Multi-Layer-Perzeptron Rechner

Technisches Handbuch und Mathematische Grundlagen

Deutsch

Wolfgang Böhme

wolfgang.boehme@economic-modeling.de

www.economic-modeling.de

2. November 2026

Dokumentationen dieser Webseite wurden mit Unterstützung der KI Claude (Anthropic) entwickelt.

Inhaltsverzeichnis

1 Einleitung **5**
1.1 Überblick . . . 5
1.2 Anwendungsbereiche . . . 5

2 Mathematische Grundlagen **5**
2.1 Netzwerk-Architektur . . . 5
2.2 Neuron-Modell . . . 6
2.3 Matrix-Notation . . . 6

3 Aktivierungsfunktionen **6**
3.1 Sigmoid-Funktion . . . 6
3.2 ReLU (Rectified Linear Unit) . . . 7
3.3 Tanh-Funktion . . . 7
3.4 Lineare Funktion . . . 7

4 Forward Propagation **8**
4.1 Algorithmus . . . 8
4.2 Beispiel-Berechnung . . . 8

5 Loss-Funktionen **8**
5.1 Mean Squared Error (MSE) . . . 8
5.2 Binary Cross-Entropy . . . 9

6 Backpropagation **9**
6.1 Grundprinzip . . . 9
6.2 Output-Layer Gradient . . . 9
6.3 Hidden Layer Gradienten . . . 10
6.4 Gewichts- und Bias-Gradienten . . . 10
6.5 Backpropagation-Algorithmus . . . 10

7 Gradient Descent **11**
7.1 Standard Gradient Descent . . . 11
7.2 Batch Gradient Descent . . . 11
7.3 Mini-Batch Gradient Descent . . . 11

8 Training-Algorithmus **12**
8.1 Vollständiger Training-Loop . . . 12
8.2 Umsetzung im Rechner v20 . . . 12

9 Datennormalisierung **12**
9.1 Motivation . . . 12
9.2 Min-Max-Normalisierung . . . 13
9.3 Denormalisierung . . . 13
9.4 Normalisierungs-Parameter . . . 13
9.5 Parameter nur aus den Trainingsdaten . . . 13

10 Gewichtsinitialisierung **13**
10.1 Zufällige Initialisierung . . . 13
10.2 Xavier/Glorot Initialisierung (empfohlen) . . . 14
10.3 Umsetzung im Rechner v20 . . . 14

11 Hyperparameter **14**
11.1 Learning Rate η . . . 14
11.2 Anzahl Epochs . . . 14
11.3 Batch Size B . . . 15
11.4 Netzwerk-Architektur . . . 15

12 Datenaufteilung und Backtesting **15**
12.1 Aufteilung in Training und Test . . . 15
12.2 Vortageswert als Input (Lag 1) . . . 15
12.3 Kennzahlen . . . 16
12.4 Vergleichsmaßstäbe . . . 16
12.5 Extrapolationsprüfung . . . 16

13 Verwendung des Rechners **16**
13.1 Workflow . . . 16
13.2 Beispiel: Kreuzkurse (UZS) . . . 17

14 Troubleshooting **17**
14.1 Training konvergiert nicht . . . 17
14.2 Overfitting . . . 18
14.3 Underfitting . . . 18
14.4 Numerische Instabilität . . . 18

15 Mathematische Beweise **18**
15.1 Universelle Approximations-Theorem . . . 18
15.2 Gradient Descent Konvergenz . . . 19

16 Implementierungs-Details **19**
16.1 Matrix-Operationen . . . 19
16.2 Komplexitätsanalyse . . . 19

17 Visualisierungen **19**
17.1 Netzwerk-Graph . . . 19
17.2 Aktive Neuronen . . . 20
17.3 Loss-Kurve . . . 20
17.4 Gewichtsverteilung . . . 20

18 Export-Formate **20**

19 Erweiterungsmöglichkeiten **21**
19.1 Zukünftige Features . . . 21
19.2 Dropout . . . 21
19.3 Momentum . . . 21
19.4 Adam Optimizer . . . 21

20 Referenzen und Weiterführende Literatur **22**
20.1 Grundlegende Werke 22
20.2 Lehrbücher 22

21 Anhang **22**
21.1 Formelsammlung 22
21.2 Glossar 23
21.3 Notation 23

22 Schlusswort **24**

Zusammenfassung

Dieses technische Handbuch beschreibt die mathematischen Grundlagen, Algorithmen und Implementierungsdetails des Multi-Layer-Perzeptron (MLP) Rechners. Das Dokument richtet sich an Anwender, die ein tieferes Verständnis der zugrundeliegenden neuronalen Netzwerk-Theorie erlangen möchten, sowie an Entwickler, die die Implementierung nachvollziehen oder erweitern möchten. Es werden alle verwendeten Formeln, der Backpropagation-Algorithmus, Aktivierungsfunktionen, Loss-Funktionen und Optimierungsverfahren detailliert erläutert.

1 Einleitung

1.1 Überblick

Das Multi-Layer-Perzeptron (MLP) ist ein fundamentales künstliches neuronales Netzwerk, das aus mehreren Schichten von Neuronen besteht. Der vorliegende Rechner implementiert ein vollständiges MLP mit folgenden Eigenschaften:

- Konfigurierbare Anzahl von Input-Neuronen (1-10)
- Variable Anzahl von Hidden Layers (2-5)
- Flexible Anzahl von Neuronen pro Hidden Layer (1-20)
- Ein Output-Neuron für Regression oder binäre Klassifikation
- Verschiedene Aktivierungsfunktionen pro Layer
- Backpropagation-basiertes Training
- Echtzeit-Visualisierung des Trainingsfortschritts
- Reproduzierbare Ergebnisse über einen Startwert (Seed)
- Train/Test-Aufteilung (chronologisch oder gleichmäßig) mit automatischem Backtesting und Vergleichsmaßstäben
- Eingabe und Ausgabe stets in Originalwerten; Normalisierung intern
- Prüfung auf Extrapolation außerhalb des Trainingsbereichs

1.2 Anwendungsbereiche

MLPs eignen sich für eine Vielzahl von Aufgaben:

- **Regression:** Vorhersage kontinuierlicher Werte (z.B. Währungskurse, Preise)
- **Binäre Klassifikation:** Zuordnung zu zwei Klassen (z.B. Spam/kein Spam)
- **Funktionsapproximation:** Lernen komplexer nichtlinearer Zusammenhänge
- **Mustererkennung:** Identifikation von Mustern in Daten

