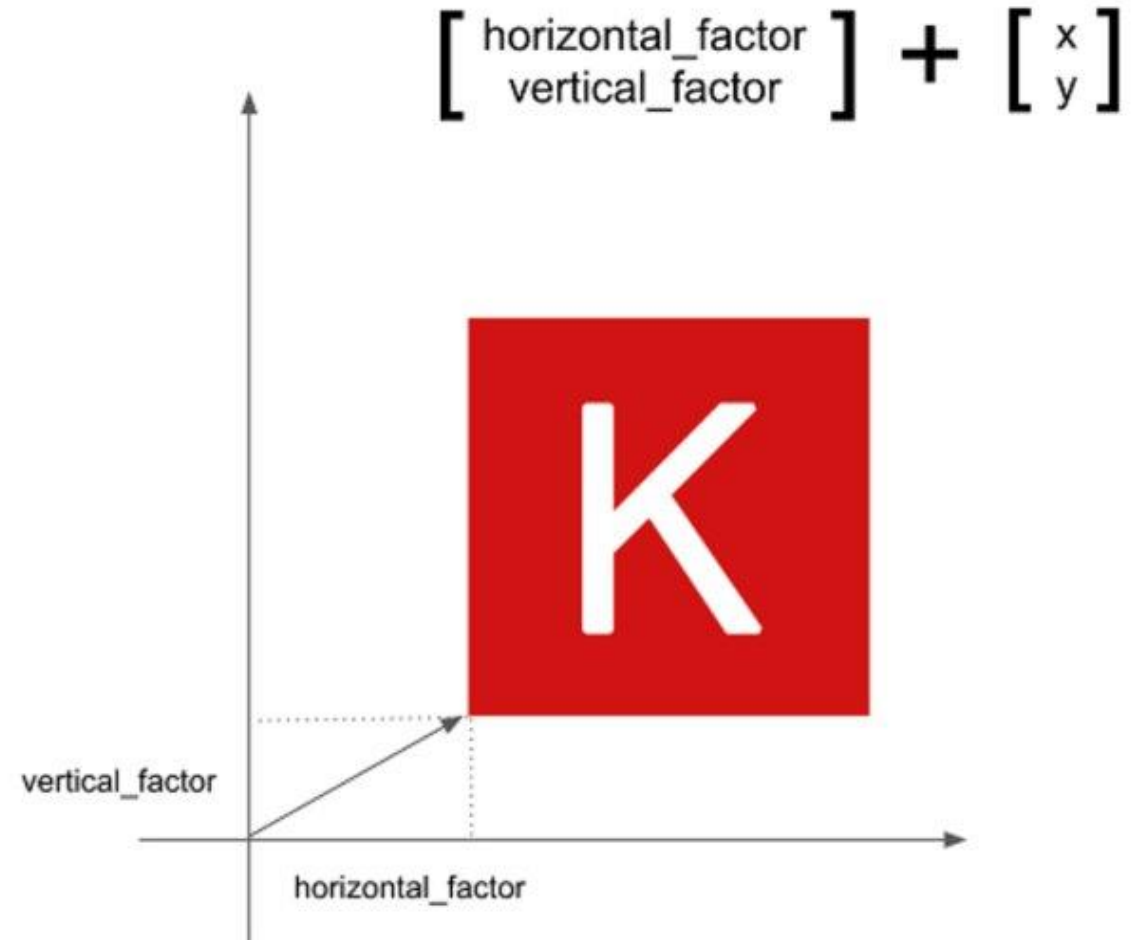
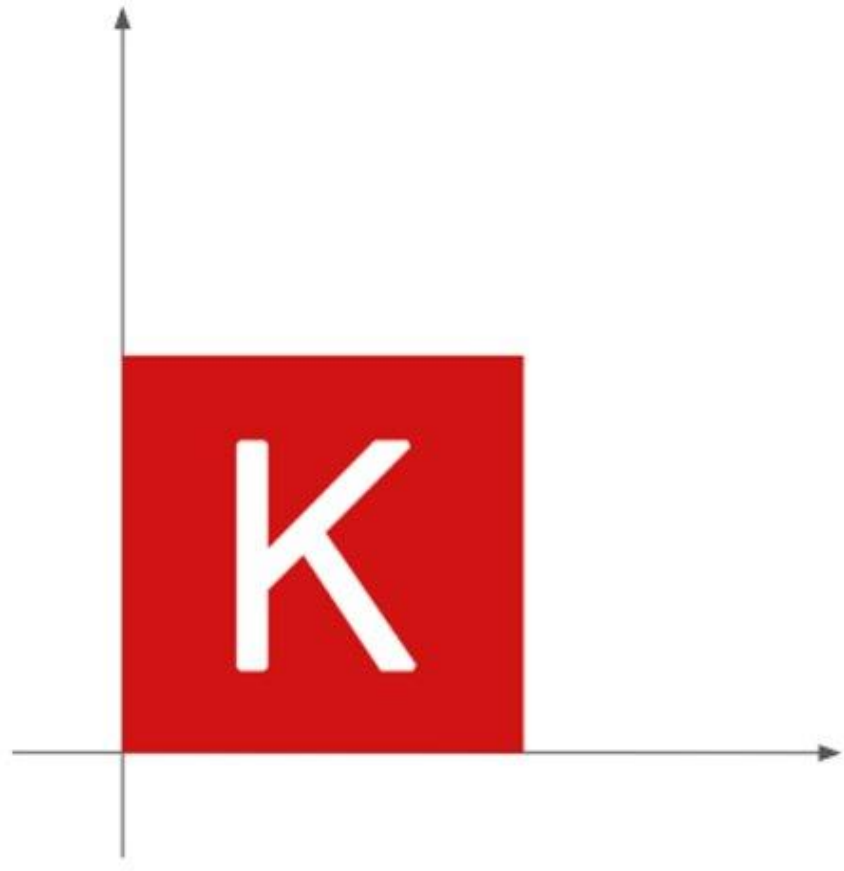
Geometrische Interpretation affiner Transformationen

Vektorarithmetik hat immer eine geometrische Interpretation:
z.b. die Addition von Vektoren.



