



קריית החינוך
פארק המדע
בית לערכים
למצוינות ולחדשנות

למידת חיזוק עמוקה

DQN



החיסרון בתכנות דינמי

- כל הפתרונות שהצגנו עד היום התבססו על מבנה נתונים שהחזיק את הערכים (Value-table, Q-table) וכלל ערכים כמספר המצבים בסביבה.
- הבעיה היא שבמקרים רבים מספר המצבים האפשריים הוא גדול כל כך שאין כל אפשרות מעשית לשמור אותו בזיכרון המחשב, ולעיתים מספר המצבים הוא אין סופי.
- מעבר לכך, גם אם ניתן היה לשמור את כל המצבים במבנה נתונים במחשב – הזמן שיערוך לעדכן אותם הוא עצום ולא מעשי.
- מספר המצבים במשחק דמקה הוא 10^{20}
- מספר המצבים במשחק שחמט הוא כ- 10^{40}
- מספר המצבים של משחק GO הוא 10^{170}
- מספר המצבים של רכב אוטונומי הוא אין סופי.

הפתרון – שימוש ברשת נוירונים

- למידת חיזוק עמוקה עושה שימוש ברשת נוירונים המחליפה את טבלאות הערכים.

- טבלת ערכים היא למעשה פונקציה: $Q(s, a) = \text{value}$; $V(s) = \text{value}$.

- באלגוריתמים שלמדנו אנחנו מעדכנים את הערכים של הטבלאות כל פעם ב"כיוון הטעות":

$$V(s) = V(s) + \alpha(R + V(s') - V(s))$$

- אנחנו מתחילים בערכים אקראיים ומבצעים עדכונים חוזרים ונשנים עד שמגיעים לתוצאה הנכונה, כך שהטעות היא קטנה ביותר.

- הדבר דומה מאוד לעדכון הפרמטרים ברשת נוירונים. גם שם אנחנו מתחילים בערכים אקראיים ובאמצעות עדכונים חוזרים ונשנים מעדכנים את המשקלים לכיוון מינימום הטעות.

רשת נוירונים בלמידת חיזוק

• באימון רשת נוירונים (supervised learning) אנחנו מקבלים זוגות של נתונים:

• הקלט X – התוצאה (המטרה - Y).

• מבצעים חלחול קדימה על ידי חישוב התוצאה המשוערת: $y_{hat} = Model(X)$

• מחשבים את הטעות $loss = (y_{hat} - y)^2$

• מבצעים חילחול לאחור לצורך חישוב הנגזרות של פונקציית הטעות.

• ומעדכנים את הפרמטרים של הרשת בהתאם לטעות על פי אלגוריתם SGD.

