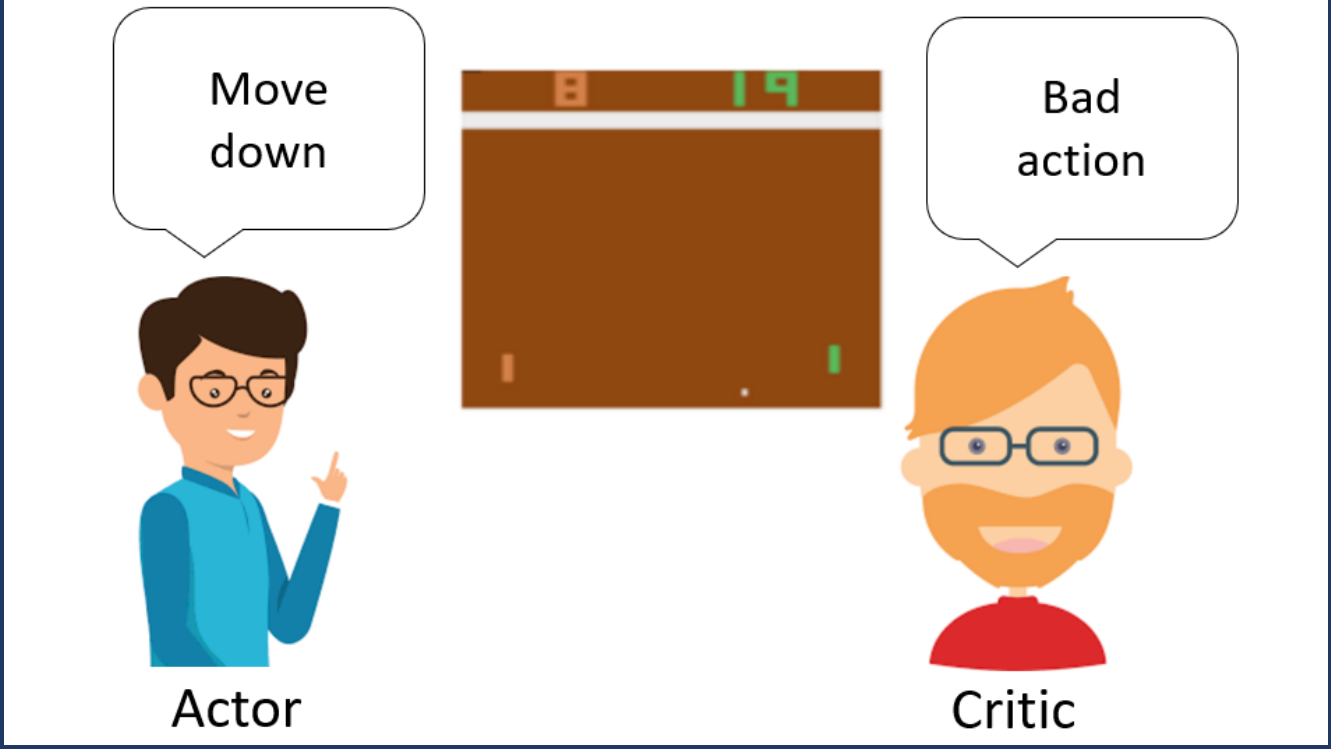
PPO - Proximal Policy Optimization



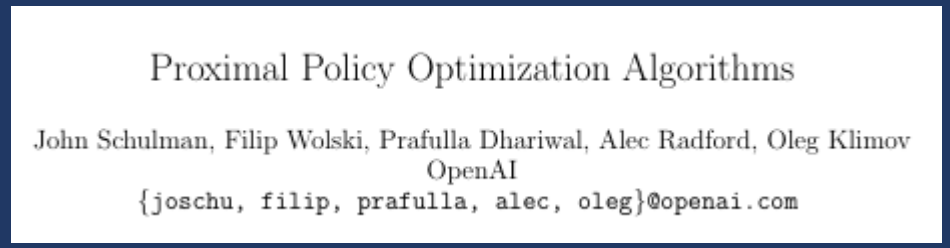
גלעד מרקמן

קישור ל GitHub

קריית החינוך
פארק המדע
בית לערכים
למצוינות ולחדשנות

PPO

- האלגוריתם PPO פורסם במאמר של חוקרים מחברת OpenAI בשנת 2017.


