Deep Learning - Uebungsblatt 1

Mathematische Grundlagen und Einfuehrung

Fachhochschule Südwestfalen

23. Oktober 2025

Hinweise

- **Keine Abgabe erforderlich** - Diese Uebungen dienen der Selbstkontrolle und Vertiefung
- Empfohlene Bearbeitungszeit: 2 Wochen parallel zur Vorlesung
- Bei Programmieraufgaben: Code selbst ausführen und Ergebnisse interpretieren
- Rechenwege bei mathematischen Aufgaben vollständig nachvollziehen
- Zusammenarbeit in Lerngruppen ausdrücklich erwünscht
- Bei Fragen: Sprechstunden oder Diskussion im Kurs
- Lösungshinweise finden Sie am Ende des Dokuments

1 Mathematische Grundlagen

1.1 Lineare Algebra

Aufgabe 1.1: Gegeben seien die Matrizen:

$$\mathbf{A} = \begin{pmatrix} 2 & 1 & -1 \\ 0 & 3 & 2 \end{pmatrix}, \quad \mathbf{B} = \begin{pmatrix} 1 & 0 \\ -1 & 2 \\ 3 & 1 \end{pmatrix} \tag{1}$$

(a) Berechnen Sie $\mathbf{C} = \mathbf{A} \cdot \mathbf{B}$

(b) Berechnen Sie $\mathbf{B}^T \cdot \mathbf{A}^T$ und verifizieren Sie $(\mathbf{A} \cdot \mathbf{B})^T = \mathbf{B}^T \cdot \mathbf{A}^T$

(c) Interpretieren Sie die Dimensionen: Was bedeutet eine 2×3 Matrix in einem neuronalen Netzwerk?

1.2 Aktivierungsfunktionen

Wichtige Ableitungsregeln - Wiederholung:

- **Kettenregel:** $(f(g(x)))' = f'(g(x)) \cdot g'(x)$
- **Quotientenregel:** $\left(\frac{f(x)}{g(x)}\right)' = \frac{f'(x)g(x)-f(x)g'(x)}{(g(x))^2}$
- **Exponentialfunktion:** $(e^{f(x)})' = e^{f(x)} \cdot f'(x)$
- **Produktregel:** $(f(x) \cdot g(x))' = f'(x)g(x) + f(x)g'(x)$
- **Spezialfälle:** $(e^x)' = e^x$, $(\ln(x))' = \frac{1}{x}$, $(x^n)' = nx^{n-1}$

Aufgabe 1.2: Analysieren Sie die folgenden Aktivierungsfunktionen:

(a) **Sigmoid-Funktion:** $\sigma(x) = \frac{1}{1+e^{-x}}$

- Berechnen Sie die Ableitung $\sigma'(x)$
- Zeigen Sie, dass $\sigma'(x) = \sigma(x)(1 - \sigma(x))$
- Berechnen Sie $\sigma(0)$, $\sigma(2)$, $\sigma(-2)$

(b) **ReLU-Funktion:** $\text{ReLU}(x) = \max(0, x)$

- Skizzieren Sie die Funktion fuer $x \in [-3, 3]$
- Geben Sie die Ableitung an (beachten Sie $x = 0$)
- Erklaeren Sie das "Dying ReLU"Problem

2 Grundlagen Neuronaler Netze

2.1 Perceptron

Aufgabe 2.1: Ein Perceptron habe die Gewichte $\mathbf{w} = (2, -1, 1)^T$ und Bias $b = -1$.

(a) Berechnen Sie die Ausgabe fuer die Eingaben:

- $\mathbf{x}_1 = (1, 0, 1)^T$
- $\mathbf{x}_2 = (0, 1, 1)^T$
- $\mathbf{x}_3 = (1, 1, 0)^T$

(b) Verwenden Sie die Heaviside-Funktion als Aktivierung: $H(z) = \begin{cases} 1 & \text{wenn } z \geq 0 \\ 0 & \text{sonst} \end{cases}$

(c) Zeichnen Sie die Entscheidungsgrenze in einem 2D-Raum (setzen Sie $x_3 = 1$)

2.2 XOR-Problem

Aufgabe 2.2: Das XOR-Problem kann nicht mit einem einzelnen Perceptron geloest werden.