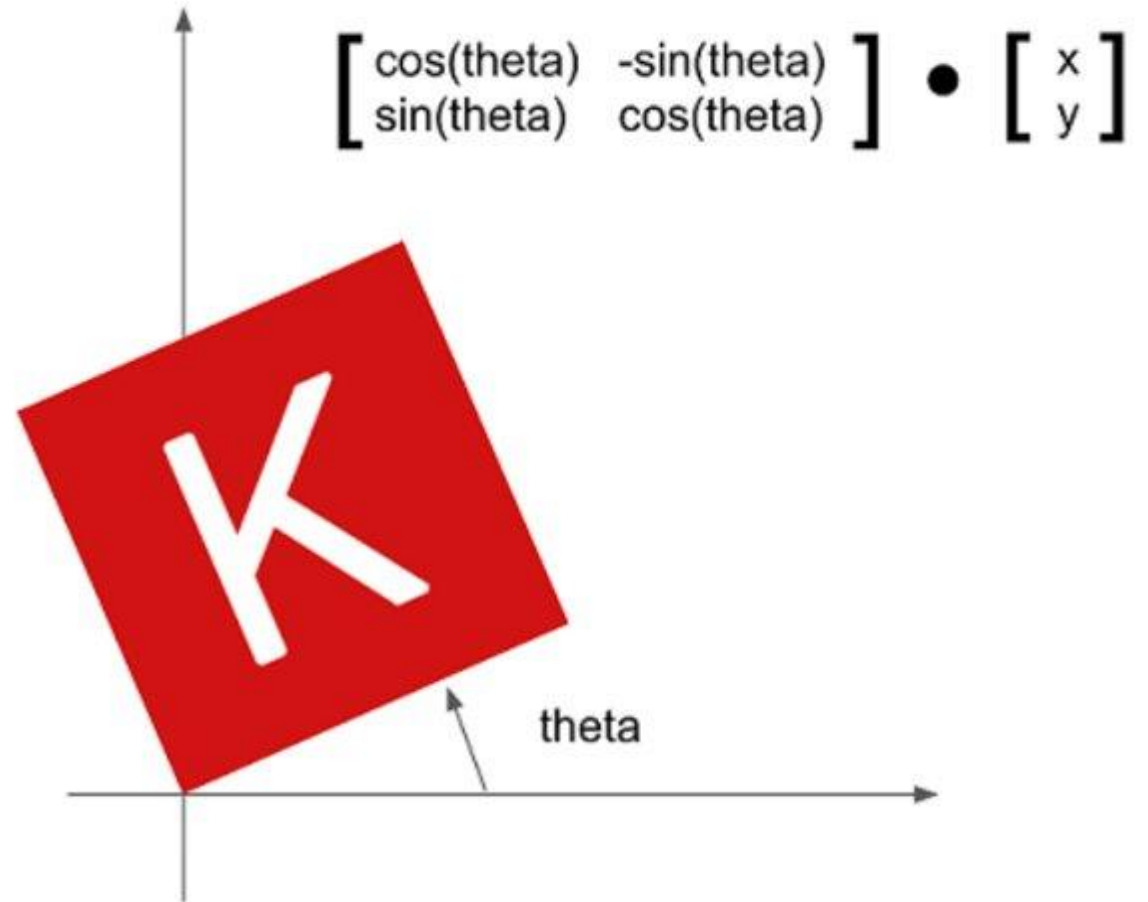
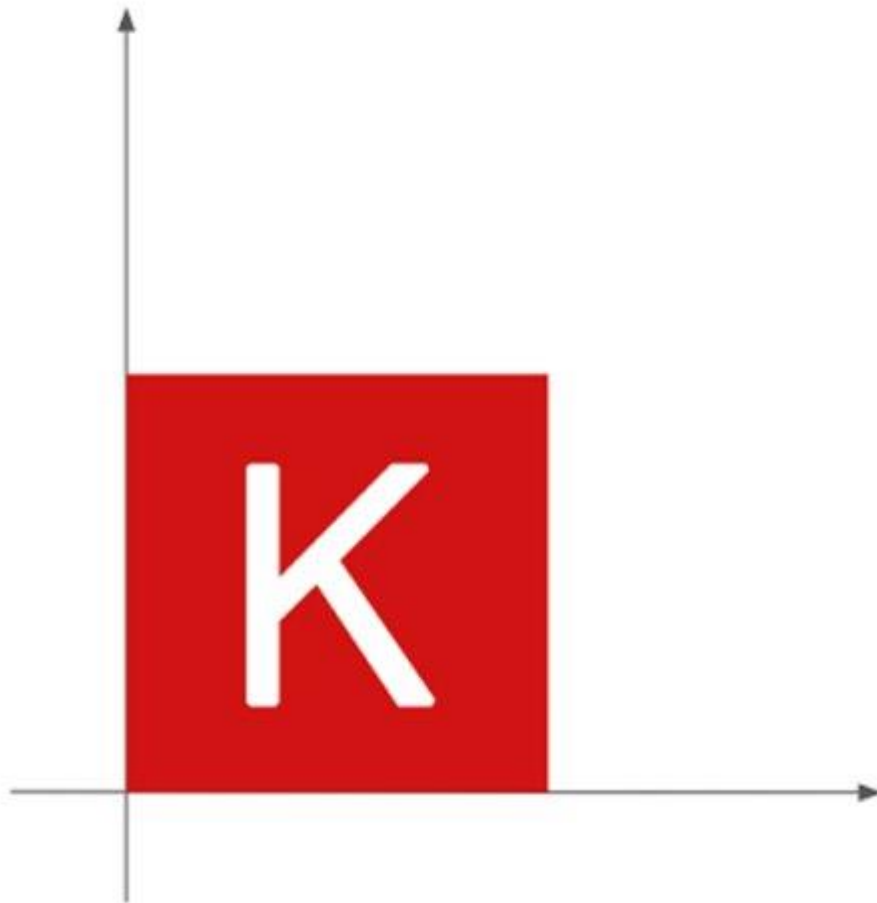
Geometrische Interpretation von Deep Learning - Affine Transformation

Einen Vektor zu einer Punktmenge zu addieren, nennt man „Translation“.



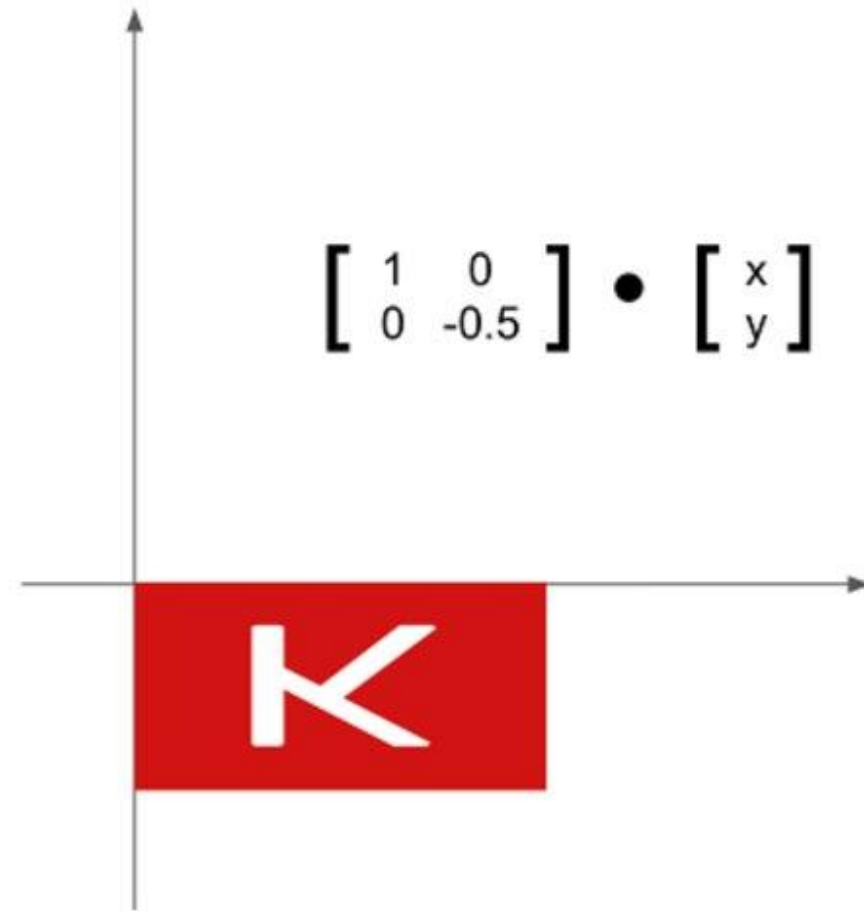
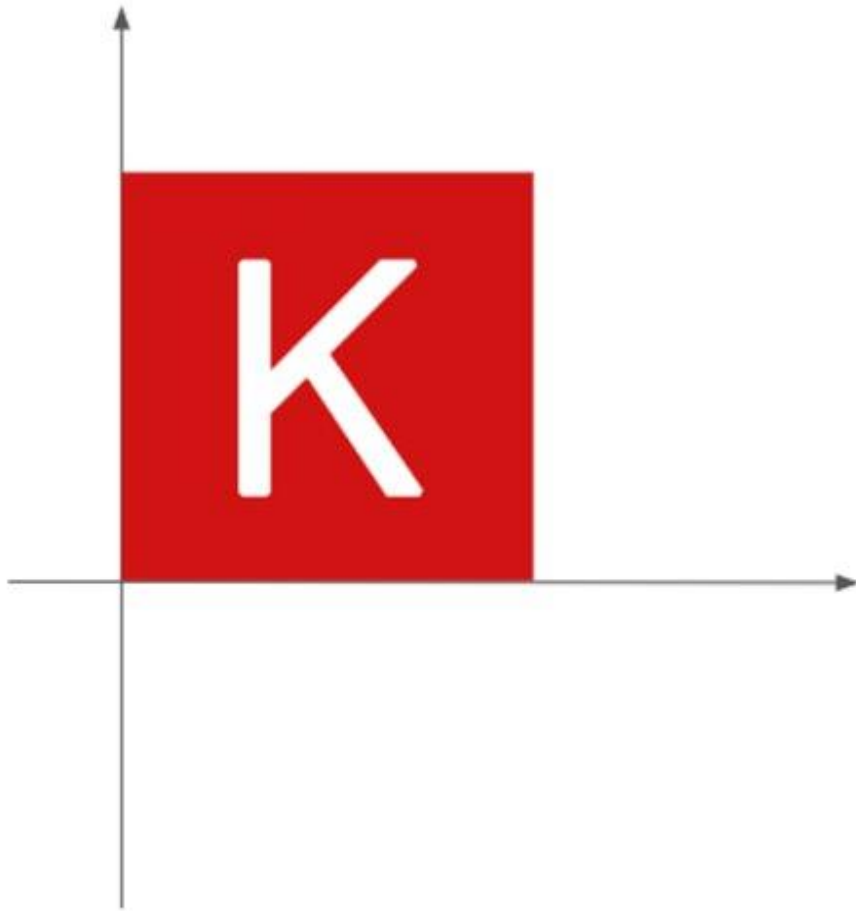
Geometrische Interpretation von Deep Learning - Affine Transformation

