4.4 Texte erzeugen: next word prediction und Sampling

Leitidee

Bisher hast du ein Sprachmodell nur **trainiert**: Du hast gezählt und Wahrscheinlichkeiten berechnet. Jetzt lässt du es **schreiben**. Die Idee ist verblüffend einfach: Das Modell schaut auf das letzte Wort, wählt mithilfe seiner Tabelle ein passendes nächstes Wort aus, hängt es an – und wiederholt das immer wieder. Spannend ist dabei die Frage, **wie** es unter mehreren möglichen Folgewörtern auswählt. Davon hängt ab, ob der Text eintönig, abwechslungsreich oder unsinnig wird. Auch die Trainingstexte und die Länge der n-Gramme verändern das Ergebnis deutlich.

Lernziele

Nach diesem Abschnitt kannst du

- beschreiben, wie ein n-Gramm-Modell Wort für Wort einen Text erzeugt (next word prediction),
- die Sampling-Strategien greedy, random und top-k benennen und von Hand anwenden,
- Textgenerierung mit greedy, random und top-k in Python umsetzen und vergleichen,
- beschreiben, wie sich die Länge der n-Gramme auf den erzeugten Text auswirkt,
- erläutern, wie der erzeugte Text von den Trainingsdaten abhängt.

4.4.1 Vom Modell zum Text

Ein Text entsteht beim n-Gramm-Modell Wort für Wort. Man gibt ein **Startwort** vor. Dann wiederholt das Modell immer dieselben drei Schritte:

1. Schaue in der Tabelle nach, welche Wörter auf das **letzte Wort** folgen können.
2. **Wähle** eines dieser Folgewörter aus.
3. Hänge es an den Text an. Es ist jetzt das neue letzte Wort.

Das Modell sagt also immer nur das **nächste Wort** voraus. Auf Englisch heißt das **next word prediction**. Hat das letzte Wort keine Folgewörter (weil es im Korpus nur ganz am Ende steht), bricht das Verfahren ab.

Definition: Textgenerierung (next word prediction)

Textgenerierung ist das Erzeugen eines neuen Textes durch ein Sprachmodell. Bei der **next word prediction** (Vorhersage des nächsten Wortes) bestimmt das Modell aus dem bisherigen Text ein passendes nächstes Wort.

Merke

Textgenerierung (next word prediction): Ausgehend von einem Startwort bzw. Startkontext wird mithilfe der n-Gramm-Tabelle immer wieder das **nächste Wort** bestimmt und an den Text angehängt.

Die naheliegendste Art, ein Folgewort auszuwählen: Man nimmt immer das **wahrscheinlichste**. Probiere das zuerst von Hand aus.

Arbeitsauftrag 34: Von Hand: die ersten Schritte der Textgenerierung

Verwende die Bigramm-Tabelle zum Hund-Katze-Korpus aus dem Abschnitt *Textanalyse mit dem Computer* („der hund läuft im park der hund spielt mit dem ball …“). Die Zeilen, die du brauchst, findest du dort in der Tabelle und in deinem Ergebnis zum Auftrag *Eigene Analyse* (Zeile „hund“). Für „im“ und „park“ gilt: Auf „im“ folgt immer „park“, auf „park“ immer „der“.

Beginne beim Wort „der“. Wähle jeweils das **wahrscheinlichste** Folgewort, bis du **5 Wörter** notiert hast. Haben mehrere Folgewörter die gleiche Wahrscheinlichkeit, nimm das, das in der Tabelle zuerst steht.

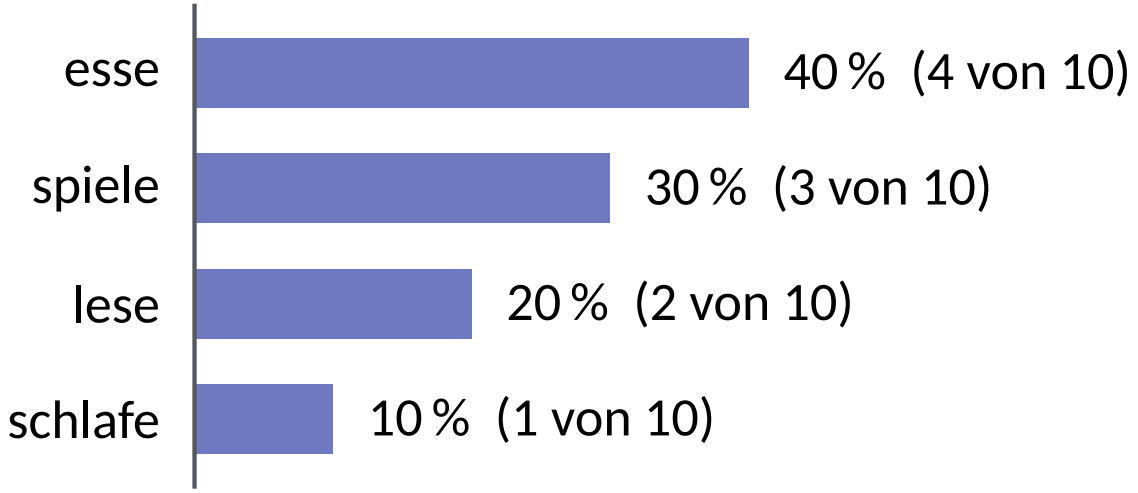
Mein Text aus 5 Wörtern:

4.4.2 Sampling: Wie wird das Folgewort ausgewählt?

Oft gibt es mehrere mögliche Folgewörter. Die Art, wie das Modell eines davon auswählt, heißt **Sampling-Strategie** (englisch *sample* = Stichprobe, Probe ziehen). Wir betrachten drei Strategien an einem kleinen Korpus über das Wochenende:

```
ich esse pizza     ich spiele fußball   ich esse eis
ich lese comics    ich spiele schach    ich esse pizza
ich schlafe lange  ich lese bücher      ich spiele fußball
ich esse nudeln
```

Das Wort „ich“ kommt zehnmal vor, und immer folgt ein Wort: viermal „esse“, dreimal „spiele“, zweimal „lese“ und einmal „schlafe“. Daraus ergeben sich diese Wahrscheinlichkeiten:



Mögliche Folgewörter nach „ich“ im Wochenend-Korpus mit ihren Wahrscheinlichkeiten

Definition: Sampling

Sampling ist die Auswahl des nächsten Wortes aus den möglichen Folgewörtern. Die **Sampling-Strategie** legt fest, **wie** ausgewählt wird.

Greedy: immer das wahrscheinlichste Wort

Bei der Strategie **greedy** (englisch für „gierig“) nimmt das Modell immer das Wort mit dem längsten Balken. Nach „ich“ kommt also **jedes Mal** „esse“. Der Name passt: Das Modell greift immer sofort nach dem größten Stück, ohne an später zu denken. Den Begriff „gierig“ kennst du vielleicht schon vom Problemlösen mit Heuristiken.

Greedy ist **deterministisch**: Gleiches Startwort, gleicher Text – jedes Mal. Das hat einen Nachteil. Im Wochenend-Korpus folgt auf „esse“ am häufigsten „pizza“, auf „pizza“ immer „ich“, und auf „ich“ wieder „esse“. Der Text dreht sich im Kreis: „ich esse pizza ich esse pizza ich esse pizza …“. Die Wörter „spiele“, „lese“ und „schlafe“ kommen nie vor, obwohl sie zusammen sogar häufiger sind als „esse“.

Random: Auslosen nach Wahrscheinlichkeit

Bei der Strategie **random** (englisch für „zufällig“) wird das Folgewort **ausgelost**. Dabei bekommt aber nicht jedes Wort die gleiche Chance. Häufige Wörter sollen öfter gezogen werden als seltene. Man sagt: Die Auswahl ist **gewichtet** nach der Wahrscheinlichkeit.

Das kannst du dir wie eine **Lostrommel** vorstellen. Für jedes Bigramm im Korpus kommt ein Los hinein. Nach „ich“ sind das 10 Lose: 4 mit „esse“, 3 mit „spiele“, 2 mit „lese“ und 1 mit „schlafe“. Wer blind ein Los zieht, erwischt „esse“ mit der Wahrscheinlichkeit $\frac{4}{10} = 40\,\%$ – genau wie in der Tabelle.



Die Lostrommel für „ich“: zehn Lose, nummeriert von 1 bis 10. Eine Zufallszahl von 1 bis 10 bestimmt das Los

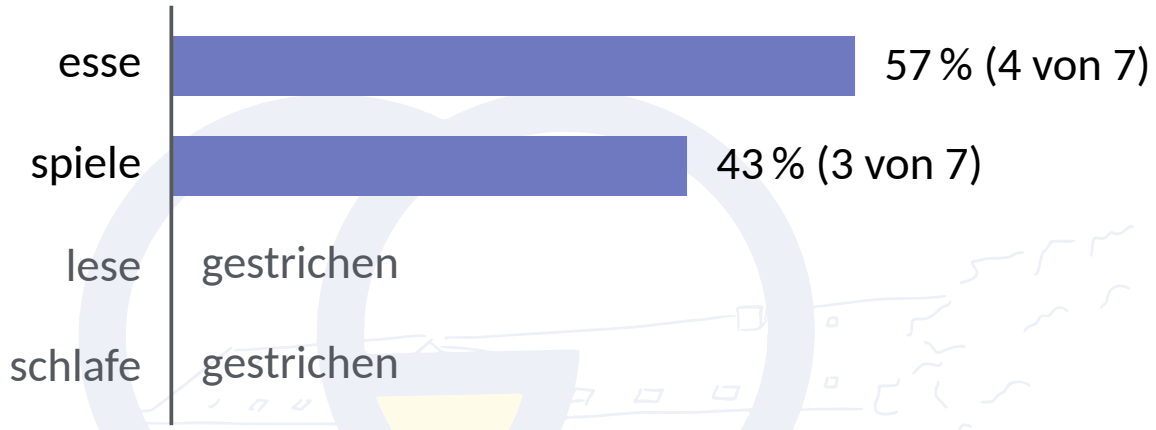
Mit einem Zehnerwürfel oder einem Zufallszahlengenerator geht das auch ohne Trommel: Ziehe eine Zufallszahl von 1 bis 10 und schau nach, welches Wort auf diesem Los steht. Die Zahlen 1 bis 4 bedeuten „esse“, 5 bis 7 „spiele“, 8 und 9 „lese“ und 10 „schlafe“.

