מבוא לבינה מלאכותית

סמסטר חורף תשפ"ו

מטלה 5 – RL and MDP

תאריך הגשה: 24.01.2026 23:55

הנחיות

- שאלות בנושא מטלה זו יש לשאול דרך המודל, בפורום "מטלה 4".
- הוראות להגשת המטלה מופיעים בסוף מסמך זה.
- הקבצים הנדרשים להרצת הקוד הינם:
  - q_learning.py
  - common.py
  - value_iteration.py
  - plotting.py
  - cliff_walking_experiment.py
  - gamblers_experiment.py
  - requirements.txt
- העבודה להגשה בזוגות בלבד אלא אם כן המגישים קיבלו אישור להגשה שאינה בזוגות.
- פתרון המטלה שתגישו ייבדק מול שאר ההגשות על ידי תוכנת העתקות.
- **מי שימצא כי העתיק יכשל בקורס וכן יועבר לוועדת משמעת אוניברסיטאית**.
- הפרויקט נכתב וייבדק בשפת התכנות python, גרסה 3.10.
- **יש להתקין את החבילות על פי הקובץ requiremts.txt.**
  - על מנת לוודא שהסביבה מותקנת במלואה יש להריץ:

    pip install "gymnasium[toy-text]"
- מסמך זה בנוי באופן הבא: תיאור המטלה, תיאור הבעיה, שאלות המטלה והסבר על המימוש הקיים והסבר על הגשת המטלה.

תיאור המטלה:

בתרגיל זה אתם מתבקשים לממש את האלגוריתמים Q Learning ו-Value Iteration על מנת לפתור שתי בעיות.

בחלק הראשון של התרגיל עליכם לממש את האלגוריתם של Q Learning כדי לפתור את בעיית ה-cliff walking ולאחר מכן עליכם לממש את האלגוריתם Value Iteration על מנת שתוכלו לפתור את בעיית המהמר (עם שינוי).

בעיית ה-Cliff Walking:

מצרף לינק הסבר על הסביבה - https://gymnasium.farama.org/environments/toy_text/cliff_walking בבעיה זו המצבים והפעולות מבוטאים על ידי מספרים המצייגים את המצב (על פי המיימדים של המפה) וכיווני התנועה של הסוכן (0 עד 3 עבור הכיוונים בהם הסוכן מסוגל לנוע).

עליכם לממש את הפונקציות הבאות במחלקה QLearning:

```
def update(self, state: int, action: int, reward: float,
new_state: int):
```

הפונקציה מקבלת את המצב הנוכחי, ה-reward והמצב הבא ומעדכנת את ערך ה-Q table בהתאם על פי משוואת בלמן שנלמדה בכיתה.

```
def select_epsilon_greedy_action(self, state: int) -> int:
```

הפונקציה מקבלת את המצב הנוכחי ומחזירה את הפעולה לביצוע על פי אלגוריתם epsilon greedy.

שימו לב, אם ישנן מספר פעולות אופטימליות מומלץ לבצע tie breaking רנדומלי.

```
def train_episode(self, env: gym.Env) ->Tuple[float, int]:
```

הפונקציה מקבלת instance של הסביבה ומאמנת את הסוכן על ה-episode. הפונקציה מחזירה את ה-reward שהתקבל ב-episode **הפונקציה צריכה לרוץ עד אשר מגיעי למצב טרמינלי**. שימו לב, פעולה זו *לא* מקבלת מספר צעדים ורצה עד אשר הסוכן נפסל או עד אשר המשימה הצליחה והתקבל reward.

שימו לב, הפונקציה *env*.step(action) מבצעת צעד יחיד על הסביבה. ערכי החזרה של פונקציה זו הם:

```
new_state, reward, terminated, truncated, info =
env.step(action)
```

כאשר המשתנה truncated מציין האם האינטראקציה עם הסביבה הסתיימה בצורה פתאומית (כישלון המשימה או כישלון בסביבה וכו') והערך terminated אומר האם המשחק הסתיים (בהצלחה). על מנת לדעת האם ה-episode הסתיים, בדקו האם מתקיים כי truncated או terminated מקבלים ערך True.

```
def run_environment(self, env: gym.Env, num_episodes: int) ->
Tuple[List[float], List[int]]:
```

הפונקציה מריצה על הסביבה את מספר ה-episodes שנתון כפרמטר.

על הפונקציה להחזיר את ה-rewards עבור כל ה-episodes כמו גם את מספר הצעדים שבוצעו בכל episode.

בעיית ה-Gambler:

בשאלה זו עליכם לממש את אלגוריתם value iteration על גירסה חדשה של בעיית המהמר.

בעיית המהמר **המקורית** מוגדרת באופן הבא:

מהמר מקבל הזדמנות להמר על התוצאות של רצף הטלות מטבע. אם המטבע נוחת על צד ה"עץ", הוא זוכה במספר דולרים כמספר הסכום שהימר; אם נוחת על צד "פלי", הוא מפסיד את הסכום שהימר. המשחק מסתיים כאשר המהמר מנצח ומגיע למטרה של 100 דולר, או מפסיד כאשר נגמר לו הכסף.

