קריית החינוך
פארק המדע
בית לערכים
למצוינות ולחדשנות

תכנון דינמי – בעזרת טבלה
משוואות בלמן

ריצארד בלמן 1984 - 1920

תכנון דינמי ומשוואות בלמן

- סיימנו את השיעור הקודם בהצבת המטרה – מציאת אלגוריתם לחישוב המדיניות המיטבית P^*.
- המתמטיקאי ריצ'ארד בלמן פיתוח אלגוריתם לפתרון הבעיה באמצעות תכנון דינמי – שהינו תכנון באמצעות טבלת מצבים (או מבנה נתונים אחר המחזיק את כל המצבים האפשריים בסביבה).
- פתרון זה מתאים רק לסביבה בה מרחב המצבים הוא קטן יחסית וניתן לשמירה בזכרון המחשב (בטבלה או במבנה נתונים אחר).
- במקרים בהם מספר המצבים גדול יותר אנחנו נצטרך לשדרג את האלגוריתם של בלמן.

משוואות בלמן בסביבה דטרמיניסטית

- משוואת הערך V(s) בסביבה דטרמיניסטית היא:
- $V_p(s_0) = G_{s_0} = \ R_0 + \gamma R_1 + \gamma^2 R_2 + \gamma^3 R_3 + \cdots + \gamma^n R_n$
- כאשר הצעד $s_t \rightarrow s_{t+1}$ נעשה בהתאם למדיניות שלנו (הפוליסי - P).
- ניתן לכתוב את המשוואה גם בצורה הבאה:
- $V_p(s_0) = \ R_0 + \gamma(R_1 + \gamma^1 R_2 + \gamma^2 R_3 \ldots + \gamma^{n-1} R_n)$
- בהתאם לתכונת מרקוב אין חשיבות למצבים הקודמים בהם ביקרנו, לכן משוואת הערך של $V(s_1)$ היא:
- $V_p(s_1) = R_1 + \gamma^1 R_2 + \gamma^2 R_3 + \cdots + \gamma^{n-1} R_n$
- שילוב שתי המשוואות לעיל מביא אותנו למשוואת בלמן:

$$\boldsymbol{V_p(s) = R + \gamma V_p(s_1)} \qquad \boldsymbol{V_p(s) = R + \gamma V_p(s')}$$

משוואות בלמן בסביבה דטרמיניסטית

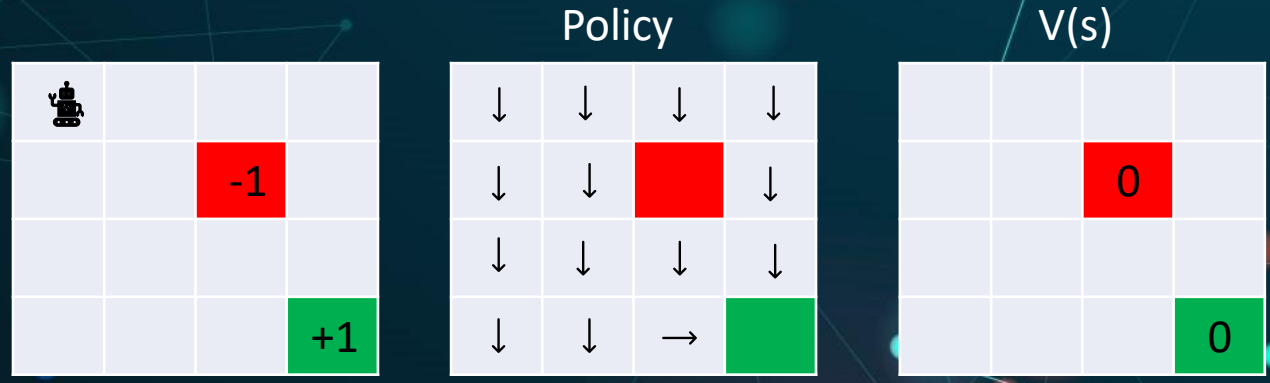
- פיתוח משוואת Q(s, a) בסביבה דטרמינסטית מביא לתוצאה דומה:
- $Q_p(s,a) = R + \gamma Q_p(s',a')$

- משוואות בלמן בסביבה אקראית (סטוכסטית)
  - בסביבה בה המעבר בין מצבים הוא הסתברותי (כלומר פעולה a במצב s לא תיתן לנו תמיד את אותו מצב s', אלא יתכנו מספר מצבים בהסתברויות שונות), המשוואות מוסיפות חישוב הסתברותי.
  - אנחנו לא נלמד נושא זה רק נביא את המשוואה המלאה (π – הפוליסי שלנו):

