Künstliche neuronale Netzwerke und Tensorflow

Hubl Lukas, Grüneis Dominik

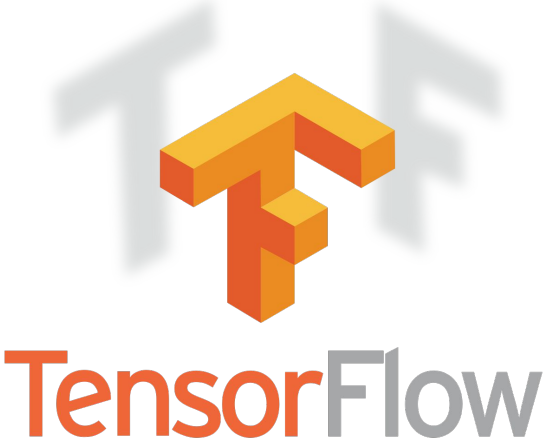
Agenda

- Installation von Tensorflow
- Theorie zu künstliche neurale Netwerke
- Anwenden von Tensorflow mit MNIST

Installation

Installation

- python3: ausführen der Installationsdatei
- pandas: pip install pandas
- matplotlib: pip install matplotlib
- tensorflow:
  - MAC: pip install tensorflow-1.7.0rc1-py3-none-any.whl
  - WIN: pip install tensorflow-1.7.0rc1-cp36-cp36m-win_amd64.whl
  - Ubuntu: pip install tensorflow-1.7.0rc1-cp36-cp36m-linux_x86_64.whl
  - Generell: pip install tensorflow



Setup check

Python 3.5.6

Befehl: “python3” bzw. python

Ausgabe:

```
Lukass-MacBook-Pro:Workshop relevant lukashubl$ python3
Python 3.6.4 |Anaconda, Inc.| (default, Jan 16 2018, 12:04:33)
[GCC 4.2.1 Compatible Clang 4.0.1 (tags/RELEASE_401/final)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Bei Problemen hilft ihnen Herr Hubl gerne weiter

MacOS, Windows & Linux

TensorFlow

Download: https://github.com/lukashubl/ann_tf_ws/blob/master/setupCheck.py

Befehl: “python3 setupCheck.py”

Ausgabe: Skript wird dir sagen ob alles passt

MacOS, Windows & Linux

Theorie

Grundlagen

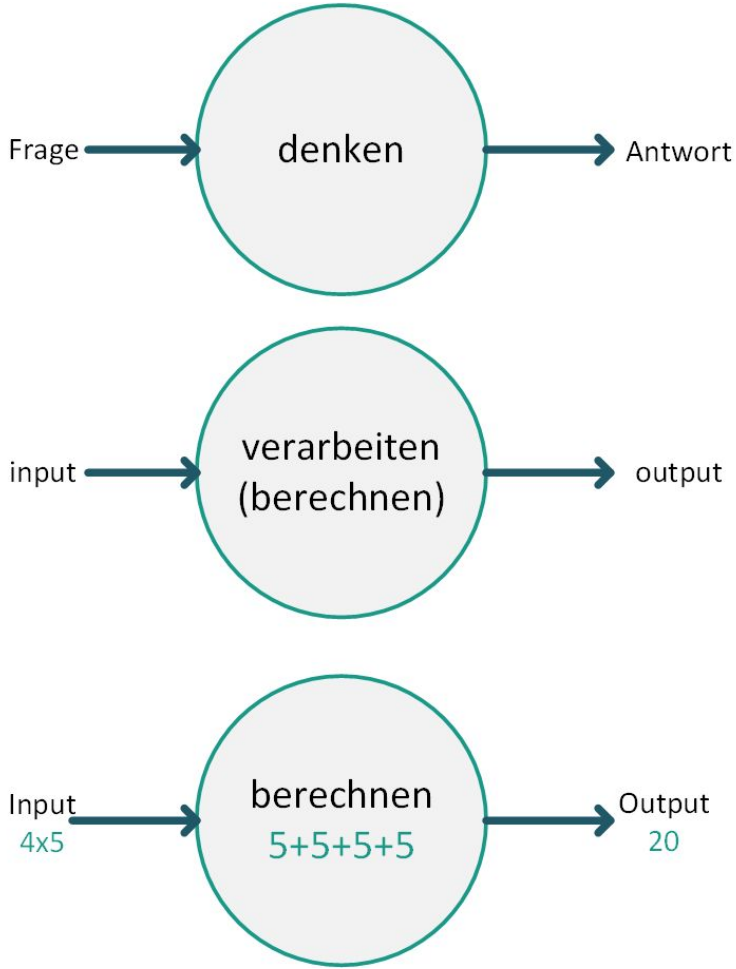
Was sind künstliche neuronale Netzwerke?

Warum möchten wir KNNs verwenden?

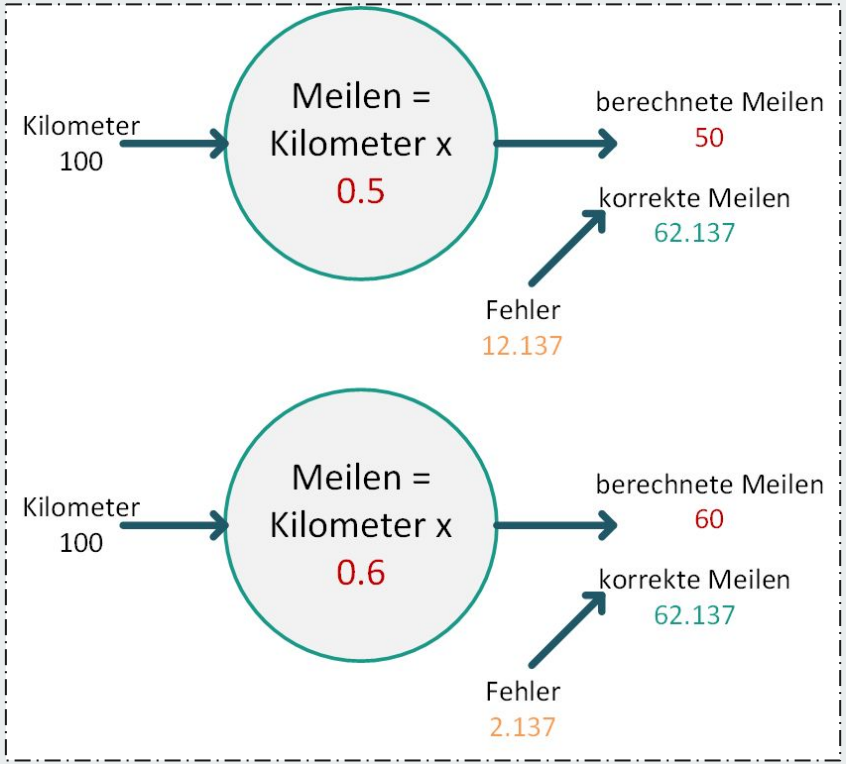
Wie entwickeln wir KNNs?

