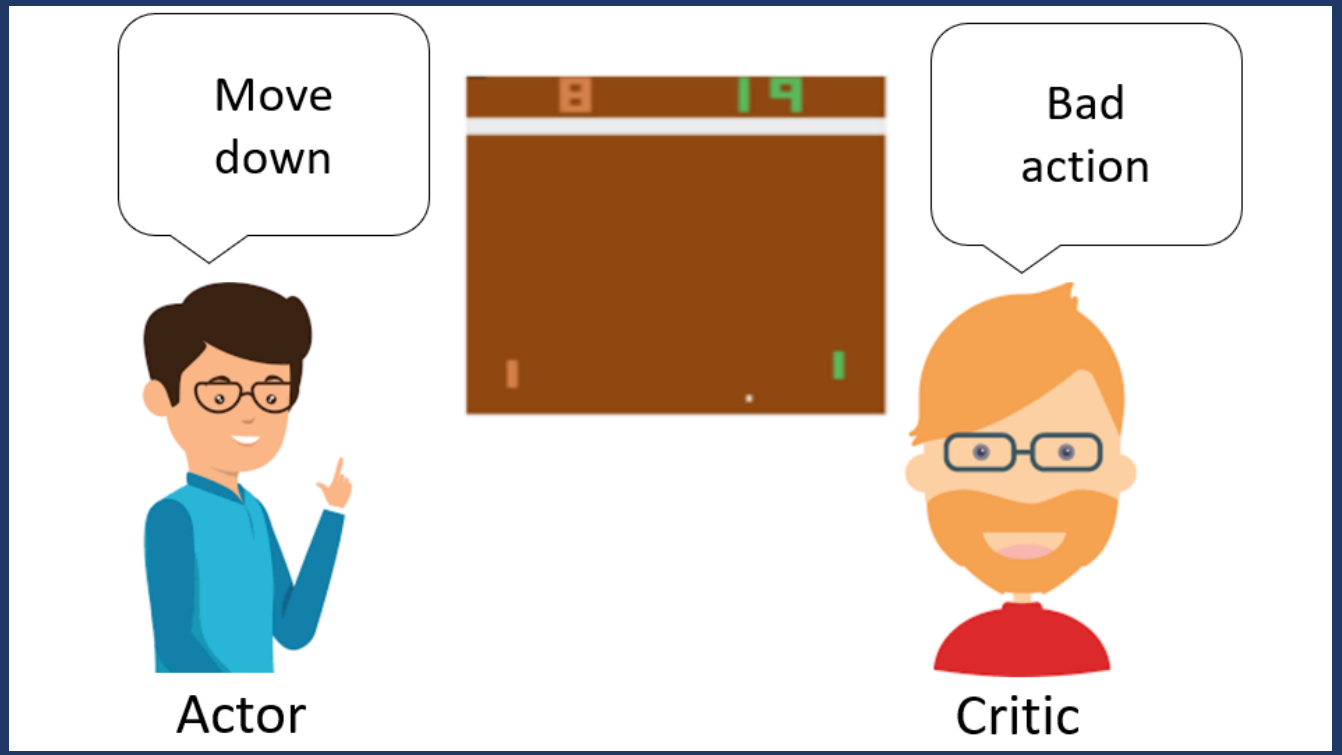
Actor Critic n-step

קריית החינוך
פארק המדע
בית לערכים
למצוינות ולחדשנות

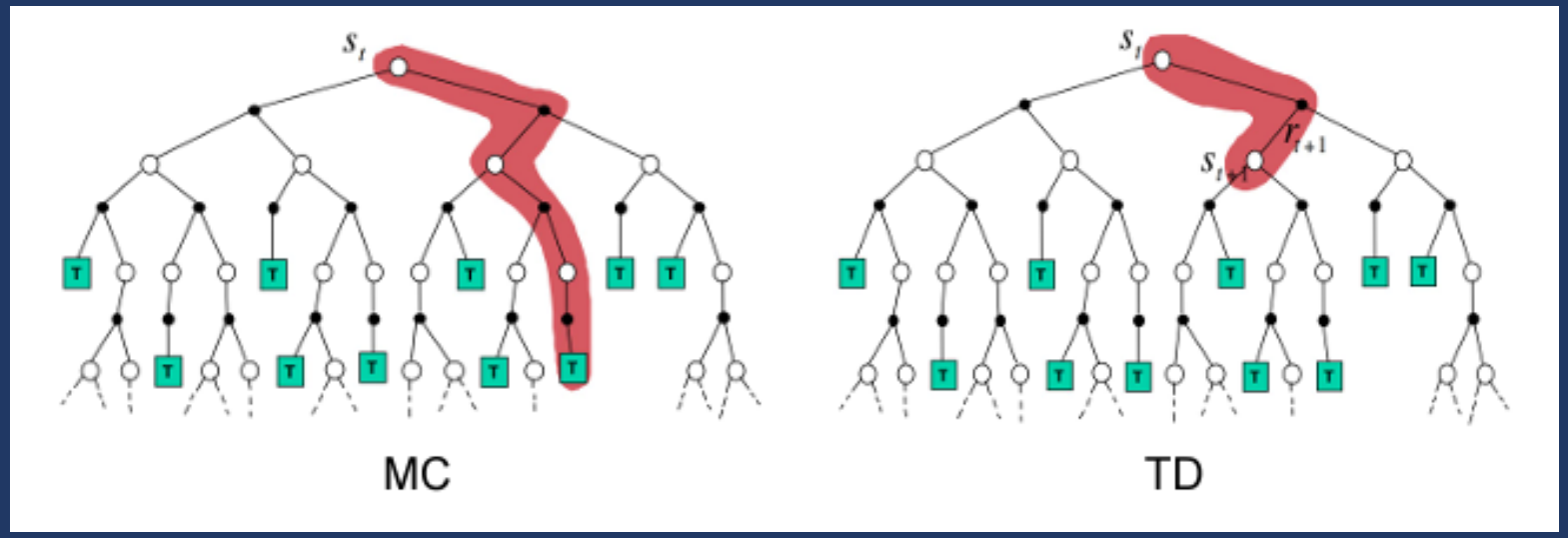


גלעד מרקמן

קישור ל GitHub

עדכון מספר צעדים קדימה

- עד עתה הכרנו שני אלגוריתמים שמבצעים את הדגימה של הסביבה ועדכון הפרמטרים של הרשת בשתי שיטות:
- REINFORCE Monte Carlo – דוגם אפיזודה (משחק) שלם ומבצע עדכון רק לאחר סיומו.
- Actor Critic – דוגם צעד אחד בלבד ומייד מבצע עדכון של הפרמטרים בהתאם לצעד זה.



החסרונות של השיטות

- לשתי השיטות יש חסרונות משמעותיים.
- מונטה קרלו:
  - מונטה קרלו מחייב אותנו לסיים אפיזודה שלמה שלעיתים היא ארוכה מאוד.
  - שונות גבוהה (variance) - שונות היא ההבדל בערכים בין כל עדכון. כיוון שכל משחק שונה משמעותית ממשחק אחר, אזי העדכונים לאחר כל משחקים הם עם שונות גבוהה ומקשים על הרשת ללמוד. יש "קפיצות" מצד לצד.
- One step Actor Critic:
  - ראיית קצרת טווח – הסוכן מסתכל רק צעד אחד קדימה ועל בסיסו מעדכן את הרשת.
  - חוסר יעילות – מעדכנים את הרשת על סמך נתון בודד ולא מספר נתונים.
  - הטיה (bias) – כיוון שמעדכנים על סמך צעד בודד נוצרת הטיה של האימון לתוצאות הספציפיות של הצעד הנדגם.

N-step Actor Critic

- בטווח שבין דגימה של כל האפיזודה (המשחק) לבין דגימה של צעד אחד בלבד, ניתן לבחור את שביל הביניים – דגימה של מספר צעדים קבוע.
- זהו האלגוריתם המכונה n-step Actor Critic.
- הרעיון הוא שאנו נדגום את הסביבה במשך מספר צעדים, ונקבל את התגמול עבור כל צעד כמו ב Monte Carlo.
- בסוף המהלך אם לא נגיע ל**מצב סופי** נעריך את שווי התגמולים העתידיים באמצעות פונקציית הערך שלנו $V(s)$.