- $V_\pi(s) = \sum_a \pi(a|s) \sum_{s',r'} p(s',r|s,a)[r + \gamma V_\pi(s')]$

הדגמה – תכנון דינמי למציאת מדיניות מיטבית

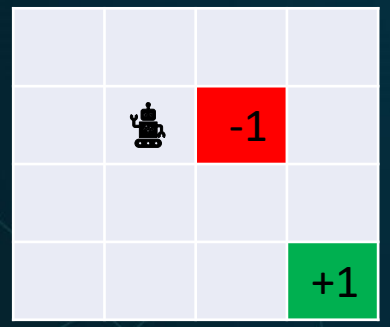
- נדגים באמצעות המבוך הפשוט שלנו. המטרה של הסוכן להגיע למשבצת הירוקה.
- המדיניות ההתחלתית של הסוכן $P(s)$ היא ללכת למטה, אלא אם כן צמוד למשבצת ירוקה (רואה את הירוק) ואז הולך לכיוונו.
- התגמול הוא 0 בכל צעד רגיל; 1 בכניסה למשבצת הירוקה; -1 בכניסה למשבצת האדומה; הפקטור שלנו הוא $\gamma = 0.9$
- נחשב את פונקציית הערך באמצעות טבלה:



חישוב פונקציית הערך למדיניות נתונה

$$V_p(s) = r + \gamma V_p(s')$$

האלגוריתם של בלמן מציע לבצע חישוב חוזר ונשנה של המשוואה על הטבלה עד אשר לא יהיו יותר שינויים בטבלה (או שהשינויים יהיו קטנים מאוד):



↓	↓	↓	↓
↓	↓		↓
↓	↓	↓	↓
↓	↓	→	

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

0	0	-1	0
0	0	0	0
0	0	0	1
0	0	1	0

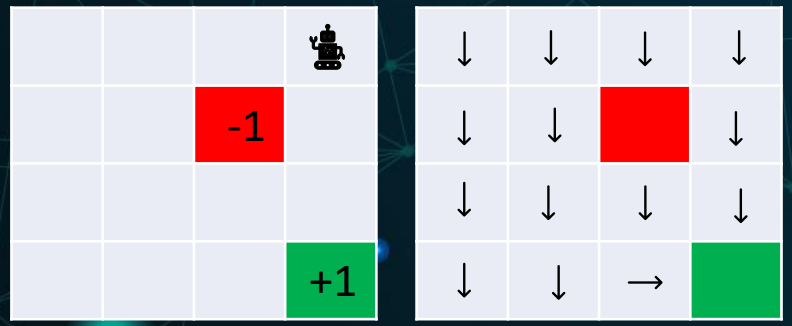
0	0	-1	0
0	0	0	0.9
0	0	0.9	1
0	0	1	0

0	0	-1	0.81
0	0	0	0.9
0	0	0.9	1
0	0	1	0

Policy Evaluation

- חישוב פונקציית הערך למדיניות נתונה נקרא: Policy Evaluation
- אנחנו מסיימים את האיטרציות כאשר תיקון הטבלה הוא קטן מסף שנקבע.
- בלמן הוכיח כי אם $\gamma < 1$ האלגוריתם מתכנס והתיקון שואף לאפס ובסוף ייעצר.

- $V(s) = r + \gamma V(s')$
- $P(s) = s'$



Policy

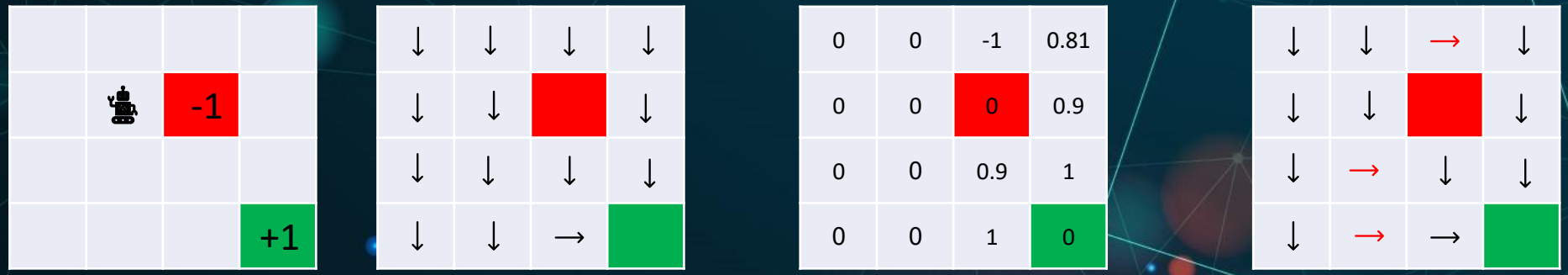

```python
# Iterative Policy Evaluation for Deterministic environment

# Initialization
for all s in S:
    V(s) = random(float) except V(terminal) = 0

def Policy_Evaluation (P, V):

    accuracy = 0.0001 (small number)
    acc = maxNumber

    while acc > accuracy:
        acc = 0
        for s in all states:
            old_value = V(s)
            action = P(s)
            s', r = environment(state, action)
            new_value = r + gamma * V(s')
            V(s) = new_value
            acc = max(acc, abs(old_value - new_value))
```