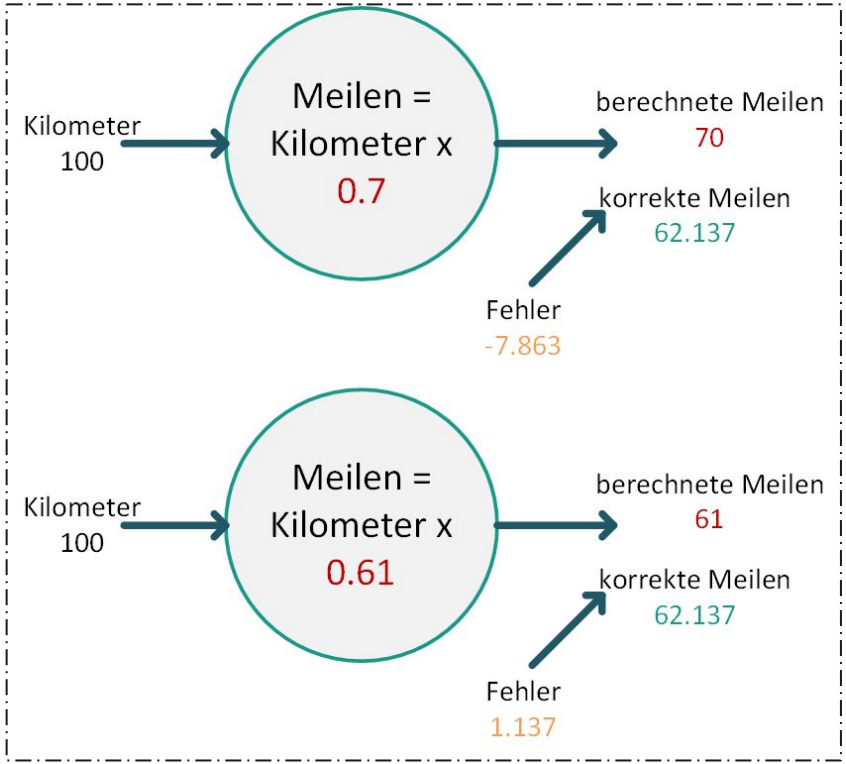
Vergleich

Mensch vs Computer

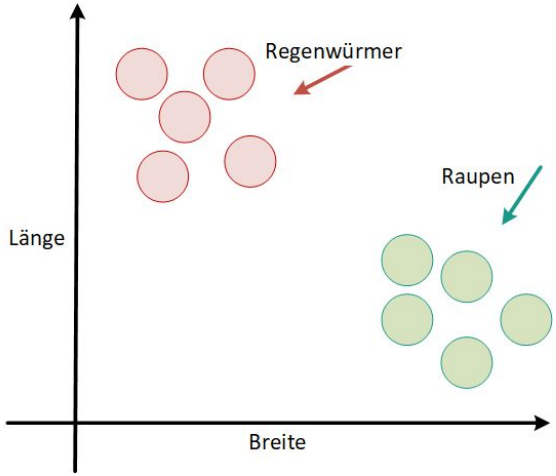


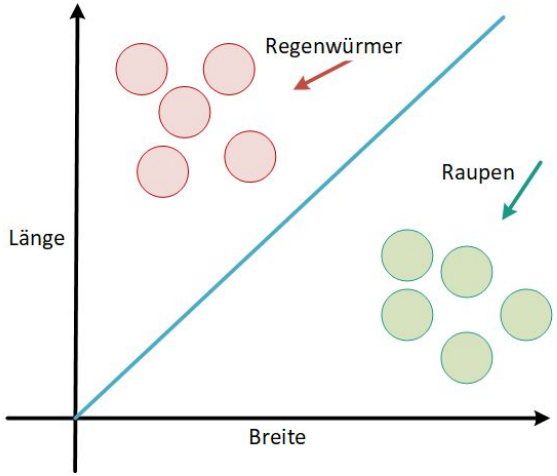
Einfacher Predictor

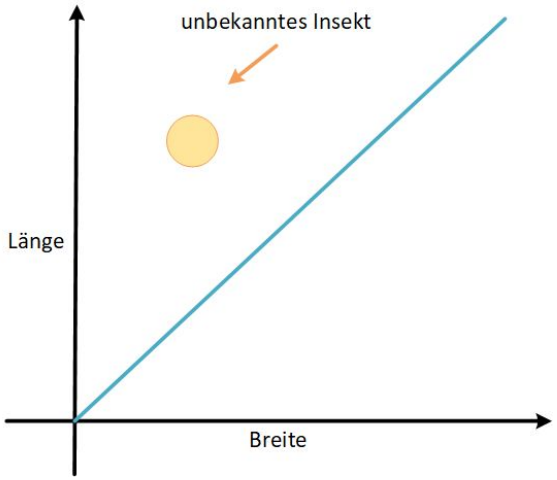




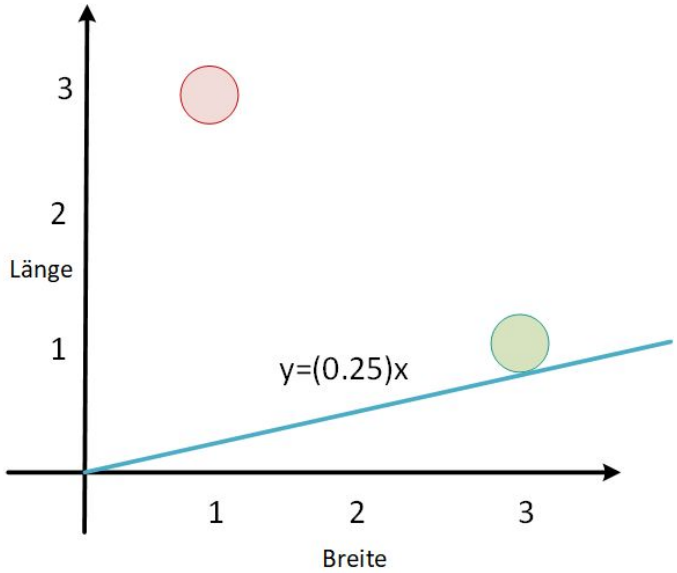
Einfacher Klassifizierer







Einfacher Klassifizierer



x	y	Typ
1	3	Regenwurm
3	1	Raupe

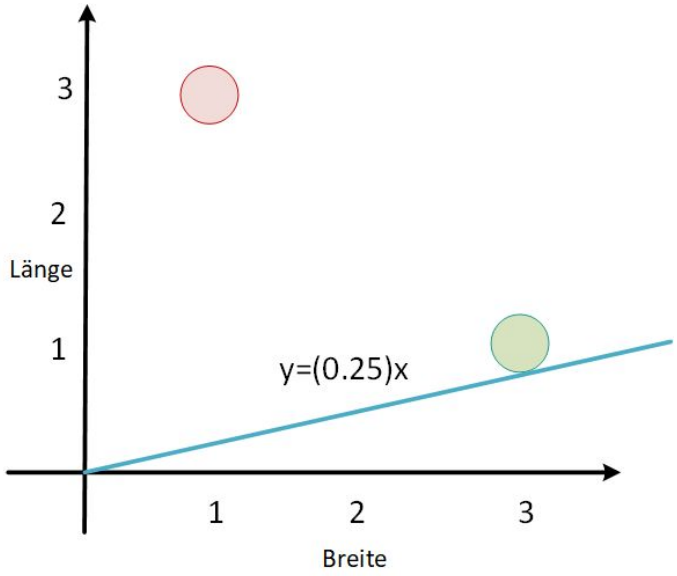
y=Ax

y=0.25x

y=(0.25)*(3.0)=0.75

error=(korrektes Ergebnis - Funktionsergebnis)
E=1.1 - 0.75 = 0.35

Einfacher Klassifizierer



$y=Ax$

$targetY=(A+\Delta A)x$

$targetY-actualY=(A+\Delta A)x - Ax$

$E=targetY-actualY$

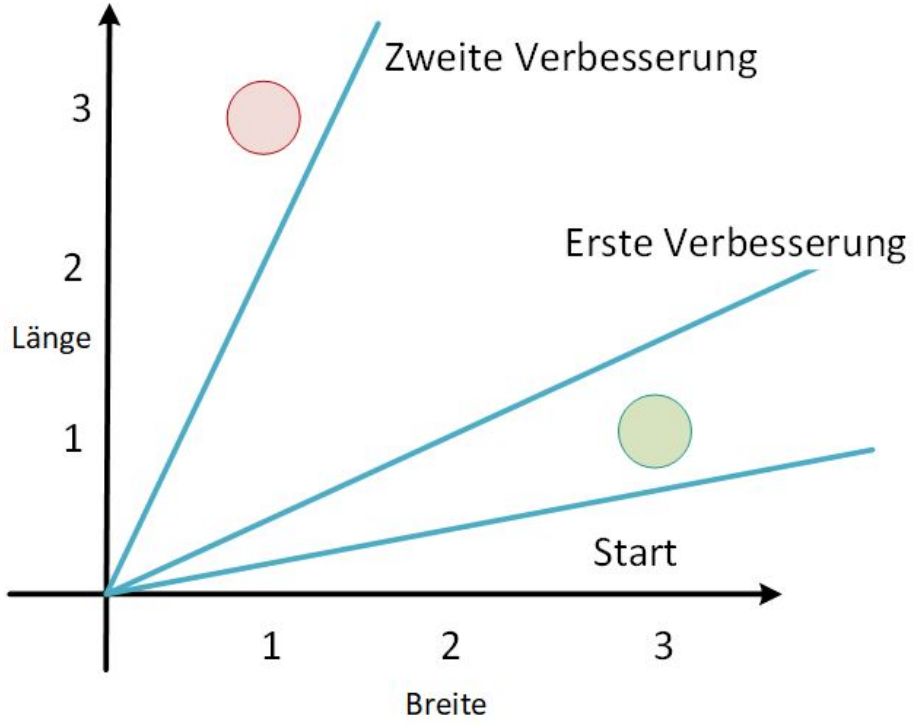
$E=Ax+(\Delta A)x - Ax$

$E=(\Delta A)x$

$\Delta A=E/x$

Einfacher Klassifizierer

x	y	Typ
1	3	Regenwurm
3	1	Raupe



$\Delta A = E / x$

$\Delta A = 0.35/3 = 0.1167$

$y = 0.3667x$

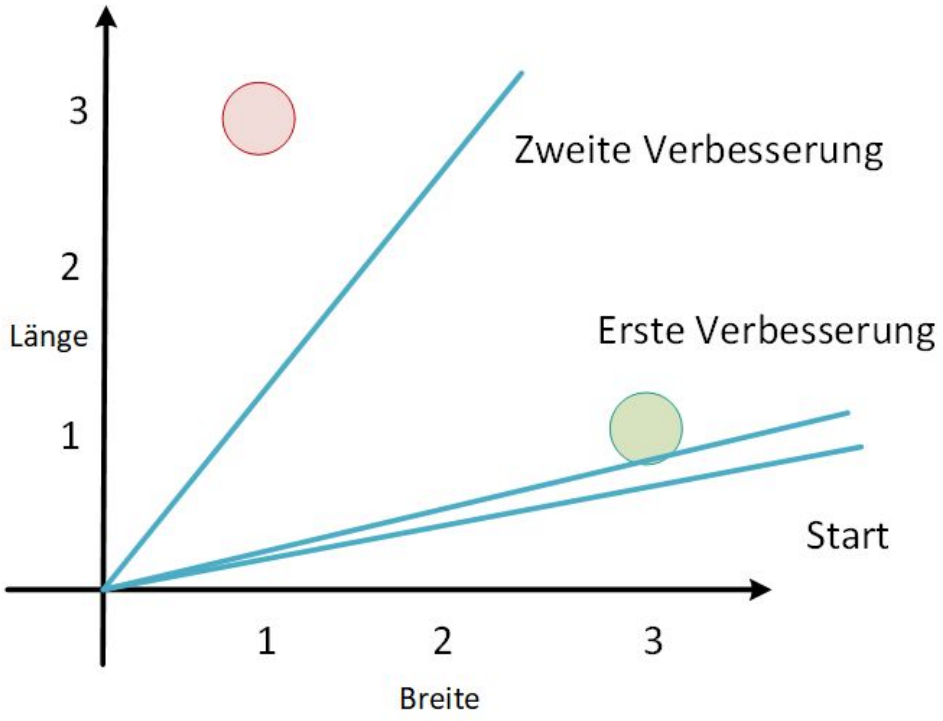
$y = 0.3667 * 1.0 = 0.3667$

$E = 2.9 - 0.3667 = 2.5333$

$\Delta A = 2.5333 / 1.0 = 2.5333$

$y = 2.9x$

Einfacher Klassifizierer