היתרונות של n-step

- איזון טוב יותר בין הטיה לשונות:
  - כיוון שאנחנו דוגמים מספר צעדים אנחנו מקבלים ערכים פחות מוטים. כיוון שלא ממשיכים עד סוף האפיזודה השונות אינה גבוה מידי.
- למידה יציבה ומהירה יותר
  - בגלל שהגרדיאנטים מבוססים על יותר מידע (n צעדים במקום אחד), העדכון של הרשת מבוסס על מידע עשיר יותר – וזה יכול להוביל ללמידה מהירה ויעילה יותר.
- ראיה של מספר צעדים קדימה
  - כיוון שאנחנו מתחשבים במספר צעדים קדימה – הראיה של הסוכן לתוצאות עתידיות של הפעולות שלו טובה יותר.
- אימון באמצעות Batches יעיל יותר
  - אנחנו נבצע את אימון הרשת באמצעות קבוצה של פעולות ולא רק פעולה אחת, דבר המנצל את החישוב המקבילי של ה GPU והוא יותר יעיל.

שווי המצב ב n-step

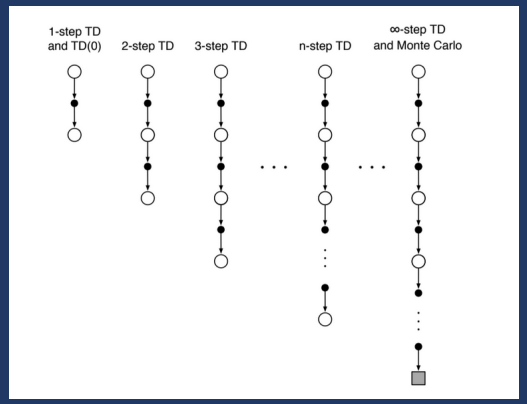
- כזכור שווי המצב – פעולה שלנו נקבע בהתאם לנוסחה הבאה:

$$G_t = Q(s,a) = R_0 + \gamma R_1 + \gamma^2 R_2 + \gamma^3 R_3 + \cdots + \gamma^T R_T$$

- כאשר R_T הוא התגמול שמתקבל כאשר מגיעים למצב הסופי.
- ב n-step אנחנו לא מגיעים למצב הסופי ולכן הנוסחה תהיה:

$$G_t = Q(s,a) = R_0 + \gamma R_1 + \gamma^2 R_2 + \cdots + \gamma^n V(s_n)$$

- כלומר באמצעות $V(s_n)$ אנחנו מעריכים מה יהיה סכום התגמולים החל ממצב S_n ועד למצב הסופי S_T.



נוסחת ה Advantage n-step

- בצעד אחת נוסחת ה Advantage היא:

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

- ב n-step נוסחת ה Advantage שלנו תשתנה כך:

$$A_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ \ldots + \gamma V(s_{n+1}) \ - V(s_t)$$

אימון באמצעות Batches

- כיוון שאנחנו מבצעים מספר צעדים בהם אנו דוגמים את הסביבה ניתן לאמן את הרשת מספר (קבוצה) של צעדים.
- אנחנו אפילו נבצע ערבוב (Shuffle) של סדר הצעדים (מבלי לפגוע באלגוריתם המתמטי), כך שהרשת לא מתעדכנת לפי סדר הפעולות שלנו.
- התוצאה - הסוכן פחות תלוי ברצפים ספציפיים ולכן הלמידה יותר כללית ויש סכנה פחותה יותר ל- overfitting לאפיזודות בודדות.

Actor Critic n-steps

- נדגום את הסביבה במשך n-steps (או סוף משחק) ונשמור. ערכים אילו אינם חלק מגרף החישוב ולא כוללים גרדיאנטים:
  - State, action, log_prob, val, reward, done
- אם הגענו ל n-step נבצע עדכון של רשת הנוירונים (למידה).
  - נעביר נתון נוסף שהוא value(s') – הצעד הבא שלא נדגם, בנוסף לנתונים שנשמרו תוך כדי הדגימה.
- בשלב הלמידה
  - נחשב returns, advantage לכל מצב.
    - Returns = V(s')
    - Advantage = Returns – V(s)
  - לאחר החישוב נחלק את הנתונים ל batches ונערבב אותם.
  - לכל batch נחשב את ה loss ונעדכן את הפרמטרים.
  - נבצע את הלמידה מספר קטן של n_epochs = 2.

