1. (3 Punkte) Gegeben ist `t = torch.arange(60)`, also ein Tensor mit den Zahlen 0 bis 59.
   (a) Welchen `shape` hat `t.reshape(3, 4, 5)`? Wie viele Dimensionen hat er?
   (b) Welchen Wert hat `t.reshape(3, 4, 5)[2][1][3]`?
   (c) Welchen Wert hat `t.reshape(6, 10)[4][7]`?
   (d) Welchen `shape` hat `t.reshape(5, -1)`?

**Lösung:**
(a) `(3, 4, 5)`, 3 Dimensionen: 3 Matrizen mit je 4 Zeilen und 5 Spalten.
(b) Jede Matrix hat 20 Elemente, jede Zeile 5. Also $2 \cdot 20 + 1 \cdot 5 + 3 = 48$.
(c) $4 \cdot 10 + 7 = 47$.
(d) `(5, 12)`, denn $60/5 = 12$.

2. (4 Punkte) Die vier Inputs von AND stehen als Zeilen in einer Matrix:

$$X = \begin{pmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{pmatrix}$$

   (a) Berechne `z = X @ w + b` für $w = (1, 1)$ und $b = -1.5$. Welche Ausgaben liefert ein Perzeptron, das bei $z \geq 0$ feuert?
   (b) Drei Neuronen haben die Gewichte $(1, -1)$, $(-1, -1)$ und $(2, 1)$ und die Biaswerte 0, 1.5 und $-2$. Schreibt man die Gewichte als Zeilen einer Matrix $W$, berechnet `X @ W.T + b` alle Summen auf einmal. Welchen `shape` hat das Ergebnis? Berechne es.

**Lösung:**
(a) $z = (-1.5, -0.5, -0.5, 0.5)$, Ausgaben $(0, 0, 0, 1)$, das ist AND.
(b) `(4, 3)`: für jeden der 4 Inputs eine Summe für jedes der 3 Neuronen.

$$\begin{pmatrix} 0 & 1.5 & -2 \\ -1 & 0.5 & -1 \\ 1 & 0.5 & 0 \\ 0 & -0.5 & 1 \end{pmatrix}$$

3. (3 Punkte) (a) Wie viele Parameter hat eine Schicht `nn.Linear(784, 120)`?
   (b) Berechne die Zahl der Parameter für ein Netz mit den Schichten 784, 120, 60, 10 (wie im Notebook) und für ein Netz mit den Schichten 784, 256, 10.
   (c) Welche Schicht hat jeweils die meisten Parameter? Warum?

**Lösung:**

(a) $784 \cdot 120 = 94080$ Gewichte und 120 Biaswerte, zusammen 94200.

(b) $94200 + (120 \cdot 60 + 60) + (60 \cdot 10 + 10) = 102070$ und $(784 \cdot 256 + 256) + (256 \cdot 10 + 10) = 203530$.

(c) Jeweils die erste Schicht. Jeder der 784 Pixel ist mit jedem Neuron der ersten Schicht verbunden.

4. (3 Punkte) Die Ausgabeschicht eines Netzes mit drei Klassen liefert die Logits $(2, 1, 0)$.
   (a) Berechne mit der Softmax-Funktion die Wahrscheinlichkeiten der drei Klassen. Welche Klasse sagt das Netz voraus?
   (b) Die richtige Klasse ist die zweite (Nummer 1). Berechne den Fehler mit Cross-Entropy.
   (c) Wie müssten sich die Logits ändern, damit der Fehler kleiner wird?

**Lösung:**

(a) $e^2 = 7.3891$, $e^1 = 2.7183$, $e^0 = 1$, Summe 11.1073. Wahrscheinlichkeiten $(0.6652, 0.2447, 0.09)$. Vorhergesagt wird die erste Klasse (Nummer 0).

(b) $L = -\ln(0.2447) = 1.4076$

(c) Der Logit der Klasse 1 müsste größer werden oder die anderen kleiner, dann steigt ihre Wahrscheinlichkeit.

5. (3 Punkte) Ein Netz hat für zwei Klassen zwei Ausgabeneuronen mit den Logits $z$ und 0.
   (a) Zeige, dass die Softmax-Funktion der Klasse mit dem Logit $z$ die Wahrscheinlichkeit $\sigma(z)$ gibt.
   (b) Was folgt daraus für Cross-Entropy und Binary Cross-Entropy?

**Lösung:**

(a) $p = \dfrac{e^z}{e^z + e^0} = \dfrac{e^z}{e^z + 1}$. Teilt man Zähler und Nenner durch $e^z$, erhält man $\dfrac{1}{1 + e^{-z}} = \sigma(z)$.

(b) Bei zwei Klassen ist Softmax mit Cross-Entropy dasselbe wie ein einzelnes Ausgabeneuron mit Sigmoid und Binary Cross-Entropy. Beide Fehler sind der negative Logarithmus der Wahrscheinlichkeit der richtigen Klasse.