Random ist **nicht deterministisch**: Jeder Durchlauf kann einen anderen Text liefern. Die Texte werden abwechslungsreicher. Aber auch seltene Folgewörter kommen manchmal dran, und dann kann der Text seltsam werden.

Top-k: nur unter den besten k auslosen

Die Strategie **top-k** ist ein Kompromiss. Zuerst werden nur die k wahrscheinlichsten Folgewörter behalten, alle anderen fliegen raus. Unter den übrigen wird dann wie bei random gewichtet ausgelost. Die Zahl k legt man vorher fest.

Bei **top-2** bleiben nach „ich“ nur „esse“ und „spiele“ übrig. In der Lostrommel liegen jetzt nur noch 4 + 3 = 7 Lose. Die Wahrscheinlichkeiten ändern sich deshalb: „esse“ hat $\frac{4}{7} \approx 57\,\%$ und „spiele“ $\frac{3}{7} \approx 43\,\%$.



Top-2 nach „ich“: „lese“ und „schlafe“ werden gestrichen. Unter „esse“ und „spiele“ wird mit 7 Losen neu gelost

Mit $k = 1$ bleibt nur das wahrscheinlichste Wort übrig – das ist greedy. Ist k mindestens so groß wie die Zahl aller Folgewörter, wird nichts gestrichen – das ist random. Top-k liegt also **zwischen** den beiden anderen Strategien.

Merke

Sampling-Strategien legen fest, **wie** aus mehreren möglichen Folgewörtern eines ausgewählt wird:

- **Greedy:** Es wird **immer** das **wahrscheinlichste** Folgewort gewählt → deterministisch, aber oft eintönig und sich wiederholend.
- **Random:** Es wird **zufällig** unter allen möglichen Folgewörtern gewählt, **gewichtet nach ihrer Wahrscheinlichkeit** → abwechslungsreicher, aber manchmal unsinnig.
- **Top-k:** Es werden nur die k wahrscheinlichsten Folgewörter betrachtet, unter denen dann zufällig (gewichtet) ausgewählt wird → Kompromiss zwischen Vorhersagbarkeit und Abwechslung.

So kannst du es sagen

So beschreibst du die Strategien in Fachsprache:

- *„Greedy nimmt das Beste.“ → „Bei greedy wird stets das Folgewort mit der höchsten Wahrscheinlichkeit gewählt.“*
- *„Random würfelt.“ → „Bei random wird das Folgewort zufällig ausgewählt, wobei häufigere Folgewörter mit höherer Wahrscheinlichkeit gezogen werden.“*

Arbeitsauftrag 35: Sampling von Hand

Verwende die Folgewörter nach „ich“ aus dem Wochenend-Korpus (4 × esse, 3 × spiele, 2 × lese, 1 × schlafe).

1. Welches Wort wählt greedy?
2. Bei random werden nacheinander die Zufallszahlen 7, 2, 10 und 9 gezogen (Lose wie in der Abbildung). Welche Wörter werden gewählt?
3. Bei top-2 sind die Lose 1 bis 4 „esse“ und 5 bis 7 „spiele“. Welche Wörter werden bei den Zufallszahlen 5 und 3 gewählt?
4. Berechne die Wahrscheinlichkeiten der Folgewörter bei **top-3** und zeichne ein Balkendiagramm.