Proximal Policy Optimization Algorithms

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov
OpenAI
{joschu, filip, prafulla, alec, oleg}@openai.com

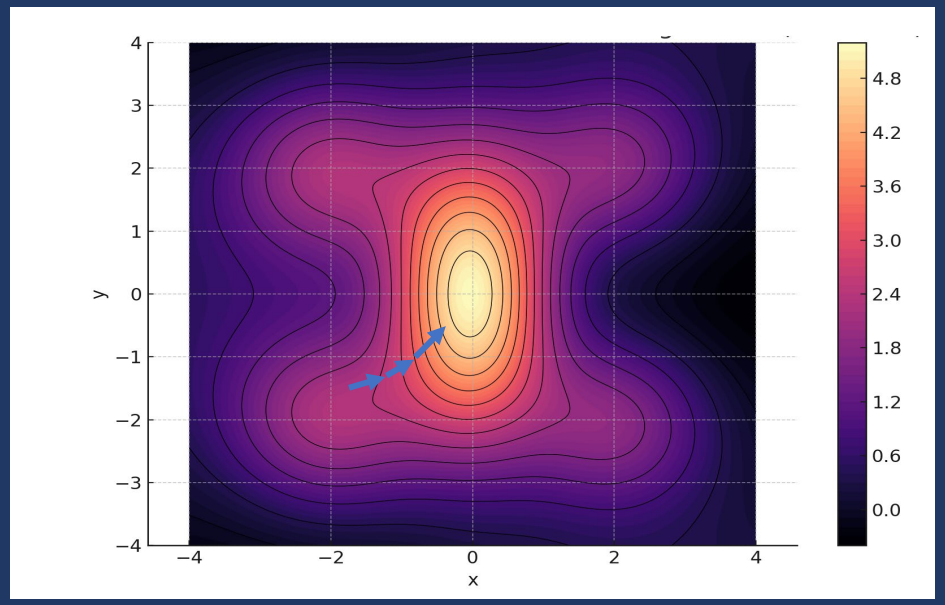
קישור למאמר

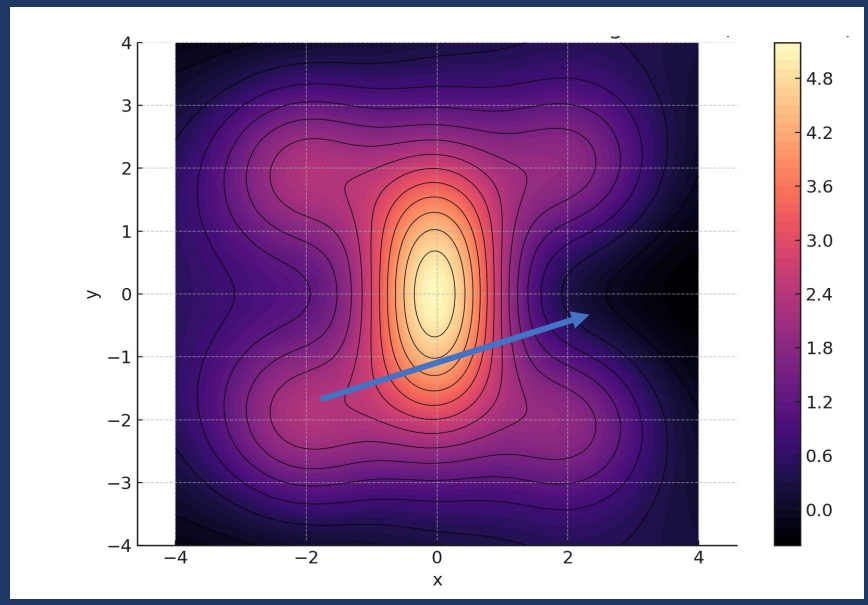
- האלגוריתם מבוסס על Actor Critic ומשפר אותו בשלוש נקודות עיקריות:
  - שימוש ב n-step & Batch optimization.
  - פיתוח טכניקה לעדכון "אמין" של רשת הנוירונים המחשבת את המדיניות Actor. הטכניקה מבוססת על TRPO – Trust Region Policy Optimization.
  - שיפור חישוב ה Advantage באמצעות GAE – Global Advantage Estimation.