מימוש האלגוריתם
N step Actor Critic

Space Invaders

עדכון מחלקת הסביבה - Environment

- מחלקת הסביבה עודכנה לעומת ההרצאות הקודמות כדי לשפר את האימון של הסוכן.
- ה state נבנה כך שמיקום האובייקטים של הסביבה הוא קבוע.
  - במימוש הקודם אם חללית אויב הושמדה כל החלליות הנותרות זזו למקום הפנוי ב state tensor, כך שהמקומות הפנויים תמיד היו בסוף ה Tensor.
  - אותה בעיה היתה גם לגבי האובייקטים האחרים (הטילים של השחקן וטילי האוייב).
  - מימוש דינאמי זה פגע ביכולת של הסוכן להבין את הסביבה, שכן עם השמדת חללית כל הנתונים על החליליות האחרות זזו למיקום אחר ב tensor ללא סיבה.
  - במימוש הנוכחי המיקום של האובייקטים ב tensor של הסביבה לא משתנה לאורך כל המשחק. אם האובייקט מושמד המיקום שלו מאופס. אם צריך להוסיף אובייקט חדש מחפשים מקום פנוי אך לא מזיזים אובייקטים אחרים.

עדכון מחלקת הסביבה - Environment

- שינוי התגמול – reward
  - הפרמטרים של התגמול (reward) מהווים את אחד הנתונים החשובים להצלחת האימון.
  - הוספנו תגמול שלילי אם החללית נמצאת מתחת לטיל אוייב. כדי ללמד את הסוכן להתחמק מטילי אוייב.
  - הוספנו תגמול שלילי קטן על יריה כדי ללמד את הסוכן לחסוך בתחמושת
  - הוספנו תגמול עבור סיום שלב.
- סוף משחק
  - סוף משחק אם החללית נפגעה, האויבים נחתו, או התחמשות נגמרה.
  - מבחינת הסוכן – סיום שלב מהווה סוף משחק עם ניצחון, והוא מתחיל משחקון חדש.

Actor Network

```python
class ActorNetwork(nn.Module):
    def __init__(self, input_dims, n_actions, lr, fc1_dims=256, fc2_dims=512, chkpt=1, optim_step = 100, optim_gamma = 0.9):
        super(ActorNetwork, self).__init__()
        self.fc1 = nn.Linear(input_dims, fc1_dims)
        self.fc2 = nn.Linear(fc1_dims, fc2_dims)
        self.fc3 = nn.Linear(fc2_dims, fc1_dims)
        self.fc4 = nn.Linear(fc1_dims, n_actions)
        self.relu = nn.ReLU()

        self.checkpoint_file = f'Data/Actor{chkpt}.pth'
        self.optimizer = optim.Adam(self.parameters(), lr=lr)
        self.scheduler = optim.lr_scheduler.StepLR(self.optimizer, step_size=optim_step, gamma=optim_gamma)
        self.device = T.device('cuda:0' if T.cuda.is_available() else 'cpu')
        self.to(self.device)

    def forward(self, state):
        x = self.fc1(state)
        x = self.relu(x)
        x = self.fc2(x)
        x = self.relu(x)
        x = self.fc3(x)
        x = self.relu(x)
        x = self.fc4(x)

        dist = Categorical(logits=x)
        return dist
```

Input_dims = state (88)

n_action=4

Categorical = softmax

Critic Nerwork

```python
class CriticNetwork(nn.Module):
    def __init__(self, input_dims, lr, fc1_dims=256, fc2_dims=512, chkpt=1, optim_step = 100, optim_gamma = 0.9):
        super(CriticNetwork, self).__init__()

        self.checkpoint_file = f'Data/Critic{chkpt}.pth'
        self.fc1 = nn.Linear(input_dims, fc1_dims)
        self.fc2 = nn.Linear(fc1_dims, fc2_dims)
        self.fc3 = nn.Linear(fc2_dims, fc1_dims)
        self.fc4 = nn.Linear(fc1_dims, 1)
        self.relu = nn.ReLU()

        self.optimizer = optim.Adam(self.parameters(), lr=lr)
        self.scheduler = optim.lr_scheduler.StepLR(self.optimizer, step_size=optim_step, gamma=optim_gamma)
        self.device = T.device('cuda:0' if T.cuda.is_available() else 'cpu')
        self.to(self.device)

    def forward(self, state):
        x = self.fc1(state)
        x = self.relu(x)
        x = self.fc2(x)
        x = self.relu(x)
        x = self.fc3(x)
        x = self.relu(x)
        x = self.fc4(x)
        return x
```

Input_dims = state (88)