(a) Erklaeren Sie mathematisch, warum ein linearer Klassifikator das XOR-Problem nicht loesen kann

(b) Entwerfen Sie ein 2-Schicht-Netzwerk (3 Neuronen in der versteckten Schicht) zur Loesung des XOR-Problems

- Geben Sie konkrete Gewichte und Biases an
- Ueberpruefen Sie Ihre Loesung fuer alle vier XOR-Eingaben

3 Programmieraufgaben

3.1 Implementierung Grundfunktionen

Aufgabe 3.1: Implementieren Sie in Python (ohne ML-Bibliotheken wie TensorFlow/-PyTorch):

(a) Eine Klasse `ActivationFunctions` mit den Methoden:

- `sigmoid(x)` und `sigmoid_derivative(x)`
- `relu(x)` und `relu_derivative(x)`
- `tanh(x)` und `tanh_derivative(x)`

(b) Plotten Sie alle drei Aktivierungsfunktionen und ihre Ableitungen fuer $x \in [-5, 5]$

Beispiel-Code-Struktur:

```python
import numpy as np
import matplotlib.pyplot as plt

class ActivationFunctions:
    @staticmethod
    def sigmoid(x):
        # Ihre Implementierung hier
        pass

    @staticmethod
    def sigmoid_derivative(x):
        # Ihre Implementierung hier
        pass

    # Weitere Funktionen...

# Test und Visualisierung
x = np.linspace(-5, 5, 100)
# Plotten Sie hier...
```

3.2 Einfaches Perceptron

Aufgabe 3.2: Implementieren Sie ein einfaches Perceptron:

(a) Erstellen Sie eine Klasse `Perceptron` mit:

- Initialisierung von Gewichten und Bias
- `forward(x)`-Methode fuer Vorhersagen
- `train(X, y, epochs, learning_rate)`-Methode

(b) Trainieren Sie das Perceptron auf dem AND-Gate:

- Input: $\mathbf{X} = \begin{pmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{pmatrix}$
- Output: $\mathbf{y} = (0, 0, 0, 1)^T$

(c) Visualisieren Sie den Lernprozess (Fehler ueber Epochen) und die finale Entscheidungsgrenze

4 Verstaendnisfragen

Aufgabe 4.1: Beantworten Sie folgende Fragen ausfuehrlich:

(a) Erklaeren Sie den Unterschied zwischen linearen und nichtlinearen Aktivierungsfunktionen. Warum sind nichtlineare Funktionen essentiell fuer Deep Learning?

(b) Was ist der Universelle Approximationssatz und welche Bedeutung hat er fuer neuronale Netze?

(c) Erklaeren Sie das Konzept der ”Lernfaehigkeitëines neuronalen Netzes. Was unterscheidet ein lernendes System von einem statischen Algorithmus?

5 Bonusaufgabe

Aufgabe 5.1: Erweiterte mathematische Analyse:
Gegeben sei ein 2-Schicht-Netzwerk mit einer versteckten Schicht:

$$\mathbf{z}_1 = \mathbf{W}_1\mathbf{x} + \mathbf{b}_1 \tag{2}$$
$$\mathbf{a}_1 = \sigma(\mathbf{z}_1) \tag{3}$$
$$z_2 = \mathbf{w}_2^T\mathbf{a}_1 + b_2 \tag{4}$$
$$\hat{y} = \sigma(z_2) \tag{5}$$

Mit der Verlustfunktion $L = \frac{1}{2}(y - \hat{y})^2$:

(a) Leiten Sie $\frac{\partial L}{\partial \mathbf{w}_2}$ und $\frac{\partial L}{\partial b_2}$ her

(b) Leiten Sie $\frac{\partial L}{\partial \mathbf{W}_1}$ und $\frac{\partial L}{\partial \mathbf{b}_1}$ her (verwenden Sie die Kettenregel)

Lösungshinweise und Tipps

Zu Aufgabe 1.1 (Lineare Algebra)

- **Matrixmultiplikation:** $(AB)_{ij} = \sum_k A_{ik}B_{kj}$
- **Dimension prüfen:** $(m \times n) \cdot (n \times p) = (m \times p)$
- **Transposition:** $(AB)^T = B^T A^T$ (Reihenfolge umkehren!)
- **Interpretation:** 2×3 Matrix = 2 Ausgabe-Neuronen, 3 Eingabe-Features

Zu Aufgabe 1.2 (Aktivierungsfunktionen)