שיפור פונקציית המדיניות

- אם נתונה לנו פונקציית הערך של מדיניות מסוימת אנחנו יכולים לשפר את המדיניות באמצעות פונקציית הערך לפי הנוסחה הבאה:

$$P(s) = argmax_a(r + \gamma Q(s,a)) \quad \text{או} \quad P'(s) = \max_a(r + \gamma V(s'))$$

- במילים אחרות, לכל מצב s אנחנו בוחרים את הפעולה שתביא אותנו לתגמול המירבי.
- Policy improvement מצריך איטרציה אחת בלבד.



Policy Improvement

- פסאודו קוד של אלגוריתם שיפור פונקציית המדיניות.
- מחזירים True אם לא היו שינויים. הפונקציה יציבה.

```
# policy improvement for Deterministic environment
# return true if policy stable

def Policy improvement (P, V):
    stable = true
    for all s in States:
        actions = legal_actions(s)
        v_max = -inf
        for all action in actions:
            s', r = environment(s, action)
            if v_max < r + gamma * V(s'):
                v_max = r + gamma * V(s')
                best_action = action
        if P(s) != best_action:
            P(s) = best_action
            stable = false
    return stable
```

שוב Policy Evaluation

- לאחר שעדכנו את המדיניות שלנו עלינו לבצע שוב עדכון של טבלת הערכים Policy Evaluation:

		-1	
			+1

↓	↓	→	↓
↓	↓		↓
↓	→	↓	↓
↓	→	→	

0	0	0	0
0	0	0	0
0	0	0	1
0	0	1	0

0	0	0	0
0	0	0	0.9
0	0	0.9	1
0	0.9	1	0

0	0	0	0.81
0	0	0	0.9
0	0.81	0.9	1
0	0.9	1	0

0	0	0.73	0.81
0	0.73	0	0.9
0	0.81	0.9	1
0	0.9	1	0

0	0.66	0.73	0.81
0	0.73	0	0.9
0	0.81	0.9	1
0	0.9	1	0

Policy Evaluation + Policy Improvement

- נעדכן שוב את המדיניות שלנו בהתאם לטבלת הערכים, לאחר מכן, נעדכן שוב את טבלת הערכים.

		-1	
			+1

↓	↓	→	↓
↓	↓		↓
↓	→	↓	↓
↓	→	→	

0	0.66	0.73	0.81
0	0.73	0	0.9
0	0.81	0.9	1
0	0.9	1	0

→	↓	→	↓
→	↓		↓
→	→	↓	↓
→	→	→	

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

0	0	0	0
0	0	0	0
0	0	0	1
0	0	1	0

0	0	0	0
0	0	0	0.9
0	0	0.9	1
0	0.9	1	0

0	0	0	0.81
0	0	0	0.9
0	0.81	0.9	1
0.81	0.9	1	0

0	0	0.73	0.81
0	0.73	0	0.9
0.73	0.81	0.9	1
0.81	0.9	1	0

0	0.66	0.73	0.81
0.66	0.73	0	0.9
0.73	0.81	0.9	1
0.81	0.9	1	0

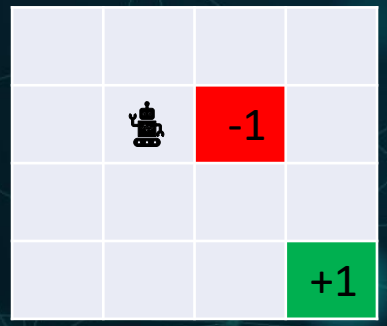
0.59	0.66	0.73	0.81
0.66	0.73	0	0.9
0.73	0.81	0.9	1
0.81	0.9	1	0

Policy Iteration

- נמשיך לבצע לסירוגין עדכון של טבלת הערכים ושל המדיניות

Policy evaluation + policy improvement = policy Iteration

- עד אשר נגיע למצב שבו המדיניות כבר לא משתנה (יציבה).
- בשלב זה – סיימנו ומצאנו את המדיניות המיטבית.



