1. (4 Punkte) Ein Bigramm-Modell wird mit den vier Namen `anna`, `nina`, `ina`, `ana` trainiert. Der Punkt markiert Anfang und Ende, aus `anna` wird `.anna.`.
   (a) Zähle alle Bigramme.
   (b) Berechne die Wahrscheinlichkeiten für das Zeichen nach `n` und nach `a` (ohne +1).
   (c) Mit welcher Wahrscheinlichkeit erzeugt das Modell den Namen `nina`? Mit welcher den Namen `nana`, der nicht in der Liste steht?
   (d) Warum würde das Modell den Namen `nia` nie erzeugen? Wozu addiert man im Notebook zu jeder Zählung 1?

Lösung:

(a) `.a`: 2, `.i`: 1, `.n`: 1, `a.`: 4, `an`: 2, `in`: 2, `na`: 4, `ni`: 1, `nn`: 1

(b) nach `n`: `a`: $\frac{2}{3}$, `i`: $\frac{1}{6}$, `n`: $\frac{1}{6}$ nach `a`: `.`: $\frac{2}{3}$, `n`: $\frac{1}{3}$

(c) `nina`: $P(n|.) \cdot P(i|n) \cdot P(n|i) \cdot P(a|n) \cdot P(.|a) = \frac{1}{4} \cdot \frac{1}{6} \cdot 1 \cdot \frac{2}{3} \cdot \frac{2}{3} = \frac{1}{54} \approx 0.0185$
`nana`: $\frac{1}{4} \cdot \frac{2}{3} \cdot \frac{1}{3} \cdot \frac{2}{3} \cdot \frac{2}{3} = \frac{2}{81} \approx 0.0247$

(d) Das Paar `ia` kommt in den Trainingsnamen nicht vor, also ist $P(a|i) = 0$. Mit der +1 bekommt jedes Paar eine kleine Wahrscheinlichkeit größer 0. Sonst wäre der Fehler $-\ln(0)$ für Testnamen mit einem neuen Paar unendlich.

2. (3 Punkte) Wir nehmen das Bigramm-Modell aus der vorigen Aufgabe (ohne +1).
   (a) Berechne für den Namen `nina` für jedes der fünf Zeichen nach dem Anfangspunkt den Fehler $-\ln(p)$ und den Mittelwert. Das ist die Cross-Entropy des Modells für diesen Namen.
   (b) Wie groß wäre der mittlere Fehler eines Modells, das jedes der fünf Zeichen `.`, `a`, `i`, `n` und `o` mit gleicher Wahrscheinlichkeit rät?

Lösung:

(a) $-\ln(\frac{1}{4}) = 1.3863$, $-\ln(\frac{1}{6}) = 1.7918$, $-\ln(1) = 0$, $-\ln(\frac{2}{3}) = 0.4055$, $-\ln(\frac{2}{3}) = 0.4055$
Mittelwert 0.7978

(b) $-\ln(\frac{1}{5}) = \ln 5 \approx 1.6094$

3. (3 Punkte) Ein Sprachmodell liefert für drei mögliche nächste Zeichen die Logits $(2, 1, 0)$. Vor der Softmax-Funktion werden die Logits durch die Temperatur T geteilt.
   (a) Berechne die Wahrscheinlichkeiten für $T = 0.5$, $T = 1$ und $T = 2$.
   (b) Was bewirkt eine kleine, was eine große Temperatur beim Erzeugen von Text?

Lösung:

(a) $T = 0.5$: $(0.8668, 0.1173, 0.0159)$ $T = 1$: $(0.6652, 0.2447, 0.09)$ $T = 2$: $(0.5065, 0.3072, 0.1863)$

(b) Bei kleiner Temperatur bekommt das wahrscheinlichste Zeichen fast alles, der Text wird vorhersehbar und wiederholt sich eher. Bei großer Temperatur werden die Wahrscheinlichkeiten gleichmäßiger, der Text wird überraschender, aber auch fehlerhafter.