2 Mathematische Grundlagen

2.1 Netzwerk-Architektur

Definition 2.1 (Multi-Layer-Perzeptron)**.** Ein Multi-Layer-Perzeptron ist ein gerichteter azyklischer Graph bestehend aus L Schichten (Layers), wobei gilt:

- Layer 0: Input-Layer mit n_0 Neuronen
- Layer l für $1 \leq l < L-1$: Hidden Layers mit n_l Neuronen
- Layer $L-1$: Output-Layer mit n_{L-1} Neuronen

Die Netzwerk-Architektur wird beschrieben durch den Vektor:

$$\mathcal{N} = (n_0, n_1, n_2, \ldots, n_{L-1}) \tag{1}$$

Beispiel: $(7, 16, 8, 1)$ beschreibt ein Netzwerk mit 7 Input-Neuronen, zwei Hidden Layers mit 16 bzw. 8 Neuronen und einem Output-Neuron.

2.2 Neuron-Modell

Definition 2.2 (Künstliches Neuron)**.** Ein künstliches Neuron j in Layer l berechnet seine Ausgabe in zwei Schritten:

1. Gewichtete Summe (Pre-Activation):

$$z_j^{(l)} = \sum_{i=1}^{n_{l-1}} w_{ji}^{(l)} \cdot a_i^{(l-1)} + b_j^{(l)} \tag{2}$$

2. Aktivierung (Post-Activation):

$$a_j^{(l)} = f^{(l)}(z_j^{(l)}) \tag{3}$$

wobei:

- $w_{ji}^{(l)}$ das Gewicht von Neuron i in Layer $l-1$ zu Neuron j in Layer l ist
- $b_j^{(l)}$ der Bias von Neuron j in Layer l ist
- $a_i^{(l-1)}$ die Aktivierung von Neuron i im vorherigen Layer ist
- $f^{(l)}$ die Aktivierungsfunktion in Layer l ist

2.3 Matrix-Notation

Für effiziente Berechnung verwenden wir Matrix-Notation. Für Layer l:

$$\mathbf{z}^{(l)} = \mathbf{W}^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)} \tag{4}$$

$$\mathbf{a}^{(l)} = f^{(l)}(\mathbf{z}^{(l)}) \tag{5}$$

wobei:

- $\mathbf{W}^{(l)} \in \mathbb{R}^{n_l \times n_{l-1}}$ die Gewichtsmatrix ist
- $\mathbf{b}^{(l)} \in \mathbb{R}^{n_l}$ der Bias-Vektor ist
- $\mathbf{a}^{(l)} \in \mathbb{R}^{n_l}$ der Aktivierungsvektor ist
- $f^{(l)}$ elementweise auf den Vektor angewendet wird

3 Aktivierungsfunktionen

Aktivierungsfunktionen ermöglichen es dem Netzwerk, nichtlineare Zusammenhänge zu lernen.

3.1 Sigmoid-Funktion

Die Sigmoid-Funktion bildet Werte auf das Intervall $(0, 1)$ ab:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{6}$$

Ableitung:

$$\sigma'(z) = \sigma(z) \cdot (1 - \sigma(z)) \tag{7}$$

Eigenschaften:

- Ausgabe zwischen 0 und 1
- Glatt und differenzierbar
- Interpretierbar als Wahrscheinlichkeit
- Problem: Vanishing Gradients bei $|z| \gg 0$

3.2 ReLU (Rectified Linear Unit)

$$\text{ReLU}(z) = \max(0, z) = \begin{cases} z & \text{wenn } z > 0 \\ 0 & \text{wenn } z \leq 0 \end{cases} \tag{8}$$

Ableitung:

$$\text{ReLU}'(z) = \begin{cases} 1 & \text{wenn } z > 0 \\ 0 & \text{wenn } z \leq 0 \end{cases} \tag{9}$$

Eigenschaften:

- Sehr effizient zu berechnen
- Kein Vanishing Gradient für $z > 0$
- Standard für tiefe Netzwerke
- Problem: "Dying ReLU"wenn $z \leq 0$

3.3 Tanh-Funktion

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} = \frac{e^{2z} - 1}{e^{2z} + 1} \tag{10}$$

Ableitung:

$$\tanh'(z) = 1 - \tanh^2(z) \tag{11}$$

Eigenschaften:

- Ausgabe zwischen -1 und 1
- Zentriert um 0
- Stärkere Gradienten als Sigmoid
- Ähnliches Vanishing-Gradient-Problem wie Sigmoid

3.4 Lineare Funktion

$$\text{linear}(z) = z \tag{12}$$

Ableitung:

$$\text{linear}'(z) = 1 \tag{13}$$

Verwendung:

- Typisch für Output-Layer bei Regression
- Keine Begrenzung des Wertebereichs

4 Forward Propagation

4.1 Algorithmus

Der Forward Pass berechnet die Netzwerk-Ausgabe für einen gegebenen Input $\mathbf{x}$.

Algorithm 1 Forward Propagation

Require: Input-Vektor $\mathbf{x} \in \mathbb{R}^{n_0}$, Gewichte $\{\mathbf{W}^{(l)}\}_{l=1}^{L-1}$, Biases $\{\mathbf{b}^{(l)}\}_{l=1}^{L-1}$
Ensure: Output $\mathbf{a}^{(L-1)}$

$\mathbf{a}^{(0)} \leftarrow \mathbf{x}$ ▷ Input-Layer
for $l = 1$ to $L-1$ **do** ▷ Für jeden Layer
  $\mathbf{z}^{(l)} \leftarrow \mathbf{W}^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}$ ▷ Gewichtete Summe
  $\mathbf{a}^{(l)} \leftarrow f^{(l)}(\mathbf{z}^{(l)})$ ▷ Aktivierung
end for
return $\mathbf{a}^{(L-1)}$ ▷ Output

4.2 Beispiel-Berechnung

Gegeben sei ein Netzwerk mit Architektur $(2, 3, 1)$ und Input $\mathbf{x} = (0.5, 0.8)^T$.

