OPTIMIERUNGSMETHODEN DES GRADIENTEN ABSTIEGVERFAHRS BEI NEURONALEN NETZEN

GLIEDERUNG

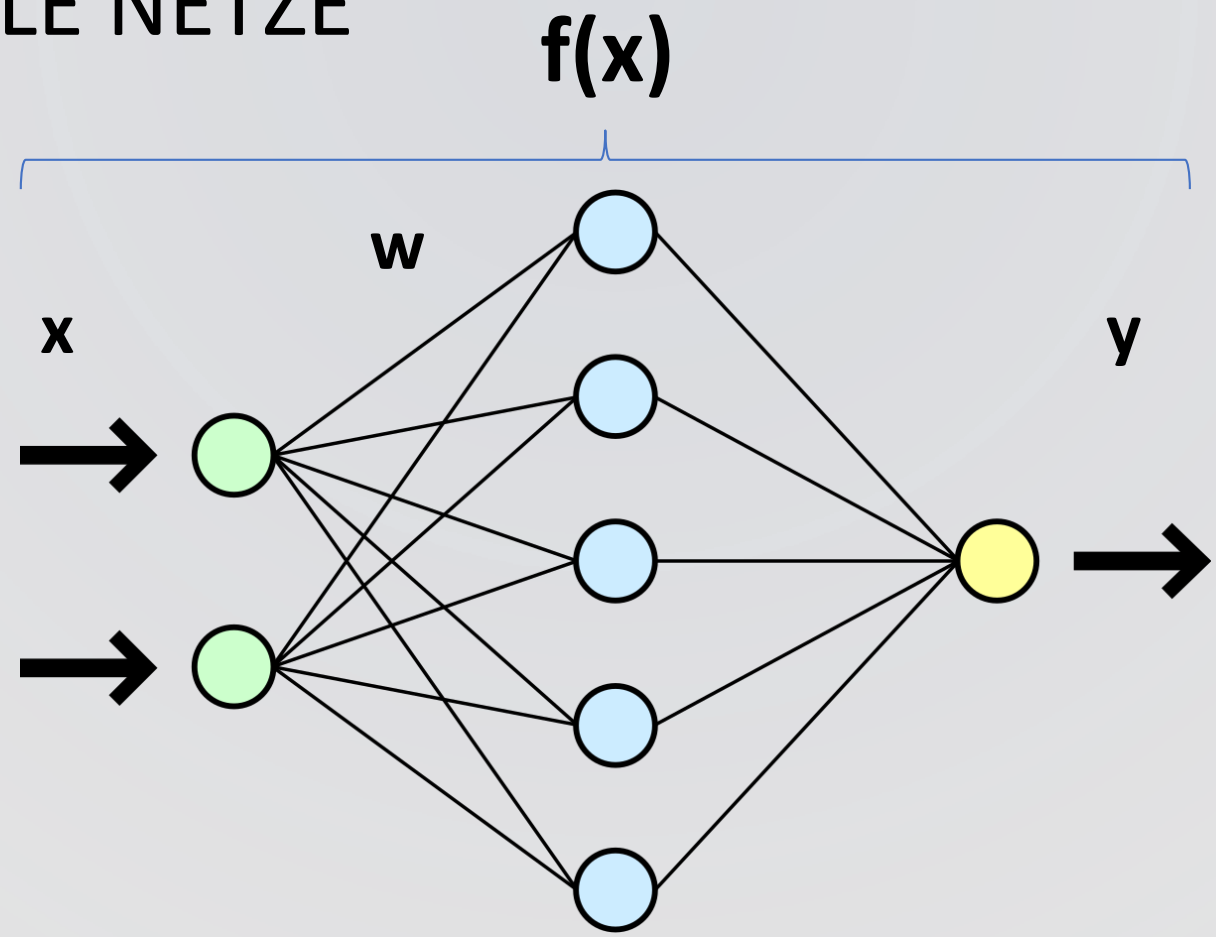
1. Motivation
2. Neuronale Netze
3. Gradient Descent
4. Optimierungsalgorithmen
5. Python Programm
6. Evaluation
7. Fazit