x	y	Typ
1	3	Regenwurm
3	1	Raupe

Learning rate L

$\Delta A = (E / x) L$

$L = 0.5$

$\Delta A = (0.35 / 3) 0.5 = 0.0583$

$y = 0.3083x$

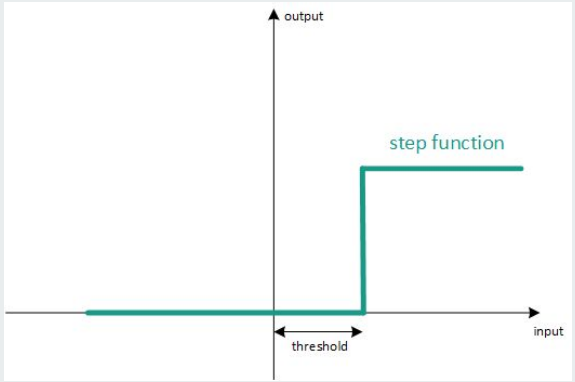
$y = 0.3083 * 1.0 = 0.3083 \rightarrow E = 2.5917$

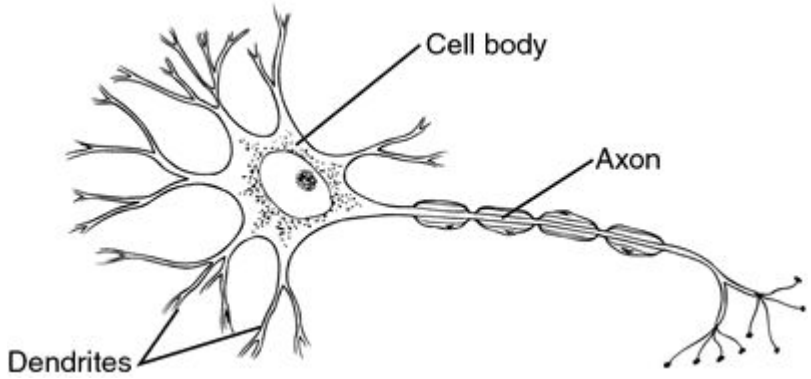
$\Delta A = (2.5917 / 1.0) * 0.5 = 1.2958$

$y = 2.9x$

Biologisches neuronales Netwerk

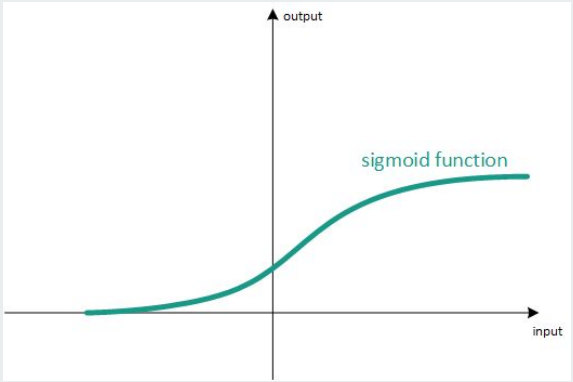
- Dendriten = Empfangseinheit
- Axon = Sendeeinheit
- Zellkern = Entscheidungseinheit
- Threshold muss überschritten werden -> Signal am Axon
- Axon über Synapsen verbindung zu anderen Neuronen

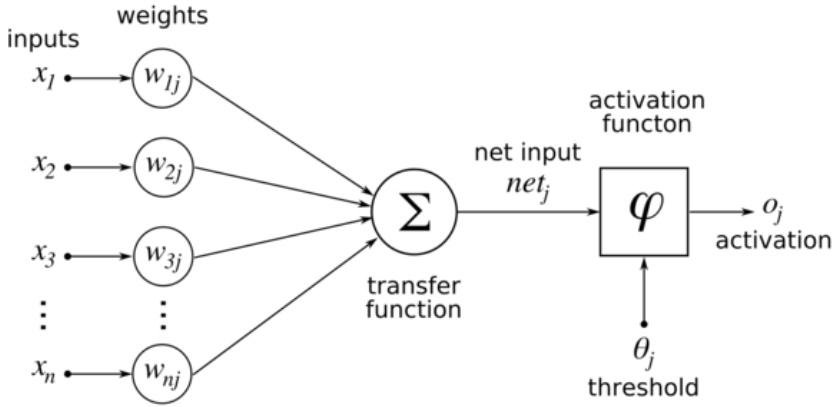


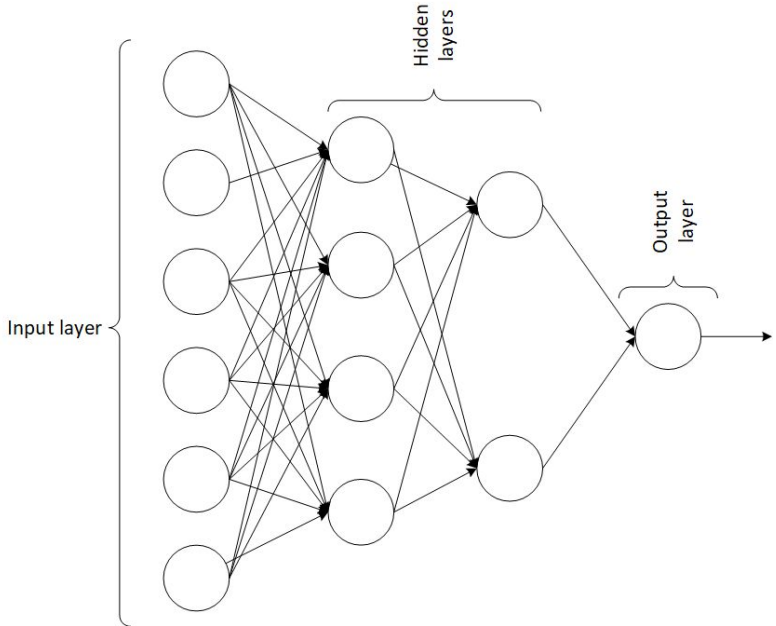


Künstliches neuronales Netwerk

- Gewichtete Inputs = Dendriten
- Aktivierungsfunktion = Threshold
- Ausgang = Axon
- Gewichtete Inputs werden summiert und dann an eine Aktivierungsfunktion gesendet







Einfaches KNN

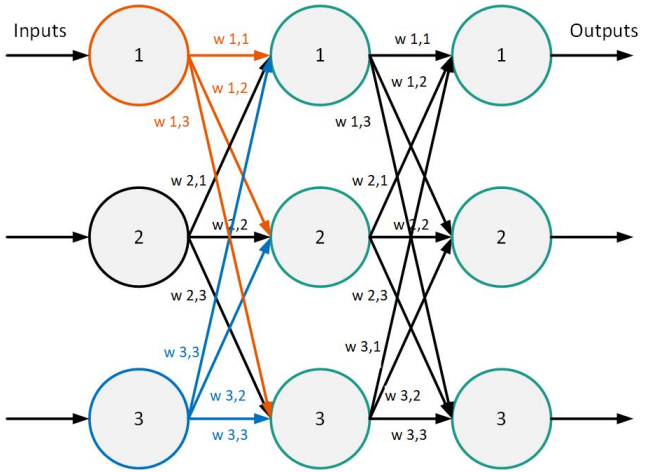
- Jede Node mit Node in nächster Layer verbunden