- **Sigmoid-Ableitung:** Nutzen Sie $\frac{d}{dx}(1 + e^{-x})^{-1}$ mit Kettenregel
- **Tipp:** $\frac{d}{dx}e^{-x} = -e^{-x}$
- **Vereinfachung:** Zeigen Sie $\sigma'(x) = \sigma(x)(1 - \sigma(x))$ durch Einsetzen
- **ReLU-Ableitung:** $\frac{d}{dx}\max(0, x) = \begin{cases} 0 & x < 0 \\ 1 & x > 0 \\ \text{undefiniert} & x = 0 \end{cases}$
- **"Dying ReLU:** Neuronen mit $z < 0$ haben Gradient 0 und lernen nicht mehr

Zu Aufgabe 2.1 (Perceptron)

- **Berechnung:** $z = \mathbf{w}^T\mathbf{x} + b = w_1x_1 + w_2x_2 + w_3x_3 + b$
- **Entscheidungsgrenze:** Lösen Sie $\mathbf{w}^T\mathbf{x} + b = 0$ nach x_2 auf
- **2D-Visualisierung:** Setzen Sie $x_3 = 1$ und plotten Sie x_1 vs. x_2

Zu Aufgabe 2.2 (XOR-Problem)

- **Linearität:** XOR ist nicht linear trennbar - zeigen Sie geometrisch
- **Lösung:** Verwenden Sie AND, OR, NAND Gates als Zwischenschicht
- **Beispiel-Architektur:**
  - Hidden Layer: 3 Neuronen (AND, OR, NAND)
  - Output: Kombination dieser Logikgatter

Zu Aufgabe 3.1 (Programmierung)

- **Numerische Stabilitaet:** $\text{sigmoid}(x) = \frac{1}{1+e^{-x}}$ fuer $x \geq 0$, $\frac{e^x}{e^x+1}$ fuer $x < 0$
- **Debugging:** Testen Sie mit bekannten Werten: $\sigma(0) = 0.5$, $\text{ReLU}(-1) = 0$
- **Visualisierung:** Verwenden Sie `plt.subplot()` fuer mehrere Plots
- **Code-Struktur:** Trennen Sie Funktion und Ableitung fuer bessere Lesbarkeit

Zu Aufgabe 3.2 (Perceptron-Implementation)

- **Gewichtsinitialisierung:** Kleine zufaellige Werte: `np.random.randn() * 0.01`
- **Learning Rule:** $\mathbf{w} = \mathbf{w} + \eta(y - \hat{y})\mathbf{x}$
- **Konvergenz:** AND-Gate sollte in wenigen Epochen konvergieren
- **Visualisierung:** Plotten Sie Entscheidungsgrenze als Linie in 2D

Zu Aufgabe 4.1 (Verständnisfragen)

- **Linearitaet:** Komposition linearer Funktionen bleibt linear
- **Nichtlinearitaet:** Aktivierungsfunktionen ermoeglichen komplexe Mappings
- **Universeller Approximationssatz:** Ein ausreichend großes Netzwerk kann jede stetige Funktion approximieren
- **Lernfaehigkeit:** Gradientenbasierte Optimierung der Parameter

Allgemeine Tipps

- **Schrittweise vorgehen:** Erst verstehen, dann implementieren
- **Kleine Beispiele:** Testen Sie mit 2×2 Matrizen vor groesseren Problemen
- **Dimensionen pruefen:** Kontrollieren Sie Array-Shapes bei jeder Operation
- **Visualisieren:** Plots helfen beim Verstaendnis der Konzepte
- **Literatur:** "Deep Learning"von Goodfellow et al., Kapitel 2-3

Haeufige Fehler vermeiden

- **Matrixmultiplikation:** Achten Sie auf korrekte Dimensionen
- **Indexierung:** Python verwendet 0-basierte Indizierung
- **Broadcasting:** NumPy-Arrays haben automatisches Broadcasting
- **Gradient Descent:** Verwenden Sie kleine Lernraten (0.01-0.1)

Weiterfuehrende Ressourcen

- **Buecher:**
  - "Deep Learning Goodfellow, Bengio, Courville
  - "Pattern Recognition and Machine Learning Bishop
- **Online:**
  - 3Blue1Brown Neural Networks Series (YouTube)

- Andrej Karpathy's "Hacker's Guide to Neural Networks"
- CS231n Stanford Lectures

- **Praxis:**
  - Jupyter Notebooks fuer interaktive Experimente
  - NumPy Documentation fuer Array-Operationen
  - Matplotlib Gallery fuer Visualisierungsideen