Layer 1 (Input → Hidden):

$$\mathbf{W}^{(1)} = \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{pmatrix}, \quad \mathbf{b}^{(1)} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} \tag{14}$$

$$\mathbf{z}^{(1)} = \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)} = \begin{pmatrix} w_{11} \cdot 0.5 + w_{12} \cdot 0.8 + b_1 \\ w_{21} \cdot 0.5 + w_{22} \cdot 0.8 + b_2 \\ w_{31} \cdot 0.5 + w_{32} \cdot 0.8 + b_3 \end{pmatrix} \tag{15}$$

$$\mathbf{a}^{(1)} = \mathrm{ReLU}(\mathbf{z}^{(1)}) \tag{16}$$

Layer 2 (Hidden → Output):

$$z^{(2)} = w_1^{(2)} a_1^{(1)} + w_2^{(2)} a_2^{(1)} + w_3^{(2)} a_3^{(1)} + b^{(2)} \tag{17}$$

$$a^{(2)} = \sigma(z^{(2)}) \tag{18}$$

5 Loss-Funktionen

Die Loss-Funktion $\mathcal{L}$ misst die Diskrepanz zwischen Vorhersage $\hat{y}$ und wahrem Wert y.

5.1 Mean Squared Error (MSE)

Für Regressionsprobleme:

$$\mathcal{L}_{\mathrm{MSE}}(y, \hat{y}) = \frac{1}{2}(y - \hat{y})^2 \tag{19}$$

Ableitung:

$$\frac{\partial \mathcal{L}_{\mathrm{MSE}}}{\partial \hat{y}} = -(y - \hat{y}) = \hat{y} - y \tag{20}$$

Für einen Datensatz mit N Samples:

$$\mathcal{L}_{\text{MSE}} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{21}$$

5.2 Binary Cross-Entropy

Für binäre Klassifikation:

$$\mathcal{L}_{\text{BCE}}(y, \hat{y}) = -[y \log(\hat{y}) + (1-y)\log(1-\hat{y})] \tag{22}$$

Ableitung:

$$\frac{\partial \mathcal{L}_{\text{BCE}}}{\partial \hat{y}} = -\frac{y}{\hat{y}} + \frac{1-y}{1-\hat{y}} = \frac{\hat{y}-y}{\hat{y}(1-\hat{y})} \tag{23}$$

Für einen Datensatz:

$$\mathcal{L}_{\text{BCE}} = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)] \tag{24}$$

6 Backpropagation

Backpropagation ist der Algorithmus zum Berechnen der Gradienten der Loss-Funktion bezüglich aller Netzwerk-Parameter.

6.1 Grundprinzip

Wir wenden die Kettenregel an, um Gradienten von der Output-Schicht rückwärts durch das Netzwerk zu propagieren.

Theorem 6.1 (Kettenregel für neuronale Netzwerke)**.** Für die Ableitung der Loss-Funktion bezüglich der Gewichte gilt:

$$\frac{\partial \mathcal{L}}{\partial w_{ji}^{(l)}} = \frac{\partial \mathcal{L}}{\partial z_j^{(l)}} \cdot \frac{\partial z_j^{(l)}}{\partial w_{ji}^{(l)}} \tag{25}$$

6.2 Output-Layer Gradient

Für den Output-Layer $L-1$ mit einem Neuron:

$$\delta^{(L-1)} = \frac{\partial \mathcal{L}}{\partial z^{(L-1)}} = \frac{\partial \mathcal{L}}{\partial a^{(L-1)}} \cdot \frac{\partial a^{(L-1)}}{\partial z^{(L-1)}} \tag{26}$$

Bei MSE-Loss und Sigmoid-Aktivierung:

$$\delta^{(L-1)} = (a^{(L-1)} - y) \cdot \sigma'(z^{(L-1)}) = (a^{(L-1)} - y) \cdot a^{(L-1)} \cdot (1 - a^{(L-1)}) \tag{27}$$

6.3 Hidden Layer Gradienten

Für Hidden Layers $l < L - 1$:

$$\delta_j^{(l)} = \left(\sum_{k=1}^{n_{l+1}} w_{kj}^{(l+1)} \delta_k^{(l+1)} \right) \cdot f'^{(l)}(z_j^{(l)}) \tag{28}$$

In Matrix-Notation:

$$\boldsymbol{\delta}^{(l)} = \left[(\mathbf{W}^{(l+1)})^T \boldsymbol{\delta}^{(l+1)} \right] \odot f'^{(l)}(\mathbf{z}^{(l)}) \tag{29}$$

wobei $\odot$ das Hadamard-Produkt (elementweise Multiplikation) ist.

6.4 Gewichts- und Bias-Gradienten

Mit den Deltas können wir die Gradienten berechnen:

Gewichte:

$$\frac{\partial \mathcal{L}}{\partial w_{ji}^{(l)}} = \delta_j^{(l)} \cdot a_i^{(l-1)} \tag{30}$$

Biases:

$$\frac{\partial \mathcal{L}}{\partial b_j^{(l)}} = \delta_j^{(l)} \tag{31}$$

In Matrix-Form:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(l)}} = \boldsymbol{\delta}^{(l)} (\mathbf{a}^{(l-1)})^T \tag{32}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{b}^{(l)}} = \boldsymbol{\delta}^{(l)} \tag{33}$$

6.5 Backpropagation-Algorithmus

Algorithm 2 Backpropagation

Require: Input $\mathbf{x}$, Target y, Aktivierungen $\{\mathbf{a}^{(l)}\}$, Pre-Activations $\{\mathbf{z}^{(l)}\}$
Ensure: Gradienten $\{\nabla_{\mathbf{W}^{(l)}} \mathcal{L}\}$, $\{\nabla_{\mathbf{b}^{(l)}} \mathcal{L}\}$

// **Output Layer**
$\boldsymbol{\delta}^{(L-1)} \leftarrow \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{(L-1)}} \odot f'^{(L-1)}(\mathbf{z}^{(L-1)})$

// **Hidden Layers (rückwärts)**
for $l = L - 2$ down to 1 **do**
    $\boldsymbol{\delta}^{(l)} \leftarrow [(\mathbf{W}^{(l+1)})^T \boldsymbol{\delta}^{(l+1)}] \odot f'^{(l)}(\mathbf{z}^{(l)})$
end for

// **Gradienten berechnen**
for $l = 1$ to $L - 1$ **do**
    $\nabla_{\mathbf{W}^{(l)}} \mathcal{L} \leftarrow \boldsymbol{\delta}^{(l)} (\mathbf{a}^{(l-1)})^T$
    $\nabla_{\mathbf{b}^{(l)}} \mathcal{L} \leftarrow \boldsymbol{\delta}^{(l)}$
end for
return Gradienten