Meine Antworten und mein Balkendiagramm:

4.4.3 Textgenerierung in Python

Jetzt bringen wir das alles in ein Programm. Wir brauchen die Funktionen aus dem Abschnitt *Textanalyse mit dem Computer*. Sie stehen hier noch einmal zusammen mit dem Hund-Katze-Korpus:

```python
def folgewoerter(woerter, wort):
    folge = []
    for i in range(len(woerter) - 1):
        if woerter[i] == wort:
            folge.append(woerter[i + 1])
    return folge

def ohne_doppelte(liste):
    ergebnis = []
    for element in liste:
        if element not in ergebnis:
            ergebnis.append(element)
    return ergebnis

def zaehle(liste, gesucht):
    anzahl = 0
    for element in liste:
        if element == gesucht:
            anzahl += 1
    return anzahl

korpus = """
der hund läuft im park der hund spielt mit dem ball
die katze sitzt auf dem baum die katze beobachtet den hund
der hund bellt laut der hund läuft zur katze
die katze springt vom baum die katze rennt weg
der hund läuft ihr hinterher der hund ist müde
"""
woerter = korpus.split()
```

Dieses Programm gibt noch nichts aus. Es stellt nur die Werkzeuge und die Wortliste bereit. Alle folgenden Programme in diesem Abschnitt bauen darauf auf: Schreibe sie in derselben Datei darunter.

Greedy in Python

Die Funktion für greedy sucht unter den Kandidaten den mit der größten Anzahl. Das ist dieselbe Idee wie die Suche nach dem Maximum in einer Liste:

```python
def naechstes_wort_greedy(woerter, wort):
    folge = folgewoerter(woerter, wort)
    if len(folge) == 0:
        return ""
    bestes = ""
    beste_anzahl = 0
    for kandidat in ohne_doppelte(folge):
        anzahl = zaehle(folge, kandidat)
        if anzahl > beste_anzahl:
            bestes = kandidat
            beste_anzahl = anzahl
    return bestes

print(naechstes_wort_greedy(woerter, "hund"))
print(naechstes_wort_greedy(woerter, "läuft"))
print(naechstes_wort_greedy(woerter, "müde"), "<- leer")
```

AUSGABE
```
läuft
im
 <- leer
```

- **Zeilen 2–4:** Hat das Wort keine Folgewörter, gibt die Funktion die leere Zeichenkette `""` zurück. Daran erkennt das aufrufende Programm später, dass es nicht weitergeht.
- **Zeilen 5–11:** Die Kandidaten werden der Reihe nach durchgegangen. Nur wenn ein Kandidat **mehr** Vorkommen hat als der bisher beste, wird er der neue beste. Bei Gleichstand bleibt der frühere Kandidat. Deshalb gewinnt nach „läuft“ das Wort „im“: Es steht unter den gleich häufigen Wörtern im Text zuerst.
- **Zeile 16:** „müde“ ist das letzte Wort des Korpus. Die Funktion gibt `""` zurück, deshalb steht vor dem Pfeil nichts.

Random in Python: die Lostrommel

Für random brauchen wir die Lostrommel. Und die haben wir schon! Die Liste `folgewoerter(woerter, wort)` enthält jedes Folgewort **so oft, wie es im Text vorkommt** – also genau ein Los pro Bigramm. Wir müssen nur eine zufällige Stelle dieser Liste auswählen:

```
1  import random
2
3  def naechstes_wort_random(woerter, wort):
4      folge = folgewoerter(woerter, wort)
5      if len(folge) == 0:
6          return ""
7      nummer = random.randint(0, len(folge) - 1)
8      return folge[nummer]
9
10 print(folgewoerter(woerter, "hund"))
11 print(naechstes_wort_random(woerter, "hund"))
12 print(naechstes_wort_random(woerter, "hund"))
13 print(naechstes_wort_random(woerter, "hund"))
```

AUSGABE
```
['läuft', 'spielt', 'der', 'bellt', 'läuft', 'läuft', 'ist']
läuft
der
läuft
```

Die Ausgabe ab der zweiten Zeile ist nur ein **Beispiel**. Bei dir kommen andere Wörter heraus, und bei jedem Start andere.

- **Zeile 1:** Das Modul `random` wird eingebunden.
- **Zeile 7:** `random.randint(0, len(folge) - 1)` liefert eine zufällige ganze Zahl von 0 bis zum letzten Index der Liste, jede mit gleicher Wahrscheinlichkeit. Das ist das blinde Ziehen eines Loses.
- **Zeile 8:** Das Wort an dieser Stelle wird zurückgegeben. Weil „läuft“ dreimal in der Liste steht, wird es mit der Wahrscheinlichkeit $\frac{3}{7}$ gezogen, jedes andere Wort mit $\frac{1}{7}$. Die Auswahl ist also **gewichtet**, ohne dass wir die Wahrscheinlichkeiten ausrechnen müssen.

Versuch: Probiere es aus: Stimmen die Wahrscheinlichkeiten?