4. (4 Punkte) Die Zeichen `.`, `a`, `i`, `n` haben die Nummern 0 bis 3 und die Embeddings `.`: $(0, 0)$, `a`: $(1, 2)$, `i`: $(0.5, -1)$, `n`: $(-1, 0.5)$.
   (a) Schreibe die Embeddings als Matrix E mit 4 Zeilen und berechne den One-Hot-Vektor von `n` mal E.
   (b) Ein Modell bekommt als Kontext die letzten zwei Zeichen `an`. Welcher Vektor geht in die versteckte Schicht? Ein Neuron dort hat die Gewichte $(1, 0, 2, -2)$ und den Bias 0.5. Berechne seine Summe z.
   (c) Das MLP im Notebook hat 30 Zeichen, Embeddings der Länge 10, 3 Zeichen Kontext, 200 versteckte Neuronen und 30 Ausgaben. Berechne die Zahl der Parameter. Wie viele sind es bei 5 Zeichen Kontext?

Lösung:

(a) $(0, 0, 0, 1) \cdot E = (-1, 0.5)$, die vierte Zeile von E.

(b) $(1, 2, -1, 0.5)$, die Embeddings von `a` und `n` hintereinander. $z = 1 \cdot 1 + 0 \cdot 2 + 2 \cdot (-1) + (-2) \cdot 0.5 + 0.5 = -1.5$

(c) Embedding $30 \cdot 10 = 300$, versteckte Schicht $30 \cdot 200 + 200 = 6200$, Ausgabe $200 \cdot 30 + 30 = 6030$, zusammen 12530. Bei 5 Zeichen: $300 + (50 \cdot 200 + 200) + 6030 = 16530$.

5. (4 Punkte) Die aktuelle Position hat die Query $q = (1, 0)$. Die drei vorherigen Positionen haben die Keys $k_1 = (2, 0)$, $k_2 = (0, 1)$, $k_3 = (1, 1)$ und die Values $v_1 = (1, 0)$, $v_2 = (0, 1)$, $v_3 = (1, 1)$.
   (a) Berechne die Scores $q \cdot k_i$ (Skalarprodukt).
   (b) Berechne mit der Softmax-Funktion die Gewichte (ohne Teilen durch eine Wurzel).
   (c) Berechne das Ergebnis, die gewichtete Summe der Values.
   (d) Welche Position bekommt die meiste Aufmerksamkeit und warum?

Lösung:

(a) $(2, 0, 1)$

(b) $e^2 = 7.3891$, $e^0 = 1$, $e^1 = 2.7183$, Summe 11.1073, Gewichte $(0.6652, 0.09, 0.2447)$

(c) $0.6652 \cdot (1, 0) + 0.09 \cdot (0, 1) + 0.2447 \cdot (1, 1) = (0.91, 0.3348)$

(d) Position 1: Ihr Key zeigt in dieselbe Richtung wie die Query, das Skalarprodukt ist am größten.

6. (3 Punkte) Im Notebook wurden verschiedene Modelle mit 4378 Trainingsnamen trainiert und mit 487 Testnamen geprüft:

Modell	Fehler Training	Fehler Test
Bigramm	2.47	2.52
MLP, 3 Zeichen Kontext	2.02	2.39
MLP, 5 Zeichen Kontext	1.75	2.61
Attention, 8 Zeichen Kontext	2.06	2.33

   (a) Warum ist das MLP mit 5 Zeichen auf den Testnamen schlechter als mit 3, obwohl es mehr sieht?
   (b) Warum misst man den Fehler auf Testnamen, die beim Training nicht vorkamen?
   (c) Warum kann das Attention-Modell 8 Zeichen nutzen, ohne so stark zu überfitten?

Lösung:

(a) Overfitting: Mit 5 Zeichen Kontext kann sich das Modell die wenigen Trainingsnamen fast auswendig merken (kleiner Trainingsfehler), es verallgemeinert aber schlechter auf neue Namen.

(b) Ein Modell soll neue Namen gut vorhersagen, nicht die bekannten aufsagen. Nur auf unbekannten Namen sieht man, ob es etwas Allgemeines gelernt hat.

(c) Beim MLP hat jede Position im Fenster eigene Gewichte, mit mehr Kontext steigt die Zahl der Parameter. Beim Attention-Modell werden Query, Key und Value für alle Positionen mit denselben Gewichten berechnet. Es hat ähnlich viele Parameter wie das MLP mit 3 Zeichen und entscheidet selbst, welche Positionen wichtig sind.