7 Gradient Descent

7.1 Standard Gradient Descent

Parameter-Update mit Learning Rate η:

$$\mathbf{W}^{(l)} \leftarrow \mathbf{W}^{(l)} - \eta \cdot \nabla_{\mathbf{W}^{(l)}} \mathcal{L} \tag{34}$$

$$\mathbf{b}^{(l)} \leftarrow \mathbf{b}^{(l)} - \eta \cdot \nabla_{\mathbf{b}^{(l)}} \mathcal{L} \tag{35}$$

7.2 Batch Gradient Descent

Für einen Datensatz $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$:

$$\nabla_{\mathbf{W}^{(l)}} \mathcal{L}_{\text{total}} = \frac{1}{N} \sum_{i=1}^{N} \nabla_{\mathbf{W}^{(l)}} \mathcal{L}(\mathbf{x}_i, y_i) \tag{36}$$

7.3 Mini-Batch Gradient Descent

Teile Datensatz in Batches der Größe B:

$$\nabla_{\mathbf{W}^{(l)}} \mathcal{L}_{\text{batch}} = \frac{1}{B} \sum_{i \in \text{Batch}} \nabla_{\mathbf{W}^{(l)}} \mathcal{L}(\mathbf{x}_i, y_i) \tag{37}$$

Vorteile:

- Schnellere Konvergenz als Batch GD
- Geringerer Speicherbedarf als Batch GD
- Regularisierungseffekt durch Rauschen

8 Training-Algorithmus

8.1 Vollständiger Training-Loop

Algorithm 3 Neural Network Training

Require: Datensatz $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$, Netzwerk-Parameter, Learning Rate η, Epochs E, Batch Size B
Ensure: Trainierte Parameter

for epoch $= 1$ to E **do**
  Shuffle $\mathcal{D}$ ▷ Datensatz mischen
  $\mathcal{L}_{\text{epoch}} \leftarrow 0$
  for each batch in $\mathcal{D}$ **do**
    $\nabla_{\mathbf{W}}\mathcal{L} \leftarrow 0$, $\nabla_{\mathbf{b}}\mathcal{L} \leftarrow 0$ ▷ Gradienten initialisieren
    for each $(\mathbf{x}, y)$ in batch **do**
      $\hat{y} \leftarrow \text{ForwardProp}(\mathbf{x})$ ▷ Forward Pass
      $\mathcal{L}_{\text{epoch}} \leftarrow \mathcal{L}_{\text{epoch}} + \mathcal{L}(y, \hat{y})$
      $\nabla_{\mathbf{W}}\mathcal{L} \leftarrow \nabla_{\mathbf{W}}\mathcal{L} + \text{BackProp}(\mathbf{x}, y)$ ▷ Gradienten akkumulieren
    end for
    $\nabla_{\mathbf{W}}\mathcal{L} \leftarrow \nabla_{\mathbf{W}}\mathcal{L}/B$ ▷ Mittelwert
    $\mathbf{W} \leftarrow \mathbf{W} - \eta \cdot \nabla_{\mathbf{W}}\mathcal{L}$ ▷ Update
    $\mathbf{b} \leftarrow \mathbf{b} - \eta \cdot \nabla_{\mathbf{b}}\mathcal{L}$ ▷ Update
  end for
  $\mathcal{L}_{\text{epoch}} \leftarrow \mathcal{L}_{\text{epoch}}/N$
  Log Fortschritt und Loss
end for

8.2 Umsetzung im Rechner v20

Der Rechner verwendet *stochastischen* Gradientenabstieg: Die Gewichte werden nach jedem einzelnen Sample aktualisiert. Die Batch Size gliedert nur die Schleife und beeinflusst das Ergebnis nicht. Für jedes Sample gilt

$$\boldsymbol{\delta}^{(L-1)} = (\mathbf{y} - \hat{\mathbf{y}}) \odot f'^{(L-1)}, \qquad \mathbf{W}^{(l)} \leftarrow \mathbf{W}^{(l)} + \eta\, \boldsymbol{\delta}^{(l)} (\mathbf{a}^{(l-1)})^T, \qquad \mathbf{b}^{(l)} \leftarrow \mathbf{b}^{(l)} + \eta\, \boldsymbol{\delta}^{(l)}. \quad (38)$$

Der Fehler wird mit den bereits aktualisierten Gewichten an die vorherige Schicht weitergegeben. Die Ableitung f' wird aus der Aktivierung a berechnet (z. B. $a(1-a)$ bei Sigmoid). Der Output-Gradient hat für MSE und BCE dieselbe Form; die gewählte Loss-Funktion bestimmt den angezeigten Loss (BCE mit Begrenzung von $\hat{y}$ auf $[10^{-4}, 1 - 10^{-4}]$). Der Epochen-Loss ist der Mittelwert über alle Trainingssamples in normierter Skala.

Mischen: Ist die Option „Reihenfolge der Trainingsdaten je Epoche mischen“ aktiv, wird die Reihenfolge zu Beginn jeder Epoche mit dem Fisher-Yates-Verfahren permutiert. Der Zufallsgenerator dafür wird mit Startwert $+7919$ initialisiert. Ohne Mischen sieht das Netz bei chronologischen Daten immer zuletzt die jüngsten Zeilen.

9 Datennormalisierung

9.1 Motivation

Bei Daten mit sehr unterschiedlichen Größenordnungen können Probleme auftreten:

- Numerische Instabilität (Overflow/Underflow)

- Langsame Konvergenz
- Verschwindende oder explodierende Gradienten

9.2 Min-Max-Normalisierung

Für Feature j mit Werten $x_j^{(1)}, \ldots, x_j^{(N)}$:

$$x_j^{\text{norm}} = \frac{x_j - \min(x_j)}{\max(x_j) - \min(x_j)} \tag{39}$$

Eigenschaften:

- Skaliert auf Intervall $[0, 1]$
- Erhält relative Abstände
- Alle Features auf gleicher Skala

9.3 Denormalisierung

Für Vorhersagen müssen wir zurück transformieren:

$$x_j^{\text{orig}} = x_j^{\text{norm}} \cdot (\max(x_j) - \min(x_j)) + \min(x_j) \tag{40}$$

9.4 Normalisierungs-Parameter

Wir speichern für jedes Feature:

$$\text{params}_j = \begin{cases} \min_j = \min(x_j^{(1)}, \ldots, x_j^{(N)}) \\ \max_j = \max(x_j^{(1)}, \ldots, x_j^{(N)}) \\ \text{range}_j = \max_j - \min_j \end{cases} \tag{41}$$

9.5 Parameter nur aus den Trainingsdaten

Minimum und Maximum je Input und für das Ziel werden *ausschließlich aus den Trainingsdaten* bestimmt und unverändert auf Testdaten, manuelle Eingaben und Batch-Dateien angewendet. Testwerte können deshalb normiert außerhalb von $[0, 1]$ liegen – das ist gewollt und entspricht einer echten Prognose, bei der die Zukunft beim Training unbekannt ist. Die Ausgabe des Netzes wird mit den Parametern des Ziels denormiert; alle Kennzahlen und Anzeigen sind in Originalwerten.