MOTIVATION

- Viele verschiedene Optimierungsalgorithmen
- Wahl des richtigen Optimierungsalgorithmus schwierig
- Evaluation Optimierungsalgorithmen

NEURONALE NETZE

NEURONALE NETZE



NEURONALE NETZE

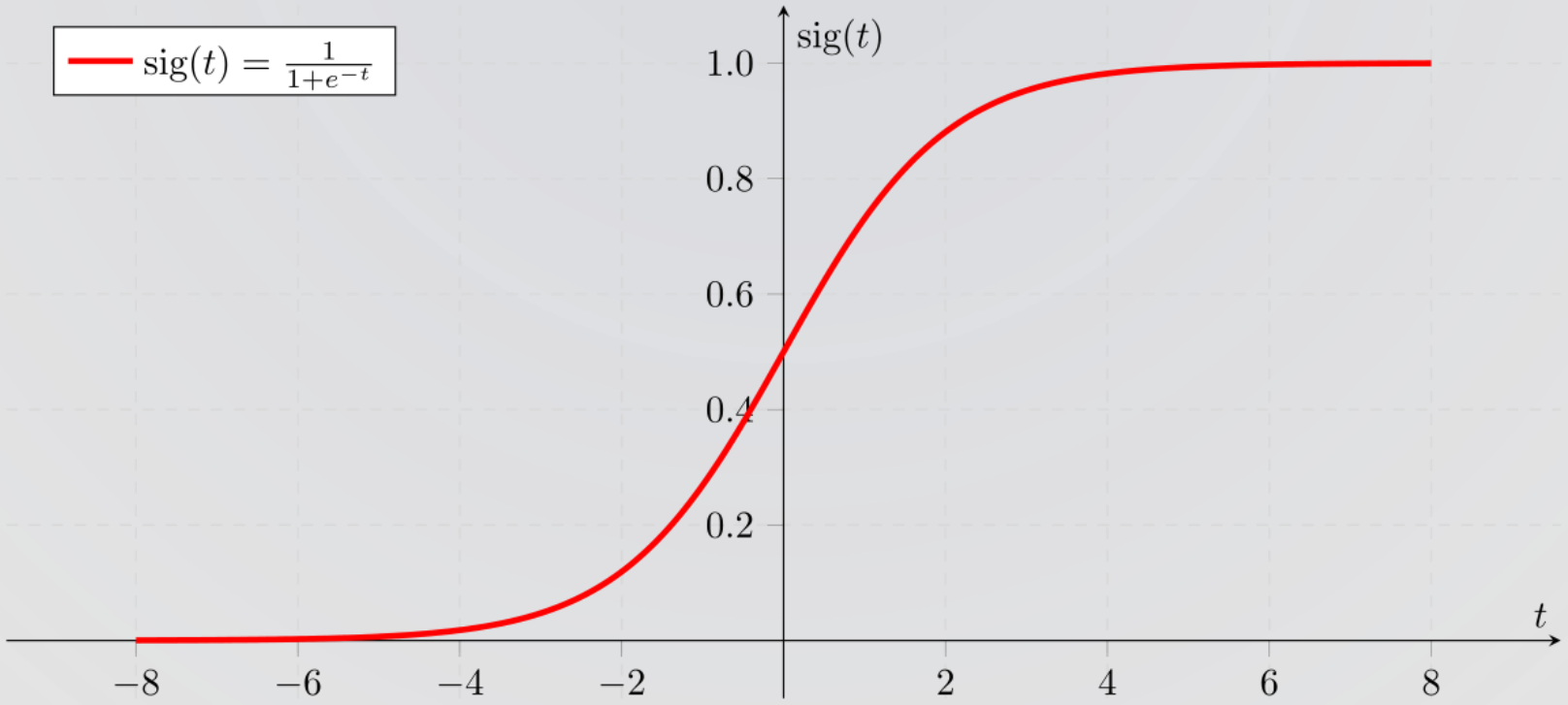
Definition 2.1 *[Bur97, Kapitel 1.2] Ein* ***(formales) Neuron*** *ist eine Funktion* $\kappa : \mathbb{R}^n \to \mathbb{R}^m$ *definiert durch:*

- *eine Aktivierungsfunktion* $T : \mathbb{R} \to \mathbb{R}$
- *ein gewichteter Vektor* $\vec{w} = \{w_1, w_2, ..., w_n\}$
- *und eine Schwelle* $\Theta \in \mathbb{R}$.