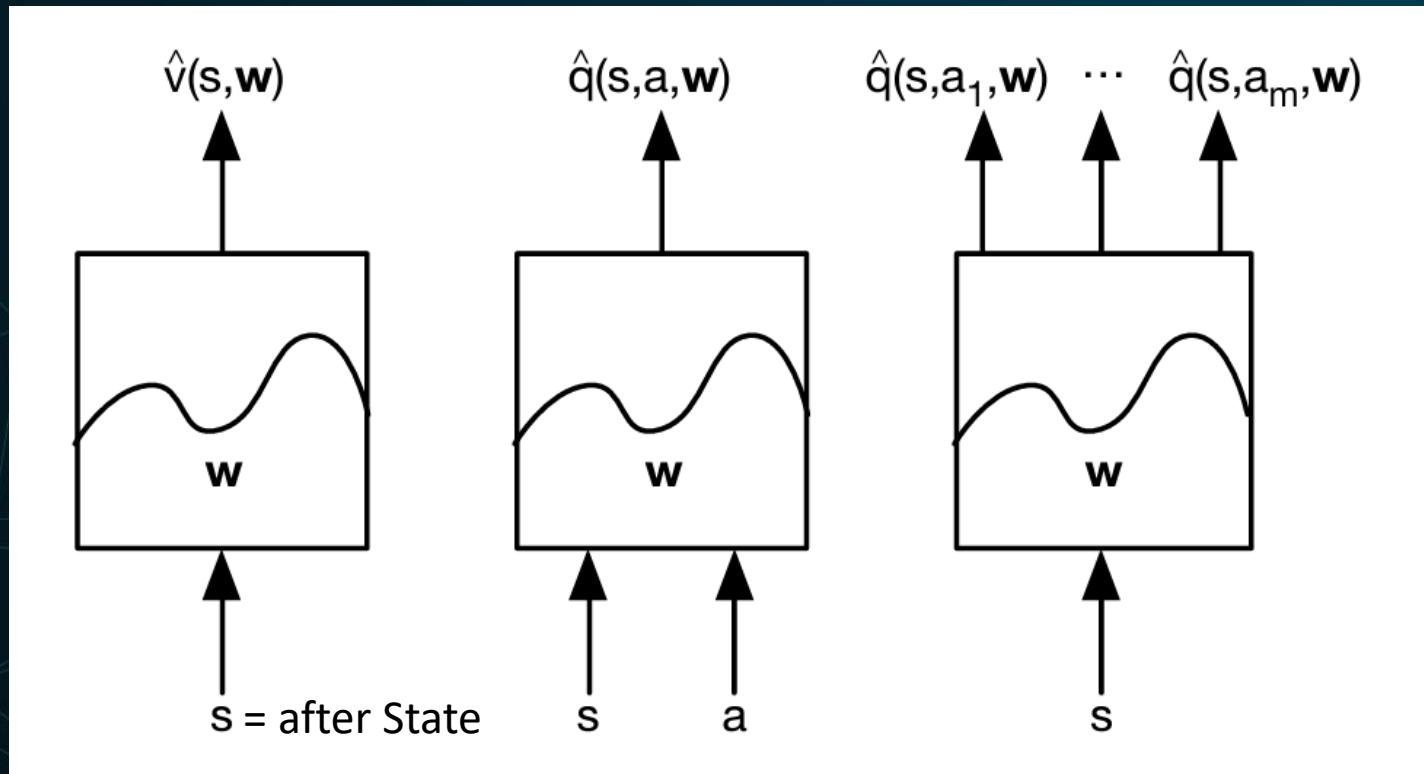
• בלמידת חיזוק אנחנו נשתמש באותו אלגוריתם ונחליף את V ברשת נוירונים $\hat{V}(s, w)$:

$$\hat{V}(s, w) = \hat{V}(s, w) + \alpha \left(R + \hat{V}(s', w) - \hat{V}(s, w) \right)$$

(s, a)	S	• הקלט X :
$\hat{Q}(s, a)$	$\hat{V}(S)$	• התוצאה המשוערת $:Y_{hat}$
$R + \hat{Q}(s', a')$	$R + \hat{V}(S')$	• התוצאה (מטרה):

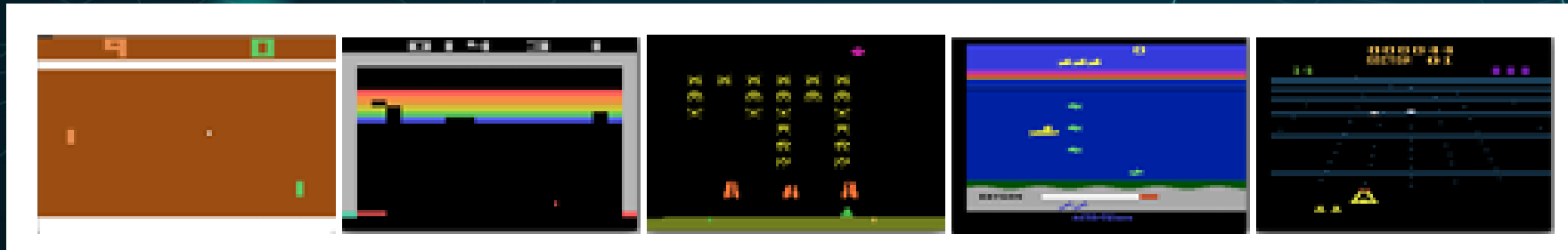
סוגי רשתות נוירונים

- אנחנו נשתמש באלגוריתם Q-Learning ונחליף את הטבלה ברשת נוירונים (שהינה פונקציה).
- במקום לעדכן את V או Q אנחנו נעדכן את הפרמטרים של רשת הנוירונים.
- סוגים של רשת נוירונים:



האלגוריתם DQN

- האלגוריתם אותו נלמד פותח על ידי צוות חוקרים ב Google DeepMind ופורסם בשני מאמרים בשנת 2013 ו-2015.
- באותם מאמרים הדגימו החוקרים כיצד ניתן ללמד סוכן AI לשחק משחקי ATARI כשהקלט הן התמונות בלבד של מסך המשחק (כמו בן אדם).
- [Playing Atari with Deep Reinforcement Learning, V. Mnih \(NIPS-DLW 2013\)](#)
- [Human-level Control Through Deep Reinforcement Learning, V. Mnih et al. \(Nature 2015\)](#)