- אלגוריתם זה משמש כיום לאימון מודלי הבינה הגדולים כגון: ChatGPT ואחרים.

Trust Region Optimization

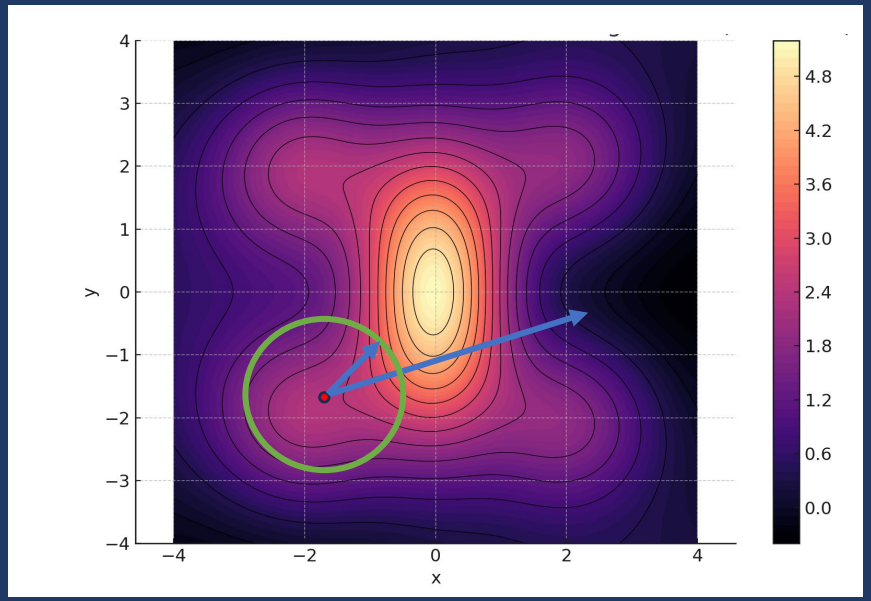
- עדכון הפרמטרים של רשת הנוירונים נעשה באמצעות SGD – Stochastic Gradient Descent או SGA – Stochastic Gradient Ascent.
- אנחנו מעדכנים את הפרמטרים בכיוון של הנגזרת "בצעדים קטנים".
- הבעיה היא ש SGD לא מונע מאיתנו לבצע צעדי עדכון "גדולים מידי" גם אם משתמשים ב learning rate נמוך.





Trust Region Policy Optimization

- אלגוריתמים ממשפחת TRPO מנסים למצוא את האזור הבטוח "Trust Region" לשינוי הפרמטרים, כדי שהעדכון לא יעשה בקפיצה גדולה מידי ויצא מ"האזור הבטוח".
- בספרות הציעו אלגוריתמים מסובכים המצריכים חישובים רבים למציאת האזור הבטוח. החידוש ב PPO הוא אלגוריתם פשוט ויעיל יותר למציאת האזור הבטוח לעדכון הפרמטרים.



PPO

- פונקציית הטעות של האלגוריתם Actor Critic הינה:

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

- מחברי המאמר הציעו לקבוע אזור בטוח בדרך הבאה:

- המחברים הגדירו את היחס האמור:

$$r_t(\theta) = \frac{\pi_\theta(a|s)}{\pi_{\theta\ old}(a|s)}$$

- היחס שבין ההסתברות של הפעולה המחושבת על ידי המדיניות המעודכנת (לאחר עדכון פרמטרים) חלקי ההסתברות של הפעולה בעת בחירת הפעולה (לפני העדכון).

PPO

- לאחר חישוב היחס :

$$r_t(\theta) = \frac{\pi_\theta(a|s)}{\pi_{\theta\ old}(a|s)}$$

- אנו מגבילים את היחס לסביבת $1 \pm \varepsilon$ (אפסילון בערך 0.2):

$$clip\_r_t = \text{clamp}(r_t, 1+\varepsilon, 1-\varepsilon)$$

- ה loss יחושב בדרך הבאה:

$$loss = \min(r_t * A_t, clip\_r_t * A_t)$$

הסבר הרציונל מאחורי ה PPO

- על מנת להבין את הרציונל מאחורי האלגוריתם PPO נחזור למושגי יסוד.