Durch eine Matrixmultiplikation kann man Punktmengen rotieren.



Geometrische Interpretation von Deep Learning - Affine Transformation

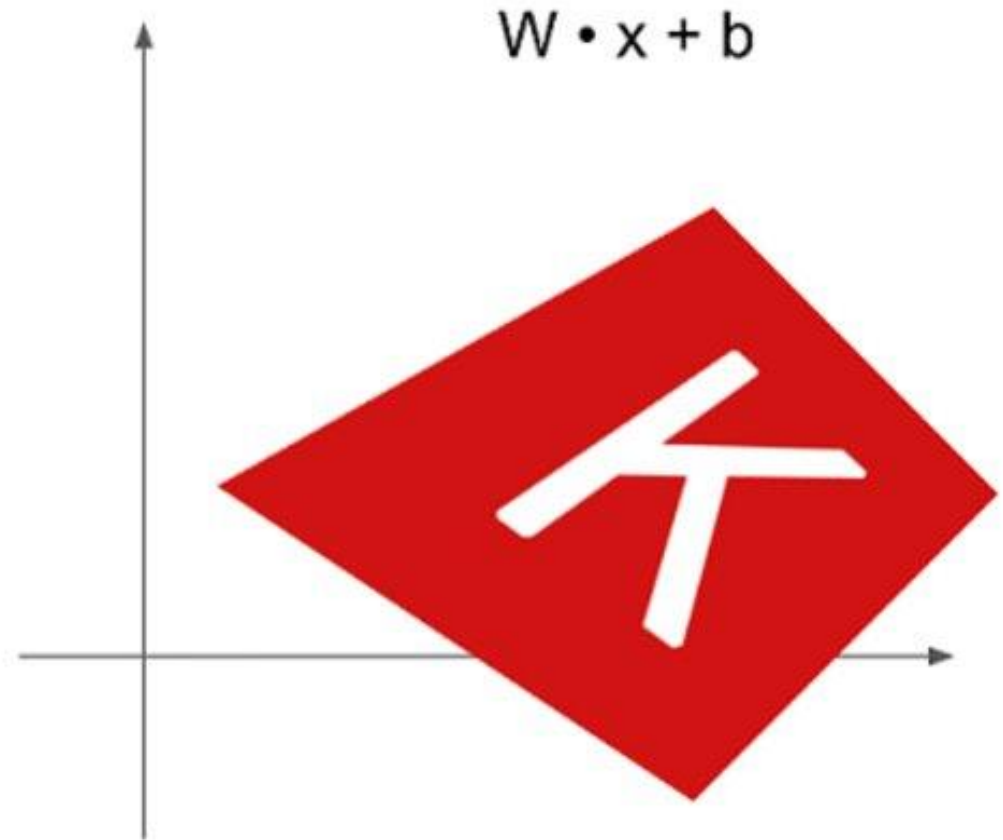
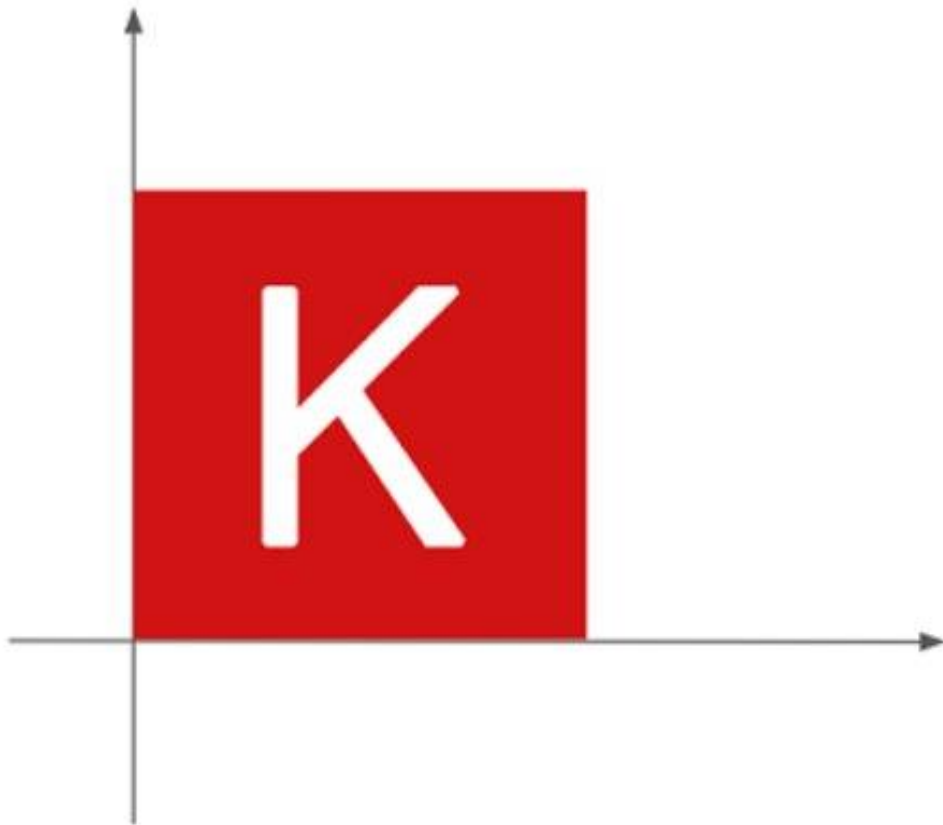
Durch eine Matrixmultiplikation kann man Punktmengen spiegeln und skalieren.



$$\begin{bmatrix} 1 & 0 \\ 0 & -0.5 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

