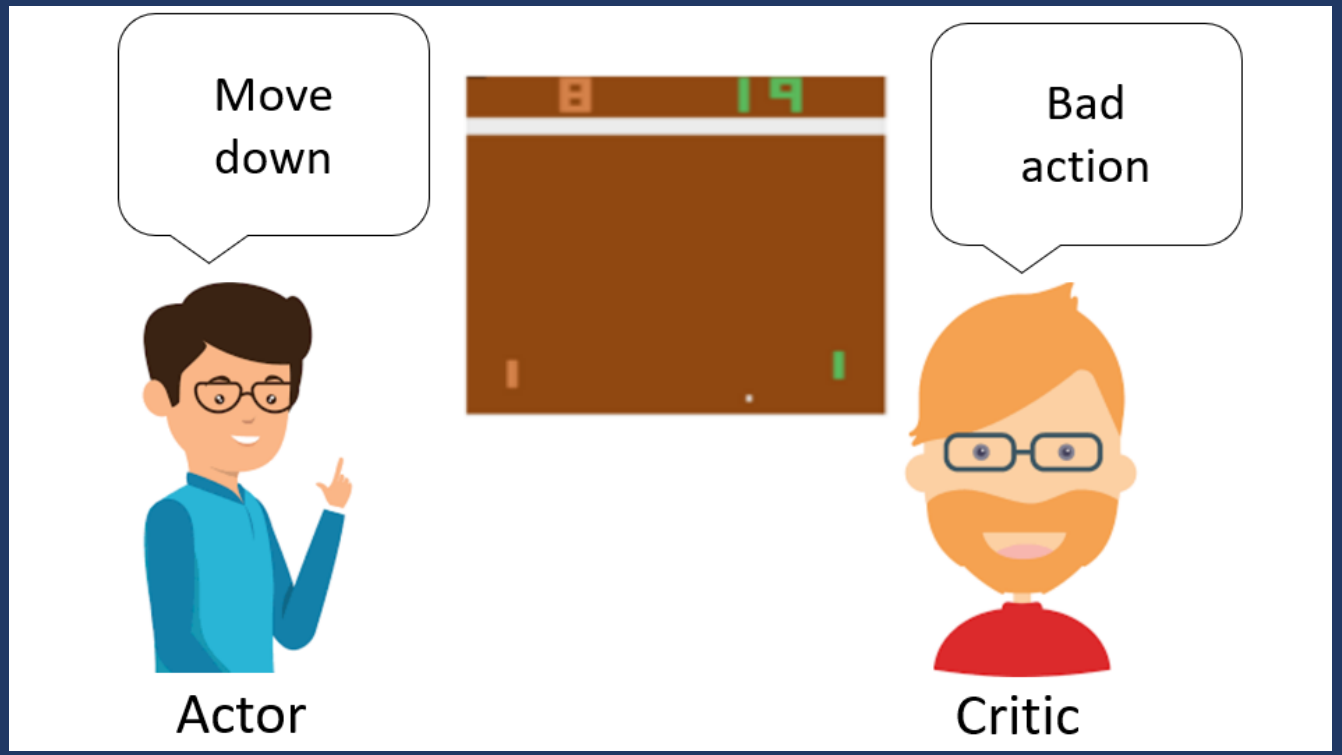
Actor Critic Method

קריית החינוך
פארק המדע
בית לערכים
למצוינות ולחדשנות



גלעד מרקמן

קישור ל GitHub

Actor Critic Method

- אלגוריתם Actor-Critic Method הינו שילוב של שני אלגוריתמים:
  - Policy Gradient Methods – REINFORCE
  - Value Methods – TD.

- המודל מבוסס על שתי רשתות נוירונים:
  - Actor – רשת לחישוב המדיניות באמצעות Policy Gradient Method
  - Critic – רשת לחישוב ערך המצב V(s) המבוססת על TD-Temporal Difference Methode.

- בפועל ניתן לבנות את המודל באמצעות שתי רשתות או באמצעות רשת אחת שמתפצלת בשכבה האחרונה לשתי רשתות.