- A – מתאר את היתרון או החיסרון בבחירת פעולה a במצב s. אם בחירת הפעולה מגדילה את התגמולים בעתיד ה $A > 0$. אם הפעולה אינה מוצלחת אזי $A < 0$.

- $\pi(s)$ – הפונקציה מחזירה לנו את ההסתברות לבחור בכל אחת מהפעולות האפשריות.

- r – מציין את היחס שבין הסיכוי לבחור פעולה a לפני העדכון של הרשת ולאחריה. אם r>1 סימן שהעדכון הביא לכך שהסתברות של a גדלה (נבחר את הפעולה יותר) ולהפך.

הסבר הרציונל מאחורי ה PPO

- אם $A > 0$ (כלומר הפעולה טובה) ו- $r > 0$ כלומר העדכון הגדיל את הסיכוי לבחור בפעולה – נאמר שהעדכון נעשה בכיוון הנכון.
- אם $A < 0$ (כלומר הפעולה אינה טובה) ו- $r < 0$ כלומר העדכון הקטין את הסיכוי שנבחר בפעולה – נאמר שהעדכון נעשה בכיוון הנכון.
- אם $A > 0$ (כלומר הפעולה טובה) ו- $r < 0$ כלומר העדכון הקטין את הסיכוי לבחור בפעולה – נאמר שהעדכון נעשה בכיוון הלא נכון.
- אם $A < 0$ (כלומר הפעולה אינה טובה) ו- $r > 0$ כלומר העדכון הגדיל את הסיכוי לבחור בפעולה – נאמר שהעדכון נעשה בכיוון הלא נכון.

הסבר הרציונל מאחורי ה PPO

PPO למעשה מבצע את העדכון כך:

- אם r אינו חורג מהתחום המותר – חישוב ה Loss יהיה שווה (בקירוב) לחישוב הרגיל (ראה הוכחה מתמטית בהמשך).
- אם r חורג מהתחום המותר ועדכון הפרמטרים היה בכיוון הנכון – אנחנו נחתוך את r למספר קבוע והנגזרת תהיה 0. כלומר לא יתבצע כל עדכון.
- אם r חורג מהתחום המותר ועדכון הפרמטרים לא היה בכיוון הנכון – אנחנו נעדכן את הפרמטרים כדי לתקן את הטעות שעשינו בעדכונים הקודמים. חישוב ה Loss יהיה שווה (בקירוב) לחישוב הרגיל.

חישוב הנגזרת

- נראה כי כשאין חיתוך clip של r – חישוב הנגזרת יהיה בקירוב שווה לחישוב הרגיל:

$$\nabla loss = \nabla A_t * \frac{\pi_\theta(a|s)}{\pi_{\theta\ old}(a|s)} = A_t * \frac{1}{\pi_{\theta\ old}(a|s)} * \nabla\pi_\theta(a|s)$$

$$\nabla loss = A_t * \frac{\nabla\pi_\theta(a|s)}{\pi_{\theta\ old}(a|s)}$$

- נזכיר שחישוב ה loss הרגיל הוא:

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

$$\nabla \ln\big(f(x)\big) = \frac{f'(x)}{f(x)}$$

- במקרה בו $r_t \approx 1$ כלומר $\pi_\theta(a|s) \approx \pi_{\theta\ old}(a|s)$ החישוב של הנגזרת קרוב לחישוב הנגזרת הרגילה של Policy Gradient Theorem.

הסבר הרציונל מאחורי ה PPO

- במקרה בו היחס בין ההסתברויות חורג מהתחום והעדכון של הפרמטרים בכיוון הנכון יבוצע חיתוך של היחס r. החיתוך נותן ליחס r מספר קבוע.

$$\nabla loss = A_t * \nabla\,(1 \pm \varepsilon) = A_t * 0 = 0$$

- וכיוון שהנוסחה מורכת רק מקבועים הנגזרת היא 0.

- אם הנגזרת 0 לא מבצוע עדכון של הפרמטרים. כלומר אם כבר התקדמנו מספיק לכיוון הנכון לא נמשיך להתקדם.

GAE – General Advantage Estimation

- ב GAE-General Advantage Estimation אנחנו מחשבים את ה Advantage בדרך שתאזן בין ההטיה (bias) לבין השונות (variance).
- כזכור, Advantage מוגדר כיתרון (או חיסרון) בבחירת פעולה מסוימת במצב בו אנו נמצאים.