10 Gewichtsinitialisierung

10.1 Zufällige Initialisierung

Die Gewichte werden zufällig initialisiert, um Symmetrie zu brechen:

$$w_{ji}^{(l)} \sim \mathcal{U}(-1, 1) \tag{42}$$

wobei $\mathcal{U}(-1, 1)$ die Gleichverteilung auf $[-1, 1]$ ist.

10.2 Xavier/Glorot Initialisierung (empfohlen)

Für bessere Konvergenz:

$$w_{ji}^{(l)} \sim \mathcal{N}\left(0, \frac{2}{n_{l-1} + n_l}\right) \tag{43}$$

oder uniform:

$$w_{ji}^{(l)} \sim \mathcal{U}\left(-\sqrt{\frac{6}{n_{l-1} + n_l}}, \sqrt{\frac{6}{n_{l-1} + n_l}}\right) \tag{44}$$

10.3 Umsetzung im Rechner v20

Gewichte und Biases werden gleichverteilt gezogen:

$$w_{ji}^{(l)},\, b_j^{(l)} \sim \mathcal{U}\Big(-\sqrt{1/n_{l-1}},\ \sqrt{1/n_{l-1}}\Big)\,. \tag{45}$$

Die Zufallszahlen liefert der Generator *mulberry32* (32-Bit-Zustand, Periode 2^{32}), initialisiert mit dem Startwert aus dem Tab „Konfiguration“ (Standard 42). Gleicher Startwert, gleiche Daten und gleiche Einstellungen ergeben damit exakt dieselben Startgewichte, dieselbe Mischreihenfolge und dasselbe Ergebnis.

11 Hyperparameter

11.1 Learning Rate η

Bedeutung: Schrittgröße bei Gradient Descent

- Zu groß: Instabilität, Oszillation, Divergenz
- Zu klein: Langsame Konvergenz, lokale Minima
- Typische Werte: 0.001 bis 0.1

$$\eta_{\text{optimal}} \approx \frac{1}{\lambda_{\max}(\mathbf{H})} \tag{46}$$

wobei $\lambda_{\max}(\mathbf{H})$ der größte Eigenwert der Hesse-Matrix ist.

11.2 Anzahl Epochs

Bedeutung: Wie oft durchläuft das Netzwerk den gesamten Datensatz

- Zu wenig: Underfitting, schlechte Generalisierung
- Zu viel: Overfitting auf Trainingsdaten
- Typische Werte: 1000 bis 10000 je nach Problem

11.3 Batch Size B

Bedeutung: Anzahl Samples pro Gradient-Update

- Klein ($B = 1$): Stochastic GD, hohes Rauschen, schnelle Updates
- Groß ($B = N$): Batch GD, stabil, langsame Updates
- Mittel ($B = 32\text{-}256$): Guter Kompromiss

$$\text{Updates pro Epoch} = \left\lceil \frac{N}{B} \right\rceil \tag{47}$$

11.4 Netzwerk-Architektur

Anzahl Hidden Layers:

- 0-1: Lineare/einfache nichtlineare Probleme
- 2-3: Komplexe nichtlineare Probleme (Standard)
- ≥ 4: Deep Learning, sehr komplexe Probleme

Neuronen pro Layer:

$$n_{\text{hidden}} \approx \frac{n_{\text{input}} + n_{\text{output}}}{2} \tag{48}$$

oder nach Kolmogorov-Nielsen-Theorem:

$$n_{\text{hidden}} \geq 2 \cdot n_{\text{input}} + 1 \tag{49}$$

12 Datenaufteilung und Backtesting

12.1 Aufteilung in Training und Test

Bei N Zeilen und Trainingsanteil q (Standard 80 %) umfasst das Training bei chronologischer Aufteilung $N_{\text{train}} = \lfloor qN \rfloor$ Zeilen (251 Zeilen, 80 % → 200 Training, 51 Test).

- **Chronologisch:** die ersten N_{train} Zeilen sind Training, die letzten Test. Für Zeitreihen die einzig realistische Wahl.
- **Gleichmäßig verteilt:** jede k-te Zeile ist Test mit $k = \max\big(2, \text{round}(1/(1-q))\big)$, also bei 80 % jede 5. Zeile.

12.2 Vortageswert als Input (Lag 1)

Optional wird der Wert der Zielgröße aus der Vorzeile als zusätzlicher Input $x_t^{\text{lag}} = y_{t-1}$ ergänzt; die erste Zeile entfällt. Damit kann das Netz mindestens die Regel „wie am Vortag“ abbilden.

12.3 Kennzahlen

Für n Zeilen mit Istwert y_i und Prognose $\hat{y}_i$ (Originalwerte), $e_i = \hat{y}_i - y_i$:

$$\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n}|e_i|, \qquad \mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}e_i^2}, \qquad \mathrm{MAPE} = \frac{100\,\%}{n}\sum_{i=1}^{n}\left|\frac{e_i}{y_i}\right|, \tag{50}$$

$$R^2 = 1 - \frac{\sum_i e_i^2}{\sum_i (y_i - \bar{y})^2}, \qquad \mathrm{Bias} = \frac{1}{n}\sum_{i=1}^{n} e_i, \qquad \mathrm{Max} = \max_i |e_i|. \tag{51}$$

$\bar{y}$ ist der Mittelwert der jeweils ausgewerteten Zeilen; $R^2 < 0$ bedeutet, dass die Prognose schlechter ist als dieser konstante Mittelwert. Bei Klassifikation (Output Sigmoid, Ziel 0/1) kommt die Trefferquote mit Schwelle 0,5 hinzu.

Der Rechner meldet *mögliche Überanpassung*, wenn $\mathrm{RMSE}_{\mathrm{Test}} > 2\,\mathrm{RMSE}_{\mathrm{Train}}$ (Regression) bzw. die Trefferquote im Test mehr als 10 Prozentpunkte unter der im Training liegt.