חזרה - REINFORCE MC

- באלגוריתם זה בנינו רשת נוירונים שתחשב את המדיניות $\pi(s)$.
- חישבנו באמצעות מונטה קרלו את סכום התגמולים במשחקון G_t
- והשתמשנו בנוסחאות הבאות לעדכון הפרמטרים:

$$loss = G_t * \ln \pi_\theta(a|s)$$
$$\nabla loss = G_t * \nabla \ln \pi_\theta(a|s)$$

- באמצעות SGA – Stochastic Gradient Ascent עדכנו את הפרמטרים θ שיביאו למקסימום של תועלת:

$$\theta_{t+1} = \theta_t + \alpha * \nabla loss(\theta)$$

חזרה - REINFORCE MC

- בלמידת מכונה מקובל להשתמש בפונקציות למציאת מינימום ולא מקסימום, לכן, הכפלנו את נוסחת הטעות במינוס, וכך נוכל להשתמש ב SGD למציאת מינימום הטעות:

$$loss = -G_t * \ln \pi_\theta(a|s)$$
$$\theta_{t+1} = \theta_t - \alpha * \nabla loss(\theta)$$

Advantage

- Advantage מוגדר כיתרון (או חיסרון) בבחירת פעולה מסוימת במצב בו אנו נמצאים.
- הגדרה:

$$A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s)$$

- Q(s,a) – זהו הערך שיקבל הסוכן אם יבחר בפעולה a וימשיך בהתאם למדיניות שלו.
- V(s,a) – זה הערך של המצב בהתחשב בכל הפעולות האפשריות באותו מצב. ממוצע (תוחלת) של כל הערכים שיתקבלו בכל אחד מהפעולות.
- אם $Q(s,a) > v(s)$ סימן שהפעולה שבחרנו בה היא פעולה טובה יותר מהפעולה הממוצעת, ולהפך. לכן, הערך נקרא "היתרון".

Advantage

- נוסחת ה advantage היא: $A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s)$
- ידוע כי: $Q(s,a) = r + \gamma V(s')$
- ונקבל: $A_\pi(s,a) = r + \gamma V(s') - V_\pi(s)$
- ובכתיב שונה: $A_t = r + \gamma V(s_{t+1}) - V(s_t)$
- הנוסחה של advantage שקיבלנו זהה לנוסחת TD-Error שהכרנו בעבר ונהוג גם לסמן אותה גם באות δ (דלתא).

Advantage in Policy Gradient

- מתברר שניתן להחליף בנוסחת ה loss של Policy Gradient את G_t בA_t . המודל נקרא A2C- Advantage Actor Critic
- לא רק שהנוסחה תישאר נכונה, אלא שהאימון של הרשת הוא הרבה יותר יציב.

$$\nabla loss = A_t * \nabla \ln \pi_\theta(a|s)$$

$$A_t = r + \gamma V(s_{t+1}) - V(s_t)$$

עדכון הפרמטרים של הרשתות

- קיבלנו שרשת ה Actor שלנו $\pi_\theta(s)$ מקיימת:

$$\nabla loss = A_t * \nabla \ln \pi_\theta(a|s)$$

$$A_t = r + \gamma V(s_{t+1}) - V(s_t)$$

- לצורך חישוב של $V(s)$ אנחנו נשתמש ברשת נוירונים נוספת, אשר תתעדכן בהתאם לנוסחת בלמן (Bootstrapping).

$$critic\_loss = \left(r + \gamma V(s_{t+1}) - V(s_t)\right)^2$$

$$critic\_loss = A^2$$

האלגוריתם

- נבנה שתי רשתות: $\pi(s) \rightarrow action\_prob, V(s) \rightarrow state\_value$
- נדגום את הסביבה צעד אחד ונמצא:

state, action, reward, value(s), s', value(s')

- לאחר כל צעד נחשב את ה TD-Error = Advantage:

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

- נעדכן את רשת ה Actor:

$$actor_{loss} = -A * \log\_prob$$

- נעדכן את רשת ה Critic:

$$critic\_loss = A^2$$

Actor Critic אלגוריתם – לסיכום

קריית החינוך פארק המדע
בית לערכים למצוינות ולחדשנות

One-step Actor–Critic (episodic)

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Input: a differentiable state-value parameterization $\hat{v}(s,\mathbf{w})$
Parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$

Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^{d}$
Repeat forever:
  Initialize S (first state of episode)
  $I \leftarrow 1$
  While S is not terminal:
    $A \sim \pi(\cdot|S,\boldsymbol{\theta})$
    Take action A, observe S', R
    $\delta \leftarrow R + \gamma \hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$ (if S' is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} I \delta \nabla_{\mathbf{w}} \hat{v}(S,\mathbf{w})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} I \delta \nabla_{\boldsymbol{\theta}} \ln \pi(A|S,\boldsymbol{\theta})$
    $I \leftarrow \gamma I$
    $S \leftarrow S'$

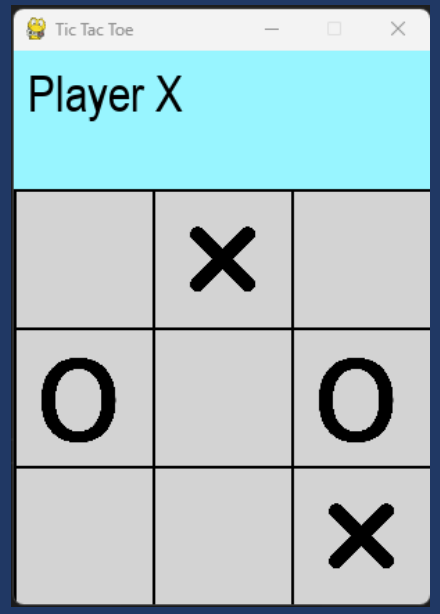
Sutton, Barto Reinforcement Learning An Intrudaction, 2nd ed. p. 332 •