7. (3 Punkte) Führe drei Schritte von Byte Pair Encoding für die Wörter `lena`, `elena`, `helena`, `leon`, `leni` durch: Am Anfang ist jedes Zeichen ein Token. In jedem Schritt wird das Paar nebeneinander stehender Tokens, das am häufigsten vorkommt, zu einem neuen Token verschmolzen. Gib nach dem dritten Schritt die Tokens jedes Wortes an.

Lösung: 1. Schritt: `l` + `e` → `le` (5-mal)
2. Schritt: `le` + `n` → `len` (4-mal)
3. Schritt: `len` + `a` → `lena` (3-mal)

`lena`: `lena`, `elena`: `e` | `lena`, `helena`: `h` | `e` | `lena`, `leon`: `le` | `o` | `n`, `leni`: `len` | `i`

8. (4 Punkte) (a) Was lernt ein großes Sprachmodell beim Vortraining? Was kommt bei der Feinabstimmung dazu?
   (b) Was sind Tokens und warum benutzt man keine ganzen Wörter?
   (c) Was ist eine Halluzination und warum entsteht sie?
   (d) Ein Chatbot erzeugt eine Antwort. Beschreibe, was dabei Schritt für Schritt passiert.

Lösung:

(a) Beim Vortraining lernt es an riesigen Textmengen, das nächste Token vorherzusagen. Bei der Feinabstimmung wird es mit Beispielen für gute Antworten und Bewertungen durch Menschen weiter trainiert, damit es Fragen beantwortet und Anweisungen befolgt.

(b) Tokens sind häufige Wortteile, z.B. gebildet mit Byte Pair Encoding. Mit ganzen Wörtern wäre der Wortschatz riesig, und neue oder seltene Wörter könnte das Modell nicht verarbeiten. Mit Tokens wird ein unbekanntes Wort in bekannte Teile zerlegt.

(c) Das Modell erzeugt eine Aussage, die überzeugend klingt, aber falsch ist, z.B. eine erfundene Quelle. Es erzeugt wahrscheinlichen Text, nicht geprüfte Fakten, so wie unser Namensmodell Namen erfindet, die es nicht gibt.

(d) Der bisherige Text (Frage und schon erzeugte Antwort) wird in Tokens zerlegt. Das Modell berechnet für jedes mögliche nächste Token eine Wahrscheinlichkeit. Ein Token wird ausgewürfelt (abhängig von der Temperatur) und angehängt. Das wiederholt sich, bis ein Ende-Token kommt.

9. (4 Punkte) Erweitere das Bigramm-Modell zu einem **Trigramm-Modell**, das die letzten zwei Zeichen kennt: Zähle, wie oft auf zwei Zeichen ein drittes folgt (am Anfang steht `..`, am Ende `.`), mit +1 für jede Kombination. Berechne den Fehler auf den Testnamen (gleiche Aufteilung wie im Notebook) und vergleiche mit dem Bigramm-Modell.

Lösung:

```
import math
import random

namen = open('vornamen.txt', encoding='utf-8').read().split()
random.seed(0)
random.shuffle(namen)
grenze = int(0.9 * len(namen))
train, test = namen[:grenze], namen[grenze:]
anzahl = len(set(''.join(namen))) + 1

zaehl = {}
kontext = {}
for name in train:
    s = '..' + name + '.'
    for i in range(len(s) - 2):
        zaehl[s[i:i + 3]] = zaehl.get(s[i:i + 3], 0) + 1
        kontext[s[i:i + 2]] = kontext.get(s[i:i + 2], 0) + 1

def p(zwei, naechstes):
    return (zaehl.get(zwei + naechstes, 0) + 1) / (kontext.get(zwei, 0) + anzahl)

werte = []
for name in test:
    s = '..' + name + '.'
    for i in range(len(s) - 2):
        werte.append(-math.log(p(s[i:i + 2], s[i + 2])))
print(round(sum(werte) / len(werte), 4))
```

Ergebnis: 2.4579. Das ist besser als das Bigramm-Modell (2.52), aber schlechter als das MLP mit drei Zeichen (2.39). Für mehr Kontext werden die Zähltabellen schnell zu groß und zu dünn besetzt.