12.4 Vergleichsmaßstäbe

Im Testzeitraum wird das Netz mit drei einfachen Regeln verglichen:

$$\hat{y}_t^{\mathrm{Mittel}} = \bar{y}_{\mathrm{Train}}, \qquad \hat{y}_t^{\mathrm{Letzter}} = y_{t_0-1}, \qquad \hat{y}_t^{\mathrm{Vortag}} = y_{t-1}, \tag{52}$$

wobei y_{t_0-1} der letzte Trainingswert vor Testbeginn t_0 ist. Mit $r = \mathrm{RMSE}_{\mathrm{MLP}} / \min_k \mathrm{RMSE}_k$ lautet das Urteil: „besser“ für $r < 0{,}95$, „etwa gleich gut“ für $0{,}95 \le r \le 1{,}05$, sonst „schlechter“. Bei Klassifikation ist der Maßstab die häufigste Klasse der Trainingsdaten.

12.5 Extrapolationsprüfung

Für jeden Input j ist der Trainingsbereich $[m_j, M_j]$ (Minimum und Maximum der Trainingsdaten). Die Abweichung eines Werts x ist

$$o_j(x) = \begin{cases} x - m_j & x < m_j \\ x - M_j & x > M_j \\ 0 & \text{sonst,} \end{cases} \qquad \text{relativ: } \frac{|o_j(x)|}{M_j - m_j} \cdot 100\,\%. \tag{53}$$

Eine Zeile gilt als Extrapolation, wenn $o_j \neq 0$ für mindestens ein j. Die Prüfung erfolgt je Input einzeln; ungewohnte *Kombinationen* innerhalb der Einzelbereiche werden nicht erkannt.

13 Verwendung des Rechners

13.1 Workflow

Schritt 1: Daten laden (Tab „Daten“)

1. Beispiel wählen – die empfohlenen Einstellungen werden übernommen und das Netz erstellt – oder eigene CSV laden (Komma oder Semikolon, Kopfzeile, Text- und Datumsspalten werden ignoriert).

2. Zielspalte wählen, Normalisierung, Lag-Option und Train/Test-Aufteilung festlegen.

Schritt 2: Konfiguration prüfen (Tab „Konfiguration“)

1. Empfehlungsbox: Soll/Ist je Parameter; Faustregel Hidden 1 = 2–4 × Inputs, Hidden 2 ≈ halb so groß.
2. Output Linear und MSE für Regression, Sigmoid und BCE für Klassifikation.
3. Startwert festhalten, um Einstellungen vergleichen zu können.

Schritt 3: Training (Tab „Training“) – Loss-Kurve beobachten.
Schritt 4: Backtesting (Tab „Backtesting“) – Test-Kennzahlen, Vergleichsmaßstäbe, Extrapolation.
Schritt 5: Vorhersage und Export – Eingaben in Originalwerten, Batch-Dateien, CSV- und PNG-Export.

13.2 Beispiel: Kreuzkurse (UZS)

Daten: 251 Handelstage (synthetisch), Inputs USD/UZS, EUR/USD, GBP/UZS, CNY/UZS, Ziel EUR/UZS mit

$$\text{EUR/UZS}_t = \text{EUR/USD}_t \cdot \text{USD/UZS}_t \cdot (1 + \varepsilon_t), \qquad \varepsilon_t \sim \mathcal{N}(0,\, 0{,}0004^2). \tag{54}$$

GBP/UZS und CNY/UZS haben keinen Einfluss auf das Ziel.
Einstellungen: Normalisierung aktiv, Architektur $(4, 12, 6, 1)$, ReLU, ReLU, Linear, MSE, $\eta = 0{,}003$, 2000 Epochen, Startwert 42, chronologische Aufteilung 200/51 – 145 Parameter.
Ergebnis (Testzeitraum, 51 Tage):

Verfahren	MAE	RMSE	MAPE	R^2
MLP	5,34	6,44	0,037 %	0,9959
Vortag	50,92	62,12	0,354 %	0,6209
Letzter Wert	80,15	100,91	0,558 %	−0,0005
Mittelwert Training	85,88	104,31	0,597 %	−0,0690

$r = 6{,}44/62{,}12 \approx 0{,}10$: Das Netz ist rund 90 % besser als die beste einfache Regel. Im Training liegen MAE 4,68 und MAPE 0,033 % – keine Überanpassung. 27 Testzeilen sind Extrapolation (USD/UZS bis 14,8 % unter, GBP/UZS bis 24,7 % über dem Trainingsbereich); ihr MAE (5,03) ist nicht größer als der der übrigen Zeilen (5,68).

Gegenbeispiel Währungsdaten (UZS): Mit USD, CNY, GBP, CHF als Inputs für EUR (Architektur $(4, 16, 8, 1)$, 3000 Epochen) ergibt sich im Test MAPE 0,557 % und $R^2 = -4{,}24$ – schlechter als die Regel „Vortag“ (MAPE 0,117 %). Mit Lag-Input erreicht das Netz MAPE 0,120 % bei $r \approx 0{,}99$, also nur die Güte dieser Regel. Ursache: Die Tagesveränderungen von EUR und den Inputs sind praktisch unkorreliert.

14 Troubleshooting

14.1 Training konvergiert nicht

Symptome: Loss bleibt konstant oder oszilliert wild
Ursachen und Lösungen:

- Learning Rate zu hoch ⇒ Reduzieren auf 0.001 oder 0.0001
- Daten nicht normalisiert ⇒ Normalisierung aktivieren

- Netzwerk zu klein ⇒ Mehr Neuronen oder Layer hinzufügen
- Vanishing Gradients ⇒ ReLU statt Sigmoid verwenden

14.2 Overfitting

Symptome: Training-Loss niedrig, aber schlechte Generalisierung
Lösungen:

- Mehr Trainingsdaten sammeln
- Kleineres Netzwerk verwenden
- Früher stoppen (weniger Epochs)
- Regularisierung hinzufügen (zukünftige Version)

14.3 Underfitting

Symptome: Hohe Training-Loss, Netzwerk lernt nicht
Lösungen:

- Größeres Netzwerk (mehr Layer/Neuronen)
- Mehr Epochs trainieren
- Learning Rate erhöhen
- Komplexere Aktivierungsfunktionen

14.4 Numerische Instabilität

Symptome: NaN oder Infinity in Loss/Gewichten
Lösungen:

- Daten unbedingt normalisieren
- Learning Rate stark reduzieren
- Gradient Clipping implementieren
- Bei BCE: Sigmoid-Outputs von 0/1 fernhalten

15 Mathematische Beweise

15.1 Universelle Approximations-Theorem

Theorem 15.1 (Cybenko, 1989; Hornik et al., 1989)**.** Ein feedforward Netzwerk mit einer Hidden Layer und endlich vielen Neuronen kann jede stetige Funktion auf einem kompakten Teilbereich des $\mathbb{R}^n$ beliebig gut approximieren, vorausgesetzt die Aktivierungsfunktion ist nichtlinear, beschränkt und monoton steigend.