Output = Value (1)

memory

- ב memory – אנו שומרים n-steps (64).
- generate_batches – יוצר לנו רשימות בגודל של batch_size (16), הכוללות מספרי אינדקסים מעורבבים אקראית.

```python
class Memory:
    def __init__(self, batch_size):
        self.states = []
        self.log_probs = []
        self.vals = []
        self.actions = []
        self.rewards = []
        self.dones = []

        self.batch_size = batch_size
```

```python
def get_arrays(self):
    return np.array(self.states),\
           np.array(self.actions),\
           np.array(self.vals),\
           np.array(self.rewards),\
           np.array(self.dones)
```

```python
def clear_memory(self):
    self.states = []
    self.log_probs = []
    self.actions = []
    self.rewards = []
    self.dones = []
    self.vals = []
```

```python
def generate_batches(self):
    n_states = len(self.states)
    batch_start = np.arange(0, n_states, self.batch_size)
    indices = np.arange(n_states, dtype=np.int64)
    np.random.shuffle(indices)
    batches = [indices[i:i+self.batch_size] for i in batch_start]

    return batches
```

```python
def store_memory(self, state, action, log_probs, vals, reward, done):
    self.states.append(state)
    self.actions.append(action)
    self.log_probs.append(log_probs)
    self.vals.append(vals)
    self.rewards.append(reward)
    self.dones.append(done)
```

Agent

- הסוכן מאתחל את הפרמטרים שלו וכן רשתות ה actor, critic והזיכרון.

```python
class Actor_Critic_Agent:
    def __init__(self, chkpt, input_dims=88, n_actions=4, wandb = None):
        self.gamma = 0.995
        self.n_epochs = 2
        self.batch_size = 16
        self.lr_actor = 1e-4
        self.lr_critic = 1e-4
        self.optim_step = 5000
        self.optim_gamma = 0.95
        self.critic_actor_ratio = 0.5
        self.wandb = None   # will be updated by Trainer
        self.learn_step = 0 # counter for number of learning
        self.entropy_coefficient = 0.1
        self.max_entropy_coeff = 0.1
        self.min_entropy_coeff = 0.02
        self.entropy_decay_rate = 0.9996


        self.actor = ActorNetwork(input_dims, n_actions, self.lr_actor, chkpt=chkpt, optim_step=self.optim_step, ···
        self.critic = CriticNetwork(input_dims, self.lr_critic, chkpt=chkpt, optim_step=self.optim_step, ···
        self.memory = Memory(self.batch_size)
```

choose action + get Value

- הפונקציה choose_action מקבלת מצב:
  - באמצעות ה actor מחזירה את הפעולה וחישוב ה log_prob של הפעולה.
  - באמצעות ה critic מחזירה את ערך המצב.
- הפונקציה get_value מחזירה באמצעות ה critic את ערך המצב.
- שתי הפונקציות אינן משתתפות בגרף החישוב.
- הפונקציה remember שומרת את הדגימה בזיכרון.

```python
def choose_action(self, state):
    state = state.to(self.actor.device)
    with T.no_grad():
        dist = self.actor(state)
        value = self.critic(state)
    action = dist.sample().item()
    log_prob = dist.log_prob(T.tensor(action, device=self.actor.device)).item()
    value = value.item()

    return action, log_prob, value
```

```python
def remember(self, state, action, probs, vals, reward, done):
    self.memory.store_memory(state, action, probs, vals, reward, done)
```

```python
def get_value (self, state):
    state = state.to(self.actor.device)
    with T.no_grad():
        value = self.critic(state)
    value = value.item()
    return value
```

trainer

```python
def train(self):
    agent = self.agent
    self.step = 0
    for epoch in range(self.start_epoch, self.epochs):
        self.env.restart()
        done = False
        self.reward = 0
        self.moves = 0
        state = self.env.state()
        if self.env.level == 1: # clearing score after l
            self.env.score = 0
        while not done:
```