תזכורת Q-learning

- נפתח בשלבים את האלגוריתם המשתמש בטבלת Q.
- הסוכן נמצא במצב S יבחר את הפעולה A בהתאם לטבלת Q ולמדיניות ϵ -greedy.
- הסוכן ידגום את הסביבה ויקבל את המצב הבא: S', A, R, S'
- בחירת A' תעשה לפי טבלת Q בלבד ללא ϵ -greedy. כלומר נבחר את הפעולה שתביא את הערך המירבי $\max_a Q(s', a)$
- אחרי כל צעד ישתמש הסוכן בנתונים כדי לעדכן את טבלת Q באמצעות הנוסחה:

$$Q(s, a) = Q(s, a) + \alpha(R + \gamma \cdot \max_a Q(s', a) - Q(s, a))$$

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Initialize $Q(s, a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\text{terminal-state}, \cdot) = 0$

Repeat (for each episode):

Initialize S

Repeat (for each step of episode):

Choose A from S using policy derived from Q (e.g., ϵ -greedy)

Take action A , observe R, S'

$Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$

$S \leftarrow S'$

until S is terminal

פיתוח האלגוריתם DQN

- כזכור, רשת נוירונים היא למעשה פונקציה המקבלת קלט ומחזירה פלט, תוך שימוש בפרמטרים W , אותם אנחנו מעדכנים תוך כדי האימון.

- נסמן פונקציית רשת הנוירונים המחליפה את Q :

$$Q(s, a) \approx \hat{Q}(s, a, w)$$

- חישוב ערך Q הוא למעשה פעולת ה $forward$ שאנחנו מבצעים ברשת הנוירונים.

- נחליף בנוסחאות בלמן את הטבלה בפונקציה של רשת נוירונים.

פונקציית הטעות loss

• קיבלנו את הנוסחאות הבאות לעדכון :

$$Q(s, a) = Q(s, a) + \alpha (R + \gamma \cdot \max_a Q(s', a) - Q(s, a))$$

$$\hat{Q}(s, a, w) = \hat{Q}(s, a, w) + \alpha (R + \gamma \cdot \max_a \hat{Q}(s', a, w) - \hat{Q}(s, a, w))$$

• לערך בסוגריים קראנו TD-error כיוון שהוא מחשב את הטעות בערכים בין הערך בטבלה (בפונקציה) לבין הערך "הנכון" יותר.

• לצורך עדכון רשת הנוירונים נעשה שימוש בטעות יחד עם פונקציית הטעות Loss=MSELoss בדרך הבאה:

$$loss = \left(R + \gamma \cdot \max_a \hat{Q}(s', a, w) - \hat{Q}(s, a, w) \right)^2$$

Deep Q-learning (start Algorithm)

Initialize $\hat{Q}(s,a,w)$ with arbitrary W .

For epoch in epochs (for each game):

Initialize s

While s is not Terminal:

Choose A From S using \hat{Q} and ϵ -greedy

Interact with environment and get S, A, R, S'

Calculate $\hat{Q}(S, A, W)$ (forward)

Calculate loss with MSE Losses :

$$\text{If } s' \text{ is Terminal: } \text{loss} = \left(R - \hat{Q}(S, A, W) \right)^2$$

$$\text{else: } \text{loss} = \left(R + \gamma \cdot \max_a \hat{Q}(S', A, W) - \hat{Q}(S, A, W) \right)^2$$

Calculate gradients (backwards): $\text{loss.backward}()$

Update W : $\text{Optim.SGD.step}()$

$S = S'$

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Initialize $Q(s, a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\text{terminal-state}, \cdot) = 0$

Repeat (for each episode):

Initialize S

Repeat (for each step of episode):

Choose A from S using policy derived from Q (e.g., ϵ -greedy)

Take action A , observe R, S'

$$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$$

$S \leftarrow S'$

until S is terminal

מצב סופי – Terminal State

- הערך של מצב סופי הוא תמיד אפס (אין תגמולים עתידיים).
- לכן, בטבלאות V או Q דאגנו כי ערכי המצב הסופי תמיד יהיו אפס.
- ברשתות נוירונים אנחנו לא יכולים לקבוע מה יהיה הערך של המצב הסופי, ובוודאי לאחר עדכון הפרמטרים אנחנו לא יכולים לדעת מה יהיה הערך של המצבים הסופיים.
- הפתרון: לבצע חישוב נפרד בקוד למצבים סופיים:

Calculate loss with MSE Loss :

$$\text{If } s' \text{ is Terminal: } \text{loss} = \left(R - \hat{Q}(S, A, W) \right)^2$$
$$\text{else: } \text{loss} = \left(R + \gamma \cdot \max_a \hat{Q}(S', A, W) - \hat{Q}(S, A, W) \right)^2$$

Replay Buffer

- אחת הבעיות המרכזיות ביישום רשת נוירונים באלגוריתם TD המשתמש ב bootstrapping הוא העובדה שהדגימות שלנו מהסביבה תלויות אחת בשניה.
- בעת האימון של הסוכן אנחנו יוצרים שרשרת של מצבים ופעולות אשר אחד נובע מהשני.
- בנוסף, הדגימות רלוונטיות למשחק ולמדיניות מסויימת ואינה אקראית.
- על מנת שלמידת מכונה תצליח על הקלט שלנו למלא אחר התנאי:

i.i.d - Independent and Identically Distributed

הפתרון: שימוש במבנה נתונים (replay buffer) השומר את הדגימות אותן אנחנו מבצעים מהסביבה. עדכון רשת הנוירונים יעשה באמצעות batches אקראיים מהדגימות שלנו ולא מייד לאחר קבלת כל דגימה.

מבנה הנתונים יהיה מוגבל בגודלו כך שיישמרו בו תמיד הנתונים האחרונים שנדגמו בהתאם למדיניות המעודכנת שלנו.

Deep Q-learning (replay buffer)

Initialize $\hat{Q}(s,a,w)$ with arbitrary W .

Initialize replay-buffer RB with size N .

For epoch in epochs (for each game):

Initialize s

While s is not Terminal:

Choose A From S using Q and ϵ -greedy

Interact with environment and get S, A, R, S'

Store transition in RB

sample random minibatch from RB

for each sample:

Calculate $\hat{Q}(S, A, W)$ - forward

Calculate loss with MSELoss :

$$\text{If } s' \text{ is Terminal: } \text{loss} = \left(R - \hat{Q}(S, A, W) \right)^2$$
$$\text{else: } \text{loss} = \left(R + \gamma \cdot \max_a \hat{Q}(S', A, W) - \hat{Q}(S, A, W) \right)^2$$

Calculate gradients (backwards): $\text{loss.backward}()$

Update W : $\text{Optim.SGD.step}()$

$S = S'$

Non-stationarity of targets

- בעיה נוספת נעוצה בעבודה שבאימון רשת נוירונים ערכי המטרה (target) הם תמיד קבועים ולא משתנים.
- האם תמונה מסויימת היא חתול או לא חתול אינה משתנה במהלך האימון.
- לעומת זאת, בלמידת חיזוק אנחנו עושים שימוש ב bootstrapping כך שהמטרות שלנו משתנות תוך כדי הלמידה.
- ערך המטרה שלנו $R + \gamma \cdot \max_a \hat{Q}(S', A, W)$ משתנה בכל פעם שאנחנו משנים את הפרמטרים של הרשת.
- הפתרון: שימוש בשני רשתות נוירונים בעלי אותו מבנה אך עם פרמטרים שונים:
 - רשת אחת משמשת לבחירת הצעדים $Q(S, A, W)$ ואותה אנחנו מעדכנים בכל איטרציה.
 - רשת שניה משמשת לבחירת המטרות $R + \gamma \cdot \max_a \hat{Q}(S', A, W^-)$ והיא קבועה למשך C צעדים ולא מתעדכנת. לאחר C צעדים מעדכנים את הרשת $W^- \leftarrow W$.

Deep Q-learning (DQN)

Initialize $Q(s,a,w)$ with arbitrary w .

Initialize $\hat{Q}(s,a,w^-)$ with arbitrary w^- .

Initialize replay-buffer RB with size N.