Der Vektor $\vec{x} = (x_1, x_2, ..., x_n) \in \mathbb{R}^n$ *wird auf den Vektor* $\vec{y} = (y, y, ..., y) \in \mathbb{R}^m$ *mit identischen Komponenten durch die folgende Rechenvorschrift abgebildet*

$$\kappa(\vec{x}) := (T(\sum_{i=1}^{n} w_i x_i - \Theta), ..., T(\sum_{i=1}^{n} w_i x_i - \Theta)) = \vec{y} \in \mathbb{R}^m \qquad (1)$$

NEURONALE NETZE



GRADIENT DESCENT

GRADIENT DESCENT

- Neuronales Netz ist Funktion f(x)
- Fehler des Netzes ist Funktion J(f(x))
- Optimale Parameter Belegung ergibt globales Minimum von J(f(x))
- Ableiten und null setzen geht nicht, da f(x) nichtlinear somit auch J(f(x))

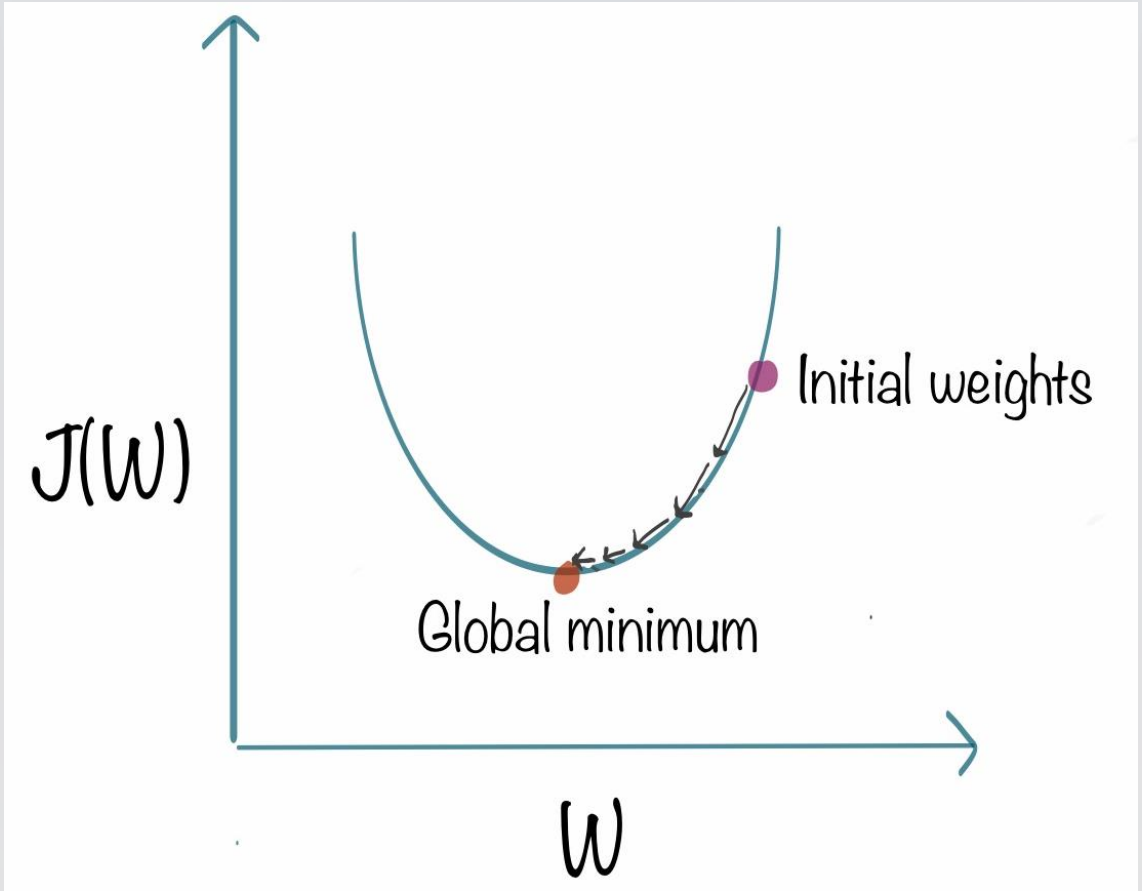
    -> **Lösung**: Gradient Descent Algorithmus