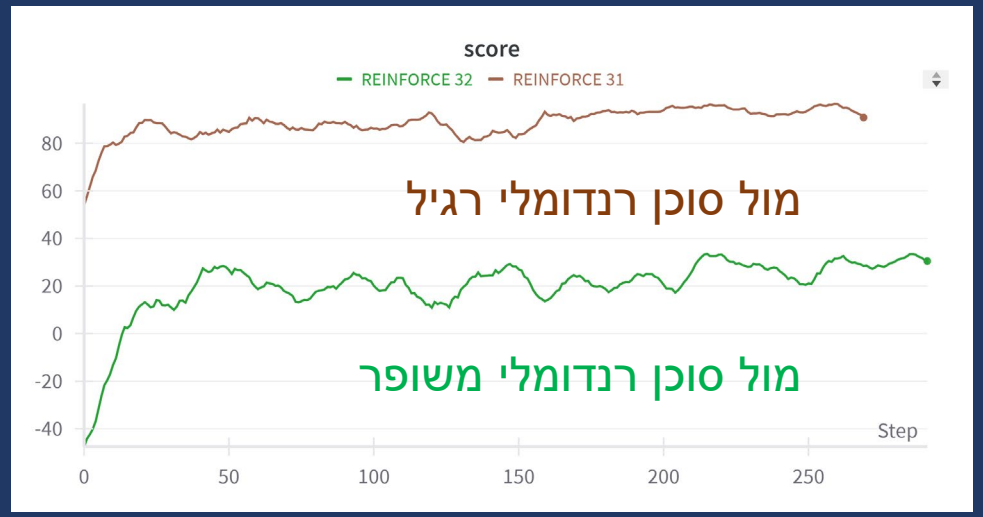
היתרונות של Actor-Critic

- אלגוריתם Actor-Critic משלב את היתרונות של שיטות מבוססות פוליסי (policy-based) יחד עם הערכת ערך (value-based) מה שמביא עמו מספר יתרונות מרכזיים:
- **<u>הפחתת שונות ועדכונים יציבים</u>**: בשיטה כמו REINFORCE (MC), העדכונים נעשים לאחר סיום כל פרק זמן, מה שגורם לעדכונים בעלי שונות גבוהה. לעומת זאת, בשיטת Actor-Critic נעשה שימוש בפונקציית הערך כבסיס (baseline) שמפחיתה את השונות בעדכונים ובכך מסייעת ללמידה יציבה ומהירה יותר.
- **<u>יעילות דגימה ושימוש ב-Bootstrapping</u>**: Actor-Critic עושה שימוש בעדכונים תוך כדי הריצה (bootstrapping), כך שלא צריך להמתין לסיום המשחק כדי לעדכן את המדיניות. דבר זה מאפשר למידת מדיניות מהירה ויעילה יותר, תוך ניצול טוב יותר של הדגימות.
- **<u>התמודדות עם מרחבי פעולה רציפים</u>**: בעוד ש-DQN הוא אלגוריתם שמיועד בעיקר למרחבי פעולה בדידים, Actor-Critic מתאים יותר לסביבות עם מרחבי פעולה רציפים. זה מאפשר להתאים את השיטה לסביבות מורכבות יותר שבהן הבחירה בפעולה אינה רק בין מספר אפשרויות קבוע.
- **<u>למידה ישירה של המדיניות</u>**: Actor-Critic מעדכן ישירות את מדיניות ה- actor תוך שימוש במשוב מה-critic, מה שמאפשר גמישות גבוהה יותר והתאמה רציפה של הפעולות הנבחרות.
- **<u>לסיכום</u>:** השילוב בין הערכת הערך למדיניות בלמידת Actor-Critic מביא ליתרונות של יציבות, יעילות ומהירות, במיוחד בסביבות מורכבות או עם מרחבי פעולה רציפים.

מימוש Actor Critic

- נדגים את מימוש האלגוריתם במשחק איקס עיגול.
- אנחנו נבנה רשת נוירונים אחת שתשמש גם ל actor וגם ל critic.
  - ניתן לממש זאת גם באמצעות שתי רשתות נפרדות – נדגים זאת בהמשך.
- נראה כי האלגוריתם מצליח להתמודד ולנצח סוכן רנדומלי מתקדם, בעוד שהאלגוריתם REINFORCE התקשה לנצח אותו.





רשת הנוירונים

- רשת הנוירונים משותפת ל Actor ול- Critic. הפיצול בשכבה האחרונה.

```python
class ActorCriticNetwork(nn.Module):
    def __init__(self, state_dim, action_dim, fc_dims=256, use_cuda=True):
        super().__init__()
        if use_cuda and torch.cuda.is_available():···
        else:···

        # Shared Layers
        self.linear1 = nn.Linear(state_dim, fc_dims)
        self.LeakyRelu = nn.LeakyReLU()
        self.linear2 = nn.Linear(fc_dims, fc_dims)

        # Actor head
        self.actor_layer = nn.Linear(fc_dims, action_dim)

        # Critic head
        self.critic_layer = nn.Linear(fc_dims, 1)

        # Move the model to the specified device
        self.to(self.device)
```

```python
def forward(self, state):
    x = self.linear1(state)
    x = self.LeakyRelu(x)
    x = self.linear2(x)
    x = self.LeakyRelu(x)

    # Actor (policy) output
    action_logits = self.actor_layer(x)

    # Critic (value) output
    value = self.critic_layer(x)

    return action_logits, value
```

מיסוך פעולות לא חוקיות

- מיסוך של הפעולות הלא חוקיות נעשה בשלבים הבאים:
  - פעולות לא חוקית היא בחירה של מיקום תפוס בלוח.
  - הפונקציה mask_illegat מחזירה מערך של כל המקומות התפוסים – לא חוקיים.
  - מיסוך הפעולה נעשה על ידי הוספת inf– לערך הפעולה כך שההסתברות שלה לאחר חישוב softmax יהיה קרוב ל- 0.
  - חישוב softmax נעשה באמצעות האובייקט Categorical.

```python
def mask_illegal (self, states_tensor, logits):
    mask = torch.zeros_like(logits, dtype=torch.float)
    mask[states_tensor != 0] = -torch.inf
    return mask
```

```python
from torch.distributions.categorical import Categorical
```

```python
mask = self.mask_illegal(state_tensor, action_logits)
masked_logits = action_logits + mask
dist = Categorical(logits=masked_logits)
```