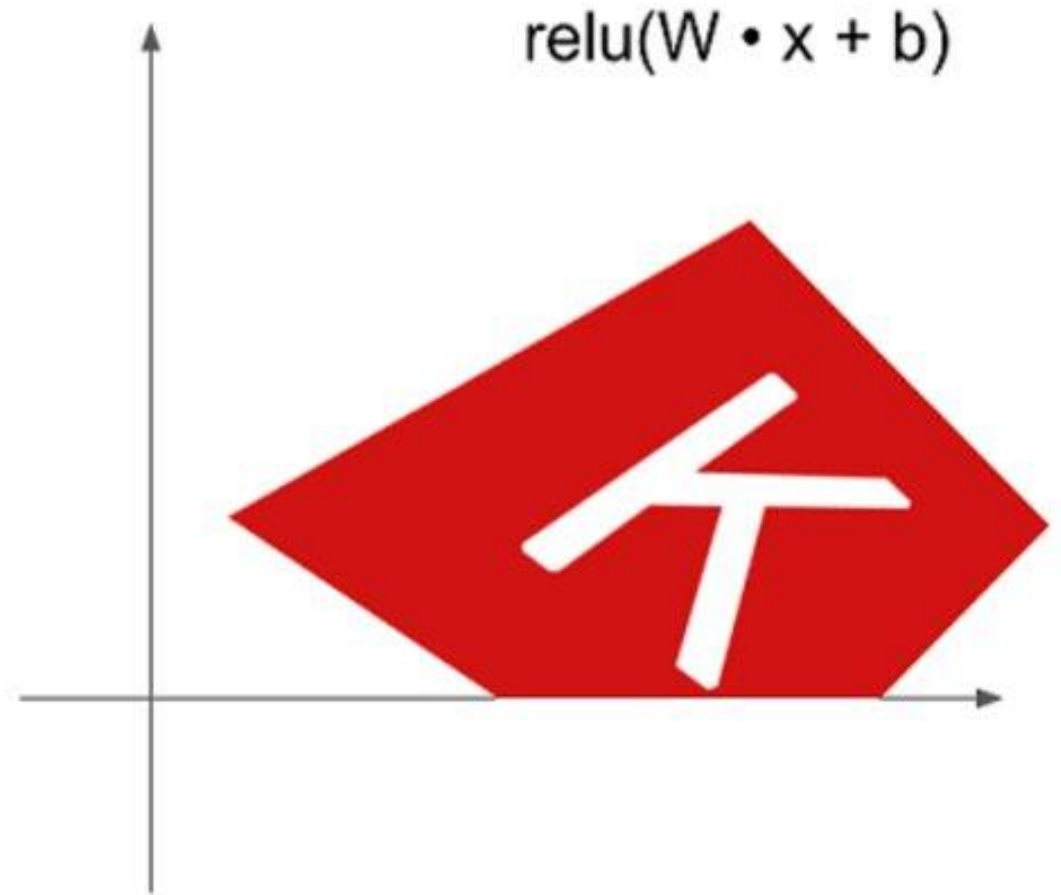
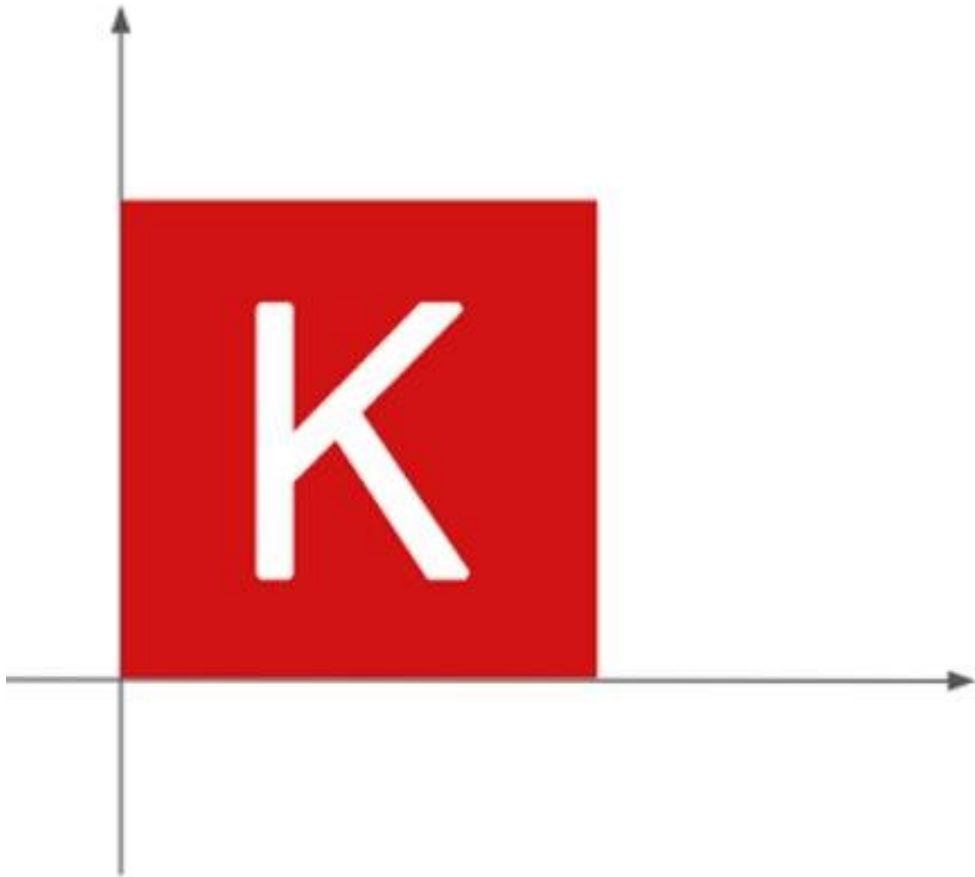
Geometrische Interpretation von Deep Learning - Affine Transformation

Wenn man diese Operationen verbindet, erhält man eine **affine Transformation** - der Input wird rotiert, gespiegelt, verzerrt und verschoben.



Geometrische Interpretation von Deep Learning - Affine Transformation

Durch die Activation-Function wird die Transformation nicht-linear.