GRADIENT DESCENT

- Benötigen Gradienten
- Gradient ist Ableitung im mehrdimensionalen
- Gradient zeigt in die Richtung des stärksten Anstiegs
- Negativer Gradient zeigt in die Richtung des stärksten Abstieg

    -> **Idee**: Folge negativen Gradienten in ein Tal

GRADIENT DESCENT



GRADIENT DESCENT

Pseudoalgorithmus

```
for i in range(nb_epochs ):
    params_grad = evaluate_gradient(loss_function , data , params)
    params = params  - learning_rate * params_grad
```

GRADIENT DESCENT

Definition

Definition 2.4 *[Rud, Kapitel 2.1] Der sogenannte* ***batch gradient descent****, berechnet den Gradienten der Kostenfunktion für den gesamten Datensatz. Jedes Update der Parameter ist definiert durch*

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta) \tag{3}$$

GRADIENT DESCENT

Probleme:

- Gradient wird für alle Eingangsdaten berechnet -> langsam
- Alle Eingangsdaten passen nicht unbedingt in den Arbeispeicher
- Langsame Konvergenz
- Wahl der Lerngeschwindigkeit schwierig

  -> Optimierung

OPTIMIERUNGSMETHODEN

- Stochastic Gradient Descent
- Adagrad
- ADAM

OPTIMIERUNGSMETHODEN

Stochastic Gradient Descent:

Definition 2.5 *[Rud, Kapitel 2.2] Der Aktualisierungsschritt des* ***stochastic gradient descent*** *ist definiert durch*

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i)}; y^{(i)}) \tag{4}$$

OPTIMIERUNGSMETHODEN

Adagrad:

Definition 2.6 *[Rud, Kapitel 4.3] Der* ***Adagrad*** *Algorithmus ist definiert durch den Aktualisierungsschritt*

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i} \tag{6}$$

wobei $g_{t,i}$ *definiert ist durch*

$$g_{t,i} = \nabla_{\theta_t} J(\theta_{t,i}) \tag{7}$$

und $G_t \in R^{d \times d}$ *ist eine diagonal Matrix, wobei die Diagonalelemente* i, i*, die Summe der Quadrate der Gradienten zum zugehörigen Parameter* θ_i *bis zum Zeitpunkt* t *sind und* $\epsilon > 0$

OPTIMIERUNGSMETHODEN

ADAM:

Definition 2.7 *Der **ADAM** Algorithmus ist definiert durch den Aktualisierungsschritt*

$$\theta_{t+1} = \theta_t + \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\hat{m}_t \tag{8}$$