הפונקציה get_action

- הפונקציה getAction מקבלת state ובאמצעות רשת הנוירונים מחזירה את הפעולה, הן כ (row, col) והן כאינדקס מתוך 9 פעולות אפשריות.
- בחירת הפעולה נעשית באמצעות דגימה הסתברותית ע"י האובייקט Categorical.

```python
def get_Action(self, state, events=None, epoch=None, train=True):
    state_tensor = state.toTensor().to(self.policy_value.device)
    with torch.no_grad():
        action_logits, _ = self.policy_value(state_tensor)

    mask = self.mask_illegal(state_tensor, action_logits)
    masked_logits = action_logits + mask
    dist = Categorical(logits=masked_logits)
    action = dist.sample().item()
    row, col = self.to_action(action)
    return (row, col), action
```

```python
def to_action(self, action_index):
    row = action_index // 3
    col = action_index % 3
    return row, col
```

הפונקציה get_action_and_value

- הפונקציה נועדה לאימון הרשת. היא מקבלת state ומחזירה:
  - Actor - Action - (row, col)
  - Actor - log_prob action
  - Critic – Value(s)
- ה log_prob הוא חישוב log(action_prob) בהתאם לנוסחת ה loss.
- חישוב הלוגריתם נעשה רק להסתברות הפעולה שנבחרה על ידי המודל.

```python
def get_action_and_value(self, state):
    state_tensor = state.toTensor().to(self.policy_value.device)
    action_logits, value = self.policy_value(state_tensor)
    mask = self.mask_illegal(state_tensor, action_logits)
    masked_logits = action_logits + mask
    dist = Categorical(logits=masked_logits)
    action = dist.sample()
    row, col = self.to_action(action.item())
    log_prob = dist.log_prob(action)
    return (row, col), log_prob, value
```

אימון המודל - הדגימה

- הדגימה של הסביבה מייצרת לנו:
  - state
  - Action
  - action_log_prob
  - Value
  - next_value

```python
def train(self, epochs=100000):
    self.init_wandb(project_name="Actor_Critic_TTO",chkpt=self.chkpt,resume=False)
    score, wins, losses = 0, 0, 0
    for epoch in range(epochs):
        print(epoch, end="\r")
        state = self.env.reset()
        done = False
        while not done:
            # Agent's move + Forward
            action, log_prob, value = self.agent.get_action_and_value(state)
            after_state, reward = self.env.next_state(state, action)
            done = self.env.end_of_game(after_state)
            # opponent move
            if not done:
                action2 = self.opponent.get_action(state=after_state)
                next_state, reward1 = self.env.next_state(after_state, action2)
                reward += reward1
                done = self.env.end_of_game(next_state)

            with torch.no_grad():
                _, _, next_value = self.agent.get_action_and_value(next_state)
```

אימון המודל – עדכון הפרמטרים

- כיוון שאנו משתמשים ברשת אחת אנחנו נסכום את שני ה loss השונים, ונבצע backward על סכום ההפסדים.

```
delta = reward + self.gamma * next_value * (1 - done) - value

# Actor loss - forward
actor_loss = -log_prob * delta.detach()

# Critic loss - forward
critic_loss = delta ** 2     # MSELoss

# Total loss for this step - forward
loss = actor_loss + critic_loss

# Backward
self.agent.optim.zero_grad()
loss.backward()
self.agent.optim.step()

state = next_state
```

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

$$actor\_loss = -A * \log\_prob$$

$$critic\_loss = \left(r + \gamma V(s_{t+1}) - V(s_t)\right)^2$$
$$critic\_loss = A^2$$

Advanced Random Agent אימון סוכן נגד

קריית החינוך פארק המדע
בית לערכים למצוינות ולחדשנות