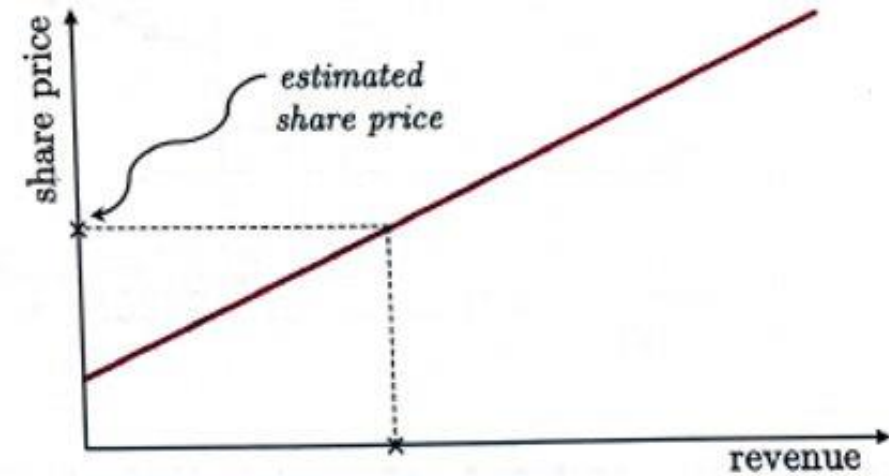
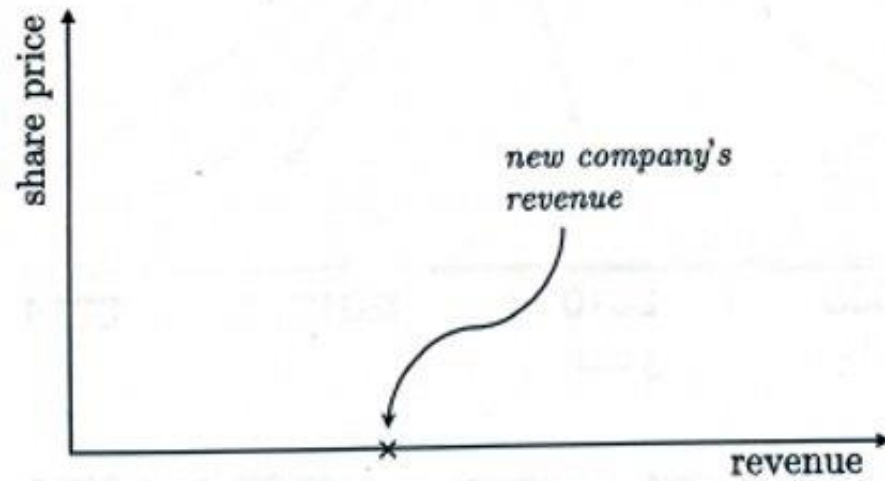
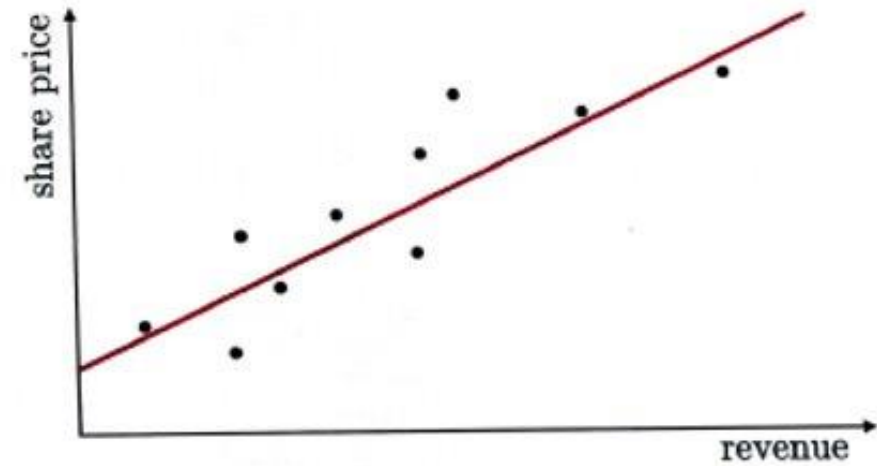
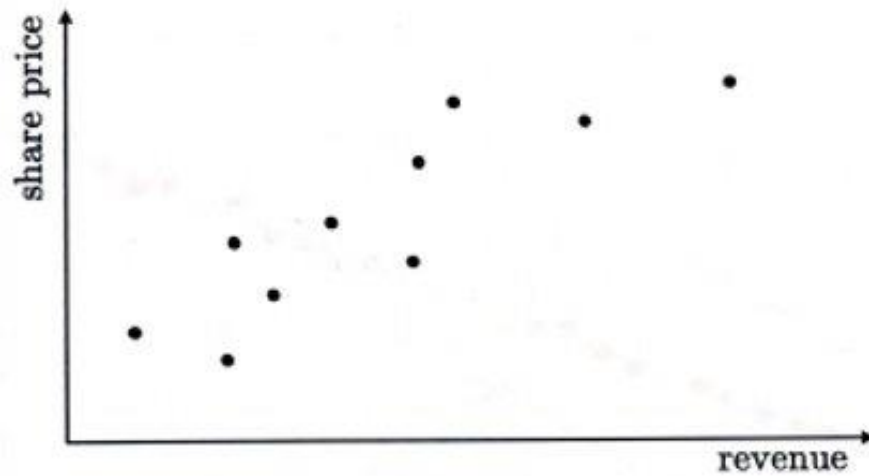


Regression vs. Klassifizierung

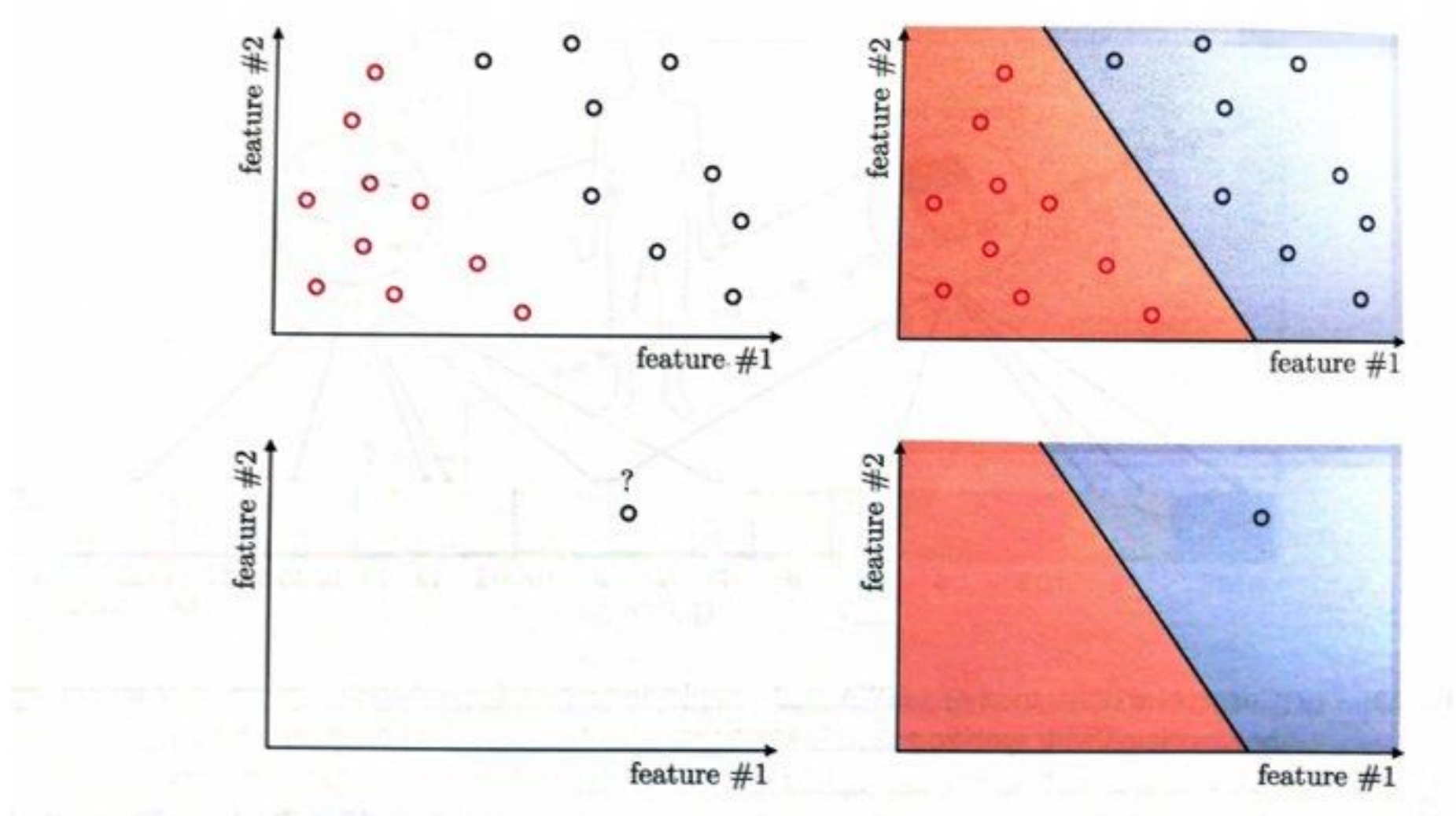
Vorhersage von kontinuierlichen oder diskreten Werten



Regression



Klassifizierung



Lossfunktionen für Klassifizierung

Für Klassifizierung erzeugt das Modell eine Wahrscheinlichkeitsverteilung über alle n Klassen.

$$f(\mathbf{x}) = (q_1, q_2, \dots, q_n)^T, \quad \sum_{i=1}^n q_i = 1$$

- Binäre Klassifizierung $n = 2$ Binäre Cross-Entropy

$$\mathcal{L} = -p \log(q) - (1 - p) \log(1 - q)$$

- Multi-Class Klassifizierung: Cross-Entropy

$$\mathcal{L} = - \sum_{i=1}^n p_i \log(q_i)$$



Output Layers

- Die Wahl der Loss-Funktion ist eng verbunden mit der Wahl der Aktivierungsfunktion der Output Layer:

Regression:

- Loss: Mean-Absolute-Error, Mean-Squared-Error, Huber Loss
- Output: Linear (keine Aktivierung)



Output Layers

Klassifizierung:

Binär:

- **Loss: Binäre Cross-Entropy**
- **Output: Sigmoid (Output ist dann im Bereich (0,1))**

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Multi-Class:

- **Loss: Cross-Entropy**
- **Output: Softmax (Output addiert sich auf 1)**

$$\text{softmax}(\mathbf{x})_i = \frac{e^{x_i}}{\sum_j e^{x_j}}$$



Notebooks

- **02_TrainingClassificationNetwork.ipynb**



Backpropagation

Die Antwort auf die Frage: Wie berechne ich den Gradienten eines neuronalen Netzwerks?