wobei

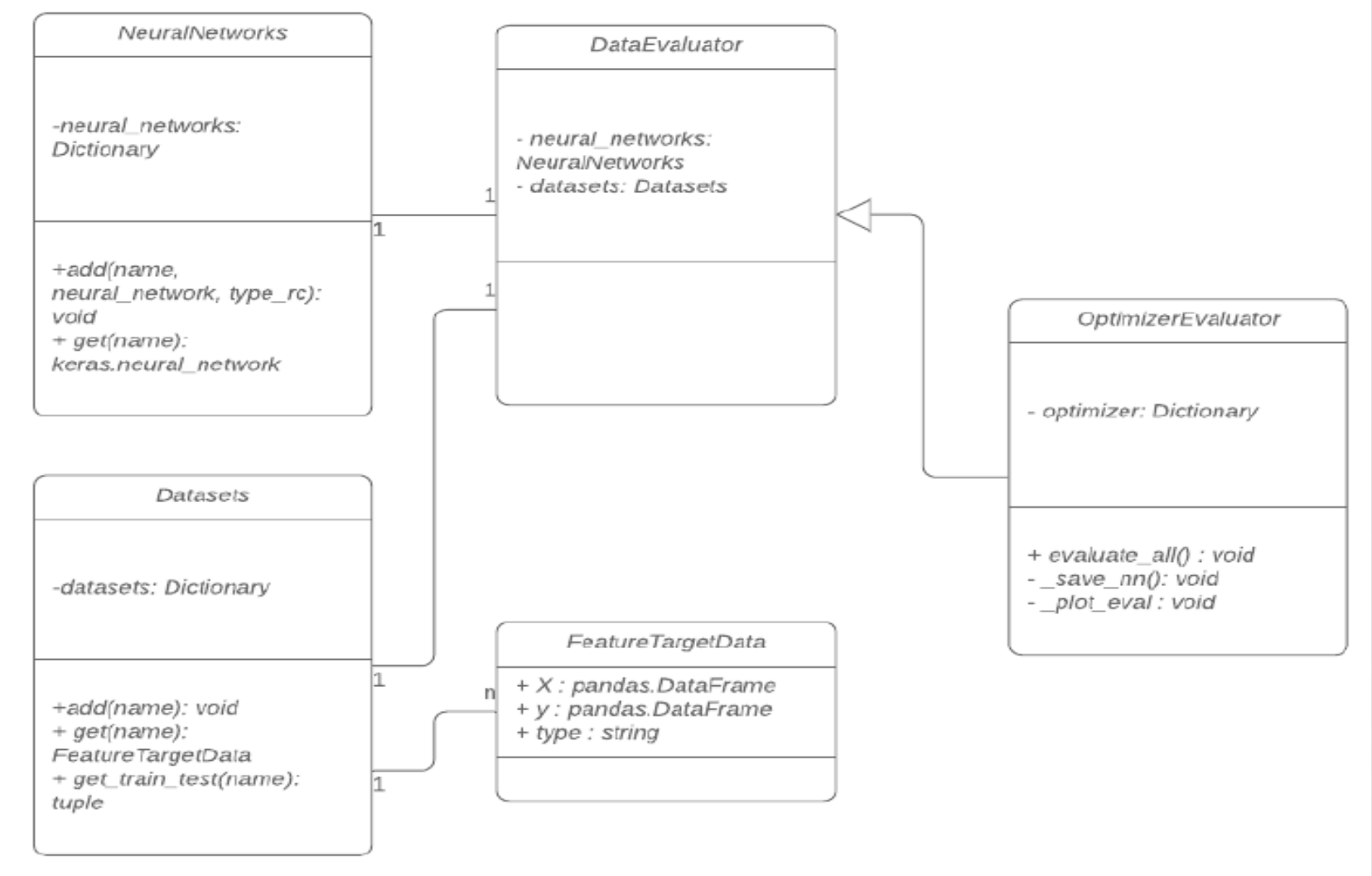
$$\hat{m}_t = \frac{\beta_1 m_{t-1} + (1-\beta_1)g_t}{1-\beta_1^t} \tag{9}$$

und

$$\hat{v}_t = \frac{\beta_2 v_{t-1} + (1-\beta_2)g_t^2}{1-\beta_1^t} \tag{10}$$

wobei $\hat{m}_t$ den Mittelwert des Gradienten über die Zeit beschreibt und $\hat{v}_t$ die Varianz. β_1 und β_2 sind dabei Zerfall Raten, die bestimmen, wieviel Einfluss die vergangenen Gradienten auf den neuen Wert nehmen.