- Ermöglicht einfache Matrixmultiplikation
- Signal wird durchgepasst

$$X_{hidden} = W_{input\_hidden} \cdot I$$

$$X_{hidden} = \begin{pmatrix} 0.9 & 0.3 & 0.4 \\ 0.2 & 0.8 & 0.2 \\ 0.1 & 0.5 & 0.6 \end{pmatrix} \cdot \begin{pmatrix} 0.9 \\ 0.2 \\ 0.1 \end{pmatrix}$$

$$X_{hidden} = \begin{pmatrix} 1.16 \\ 0.42 \\ 0.62 \end{pmatrix}$$

$$O_{hidden} = sigmoid \begin{pmatrix} 1.16 \\ 0.42 \\ 0.62 \end{pmatrix} = \begin{pmatrix} 0.761 \\ 0.603 \\ 0.650 \end{pmatrix}$$



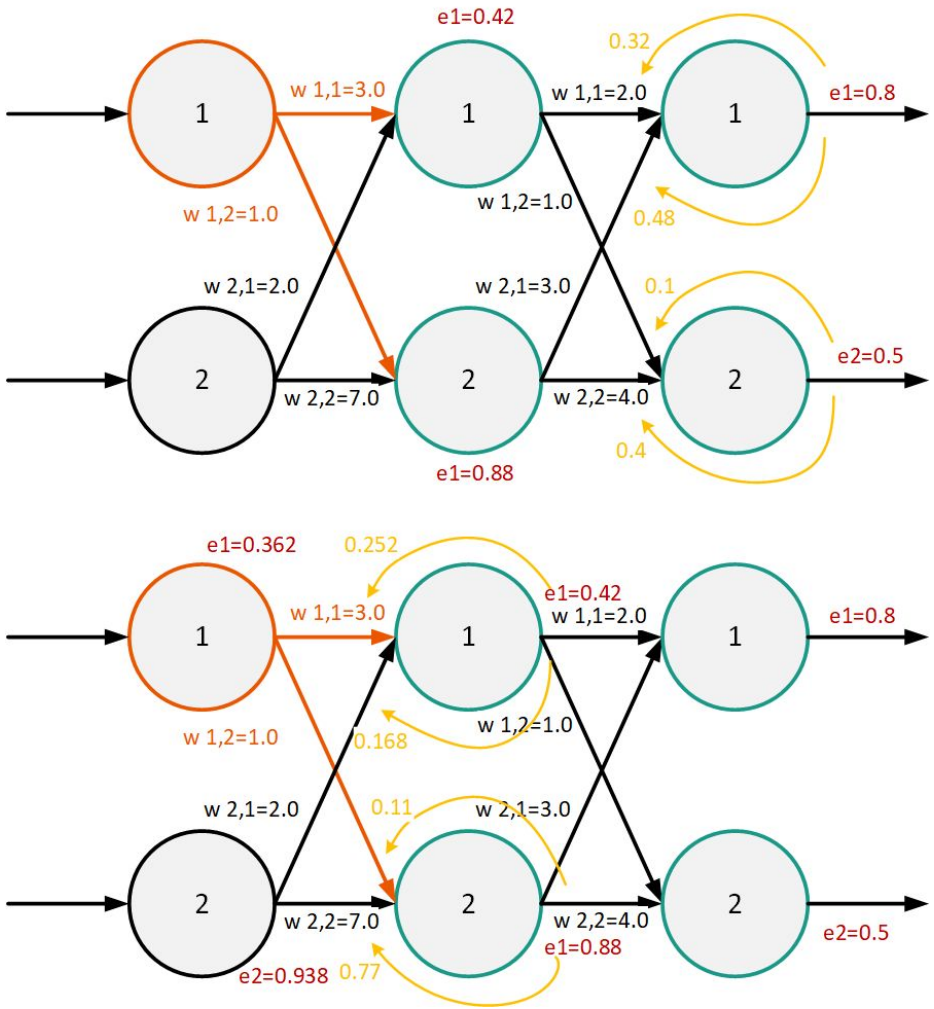
w 1,1	w 1,2	w 1,3
w 2,1	w 2,2	w 2,3
w 3,1	w 3,2	w 3,3

0.9	0.3	0.4
0.2	0.8	0.2
0.1	0.5	0.6

Inputs
0.9
0.1
0.8

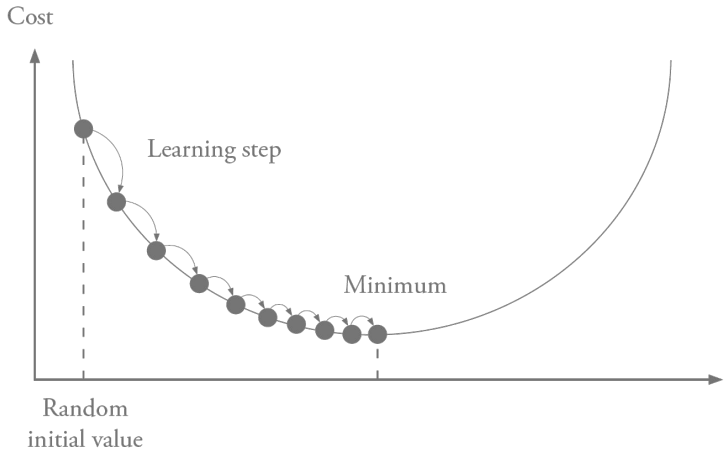
Einfaches KNN

- Fehler Backpropagation
- Aufteilung des Fehler je nach gewichtung
- Summieren der Fehler
- Zurückpropagieren des Fehlers auf den nächsten Layer
- Vereinfacht uns Tensorflow



Updaten der Gewichte

- Kompliziertester Teil
- Schritt für Schritt
- Berechnung über die Steigung Δ
- Vereinfacht uns Tensorflow