- **Backpropagation ist der Algorithmus, welcher den Gradienten in einem neuronalen Netzwerk berechnet**
- **Nicht zu verwechseln mit dem Gradientenabstieg, welcher mithilfe des Gradienten den Loss minimiert**



Backpropagation '86

Learning representations by back-propagating errors

David E. Rumelhart*, Geoffrey E. Hinton†
& Ronald J. Williams*

* Institute for Cognitive Science, C-015, University of California,
San Diego, La Jolla, California 92093, USA

† Department of Computer Science, Carnegie-Mellon University,
Pittsburgh, Philadelphia 15213, USA

We describe a new learning procedure, back-propagation, for networks of neurone-like units. The procedure repeatedly adjusts the weights of the connections in the network so as to minimize a measure of the difference between the actual output vector of the net and the desired output vector. As a result of the weight adjustments, internal 'hidden' units which are not part of the input or output come to represent important features of the task domain, and the regularities in the task are captured by the interactions of these units. The ability to create useful new features distinguishes back-propagation from earlier, simpler methods such as the perceptron-convergence procedure¹.

There have been many attempts to design self-organizing neural networks. The aim is to find a powerful synaptic modification rule that will allow an arbitrarily connected neural network to develop an internal structure that is appropriate for a particular task domain. The task is specified by giving the

more difficult when we introduce hidden units whose actual or desired states are not specified by the task. (In perceptrons, there are 'feature analysers' between the input and output that are not true hidden units because their input connections are fixed by hand, so their states are completely determined by the input vector: they do not learn representations.) The learning procedure must decide under what circumstances the hidden units should be active in order to help achieve the desired input-output behaviour. This amounts to deciding what these units should represent. We demonstrate that a general purpose and relatively simple procedure is powerful enough to construct appropriate internal representations.

The simplest form of the learning procedure is for layered networks which have a layer of input units at the bottom; any number of intermediate layers; and a layer of output units at the top. Connections within a layer or from higher to lower layers are forbidden, but connections can skip intermediate layers. An input vector is presented to the network by setting the states of the input units. Then the states of the units in each layer are determined by applying equations (1) and (2) to the connections coming from lower layers. All units within a layer have their states set in parallel, but different layers have their states set sequentially, starting at the bottom and working upwards until the states of the output units are determined.

The total input, x_j , to unit j is a linear function of the outputs, y_i , of the units that are connected to j and of the weights, w_{ji} , on these connections

$$x_j = \sum_i y_i w_{ji} \quad (1)$$

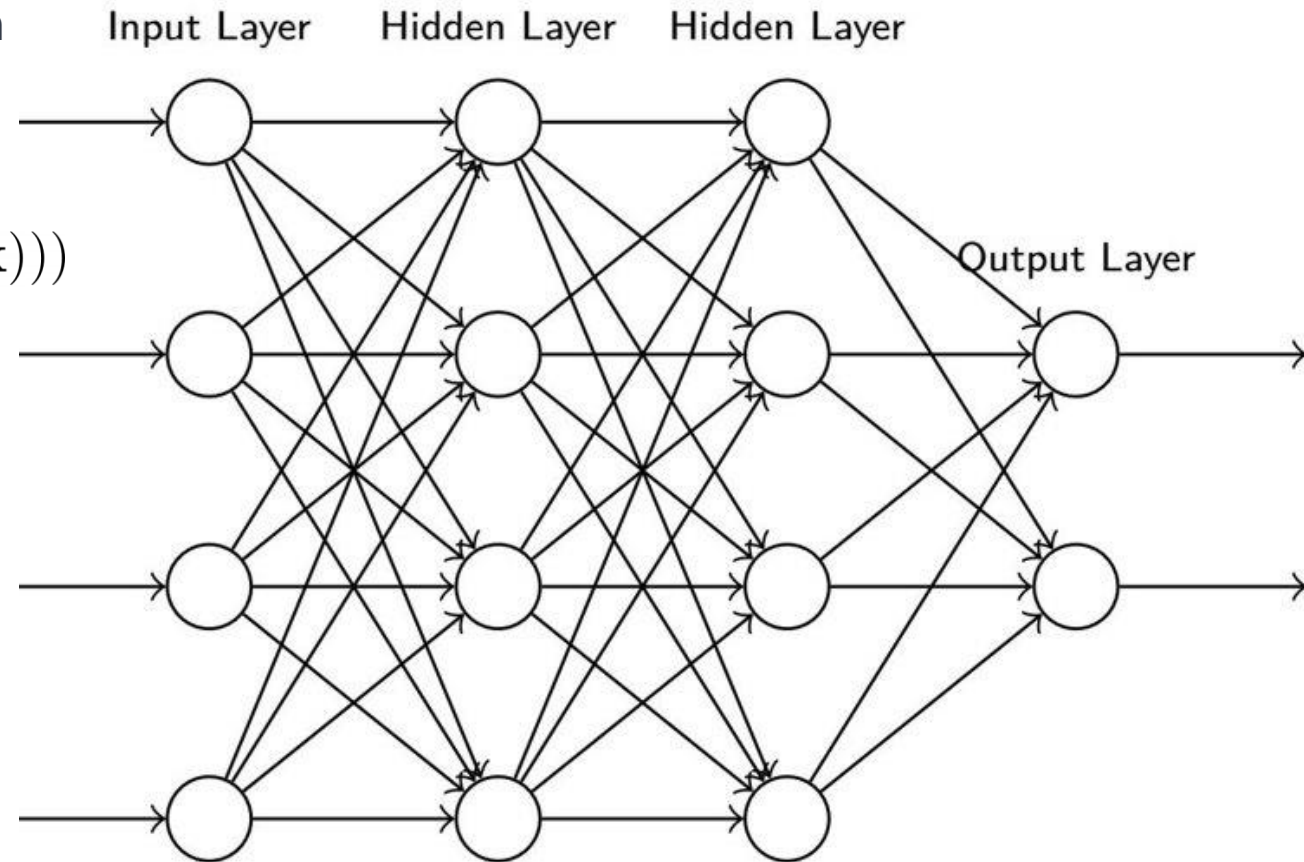


Grundlagen des Deep Learning

- Eine Art über neuronale Netzwerke nachzudenken ist es die Layers als Funktionen mit vektorwertigen Output aufzufassen:

$$f_{\text{Netzwerk}}(\mathbf{x}) = f_{\text{output}}(f_{\text{hidden 2}}(f_{\text{hidden 1}}(\mathbf{x})))$$

- Wir brauchen also eine Regel für die Berechnung der Ableitung verschachtelter Funktionen