6. (4 Punkte) Was gibt das Programm aus? Rechne von Hand mit einem Rechengraphen.

```
a = torch.tensor(2.0, requires_grad=True)
b = torch.tensor(-3.0, requires_grad=True)
c = a * b
d = c + a
e = d * d
e.backward()
print(e.item(), a.grad.item(), b.grad.item())
```

**Lösung:** Vorwärts: $c = -6$, $d = -4$, $e = 16$.

Rückwärts: $\frac{\partial e}{\partial d} = 2d = -8$. Am Plus-Knoten geht das unverändert an $c$ und an $a$. Am Mal-Knoten $c = a \cdot b$: $\frac{\partial e}{\partial b} = -8 \cdot a = -16$, über $c$ kommt zu $a$ der Beitrag $-8 \cdot b = 24$.

$a$ kommt auf zwei Wegen zu $e$, die Beiträge werden addiert: $24 + (-8) = 16$.

Ausgabe: `16.0 16.0 -16.0`

7. (3 Punkte) Gegeben ist das Programm

```
x = torch.tensor(3.0, requires_grad=True)
for i in range(3):
    f = x * x
    f.backward()
    print(x.grad.item())
```

(a) Was gibt das Programm aus?

(b) Warum steht in jedem Trainingsschritt `optimizer.zero_grad()`? Was würde ohne passieren?

**Lösung:**

(a) 6.0, 12.0, 18.0. Die Ableitung $2x = 6$ wird bei jedem `backward()` zu `x.grad` addiert.

(b) `zero_grad()` setzt die Ableitungen auf 0, bevor `backward()` die neuen berechnet. Ohne würden sich die Ableitungen aller bisherigen Schritte aufsummieren, und die Schritte des Gradientenabstiegs würden immer größer und falsch.

8. (4 Punkte) So sieht das Training im Notebook aus:

```
for epoche in range(5):
    for bilder, labels in train_loader:
        loss = F.cross_entropy(netz(bilder), labels)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```

(a) Erkläre die vier Zeilen in der inneren Schleife.

(b) Es gibt 60000 Trainingsbilder, die Batchgröße ist 100. Wie oft werden die Parameter insgesamt verändert?

(c) Warum hat die letzte Schicht des Netzes keine Aktivierungsfunktion?

**Lösung:**

(a) Das Netz berechnet für die 100 Bilder des Batches die Logits, `cross_entropy` den mittleren Fehler. `zero_grad` setzt die alten Ableitungen auf 0. `backward` berechnet mit Backpropagation die Ableitungen des Fehlers nach allen Parametern. `step` verschiebt alle Parameter gegen ihre Ableitung.

(b) 600 Batches pro Epoche, 5 Epochen, also 3000 Mal.

(c) `F.cross_entropy` wendet die Softmax-Funktion selbst auf die Logits an.

9. (3 Punkte) Ein Netz mit 102070 Parametern wurde mit nur 1000 Trainingsbildern trainiert. Die Genauigkeiten:

| Epoche | Trainingsdaten | Testdaten |
|---|---|---|
| 10 | 0.845 | 0.7684 |
| 30 | 0.936 | 0.7962 |
| 60 | 0.99 | 0.7993 |

(a) Was fällt auf? Wie nennt man das?

(b) Nenne zwei Maßnahmen dagegen.

(c) Warum misst man die Genauigkeit überhaupt auf Testdaten?

**Lösung:**

(a) Auf den Trainingsdaten steigt die Genauigkeit fast auf 100 Prozent, auf den Testdaten bleibt sie bei etwa 80 Prozent. Das Netz lernt die Trainingsbilder auswendig: Overfitting.

(b) Mehr Trainingsdaten, ein kleineres Netz mit weniger Parametern, früher aufhören.

(c) Das Netz soll neue Bilder richtig einordnen. Auf den Trainingsdaten sieht man nur, wie gut es die bekannten Bilder kann.

10. (4 Punkte) Schreibe in Colab eine Klasse `NetzKlein` für Fashion-MNIST mit den Schichten 784, 64, 32, 10. Gib die Zahl der Parameter aus und prüfe mit `torch.randn(5, 1, 28, 28)`, dass das Netz für 5 Bilder die richtige Gestalt liefert. Trainiere es dann mit den Funktionen `trainiere` und `genauigkeit` aus dem Notebook fünf Epochen lang und vergleiche die Genauigkeit auf den Testdaten mit `NetzFashion`.

**Lösung:**

```
class NetzKlein(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(784, 64)
        self.fc2 = nn.Linear(64, 32)
        self.fc3 = nn.Linear(32, 10)

    def forward(self, t):
        t = t.reshape(-1, 784)
        t = torch.relu(self.fc1(t))
        t = torch.relu(self.fc2(t))
        return self.fc3(t)

klein = NetzKlein()
print(sum(p.numel() for p in klein.parameters()))
print(klein(torch.randn(5, 1, 28, 28)).shape)
```

Ergebnis: 52650 Parameter, Gestalt `torch.Size([5, 10])`.

Training wie im Notebook, mit dem Optimizer Adam und $lr = 0.001$:

```
optimizer = torch.optim.Adam(klein.parameters(), lr=0.001)
for epoche in range(5):
    trainiere(klein, train_loader, optimizer)
    print(genauigkeit(klein, test_loader))
```

Das kleine Netz erreicht auf den Testdaten ungefähr dieselbe Genauigkeit wie `NetzFashion` (bei uns 0.854 gegen 0.856), obwohl es nur halb so viele Parameter hat.