For epoch in epochs (for each game):

Initialize s

While s is not Terminal:

Choose A From S using Q and e-greedy

Interact with environment and get s, a, r, s'

Store transition in RB

$s = s'$

sample random minibatch from RB

Calculate $Q(s, a, w)$ - forward

Calculate loss with MSELoss :

$$\text{If } s' \text{ is Terminal: } \text{loss} = (r - Q(s, a, w))^2$$
$$\text{else: } \text{loss} = (r + \gamma \cdot \max_a \hat{Q}(s', a, w^-) - Q(s, a, w))^2$$

Calculate gradients (backwards): $\text{loss.backward}()$

Update W : $\text{Optim.SGD.step}()$

Every C epochs $w^- \leftarrow w$

DQN – with Replay

Algorithm 1 : Double Q-learning (Hasselt et al., 2015)

Initialize primary network Q_θ , target network $Q_{\theta'}$, replay buffer \mathcal{D} , $\tau \ll 1$

for each iteration **do**

for each environment step **do**

 Observe state s_t and select $a_t \sim \pi(a_t, s_t)$

 Execute a_t and observe next state s_{t+1} and reward $r_t = R(s_t, a_t)$

 Store (s_t, a_t, r_t, s_{t+1}) in replay buffer \mathcal{D}

for each update step **do**

 sample $e_t = (s_t, a_t, r_t, s_{t+1}) \sim \mathcal{D}$

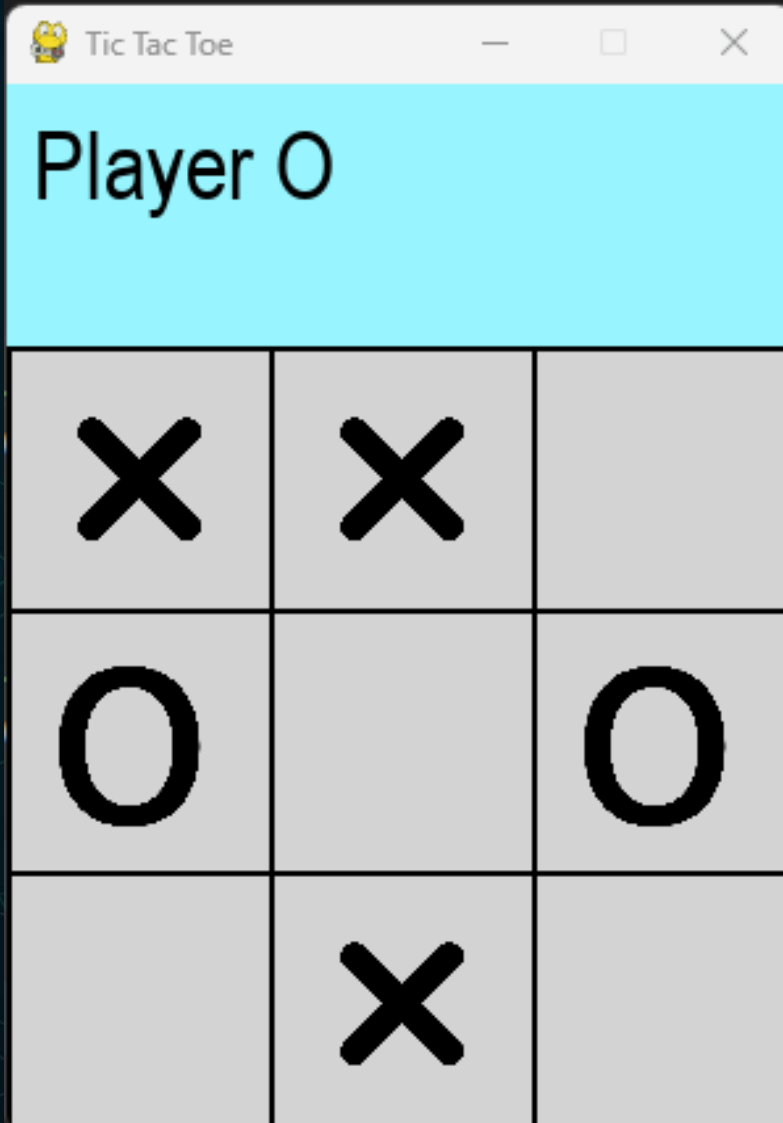
 Compute target Q value:

$$Q^*(s_t, a_t) \approx r_t + \gamma Q_{\theta'}(s_{t+1}, \operatorname{argmax}_{a'} Q_{\theta'}(s_{t+1}, a'))$$

 Perform gradient descent step on $(Q^*(s_t, a_t) - Q_\theta(s_t, a_t))^2$

 Update target network parameters:

$$\theta' \leftarrow \tau * \theta + (1 - \tau) * \theta'$$



הדגמה משחק איקס עיגול

DQN

Deep Q-learning
Neural Network