Die Ketten-Regel

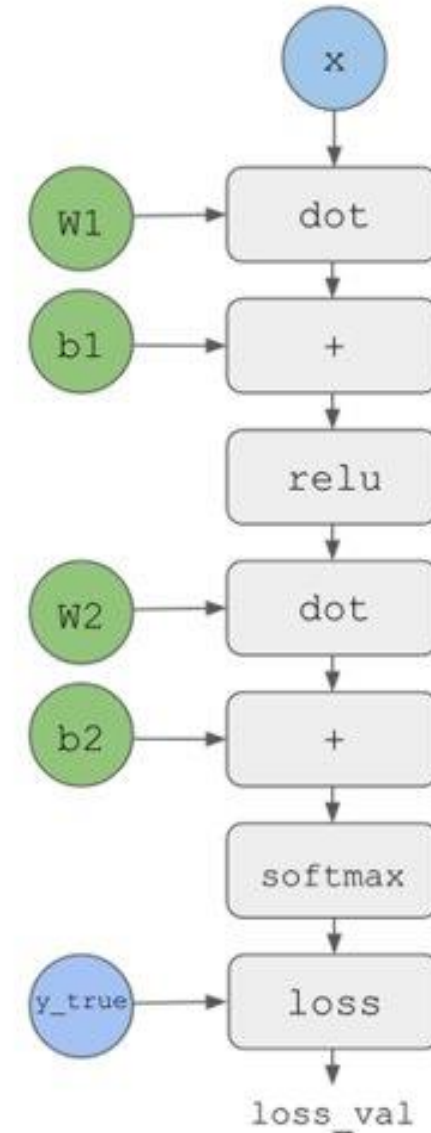
- Neuronale Netze sind ineinander geschachtelte Funktionen
 - Affine Transformationen werden in Activation-Funktionen gesteckt
 - Ein Layer wird in das nächste gesteckt
- Solche Ketten von Funktionen werden nach der Kettenregel abgeleitet
 $(u \circ v)'(x_0) = u'(v(x_0)) \cdot v'(x_0)$ ere Ableitung mal innere Ableitung“.
- In Code mit mehr Funktionen:

```
def fghj(x):  
    x1 = j(x)  
    x2 = h(x1)  
    x3 = g(x2)  
    y = f(x3)  
    return y  
  
grad(y, x) == grad(y, x3) * grad(x3, x2) * grad(x2, x1) * grad(x1, x)
```



Computation Graphs

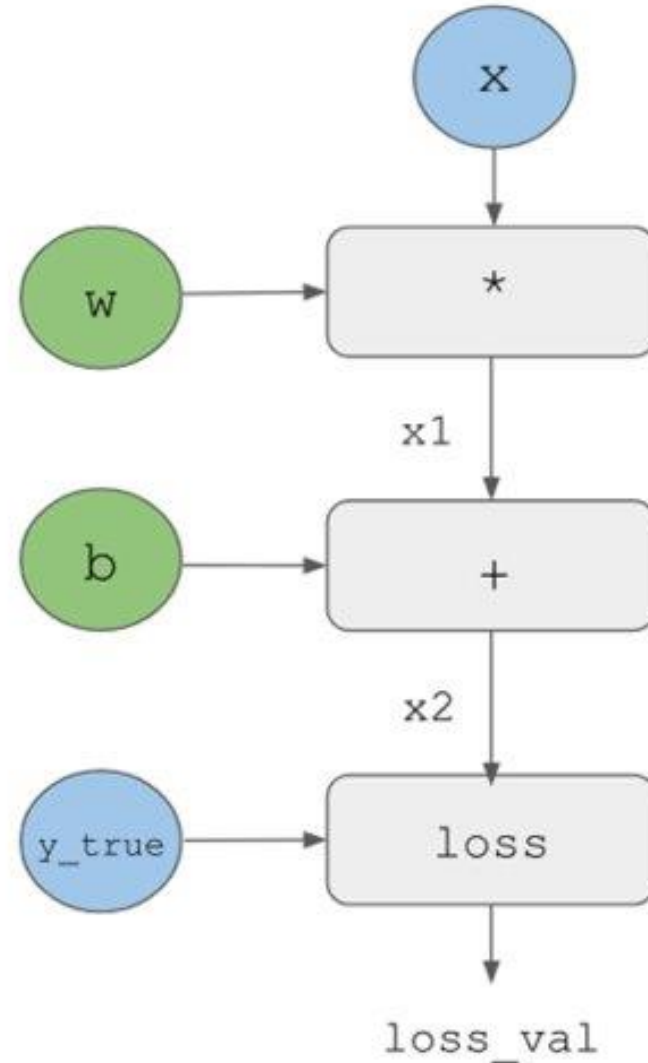
Ein Computation Graph ist ein DAG aus Operationen also Tensoroperationen oder Anwendungen von Funktionen.



Backpropagation im Computation Graph

Vorwärtsthroughgang: `loss_val` wird berechnet

```
loss_val = abs(y_true - y)
```



Backpropagation im Computation Graph

- Vorwärtsthroughgang: loss_val wird berechnet
$$\text{loss_val} = \text{abs}(y_{\text{true}} - y)$$
- Rückwärtsthroughgang: Gradienten für W und b werden berechnet

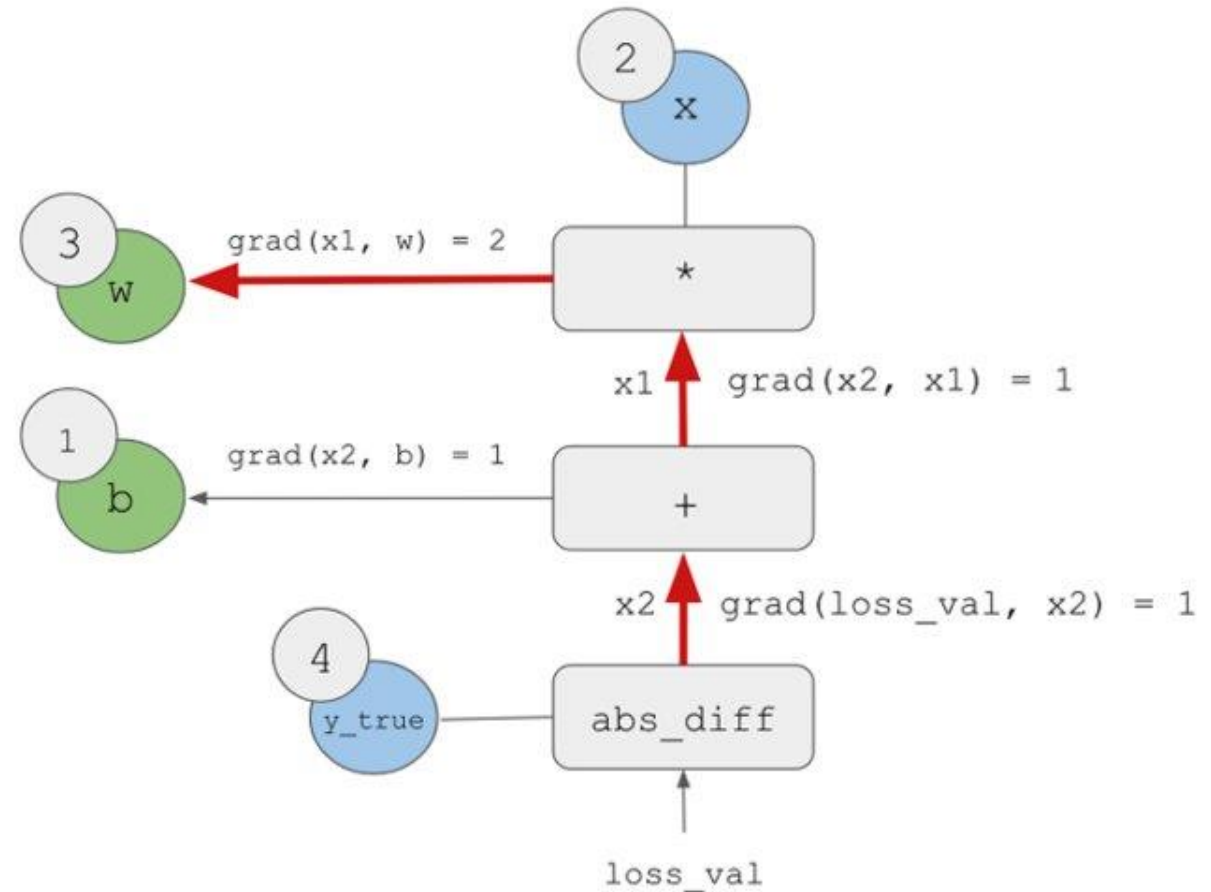
$$\text{grad}(\text{loss_val}, x_2) = 1$$

$$\text{grad}(x_2, x_1) = 1$$

$$\text{grad}(x_2, b) = 1$$

$$\text{grad}(x_1, w) = 2$$

→ Kettenregel: Wir erhalten den Gradienten z.B. von loss_val bezüglich W indem wir die Gradienten der Kanten auf dem Weg von loss_val zu W multiplizieren.