Mit dem folgenden Programm (unter die bisherigen Programme schreiben) lässt du 1000-mal ein Folgewort nach „hund“ auslosen und zählst, wie oft „läuft“ kam. Erwartet sind etwa $\frac{3}{7} \cdot 1000 \approx 429$ Treffer. Starte es mehrmals.

```python
treffer = 0
for versuch in range(1000):
    if naechstes_wort_random(woerter, "hund") == "läuft":
        treffer += 1
print(f"läuft wurde {treffer}-mal gezogen")
```

AUSGABE

```
läuft wurde 437-mal gezogen
```

Die Zahl schwankt von Start zu Start (hier ein Beispiel), liegt aber fast immer in der Nähe von 429.

Einen ganzen Text erzeugen

Jetzt fehlt nur noch die Schleife, die Wort für Wort anhängt. Die Funktion `generiere_text` bekommt die Wortliste, ein Startwort, die gewünschte Länge (Anzahl der Wörter) und den Namen der Strategie als Zeichenkette:

```python
def generiere_text(woerter, startwort, laenge, strategie):
    text = startwort
    aktuelles_wort = startwort
    for schritt in range(laenge - 1):
        if strategie == "greedy":
            naechstes = naechstes_wort_greedy(woerter, aktuelles_wort)
        else:
            naechstes = naechstes_wort_random(woerter, aktuelles_wort)
        if naechstes == "":
            break
        text = text + " " + naechstes
        aktuelles_wort = naechstes
    return text

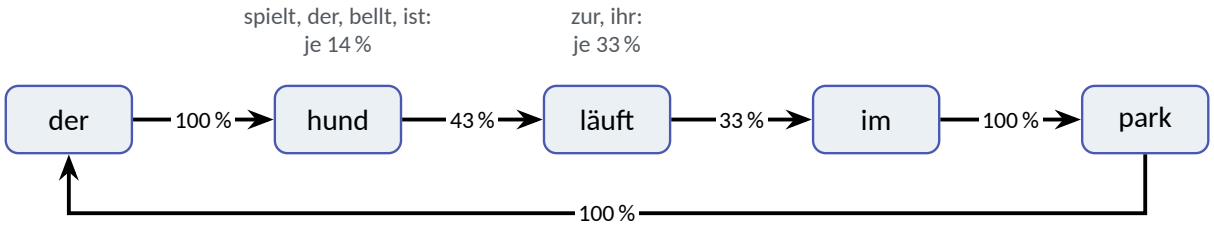
print("Greedy:", generiere_text(woerter, "der", 10, "greedy"))
```

AUSGABE

```
Greedy: der hund läuft im park der hund läuft im park
```

- **Zeilen 2–3:** Der Text beginnt mit dem Startwort. `aktuelles_wort` ist das jeweils letzte Wort.
- **Zeile 4:** Das Startwort ist schon da, also fehlen noch `laenge - 1` Wörter.
- **Zeilen 5–8:** Je nach Strategie wird das nächste Wort mit greedy oder random bestimmt.
- **Zeilen 9–10:** Gibt es kein Folgewort, wird die Schleife mit `break` vorzeitig beendet. Der Text ist dann kürzer als gewünscht.
- **Zeilen 11–12:** Das neue Wort wird mit einem Leerzeichen angehängt und wird zum aktuellen Wort.

Die ersten fünf Wörter stimmen mit deinem Ergebnis aus dem Auftrag *Von Hand: die ersten Schritte der Textgenerierung* überein. Danach beginnt greedy wieder von vorn: Der Text läuft in einer Schleife, wie die Abbildung zeigt.



Greedy im Hund-Katze-Korpus: Der Weg mit den jeweils höchsten Wahrscheinlichkeiten führt im Kreis. Die anderen Folgewörter (grau) werden nie gewählt

Und so sieht es mit random aus. Weil der Zufall mitspielt, sind die folgenden Ausgaben nur **Beispiele**:

```
print("Random:", generiere_text(woerter, "der", 10, "random"))
print("Random:", generiere_text(woerter, "der", 10, "random"))
print("Random:", generiere_text(woerter, "der", 10, "random"))
```

AUSGABE

```
Random: der hund läuft zur katze rennt weg der hund läuft
Random: der hund spielt mit dem baum die katze sitzt auf
Random: der hund ist müde
```

Im zweiten Beispiel ist ein neuer Satz entstanden, der so nicht im Korpus steht: „der hund spielt mit dem baum“. Jedes einzelne Wortpaar kommt im Korpus vor, die Kombination aber nicht. Im dritten Beispiel bricht der Text nach vier Wörtern ab, weil „müde“ keine Folgewörter hat.

Stolperstein

Wenn `generiere_text` einen `NameError` meldet, etwa `name 'naechstes_wort_random' is not defined`, fehlt eine der Funktionen in deiner Datei – oder sie steht **unter** dem Aufruf. Python muss eine Funktion kennen, bevor sie aufgerufen wird. Schreibe alle Funktionen an den Anfang der Datei.

