Dr. Konrad Völkel
Heinrich-Heine-Universität Düsseldorf
Wintersemester 2026/27

Einführung in Deep Learning
Blatt 6
(8 Punkte, abzugeben bis 12:00 am Mittwoch, dem 25.11.2026)

Jede Aufgabe gibt 4 Punkte.

Am Ende des Blattes steht eine **Anwesenheitsaufgabe**: Die rechnen wir gemeinsam in der Übung, sie wird nicht abgegeben und gibt keine Punkte. Sie ist Aufgabe 1 in klein – wer sie mitgerechnet hat, hat das Muster für Aufgabe 1c schon in der Hand.

Aufgabe 1. *Gegeben sei ein neuronales Netzwerk mit zwei ReLU Layern, wobei jeder Layer genau zwei Neuronen hat. Führe manuell Backpropagation für eine gegebene Eingabe $x = (x_1, x_2)$ und einen Zielwert $t = (t_1, t_2)$ durch. Bezeichne die Werte des ersten Layers als $y = (y_1, y_2)$ und die Werte des zweiten Layers (und damit des ganzen Netzes) als $z = (z_1, z_2)$. Bezeichne die Gewichte des ersten Layers mit der Matrix V und die Gewichte des zweiten Layer mit W. Verlustfunktion: $L = MSE$, also für eine einzelne Eingabe $L = \frac{1}{2}\|t - z\|^2$.*

a) Berechne den Forward Pass, also den Wert von z in Abhängigkeit von V, W und x. Das bedeutet: stelle eine Formel auf.

b) Berechne den Fehler L

c) Berechne den Gradienten des Fehlers bezüglich aller Gewichte.

d) Füge Bias-Terme b und c hinzu und mache die Teilaufgaben zuvor mit Bias.

Tipps:

Die (schwache) Ableitung von ReLU an einer Stelle x ist $[x > 0]$.

Nutze die Kettenregel.

Der Gradient eines Gewichts ist das Produkt aus der Eingabe des Neurons und der Ableitung des Fehlers bezüglich der Ausgabe des Neurons.

Die Ableitung von L nach der Ausgabe z ist $z - t$.

Die Updateregel für V_{11} ist beispielsweise $V_{11,neu} = V_{11,alt} - \rho * (\text{Ableitung von } L \text{ nach } V_{11})$, daher interessieren wir uns für diese Ableitung, sie ist gegeben als Ableitung von L nach y_1 mal der Ableitung von y_1 nach V_{11}. Konkret resultiert das in der Formel

$\left(\sum_{k=1}^{2} (z_k - t_k) * W_{k1} * [z_k > 0]\right) * [y_1 > 0] * x_1$.

Prüfen Sie, ob sie für V_{11} auch dieses Ergebnis haben.

Aufgabe 2. *Implementiere einen Linear Layer selbst: die Klasse `MeinLayer` soll von `torch.nn.Module` erben. Das "Vorbild" ist die Klasse `torch.nn.Linear`.*

Der Konstruktor `__init__` soll die Argumente `input_features` und `output_features` annehmen. Die Gewichte sollen zufällig (zwischen 0 und 1 uniform, d.h. gleichverteilt) gewählt werden. Die forward Methode sollte $Y = WX$ berechnen, wenn W die Gewichtsmatrix ist und X der Input. Benutze `torch.nn.Parameter` für die Gewichte, damit diese für die Backpropagation berücksichtigt werden. Teste das Verhalten des MeinLayer durch Substitution des Encoder&Decoder in der Datei `iris-autoencoder.py` und stelle sicher, dass der Code läuft. Geben Sie den Output mit ab und schreiben Sie auf, ob Ihre Version sich genau so verhält wie `torch.nn.Linear`.

***Zum Laufenlassen:** Die nötigen Pakete stehen im Kopf von `iris-autoencoder.py`, mit `uv` genügt daher `uv run iris-autoencoder.py` – ohne Grafikkarte besser mit dem Zusatz `--index https://download.pytorch.org/whl/cpu` (200 MB statt 2 GB, keine CUDA-Warnung). Das Programm öffnet kein Fenster, sondern legt `iris-autoencoder.png` neben sich ab – dieses Bild ist der Output.*

Einführung in Deep Learning
Blatt 6
(8 Punkte, abzugeben bis 12:00 am Mittwoch, dem 25.11.2026)

Bonusaufgabe: implementiere ein viel einfacheres Testverfahren, ob der `MeinLayer` funktioniert.

Bonusaufgabe 2: implementiere Bias-Terme im MeinLayer

Bonusaufgabe 3: implementiere die Möglichkeit der manuellen Festlegung der Gewichte im Konstruktor von MeinLayer.

Anwesenheitsaufgabe, ohne Abgabe

Diese Aufgabe rechnen wir gemeinsam in der Übung. Sie wird nicht abgegeben und gibt keine Punkte.

Anwesenheitsaufgabe 1. *Dasselbe Netz wie in Aufgabe 1, eine Dimension kleiner: In jeder Schicht steht statt einer Matrix ein einzelnes Gewicht, und statt Vektoren stehen Zahlen. Das Netz rechnet*

$$y = \mathrm{ReLU}(v\,x), \qquad z = \mathrm{ReLU}(w\,y), \qquad L = \tfrac{1}{2}(z-t)^2.$$

Gegeben sind $x = 2$, $t = 1$ und $v = 1{,}5$. Die (schwache) Ableitung von ReLU an einer Stelle s ist wieder $[s > 0]$.

a) Sei $w = -1$. Rechnen Sie vorwärts (y, z, L) und dann rückwärts: Wie groß sind $\partial L/\partial w$ und $\partial L/\partial v$? Was fällt auf?

b) Sei nun $w = +1$. Rechnen Sie wieder vorwärts und rückwärts und machen Sie für beide Gewichte einen Gradientenschritt mit der Lernrate $\rho = 0{,}1$. Wie groß ist der Fehler danach?

c) Gehen Sie noch einmal von $v = 1{,}5$ und $w = +1$ aus, machen den Schritt aber mit $\rho = 0{,}5$. Was passiert mit dem Netz?