$$\frac{\delta E}{\delta w_{jk}} = -(t_k - O_k) \cdot afunc(\sum w_{jk} \cdot O_j)(1 - afunc(\sum w_{jk} \cdot o_j)) \cdot o_j$$

$$w_{jkneu} = w_{jkold} - \alpha \cdot \frac{\delta E}{\delta w_{jk}}$$

$$\Delta w_{jk} = \alpha \cdot E_k \cdot O_k(1 - O_k) \cdot O_j^T$$

Tensorflow-Workshop

Agenda

Kennenlernen und verwenden von MNIST

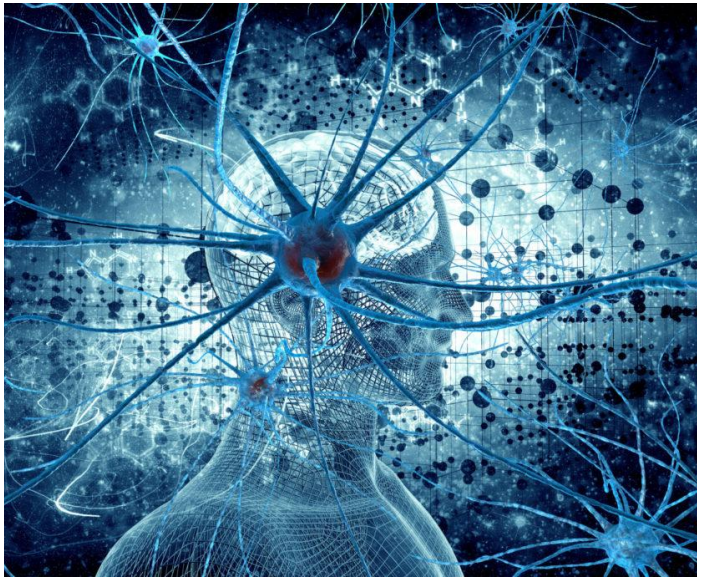
Erstellen des Models

Festlegen des Bewertungsschemas

Das Model trainieren

Evaluieren der Ergebnisse

Visualisieren und wiederverwenden des Models



MNIST Klassifizierung in TensorFlow

MNIST | Model | Bewertungsschema | Training | Evaluieren & Visualisieren | Verwenden in Realtime

MNIST

- Was ist MNIST
- Aufbau der Daten
- Code

MNIST Datenbank

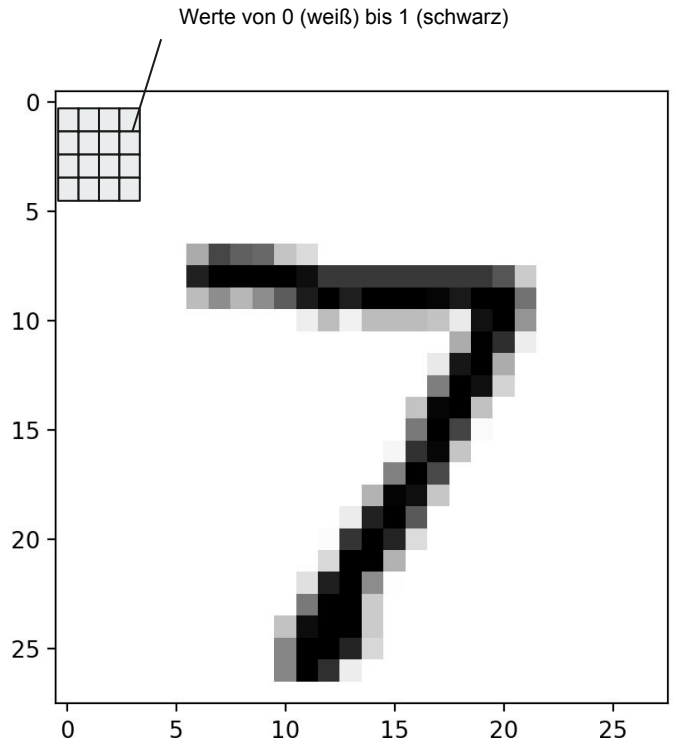
- Modified Institute of Standards and Technology

Warum sind Test- und Trainings-Datensätze getrennt?

- Bilder von handschriftlichen Ziffern
- 60.000 Trainings-Datensätze
- 10.000 Test-Datensätze

MNIST Datenbank

- Bitmap
- 28 x 28 Pixel