4.4.4 Übungen zum Sampling

Arbeitsauftrag 36: Greedy und random vergleichen

Führe das Programm mit den drei Random-Zeilen und der Greedy-Zeile **mehrfach** aus. Was fällt dir beim Vergleich der Greedy- und der Random-Ausgaben auf? Erkläre, warum greedy immer denselben Text liefert, random aber unterschiedliche.

Meine Beobachtung und Erklärung:

Arbeitsauftrag 37 (Zusatz): Top-k-Sampling implementieren

Implementiere eine Funktion `naechstes_wort_top_k(woerter, wort, k)`, die nur unter den `k` häufigsten Folgewörtern zufällig auswählt, gewichtet nach ihrer Häufigkeit. Teste sie mit „hund“ und $k = 2$. Schreibe außerdem eine Funktion `generiere_text_top_k(woerter, startwort, laenge, k)`, die damit einen Text erzeugt.

Tipp

Ein möglicher Weg:

1. Lege zwei Listen an, `kandidaten` (ohne Doppelte) und `anzahlen`, die zusammengehören wie die zwei Spalten einer Tabelle.
2. Wiederhole k-mal: Suche die Position des größten Wertes in `anzahlen`. Nimm das Wort und seine Anzahl mit `pop` aus beiden Listen heraus und lege so viele Lose dieses Wortes in eine neue Liste `lostrommel`.
3. Ziehe zum Schluss ein zufälliges Los aus `lostrommel`.

4.4.5 Wie die Länge der n-Gramme den Text verändert

Bisher hat unser Modell nur auf **ein** Wort zurückgeschaut. Das ist der Grund für Sätze wie „der hund spielt mit dem baum“: Nach „dem“ weiß das Modell nicht mehr, dass vorher „spielt mit“ stand, und wählt „baum“ genauso oft wie „ball“.

Ein **Trigramm-Modell** schaut auf die letzten **zwei** Wörter. Nach „mit dem“ folgt im Korpus nur „ball“. Der seltsame Satz kann also nicht mehr entstehen. Der Text wird **zusammenhängender** – man sagt auch **kohärenter**.

Das hat aber einen Preis. Ein Kontext aus zwei Wörtern ist **spezifischer**: Er kommt seltener vor, und nach ihm gibt es weniger Auswahl. Im Hund-Katze-Korpus sieht das so aus:

	Bigramm-Modell	Trigramm-Modell
Kontext	1 Wort	2 Wörter
verschiedene Kontexte mit Folgewort	26	36
davon mit nur **einem** möglichen Folgewort	22	33
davon mit **Auswahl**	4	3

Beim Trigramm-Modell gibt es fast nirgends eine Auswahl. Der erzeugte Text ist dann fast immer ein Stück aus dem Korpus, Wort für Wort abgeschrieben. Er wird **weniger variantenreich**. Je länger die n-Gramme, desto stärker ist dieser Effekt. Außerdem findet ein Modell mit langen n-Grammen zu einem neuen Startkontext oft **gar keine** Fortsetzung, weil genau diese Wortfolge im Korpus nie vorkam.

Merke

Die **Länge des n-Gramms** beeinflusst die Textqualität: Größere n (z. B. Trigramme statt Bigramme) erzeugen meist **kohärentere**, aber dafür **weniger variantenreiche** Texte, da der Kontext spezifischer ist und damit seltener im Korpus vorkommt.

Arbeitsauftrag 38 (Zusatz): Einfluss der n-Gramm-Länge

Baue ein **Trigramm-Modell**, bei dem der Kontext aus einem Wortpaar besteht:

1. Schreibe eine Funktion `folgewoerter_2(woerter, wort1, wort2)`, die alle Wörter sammelt, die im Korpus direkt nach dem Wortpaar `wort1 wort2` stehen.
2. Schreibe eine Funktion `generiere_text_trigramm(woerter, wort1, wort2, laenge)`, die ab dem Startpaar mit random-Sampling einen Text erzeugt.
3. Vergleiche die erzeugten Texte mit denen des Bigramm-Modells: Wirken sie mit Trigrammen „sinnvoller“? Warum kann die Trigramm-Generierung bei einem kleinen Korpus schneller abbrechen? Probiere dazu auch das Startpaar „der ball“ aus.

Deine Antworten zu Teil 3 (die Funktionen schreibst du am Computer):

Gymnasium Ottweiler

4.4.6 Die Trainingsdaten bestimmen den Text

Ein n-Gramm-Modell kann nur Wörter und Wortfolgen erzeugen, die in seinem Korpus vorkommen. Was das bedeutet, untersuchst du im nächsten Auftrag.