- הסוכן מתאמן מספר קבוע של משחקים.
- בכל משחק הסוכן דוגם את הסביבה ושמור: s, a, a_log, v, r, done
- כל n_step צעדים מתבצע אימון של הרשת.
- אם n_step אינו סוף משחק מועבר לפונקציית הלמידה גם הערך של המצב הבא (ראה הסבר בהמשך).

```python
while not done:
    self.graphics.clear()
    self.graphics.event_pump()
    self.graphics.events()
    action, log_prob, val = agent.choose_action(state)
    reward, done = self.env.move(action=action)
    self.reward += reward        # for logging
    agent.remember(state, action, log_prob, val, reward, done)
    self.step += 1
    self.moves += 1
    state = self.env.state()
```

```python
if self.step % self.n_steps == 0 or done:
    if not done:                          # calculate next state value
        val = agent.get_value(state)
        agent.learn(val)
    else:
        agent.learn(0.0)              # next state value is 0.0

self.graphics.header_writing(env=self.env, epoch=epoch, chkpt=chkpt)
self.graphics.update()
```

learn

- אימון הרשתות נעשה לאחר דגימה של n_step הנשמרת בזיכרון.
- שולפים מהזיכרון את ערכי הצעדים שנשמרו.
- מחשבים את ה returns V(s’), advantges V(s’)-V(s) – כל אחד הוא מערך כמספר הצעדים שנשמרו.
- החישוב נעשה **<u>לפני</u>** החלוקה ל batches והערבוב של הצעדים.
- ה entropy שנועד להגביר exploration מחושב עם הפחתה כך שככל שהאימון מתקדם הערך יורד, עד למינימום שנקבע.

```python
def learn(self, next_val):
    #region For logging…
    self.learn_step += 1
    state_arr, action_arr, val_arr, reward_arr, done_arr = self.memory.get_arrays()

    # skipping learning if too few samples
    if len(state_arr) < 2:
        print(f"Skipping learning: only {len(state_arr)} samples, need at least 2")
        self.memory.clear_memory()
        return

    # entropy coefficient decay
    self.entropy_coefficient = max(self.min_entropy_coeff, self.entropy_coefficient * self.entropy_decay_rate)
    # Compute advantage and returns
    advantage, returns = self.calculate_advantage_and_returns(reward_arr, val_arr, done_arr, next_val)
```

Learn – cont.

```python
for i in range(self.n_epochs):
    batches = self.memory.generate_batches()
    for batch in batches:
        states = T.tensor(state_arr[batch], dtype=T.float).to(self.actor.device)
        actions = T.tensor(action_arr[batch]).to(self.actor.device)
        batch_advantage = advantage[batch].to(self.actor.device)
        batch_returns = returns[batch].to(self.critic.device)

        # Get policy distribution and value predictions
        dist = self.actor(states)
        log_probs = dist.log_prob(actions)
        critic_value = self.critic(states).squeeze(-1)

        # Calculate actor loss
        actor_loss = -(log_probs * batch_advantage).mean()

        # Calculate critic loss
        critic_loss = F.mse_loss(critic_value, batch_returns)

        # calc entropy bonus for exploration
        dist_entropy = dist.entropy().mean()

        # Combine all losses
        total_loss = actor_loss + self.critic_actor_ratio * critic_loss - self.entropy_coefficient * dist_entropy

        #region logging loss and entropy…
        # Perform backward and optimization
        self.actor.optimizer.zero_grad()
        self.critic.optimizer.zero_grad()
        total_loss.backward()
        self.actor.optimizer.step()
        self.critic.optimizer.step()
```

- מבצעים את האימון מספר של n_epochs (2-4 פעמים).
- מקבלים מהזיכרון מערכים אקראיים של אינדקסים בגודל batch.
- באמצעות batch של אינדקסים אנחנו שולפים את הערכים מהזיכרון.
- מחשבים שוב באמצעות ה critic וה-actor את הערכים (הפעם כחלק מגרף החישוב).

- מחשבים loss לכל אחד מהרשתות וכן loss מאוחד הכולל entropy (ל- exploration).
- מבצעים פעפוע לאחור ועדכון פרמטרים.