- Pixelwert zwischen 0 - 1
- Jedes Bild hat dazugehöriges Label



Coding time

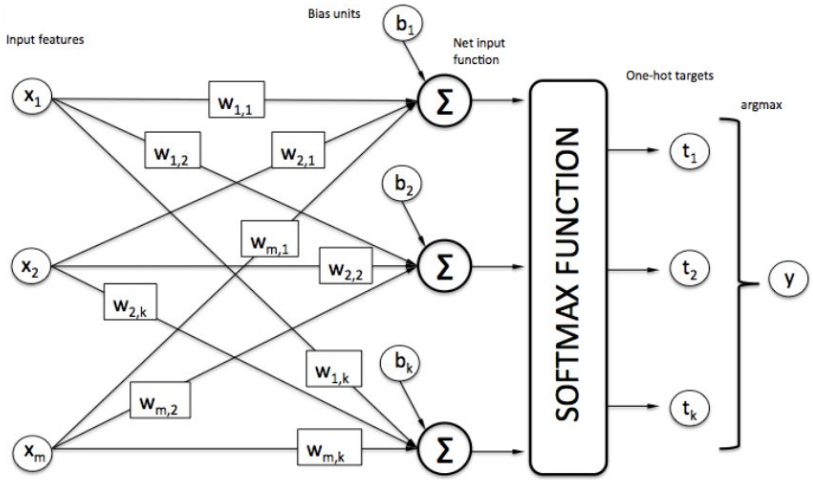
MNIST Klassifizierung in TensorFlow

Modell

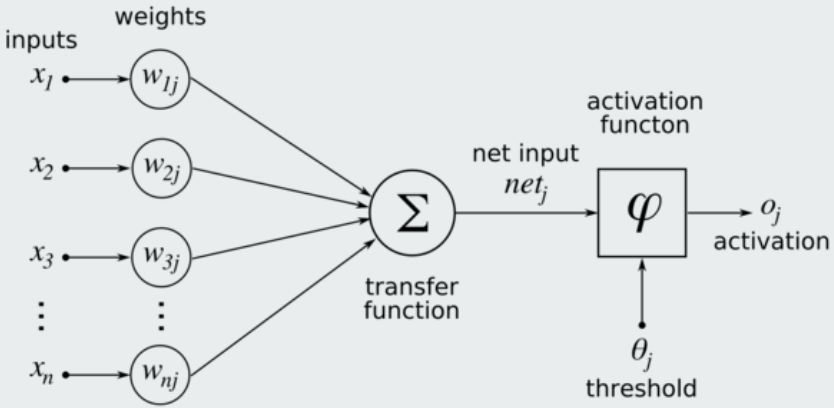
- Aufgabe des Models
- Softmax
- Code

Model

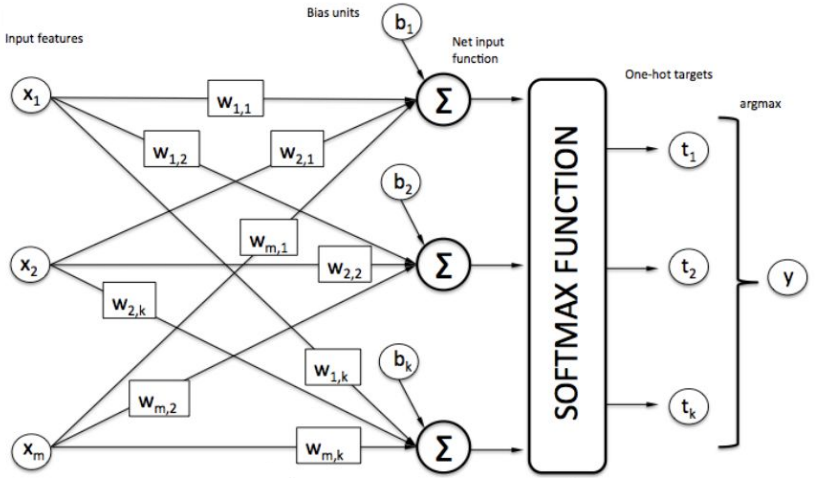
- liefert Output zu Input
- muss trainiert werden
- Logistic Regression
- Softmax Regression



Model



Logistic Regression

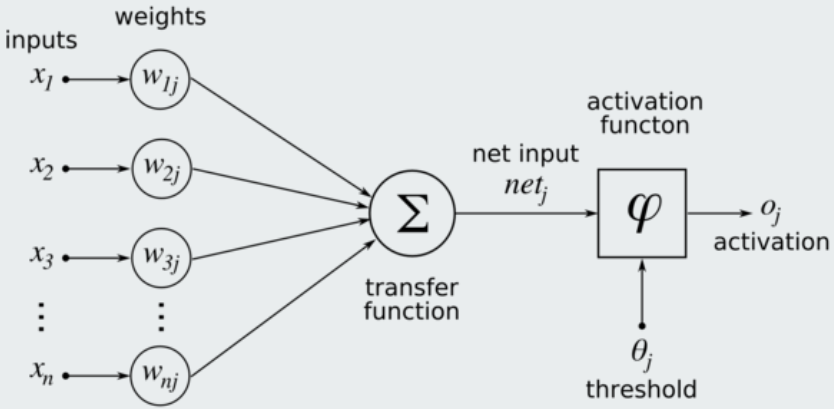


Softmax Regression

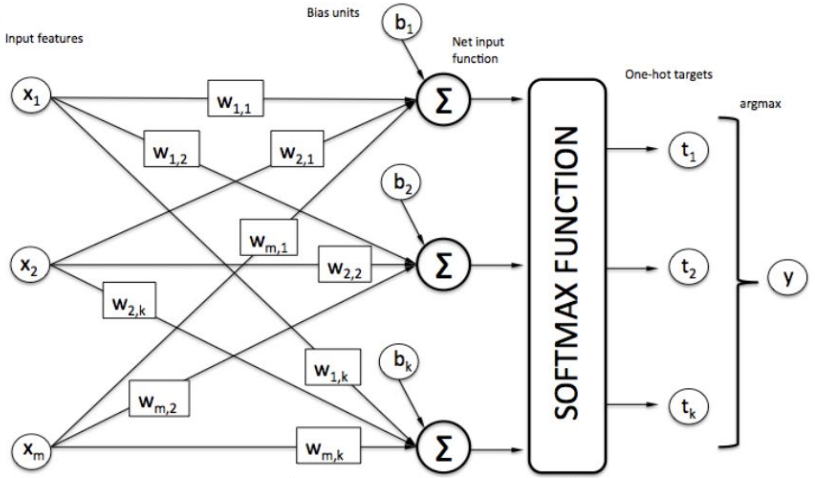
Modell

Was ist der grundlegende Unterschied zwischen den beiden Modellen?

Model



Logistic Regression



Softmax Regression

Coding time

MNIST Klassifizierung in TensorFlow

MNIST | Model | Bewertungsschema | Training | Evaluieren & Visualisieren | Verwenden in Realtime

Bewertungsschema

- Größe die es zu optimieren gilt
- Cross-Entropy
- Code

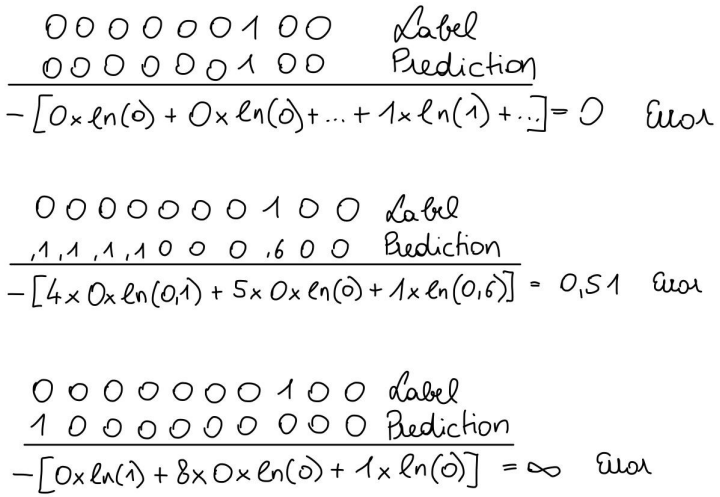
Bewertungsschema

- Wie können die Gewichte optimiert werden?
- Welche Größe ist ausschlaggebend?