Arbeitsauftrag 39: Abhängigkeit von den Trainingsdaten

Ersetze im Programm den Korpus durch einen **inhaltlich ganz anderen** Text, zum Beispiel:

```
korpus = """
heute scheint die sonne heute regnet es stark
morgen scheint die sonne morgen ist es bewölkt
heute ist es bewölkt heute regnet es leicht
"""
```

Zerlege ihn mit `split()` und erzeuge mehrmals Texte mit `generiere_text(woerter, "heute", 8, "random")`.

1. Wie stark unterscheidet sich der erzeugte Text vom Hund-Katze-Beispiel?
2. Was bedeutet das für die Aussage „Ein Sprachmodell erzeugt objektiven, neutralen Text“?
3. *Freiwillig:* Beobachte dasselbe mit SoekiaGPT, indem du dort zwischen zwei unterschiedlichen Textsammlungen wechselst und denselben Satzanfang eingibst. Das Prinzip ist identisch.

Meine Antworten:

Gymnasium Ottweiler

4.4.7 Weitere Übungen

Hier kannst du das Gelernte weiter üben. Die Punkte im Titel zeigen, wie schwer eine Aufgabe ist: ●○○ leicht, ●●○ mittel, ●●● knifflig. Fang mit den leichten an.

Mehrere Aufträge verwenden diesen Ausschnitt aus der Bigramm-Tabelle eines Korpus mit Chatnachrichten:

Kontext	mögliche Folgewörter (Anzahl)
ich	gehe (5), bin (3), habe (2)
gehe	heute (4), nach (1)
heute	ins (3), nicht (2)
ins	kino (4), bett (3)
kino	und (2)
und	ich (3), dann (1)
bin	müde (3), heute (1)
müde	und (2)

Arbeitsauftrag 40: Lückentext: next word prediction ●○○

Ergänze die Lücken. Verwende diese Wörter: *angehängt* · *endet* · *Folgewörter* · *nächste* · *Startwort* · *Tabelle*.

Bei der Textgenerierung gibt man ein _____ vor. Das Modell schaut in der n-Gramm-_____ nach, welche _____ auf das letzte Wort folgen können. Eines davon wird ausgewählt und an den Text _____. Es ist jetzt das neue letzte Wort. So wird immer wieder das _____ Wort vorhergesagt. Hat das letzte Wort keine Folgewörter, _____ der Text.

Arbeitsauftrag 41: Welche Strategie? ●○○

Ordne jeder Beschreibung die passende Sampling-Strategie zu: greedy, random oder top-k.

Nr.	Beschreibung	Strategie
1	Es wird immer das Folgewort mit der höchsten Wahrscheinlichkeit gewählt.	
2	Es wird unter allen Folgewörtern gelost. Häufige Folgewörter haben eine größere Chance.	
3	Nur die drei wahrscheinlichsten Folgewörter kommen in Frage. Unter ihnen wird gewichtet gelost.	
4	Mit demselben Startwort entsteht jedes Mal derselbe Text.	
5	Jedes Folgewort, das im Korpus vorkam, kann gewählt werden – auch das seltenste.	
6	Diese Strategie ist ein Kompromiss zwischen Vorhersagbarkeit und Abwechslung.	

Arbeitsauftrag 42: Greedy von Hand ●○○

Verwende den Tabellenausschnitt vom Anfang dieser Übungen.

1. Erzeuge mit **greedy** einen Text aus **8 Wörtern**, der mit „ich“ beginnt.
2. Beschreibe, was dir an dem Text auffällt.
3. Erkläre, warum greedy bei jedem Start genau denselben Text liefert.

Mein Text und meine Antworten:

Arbeitsauftrag 43: Wahrscheinlichkeiten bei random und top-k ●○○

In einem Korpus folgen auf „heute“ diese Wörter: „abend“ (6-mal), „nicht“ (3-mal), „schon“ (2-mal) und „mal“ (1-mal).

1. Berechne die Wahrscheinlichkeiten der Folgewörter bei **random**.
2. Berechne sie bei **top-2** und bei **top-3**.
3. Welches Wort wählt greedy?

Runde auf ganze Prozent.

Folgewort	Anzahl	random	top-2	top-3
abend	6			
nicht	3			
schon	2			
mal	1			

Arbeitsauftrag 44: Random mit Zufallszahlen ●●○

Mit random wird aus dem Tabellenausschnitt vom Anfang dieser Übungen ein Text erzeugt, der mit „ich“ beginnt. Für jeden Schritt wird eine Zufallszahl gezogen. Jedes Folgewort bekommt so viele Lose, wie es vorkommt, durchnummeriert in der Reihenfolge der Tabelle. Bei „ich“ heißt das: Lose 1 bis 5 sind „gehe“, 6 bis 8 „bin“, 9 und 10 „habe“.

Die Zufallszahlen sind der Reihe nach: **7, 4, 2, 1, 2, 4**.