Calculate Advantage

$$A_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ \ldots + \gamma V(s_{n+1}) \ - V(s_t)$$

```python
def calculate_advantage_and_returns (self, reward_arr, val_arr, done_arr, next_val):

    returns = np.zeros_like(reward_arr, dtype=np.float32)

    # n-step return calculation
    future_return = next_val
    for t in reversed(range(len(reward_arr))):
        if done_arr[t]:
            future_return = 0.0  # No bootstrap if episode ends

        future_return = reward_arr[t] + self.gamma * future_return
        returns[t] = future_return

    # Advantage = Return - Value
    advantage = returns - val_arr

    # Convert to tensors and move to device
    advantage = T.tensor(advantage).to(self.actor.device)
    returns = T.tensor(returns).to(self.actor.device)

    #region Log for debugging and monitoring…
    return advantage, returns
```

- חישוב ה Advantage נעשה בשני שלבים:
  - חישוב ה returns הנעשה בלולאה מהסוף להתחלה.
  - returns – מערך של ערכי התגמול לכל מצב V(s’).
  - אם המצב האחרון אינו סוף משחק – מתווסף next_val שהינו הערך של המצב הבא. זה ה bootstraping.
- לאחר מכן, מחושב Advantage באמצעות חיסור של ה value של המצב V(s).

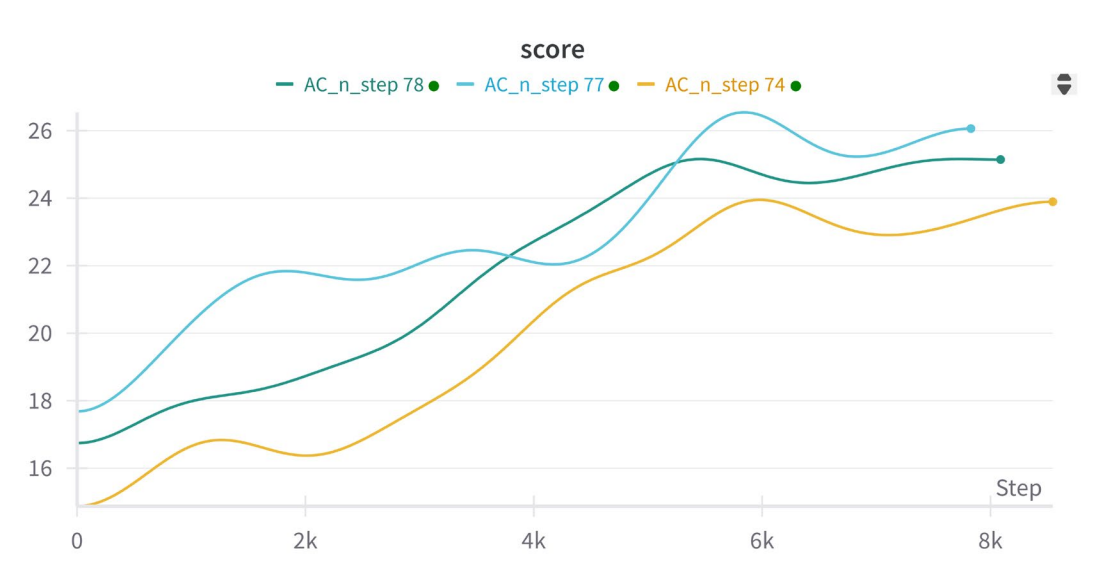
פרמטרים

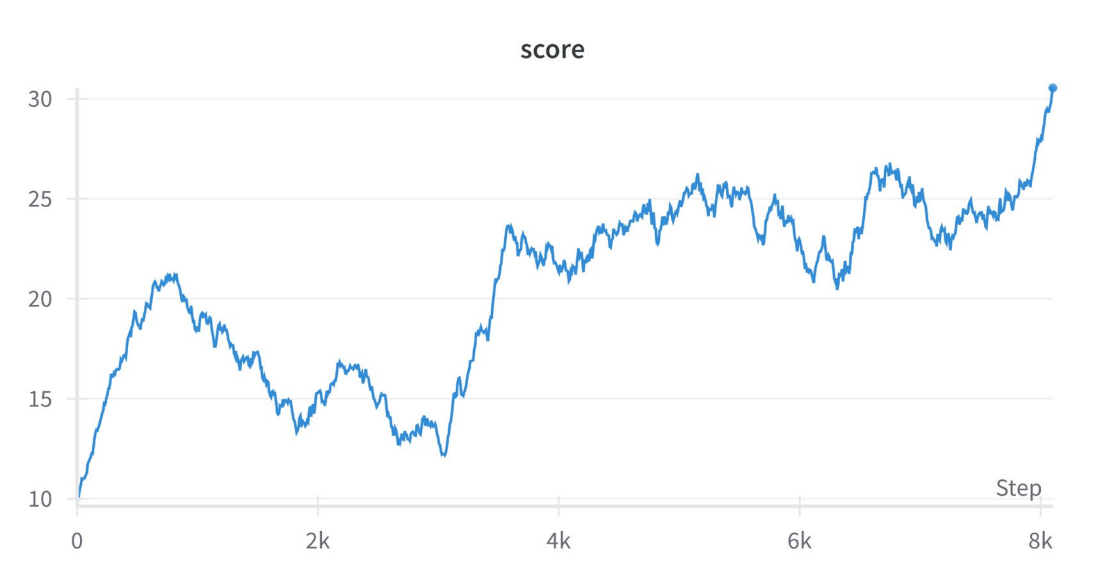
קריית החינוך פארק המדע
בית לערכים למצוינות ולחדשנות