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

- נוסחת ה advantage בחישוב באמצעות MC היא (בעיית שונות):

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

- נוסחת ה advantage בחישוב צעד אחד קדימה (TD(0) - בעזרת Bootstrap) היא (בעיית הטיה):

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

GAE

- לצורך חישוב GAE אנו מגדירים פרמטר חדש λ שבאמצעותו ניתן לאזן בין השונות של MC להטיה של TD(0).
- נוסחת ה GAE היא:

$$GAE = \delta_t + (\lambda\gamma)\delta_{t+1} + (\lambda\gamma)^2\delta_{t+1} + \ \ldots (\lambda\gamma)^T\delta_T - V(S_t)$$

- אם $\lambda = 0$ אנחנו נקבל את נוסחת TD(0) (הטיה רבה):

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

- אם $\lambda = 1$ אנחנו נקבל את נוסחת MC (שונות רבה):

$$GAE = r_t + \ \gamma r_{t+1} + \gamma^2 r_{t+2} + \ \ldots + \ \gamma^T r_T - V(s_t)$$

- אם $0 < \lambda < 1$ אנחנו נקבל תוצאת ביניים שמאזנת בין ההטיה של TD(0) לשונות של MC.

GAE and N-step Optimizatiom

- ניתן להשתמש בנוסחת GAE בחישוב Advantage באמצעות n-step optimization, היינו שאנחנו מבצעים דגימה של מספר צעדים קדימה אך לאו דווקא עד מצב סופי.
- הנוסחה במקרה של GAE n-step:

$$GAE = \delta_t + (\lambda\gamma)\delta_{t+1} + (\lambda\gamma)^2\delta_{t+1} + \ \ldots + \gamma^n V(s_n) - V(s_t)$$

קריית החינוך
פארק המדע
בית לערכים
למצוינות ולחדשנות

מימוש PPO במשחק Space Invaders

Space Invaders PPO

- המימוש של סוכן הלומד לשחק Space Invaders זהה כמעט לחלוטין למימוש של Actor Critic n-step אותו הצגנו בהרצאה הקודמת.
- ההבדל בין השניים מתרכז בשתי נקודות:
  - PPO – עדכון הפרמטרים של רשת ה Actor באמצעות Trust Region.
  - GAE – חישוב ה advantage באמצעות General Advantage Estimation.

- על כן, נציג רק את הפונקציות שהשתנו מהמימוש הקודם.

Learn

- כפי שניתן לראות פונקציית הלימוד דומה מאוד לפונקציה של Actor Critic ה- step.

```python
def learn(self, next_val):
    #region for login···

    state_arr, action_arr, log_prob_arr, val_arr, reward_arr, done_arr = self.memory.get_arrays()

    # Skip learning if too few samples
    if len(state_arr) < 2:···

    # Entropy coefficient decay
    self.entropy_coefficient = max(self.min_entropy_coeff, self.entropy_coefficient * self.entropy_decay_rate)

    # Compute advantages and returns using GAE
    advantage, returns = self.calculate_advantage_and_returns(reward_arr, val_arr, done_arr, next_val)

    for i in range(self.n_epochs):···

    self.memory.clear_memory()

    self.wandb(values = val_arr.mean(), returns = returns.mean(), advantage=advantage.mean(),
               critic_losses= stat.mean(critic_losses), actor_losses=stat.mean(actor_losses),
               total_losses= stat.mean(total_losses), entropy= stat.mean(entropy_values))
    self.critic.scheduler.step()
    self.actor.scheduler.step()

    self.memory.clear_memory()
```

Learn

- השוני מתמקד רק בחישוב ה Actor_loss באמצעות ratio.
- נשים לב לכך ש: $r = \frac{\pi_\theta(a|s)}{\pi_{\theta\ old}(a|s)} = e^{(\ln(\pi_\theta(a|s) - \ln(\pi_{\theta\ old}(a|s)))}$

```python
for i in range(self.n_epochs):
    batches = self.memory.generate_batches()

    for batch in batches:
        # Prepare tensors from the batch indices
        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)
        old_log_probs = T.tensor(log_prob_arr[batch]).to(self.actor.device)

        # Compute the current policy distribution and associated log probabilities
        dist = self.actor(states)
        new_log_probs = dist.log_prob(actions)
        critic_value = self.critic(states).squeeze(-1)

        # Compute the probability ratio
        ratio = T.exp(new_log_probs - old_log_probs)
        # Compute the clipped objective
        surr1 = ratio * batch_advantage
        surr2 = T.clamp(ratio, 1 - self.clip_epsilon, 1 + self.clip_epsilon)\
            * batch_advantage
        actor_loss = -T.min(surr1, surr2).mean()
```