Formal: Für jede stetige Funktion $f : K \to \mathbb{R}$ auf kompaktem $K \subset \mathbb{R}^n$ und $\epsilon > 0$ existiert ein Netzwerk F mit:

$$\sup_{x \in K} |f(x) - F(x)| < \epsilon \tag{55}$$

15.2 Gradient Descent Konvergenz

Theorem 15.2 (Konvergenz bei konvexer Loss-Funktion)**.** Für eine L-Lipschitz-stetige, konvexe Funktion $\mathcal{L}$ konvergiert Gradient Descent mit konstantem Learning Rate $\eta \leq \frac{1}{L}$ zu einem globalen Minimum:

$$\lim_{t \to \infty} \mathcal{L}(\mathbf{w}_t) = \mathcal{L}^* \tag{56}$$

Anmerkung: Für neuronale Netzwerke ist die Loss-Funktion im Allgemeinen nicht konvex, daher wird nur Konvergenz zu lokalen Minima garantiert.

16 Implementierungs-Details

16.1 Matrix-Operationen

Die Implementierung nutzt eine Matrix-Klasse mit folgenden Operationen:

Matrix-Multiplikation:

$$(\mathbf{AB})_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} \tag{57}$$

Komplexität: $\mathcal{O}(n^3)$

Transpose:

$$(\mathbf{A}^T)_{ij} = a_{ji} \tag{58}$$

Hadamard-Produkt:

$$(\mathbf{A} \odot \mathbf{B})_{ij} = a_{ij} \cdot b_{ij} \tag{59}$$

16.2 Komplexitätsanalyse

Forward Pass:

$$\mathcal{O}\left(\sum_{l=1}^{L-1} n_l \cdot n_{l-1}\right) = \mathcal{O}(W) \tag{60}$$

wobei W die Gesamtzahl der Gewichte ist.

Backward Pass:

$$\mathcal{O}(W) \tag{61}$$

Training (eine Epoch):

$$\mathcal{O}(N \cdot W) \tag{62}$$

wobei N die Anzahl der Trainingsdaten ist.

17 Visualisierungen

17.1 Netzwerk-Graph

Die Visualisierung zeigt:

- **Neuronen:** Kreise, farbcodiert nach Layer
- **Gewichte:** Linien zwischen Neuronen
- **Farbe:** Rot = positive Gewichte, Blau = negative Gewichte
- **Dicke:** Proportional zur Größe des Gewichts

17.2 Aktive Neuronen

Für eine Eingabe $\mathbf{x}$ (aktuelle Werte im Tab „Vorhersage“, sonst letzte Datenzeile) wird jedem Neuron ein Status nach $|a_j^{(l)}|$ zugeordnet:

$$\text{inaktiv: } |a| = 0, \quad \text{schwach: } 0 < |a| < 0{,}2, \quad \text{normal: } 0{,}2 \le |a| < 0{,}8, \quad \text{stark: } |a| \ge 0{,}8. \tag{63}$$

Die Grafik zeigt standardmäßig nur normale Neuronen, schwache blass und starke optional; inaktive erscheinen als grauer Punkt. Eine Verbindung $i \to j$ wird mit der Dicke $|w_{ji}^{(l)} a_i^{(l-1)}|$ gezeichnet (rot positiv, blau negativ), sofern der Beitrag mindestens 2 % des größten Beitrags erreicht. So wird sichtbar, über welche Pfade die konkrete Prognose zustande kommt.

17.3 Loss-Kurve

Die Loss-Kurve $\mathcal{L}(t)$ zeigt:

- Y-Achse: Loss-Wert
- X-Achse: Epoch-Nummer
- Idealverlauf: Monoton fallend und konvergierend

Interpretation:

- Steiler Abstieg am Anfang: Gutes Lernen
- Plateau: Konvergenz erreicht
- Oszillation: Learning Rate zu hoch
- Anstieg: Overfitting oder numerische Probleme

17.4 Gewichtsverteilung

Histogramm aller Gewichte:

- Gut trainiert: Normalverteilt um 0
- Probleme: Sehr kleine oder sehr große Gewichte
- Dead Neurons: Viele Gewichte genau 0

18 Export-Formate

Alle Exporte sind CSV-Dateien (UTF-8 mit BOM) bzw. PNG-Grafiken.

Datei	Spalten	Format
`mlp_gewichte.csv`	Layer, From_Neuron, To_Neuron, Weight, Bias	Komma, Punkt
`mlp_architektur.csv`	Parameter, Wert	Komma, Punkt
`mlp_aktivierungen.csv`	Layer, Neuron, Activation_Value	Komma, Punkt
`training_verlauf.csv`	Epoch, Loss	Komma, Punkt
Backtesting	Zeile, Verwendung, Inputs, Prognose, Ist, Fehler, Fehler %	Semikolon, Dezimalkomma
Batch-Vorhersagen	Index, Inputs, Prognose [, Ist, Fehler, Fehler %] [, Klasse]	Semikolon, Dezimalkomma
Batch-Datei Testzeitraum	Inputs, Ziel (Kopfzeile mit Spaltennamen)	Komma, Punkt

```
Epoch,Loss
1,0.524892
2,0.487321
...
```

19 Erweiterungsmöglichkeiten

19.1 Zukünftige Features

- **Dropout:** Regularisierung durch zufälliges Deaktivieren von Neuronen
- **Momentum:** Beschleunigte Konvergenz
- **Adam Optimizer:** Adaptive Learning Rates
- **Batch Normalization:** Stabileres Training
- **Multi-Output:** Mehrere Output-Neuronen
- **Convolutional Layers:** Für Bilddaten
- **Recurrent Layers:** Für Sequenzdaten
- **Cross-Validation:** Bessere Evaluierung
- **Early Stopping:** Automatisches Stoppen bei Overfitting
- **Learning Rate Scheduling:** Anpassung während Training

19.2 Dropout

$$\tilde{a}_j^{(l)} = \begin{cases} 0 & \text{mit Wahrscheinlichkeit } p \\ \frac{a_j^{(l)}}{1-p} & \text{mit Wahrscheinlichkeit } 1-p \end{cases} \tag{64}$$

19.3 Momentum

$$v_t = \gamma v_{t-1} + \eta \nabla_{\mathbf{W}} \mathcal{L} \tag{65}$$

$$\mathbf{W}_t = \mathbf{W}_{t-1} - v_t \tag{66}$$

wobei $\gamma \in [0, 1]$ der Momentum-Parameter ist (typisch: $\gamma = 0.9$).