- Notwendig um zu lernen
- Cross-Entropy

Cross-Entropy

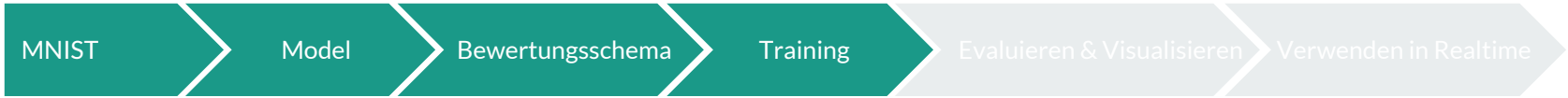
- Unterschied zwischen der echten Verteilungsfunktion und jener des Models
- Cross-Entropy als ausschlaggebende Größe
- Wird dieser Wert minimiert entspricht Prediction dem Label → Ergebnis ist korrekt

Rechenbeispiel: Zahl 7



Coding time

MNIST Klassifizierung in TensorFlow



Training

- Gradient Descent
- Parameter
- Code

Training

Bereits geschafft:

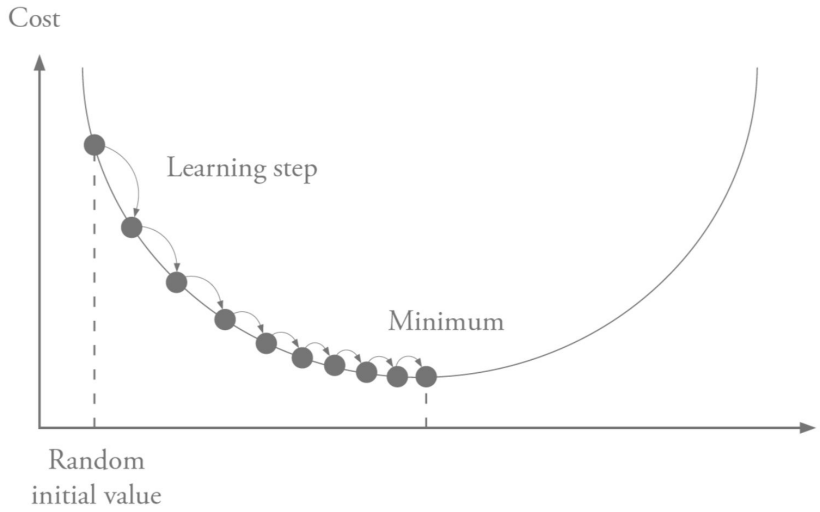
- Daten laden
- Model erstellt
- Bewerten der Ergebnisse

Nächster Schritt:

- Verbessern des Models durch Training

Gradient Descent

- Bergsteiger-Beispiel
- Berg → Fehlerfunktion (Cross-Entropy)
- Position des Bergsteigers → Fehler
- Schritt Bergab → Lernschritt



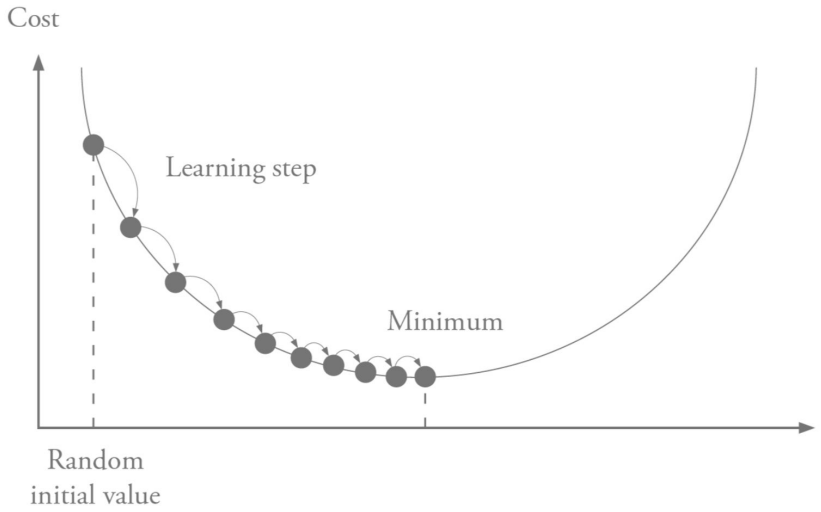
Initiale Gewichte

Was wird dadurch bestimmt?

- Startposition des Bergsteigers
- Initiale Fehler welchen es zu optimieren gilt

Lernrate

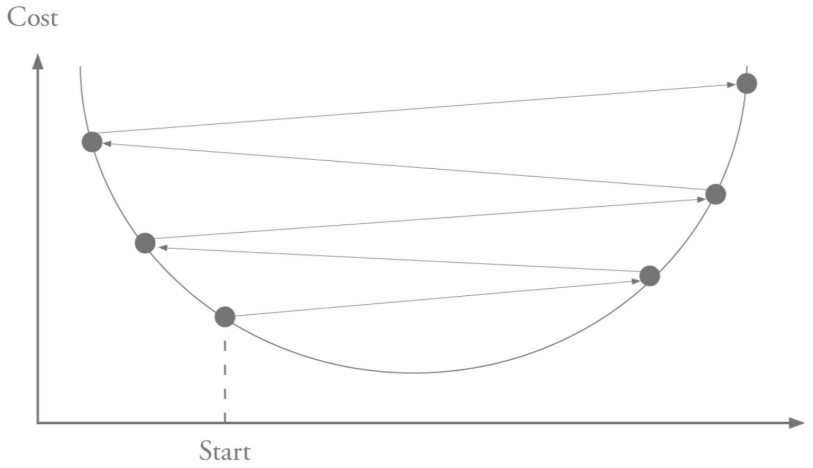
- Bestimmt wie “schnell” gelernt wird
- Die Größe der Schritte des Bergsteigers

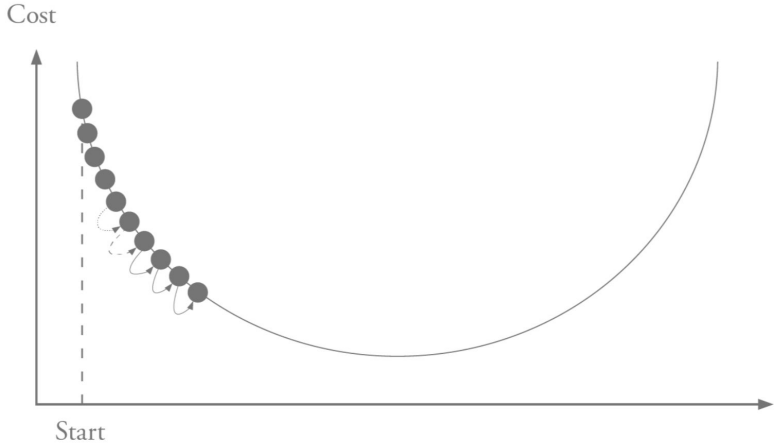


Lernrate

Was passiert wenn die Lernrate

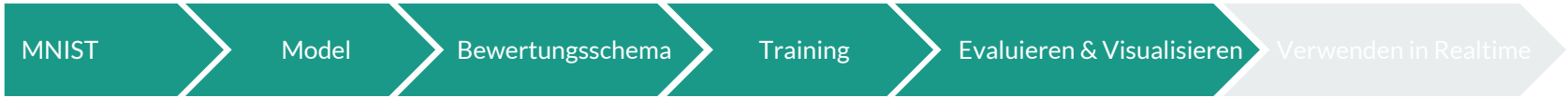
- zu groß ist?
- zu klein ist?





Coding time

MNIST Klassifizierung in TensorFlow



Evaluieren