PYTHON PROGRAMM


NeuralNetworks
-neural_networks: Dictionary
+add(name, neural_network, type_rc): void
+ get(name): keras.neural_network
DataEvaluator
- neural_networks: NeuralNetworks
- datasets: Datasets
OptimizerEvaluator
- optimizer: Dictionary
+ evaluate_all() : void
- _save_nn(): void
- _plot_eval : void
Datasets
-datasets: Dictionary
+add(name): void
+ get(name): FeatureTargetData
+ get_train_test(name): tuple
FeatureTargetData
+ X : pandas.DataFrame
+ y : pandas.DataFrame
+ type : string
1
1
1
1
1
n
1

PYTHON PROGRAM

Frameworks:

- Keras
- Scikit-learn
- Pandas
- Numpy

EVALUATION

EVALUATION

- Zwei Datensätze, ein Regressionsdatensatz, ein Klassifikationsdatensatz
- Metriken:
  - MSE - mittlere quadratische Abweichung
  - MAE - mittlere absolute Abweichung
  - Binäre Kreuzentropie
  - Accuracy

EVALUATION

Netzstruktur:

```
_regression_NN_boston = Sequential()
_regression_NN_boston.add(
    Dense(units=160, activation='relu', input_shape=(13,)))
_regression_NN_boston.add(Dense(units=64, activation='relu'))
_regression_NN_boston.add(Dense(units=1, activation='linear'))
```

EVALUATION

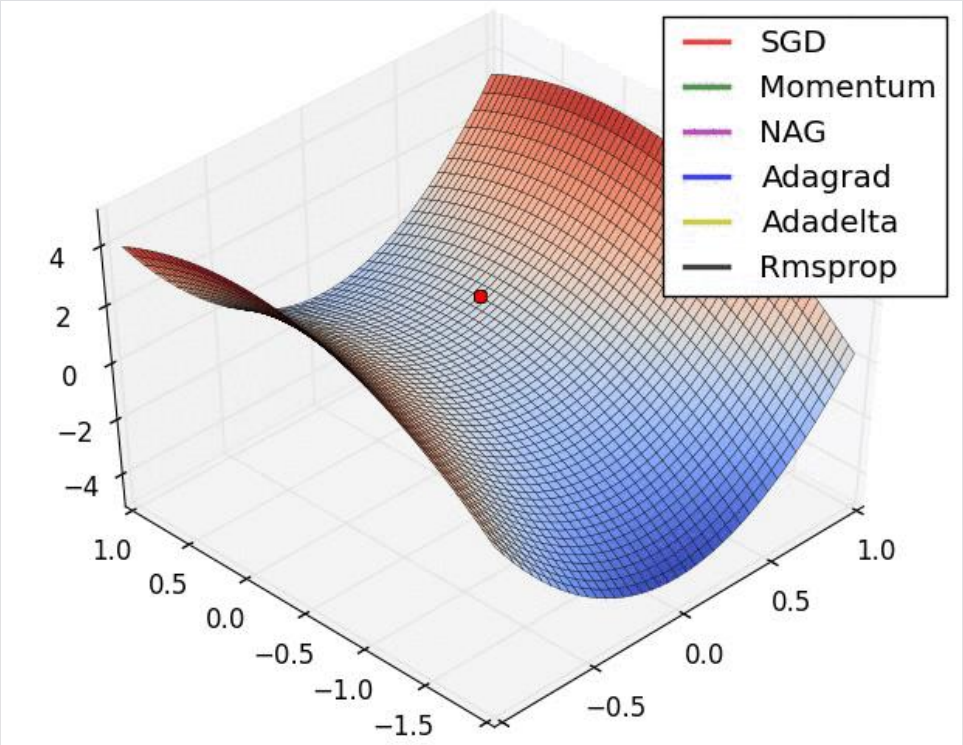
Erwartung:

- ADAM sollte am besten performen
- Adagrad ebenfalls gut
- Stochastic Gradient Descent eher schlecht

EVALUTATION

	Loss	Mae
adam	11.83	2.64
sgd		
adagrad		

EVALUATION


SGD
Momentum
NAG
Adagrad
Adadelta
Rmsprop
4
2
0
−2
−4
1.0
0.5
0.0
−0.5
−1.0
−1.5
1.0
0.5
0.0
−0.5

EVALUATION

	Loss	Accuracy
adam	0.12	0.96
sgd	17.2	0.5
adagrad	22.06	0.5

FAZIT