hit_reward = 2	gamma = 0.995
stage_reward = 5	n_step = 64
done_reward = -0.5	n_epochs = 2
zero_ammunition_reward = 0	batch_size = 16
shoot_reward = -0.01	lr_actor = 1e-4
survival_reward = 0.00	lr_critic = 1e-4
misile_above_reward = -0.3	optim_step = 5000
	optim_gamma = 0.95
Actor_net: 88->256->512->256->4	critic_actor_ratio = 0.5
Critic_net: 88->256->512->256->1	entropy_coefficient = 0.1
Activation func: Relu	max_entropy_coeff = 0.1
	min_entropy_coeff = 0.02
	entropy_decay_rate = 0.9996

תוצאות הרצה – Space Invaders

- ניתן לראות תוצאות של שלוש הרצות עם פרמטרים קצת שונים של סוכן הלומד לשחק Space Invaders.
- ניתן לראות את השיפור של הסוכנים בתוצאות המשחק לאחר כ- 8000 משחקים.
- ה entropy עדיין גבוה ולכן נראה שהסוכן לא סיים את הלמידה.

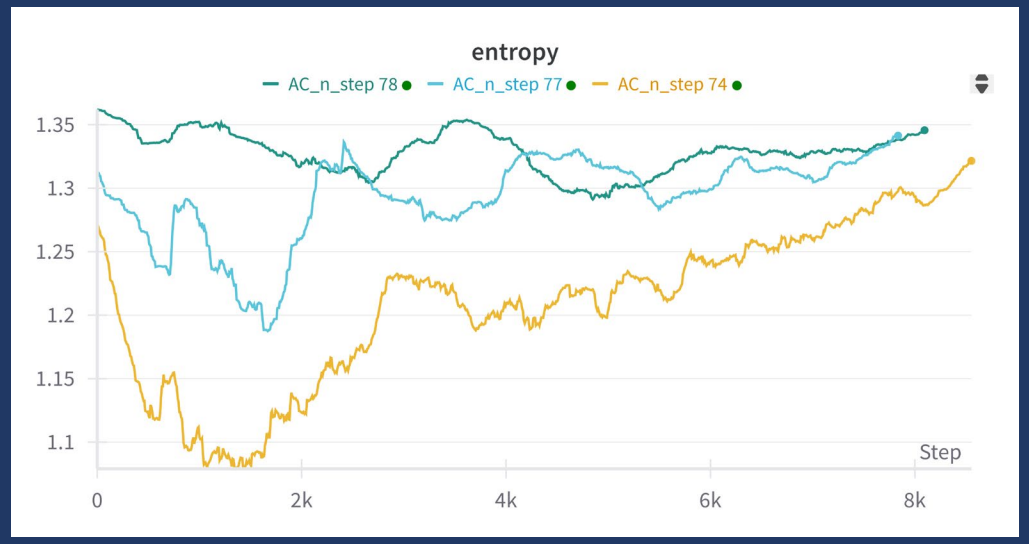




תוצאות הרצה


actor_losses
AC_n_step 78
AC_n_step 77
AC_n_step 74
-4.5
-5
-5.5
-6
-6.5
-7
-7.5
Step
0
2k
4k
6k
8k


critic_losses
AC_n_step 78
AC_n_step 77
AC_n_step 74
45
40
35
30
25
20
Step
0
2k
4k
6k
8k


entropy
AC_n_step 78
AC_n_step 77
AC_n_step 74
1.35
1.3
1.25
1.2
1.15
1.1
Step
0
2k
4k
6k
8k


total_losses
AC_n_step 78
AC_n_step 77
AC_n_step 74
12
10
8
6
4
Step
0
2k
4k
6k
8k