1. Bestimme den erzeugten Text. Notiere bei jedem Schritt, welche Lose zu welchem Wort gehören.
2. Kann greedy diesen Text erzeugen? Begründe.
3. Mit welcher Wahrscheinlichkeit wählt random nach „ich“ das Wort „bin“?

Meine Schritte und Antworten:

Arbeitsauftrag 45: Welcher Text ist möglich? ●●○

Ein Bigramm-Modell wurde mit diesem Korpus trainiert:

> lisa fährt mit dem bus tim fährt mit dem rad lisa geht zu fuß

Entscheide für jeden Text, ob das Modell ihn mit **random** erzeugen kann. Wenn nicht, nenne ein Wortpaar im Text, das im Korpus nicht vorkommt.

Text	möglich?	fehlendes Wortpaar
tim fährt mit dem bus		
lisa fährt mit dem auto		
dem rad lisa geht zu fuß		
lisa geht mit dem bus		

Welcher der möglichen Texte kommt so **nicht** im Korpus vor?

Antwort zur letzten Frage:

Arbeitsauftrag 46: Bigramme oder Trigramme? ●●○

Gegeben ist der Korpus:

> ich trinke einen kakao ich esse einen apfel ich trinke einen tee

1. Bestimme im **Bigramm-Modell** die Folgewörter von „einen“ mit ihren Wahrscheinlichkeiten.
2. Bestimme im **Trigramm-Modell** die Folgewörter der Kontexte „trinke einen“ und „esse einen“ mit ihren Wahrscheinlichkeiten.
3. Kann das Bigramm-Modell den Text „ich trinke einen apfel“ erzeugen? Und das Trigramm-Modell?
4. Erkläre an diesem Beispiel, warum größere n kohärentere, aber weniger variantenreiche Texte erzeugen.

Meine Antworten:

Gymnasium Ottweiler

Arbeitsauftrag 47: Die Trainingsdaten entscheiden ●●●

Zwei Bigramm-Modelle wurden mit verschiedenen Korpora trainiert:

- **Verein:** „wir trainieren hart wir trainieren lange wir gewinnen das spiel wir trainieren hart“
- **Schule:** „wir lernen vokabeln wir lernen mathe wir schreiben eine arbeit wir lernen vokabeln“

1. Bestimme für beide Modelle den Text, den **greedy** ab „wir“ mit **5 Wörtern** erzeugt.
2. Erkläre, warum dasselbe Verfahren mit demselben Startwort ganz verschiedene Texte erzeugt.
3. An den Schul-Korpus wird mehrmals der Satz „wir schreiben eine arbeit“ angehängt. Wie oft mindestens, damit greedy nach „wir“ das Wort „schreiben“ wählt? Bei Gleichstand wählt greedy das Folgewort, das im Korpus zuerst vorkommt.
4. Prüfe deine Antworten mit der Funktion `generiere_text` aus diesem Abschnitt.

Meine Antworten zu 1 bis 3:

4.4.8 Das Wichtigste in Kürze

- Bei der **Textgenerierung (next word prediction)** wird ab einem Startwort immer wieder das nächste Wort bestimmt und angehängt. Hat ein Wort keine Folgewörter, endet der Text.
- Die **Sampling-Strategie** legt fest, wie das Folgewort ausgewählt wird:
  - **greedy:** immer das wahrscheinlichste → deterministisch, aber eintönig, oft Wiederholungen,
  - **random:** gewichtet zufällig (Lostrommel) → abwechslungsreich, manchmal unsinnig,
  - **top-k:** gewichtet zufällig unter den k wahrscheinlichsten → Kompromiss.
- Bei top-k werden die Wahrscheinlichkeiten der übrigen Wörter neu berechnet (z. B. $\frac{4}{7}$ statt $\frac{4}{10}$).
- **Längere n-Gramme** liefern kohärentere, aber weniger variantenreiche Texte. Lange Kontexte kommen im Korpus seltener vor.
- Der erzeugte Text hängt vollständig von den **Trainingsdaten** ab. Ein Sprachmodell ist deshalb nicht automatisch neutral.

Selbstcheck

- ☐ Ich kann beschreiben, wie ein n-Gramm-Modell Wort für Wort einen Text erzeugt.
- ☐ Ich kann die Strategien greedy, random und top-k benennen und ihre Vor- und Nachteile nennen.
- ☐ Ich kann mit Zufallszahlen von Hand ein Folgewort nach random oder top-k auslosen.
- ☐ Ich kann die Wahrscheinlichkeiten bei top-k neu berechnen und als Balkendiagramm darstellen.
- ☐ Ich kann erklären, warum die Liste `folgewoerter(...)` eine Lostrommel für gewichtetes Ziehen ist.
- ☐ Ich kann beschreiben, wie sich längere n-Gramme auf den erzeugten Text auswirken.
- ☐ Ich kann erläutern, warum der erzeugte Text von den Trainingsdaten abhängt.