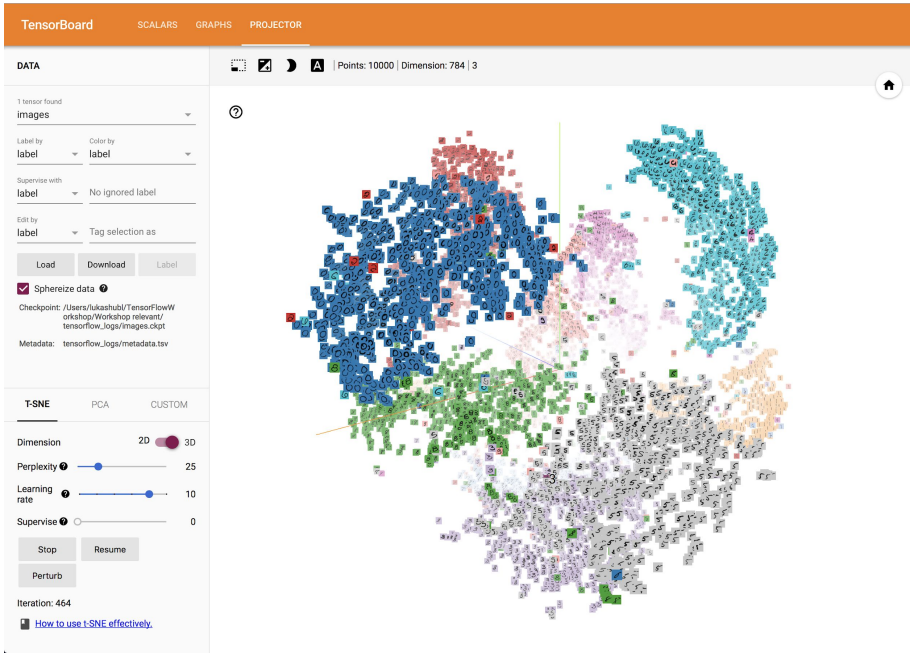
- Accuracy
- TensorBoard
- Code

Evaluieren

- Accuracy
- Wie viele Datensätze werden richtig klassifiziert

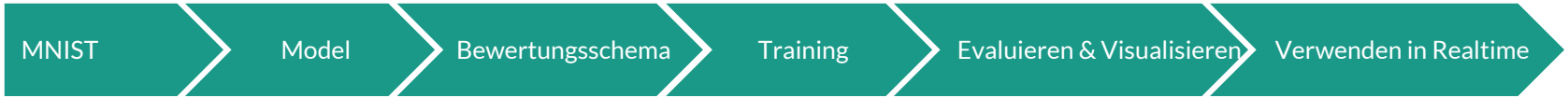
TensorBoard

- Visualisierungen aller Art
- Graph
- Skalare
- Projektor



Coding time

MNIST Klassifizierung in TensorFlow



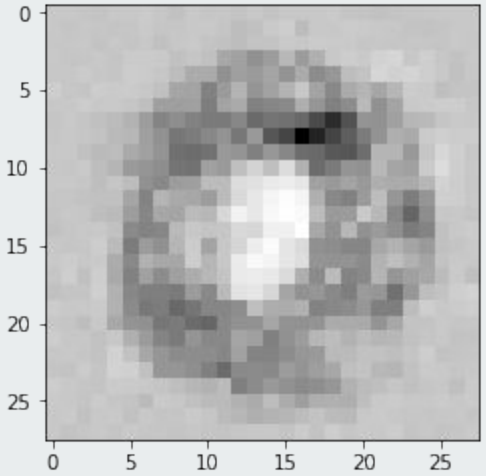
Verwenden des Models in Realtime-Anwendung

- Model speichern
- Model wiederverwenden
- Code

Coding time

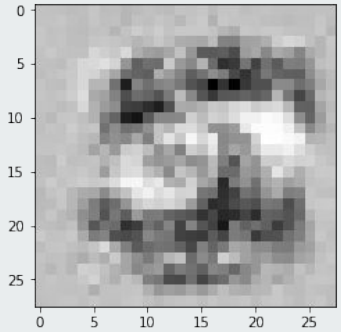
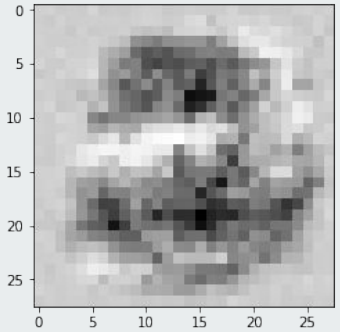
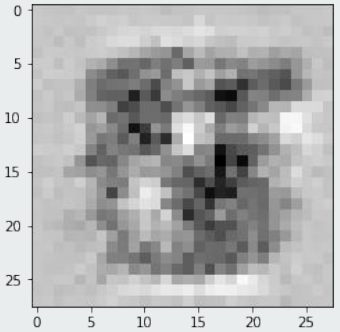
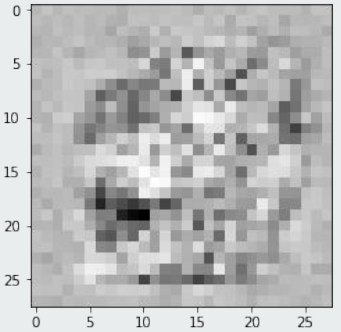
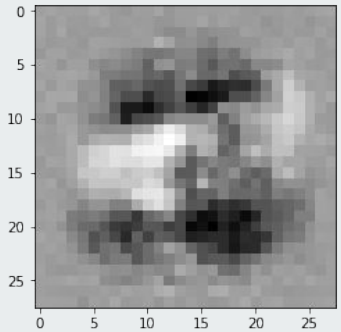
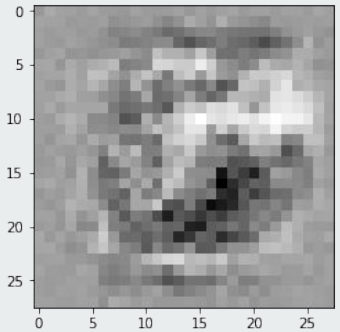
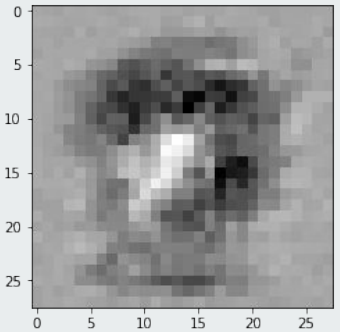
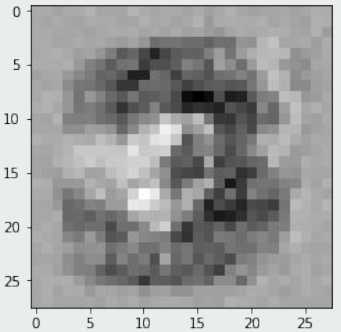
Vielen Dank für ihre Aufmerksamkeit

Fun Facts



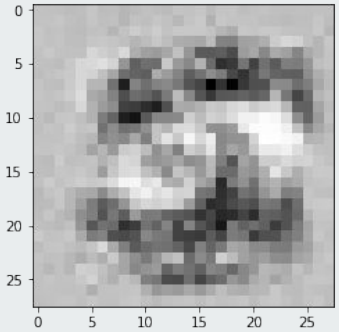
Was ist das?

Ratespiel

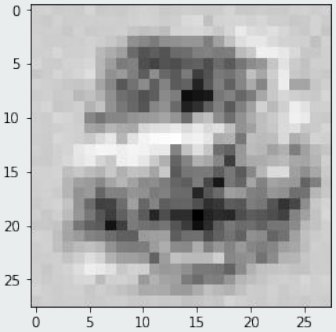









Ratespiel

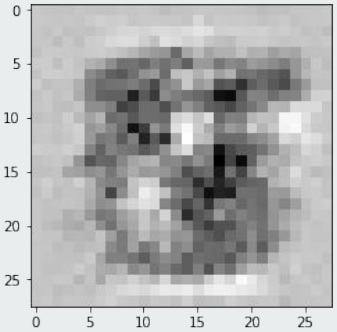
5



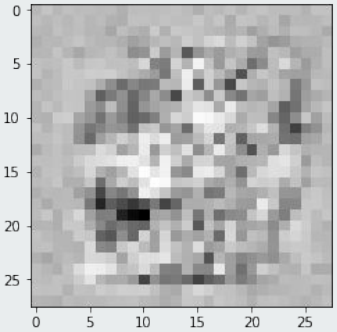
2



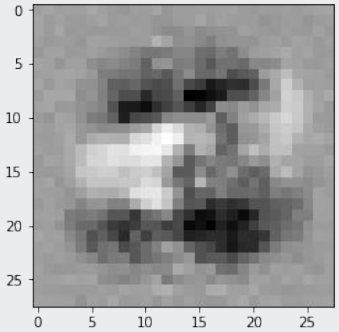
4



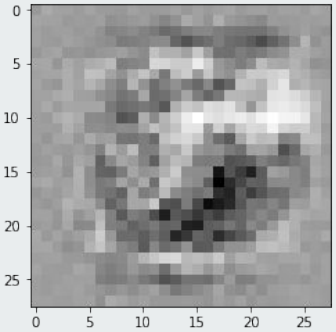
8



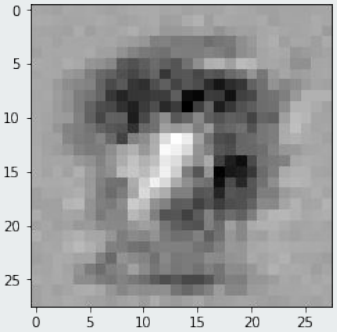
1



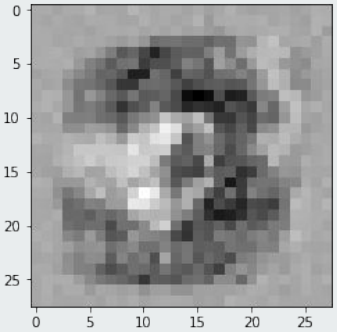
7



6



3



Wirklich vielen Dank für ihre Aufmerksamkeit