19.4 Adam Optimizer

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \nabla_{\mathbf{W}} \mathcal{L} \tag{67}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) (\nabla_{\mathbf{W}} \mathcal{L})^2 \tag{68}$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{69}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{70}$$

$$\mathbf{W}_t = \mathbf{W}_{t-1} - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t \tag{71}$$

Typische Werte: $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$

20 Referenzen und Weiterführende Literatur

20.1 Grundlegende Werke

1. Rosenblatt, F. (1958). "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain"
2. Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). "Learning representations by back-propagating errors"
3. Cybenko, G. (1989). Äpproximation by superpositions of a sigmoidal function"
4. Hornik, K., Stinchcombe, M., & White, H. (1989). "Multilayer feedforward networks are universal approximators"
5. LeCun, Y., Bengio, Y., & Hinton, G. (2015). "Deep learning"

20.2 Lehrbücher

1. Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.
2. Nielsen, M. A. (2015). *Neural Networks and Deep Learning*. Determination Press.
3. Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.

21 Anhang

21.1 Formelsammlung

Aktivierungsfunktionen:

$$\sigma(z) = \frac{1}{1+e^{-z}}, \quad \sigma'(z) = \sigma(z)(1-\sigma(z)) \tag{72}$$

$$\mathrm{ReLU}(z) = \max(0, z), \quad \mathrm{ReLU}'(z) = \mathbb{1}_{z>0} \tag{73}$$

$$\tanh(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}, \quad \tanh'(z) = 1 - \tanh^2(z) \tag{74}$$

Loss-Funktionen:

$$\mathcal{L}_{\mathrm{MSE}} = \frac{1}{2}(y-\hat{y})^2, \quad \frac{\partial \mathcal{L}_{\mathrm{MSE}}}{\partial \hat{y}} = \hat{y} - y \tag{75}$$

$$\mathcal{L}_{\mathrm{BCE}} = -[y \log \hat{y} + (1-y)\log(1-\hat{y})], \quad \frac{\partial \mathcal{L}_{\mathrm{BCE}}}{\partial \hat{y}} = \frac{\hat{y}-y}{\hat{y}(1-\hat{y})} \tag{76}$$

Backpropagation:

$$\boldsymbol{\delta}^{(L-1)} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{(L-1)}} \odot f'^{(L-1)}(\mathbf{z}^{(L-1)}) \tag{77}$$

$$\boldsymbol{\delta}^{(l)} = [(\mathbf{W}^{(l+1)})^T \boldsymbol{\delta}^{(l+1)}] \odot f'^{(l)}(\mathbf{z}^{(l)}) \tag{78}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(l)}} = \boldsymbol{\delta}^{(l)} (\mathbf{a}^{(l-1)})^T \tag{79}$$

Gradient Descent:

$$\mathbf{W}^{(l)} \leftarrow \mathbf{W}^{(l)} - \eta \cdot \frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(l)}} \tag{80}$$

Normalisierung:

$$x_{\text{norm}} = \frac{x - x_{\min}}{x_{\max} - x_{\min}} \tag{81}$$

$$x_{\text{orig}} = x_{\text{norm}} \cdot (x_{\max} - x_{\min}) + x_{\min} \tag{82}$$

21.2 Glossar

Activation Function Nichtlineare Funktion, die auf Pre-Activations angewendet wird

Backpropagation Algorithmus zur Berechnung von Gradienten

Batch Teilmenge des Trainingsdatensatzes

Bias Additiver Parameter in Neuronen (Schwellenwert)

Epoch Ein vollständiger Durchlauf durch den Trainingsdatensatz

Forward Pass Berechnung der Netzwerk-Ausgabe für einen Input

Gradient Vektor der partiellen Ableitungen

Hidden Layer Schicht zwischen Input und Output

Learning Rate Schrittgröße für Parameter-Updates

Loss Function Funktion zur Messung des Fehlers

Neuron Berechnungseinheit im Netzwerk

Overfitting Modell passt sich zu sehr an Trainingsdaten an

Weight Parameter, der Verbindungen zwischen Neuronen gewichtet

21.3 Notation

Symbol	Bedeutung
L	Anzahl Layers
n_l	Anzahl Neuronen in Layer l
$w_{ji}^{(l)}$	Gewicht von Neuron i (Layer $l-1$) zu j (Layer l)
$b_j^{(l)}$	Bias von Neuron j in Layer l
$z_j^{(l)}$	Pre-Activation von Neuron j in Layer l
$a_j^{(l)}$	Activation von Neuron j in Layer l
$\mathbf{W}^{(l)}$	Gewichtsmatrix für Layer l
$\mathbf{b}^{(l)}$	Bias-Vektor für Layer l
$f^{(l)}$	Aktivierungsfunktion in Layer l
$\mathcal{L}$	Loss-Funktion
η	Learning Rate
$\delta_j^{(l)}$	Fehler-Signal von Neuron j in Layer l
$\nabla_{\mathbf{W}}\mathcal{L}$	Gradient der Loss bezüglich Gewichten
$\odot$	Hadamard-Produkt (elementweise Multiplikation)
$\mathbb{1}$	Indikatorfunktion

22 Schlusswort

Dieses Handbuch bietet eine umfassende mathematische und praktische Einführung in Multi-Layer-Perzeptronen. Die Implementierung im Rechner folgt den hier beschriebenen Algorithmen und ermöglicht es, die Theorie direkt in der Praxis anzuwenden.

Für Fragen, Verbesserungsvorschläge oder Fehlerberichte wenden Sie sich bitte an die Entwickler oder nutzen Sie das Feedback-System im Rechner.

Viel Erfolg beim Experimentieren mit neuronalen Netzwerken!