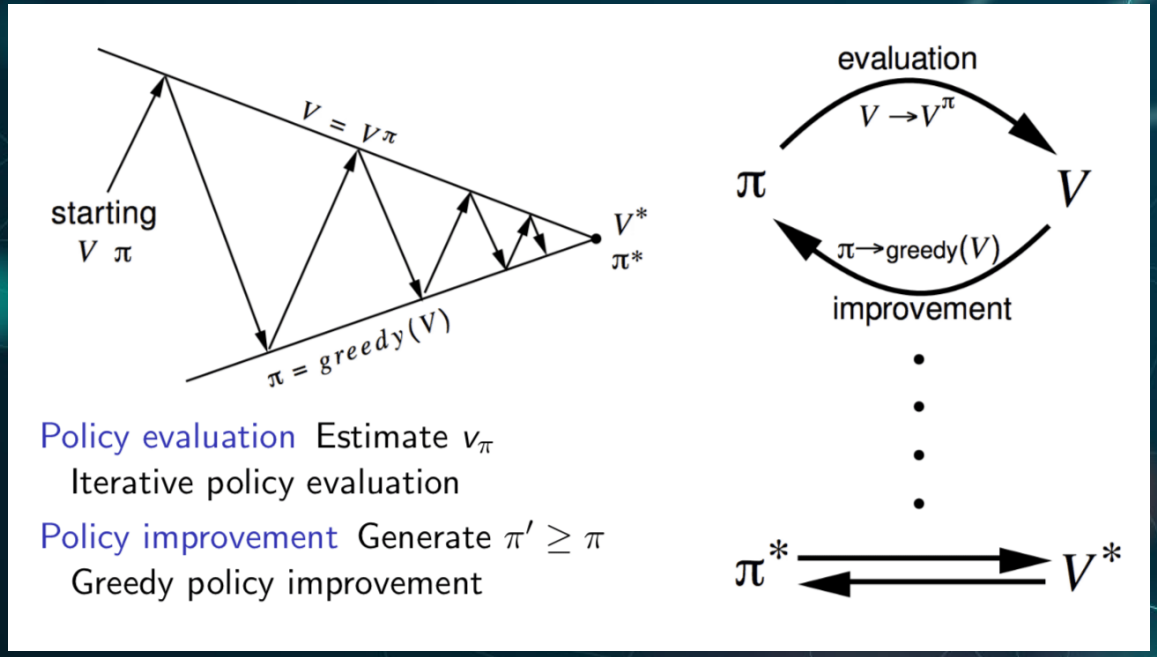
↓	↓	→	↓
↓	↓		↓
↓	→	↓	↓
↓	→	→	

0.59	0.66	0.73	0.81
0.66	0.73	0	0.9
0.73	0.81	0.9	1
0.81	0.9	1	0

↓	↓	→	↓
↓	↓		↓
↓	→	↓	↓
↓	→	→	

Policy Iteration

- Policy iteratin
- אלגוריתם למציאת המדיניות המיטבית.
- אנו מבצעים לסירוגין:
  - Policy Evaluation – לעדכון טבלת הערכים לפי המדיניות.
  - Policy improvement – לעדכון המדיניות לפי טבלת הערכים החדשה.
- בלמן הוכיח כי בסופו של דבר אנחנו נתכנס למדיניות המיטבית.



Policy Iteration

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Loop:
   $\Delta \leftarrow 0$
   Loop for each $s \in \mathcal{S}$:
   $v \leftarrow V(s)$
   $V(s) \leftarrow \sum_{s',r} p(s', r \mid s, \pi(s)) \left[r + \gamma V(s') \right]$
   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   old-action $\leftarrow \pi(s)$
   $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma V(s') \right]$
   If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

```
# Policy iteration for Deterministic environment

def Policy iteration ():

    #1. Initialization
    for all s in S:
        V(s) = random(float) except V(terminal) = 0
        P(s) = random(actions)
    policy_stable = False

    2# loop
    while not policy_stable:

        Policy_Evaluation (P, V)

        policy_stable = Policy Improvement (P, V)
```

תרגיל

- נתון לכם ממשק של Grid World לפי מודל סוכן-סביבה.
  - https://github.com/MarkmanGilad/GridWorld_2
  - State – מיקום הרובוט (row, col).
  - הפונקציה move בסביבה מחזירה שני ערכים: new_state, reward.
  - בקובץ Agent יש שני סוכנים: סוכן רנדומלי מוכן לעבודה. סוכן AI להשלמת הקוד

- אתחלו את המדיניות של הסוכן AI במדיניות אקראית לפי בחירתכם.
- כיתבו את הפונקציות הבאות של הסוכן AI והריצו אותם:
  - def policy_eval ()
  - def Policy_improv ()
  - def Policy_iteration ()