Backpropagation im Computation Graph

Warum ist das toll?

- Automatic Differentiation = die automatische Berechnung von Gradienten
- Backpropagation bedeutet, die Komplexität der Gradientenberechnung ist gleich der Komplexität des Vorwärtsthroughs.

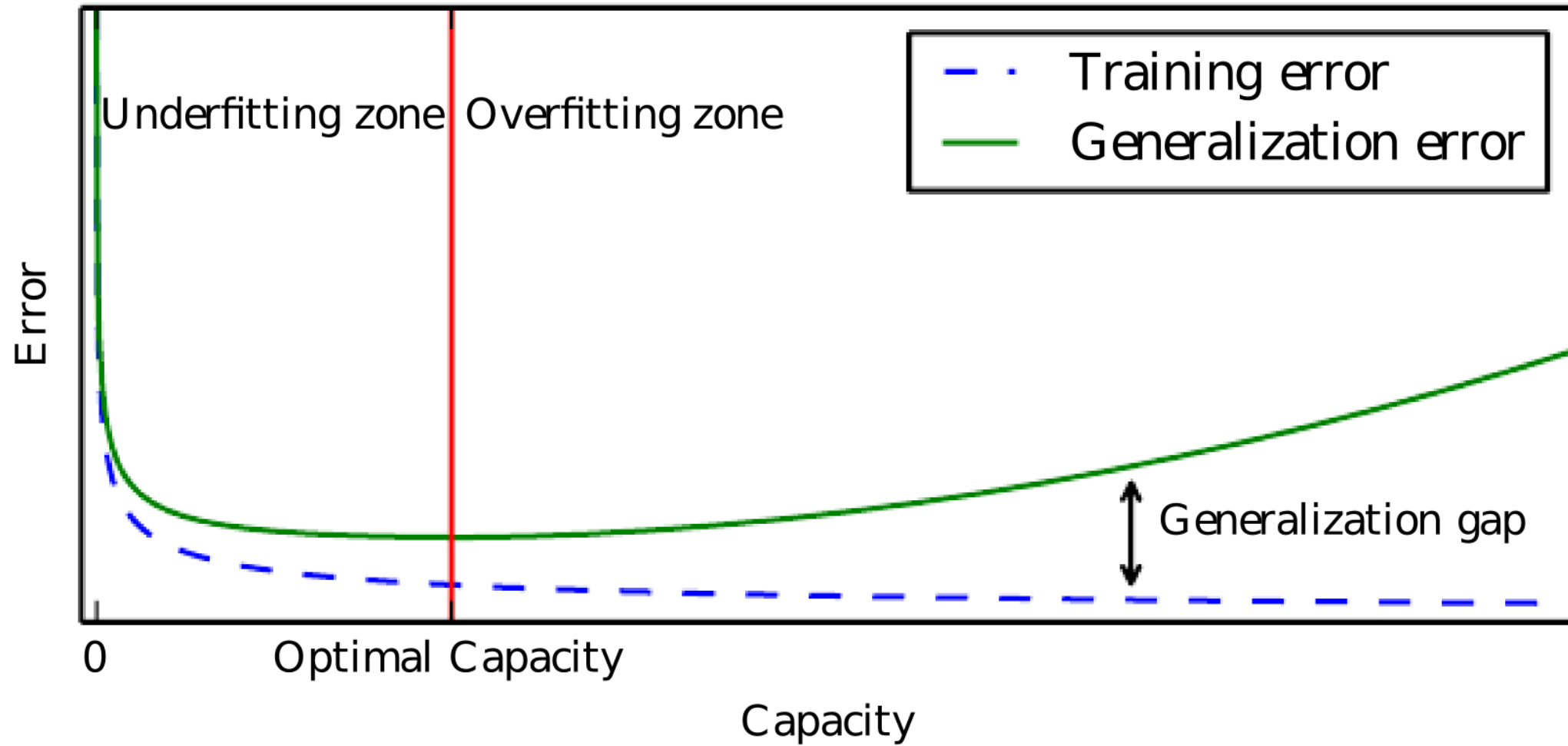


Generalisierung: Overfitting vs. Underfitting

- **Fähigkeit eines Modells auf neue/ungesehene Beispiele zu generalisieren**
- **Overfitting vs. Underfitting**



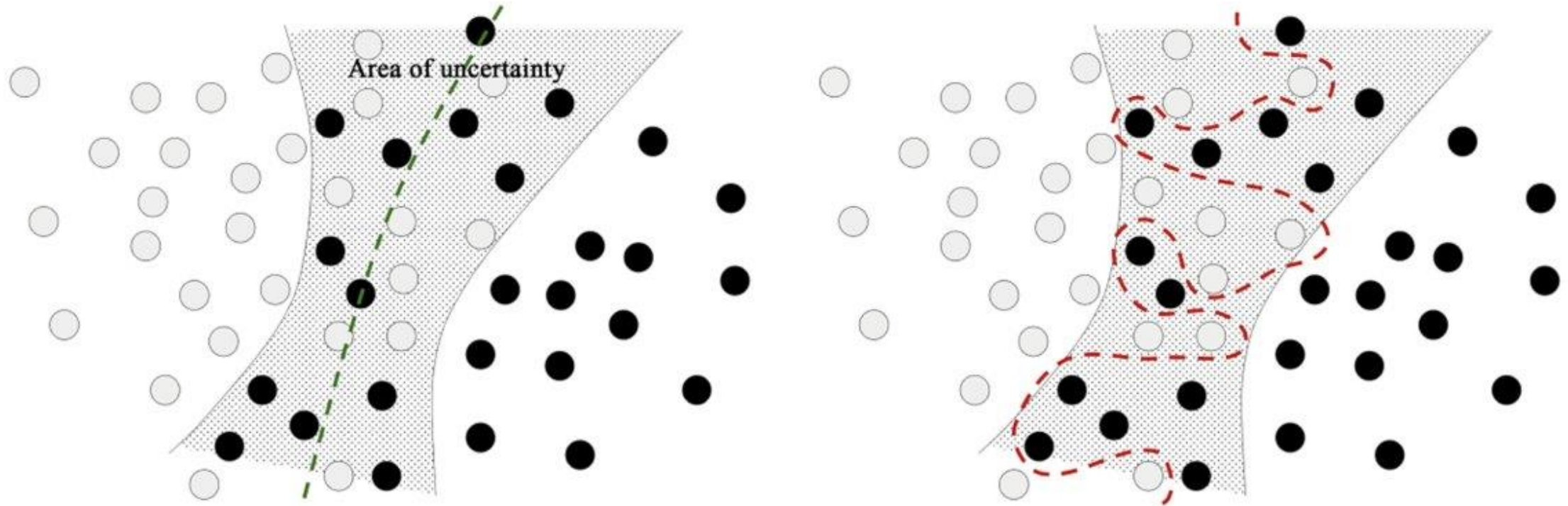
Overfitting vs. Underfitting



Source: Deep Learning from Goodfellow et al.



Das Ergebnis: Schlechte Generalisierung



Einflüsse auf Generalisierung

- Learning rate • Saturisierung
- Datenmenge / Datenkomplexität
- Modelgröße

Regularisierung: Techniken welche die Generalisierung eines Modells verbessern

- L1/L2 - Regularisierung
- Dropout
- Early Stopping



Notebooks

- **03_OneHotEncoding.ipynb**





CONNECTING WORLDS

Kontakt



Marvin Göbel

Data Scientist

@ marvin.goebel@EXXETA.com

m +491522 2690118