בכל הטלה, המהמר חייב להחליט איזה חלק מההון שלו להמר, במספרים שלמים של דולרים. בעיה זו יכולה להיות מנוסחת כ MDP-סופי, אפיזודי ולא מוזל.

המצב הוא כמות הכסף הנוכחית של המהמר. $s \in \{1, 2, ..., 99\}$,

הפעולות הן סכומי ההימור. $a \in \{0, 1, ..., \min(s, 100 - s)\}$, התגמול הוא אפס בכל המעברים פרט לאלה שבהם המהמר מגיע למטרה שלו, שאז הוא +1.

עבור **הבעיה שלנו** אנחנו נניח כי במקום מטבע עם שני צדדים, למהמר יש 2 קוביות הוגנות כאשר התוצאות של ההטלה הן כדלהלן:

1. כאשר יוצא סכום הקטן מ 7 – המהמר מפסיד את ההימור שהוא שם. תזכורת: הסיכוי לכך הוא 5/12.
2. כאשר יוצא סכום השווה ל 7 – המהמר מרוויח את הסכום שהימר עליו. תזכורת: הסיכוי לכך הוא 1/6.
3. כאשר יוצא סכום הגדול מ 7 – המהמר מפסיד מחצית מההימור שהוא שם (מעוגל כלפי מעלה).
   תזכורת: הסיכוי לכך הוא 5/12.

עליכם לממש את הפעולות הבאות במחלקה ValueIteration:

```
def calculate_q_values(self, current_capital: int,
value_function: np.ndarray, rewards: np.ndarray) ->
np.ndarray:
```

הפונקציה מחשבת את ה- value בהינתן המצב הנוכחי, **<u>אוסף</u>** ה-rewards (מוגדרים להיות אפס לכל אחד מהמצבים פרט למצב האחרון שה-reward שלו 100).

שימו לב – עליכם להשתמש בהסתברויות הנ"ל על מנת לחשב את הערכי ה-Q[s,a] כאשר ה-s הינו המצב הנוכחי (current_capital) הניתן כקלט לפונקציה.

עליכם לממש גם את הפונקציה

```
def value_iteration_for_gamblers(self) -> Tuple[np.ndarray,
np.ndarray]:
```

אשר מריצה את תהליך ה- value iteration עד להתכנסות.

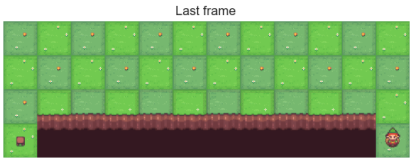
שימו לב, על הפעולה להחזיר את ה-values כמו גם את ה-policy האופטימלי שחושב. מצופה להחזיר באופן הבא:

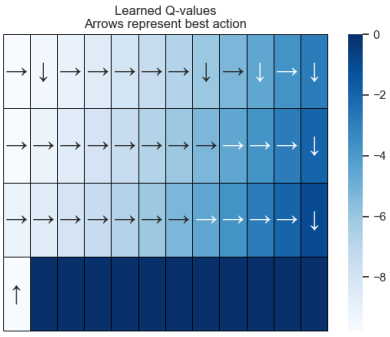
return policy, V

שימו לב, ישנם 101 ערכים אפשריים ל-rewards – 0 עד 100 *כולל* .

בדיקות נכונות:

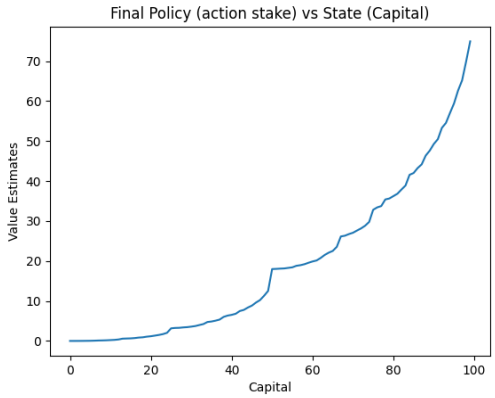
בקבצים gamblers_experiment.py ו-cliff_walking_experiment.py מצויים תרחישים אשר יבדקו את נכונות האלגוריתם שאתם כותבים.





עבור הרצת הסוכן על בעיית ה-clif walking מצופה מכם לקבל תוצאה כזו לאחר סיום הריצה.

עבור בעיית המהמר עם הפרמטרים שבקובץ מצופה מכן לקבל גרף כדלהלן:



הגשת המטלה:

המטלה תיבדק ב-VPL, הקבצים שעליכם להגיש הם value_iteration.py ו-q_learning.py. הקוד שלכם יעבור בדיקות אוטומטיות על תרחישי בדיקה שמדמים את התרחישים שאתם מריצים פה.

הסביבה מריצה את הקוד שלכם על מספר תרחישים ולכן הריצה אמורה לקחת מספר שניות.

בהצלחה 😄