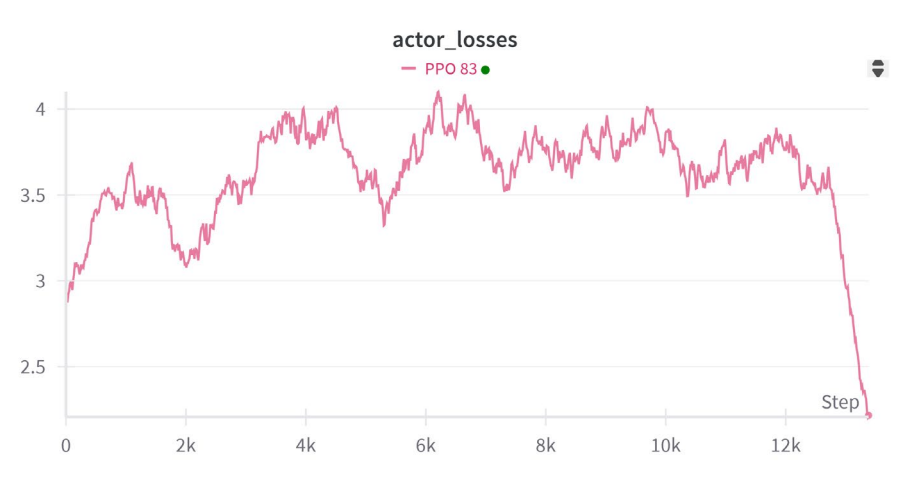
```python
        # Calculate critic loss (value function loss)
        critic_loss = F.mse_loss(critic_value, batch_returns)

        # Entropy bonus for exploration
        entropy = dist.entropy().mean()

        # Combine losses: note that critic_actor_ratio scales the value loss,
        # and the entropy bonus is subtracted (to maximize entropy)
        total_loss = actor_loss + self.critic_actor_ratio * critic_loss \
            - self.entropy_coefficient * entropy

        #region Logging···
        # Perform backward pass and optimization
        self.actor.optimizer.zero_grad()
        self.critic.optimizer.zero_grad()
        total_loss.backward()
        self.actor.optimizer.step()
        self.critic.optimizer.step()
```

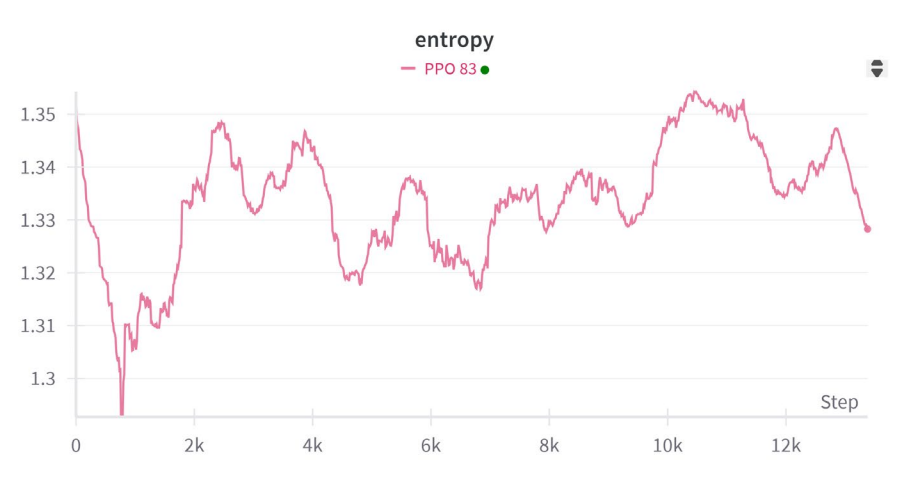
GAE Advantage

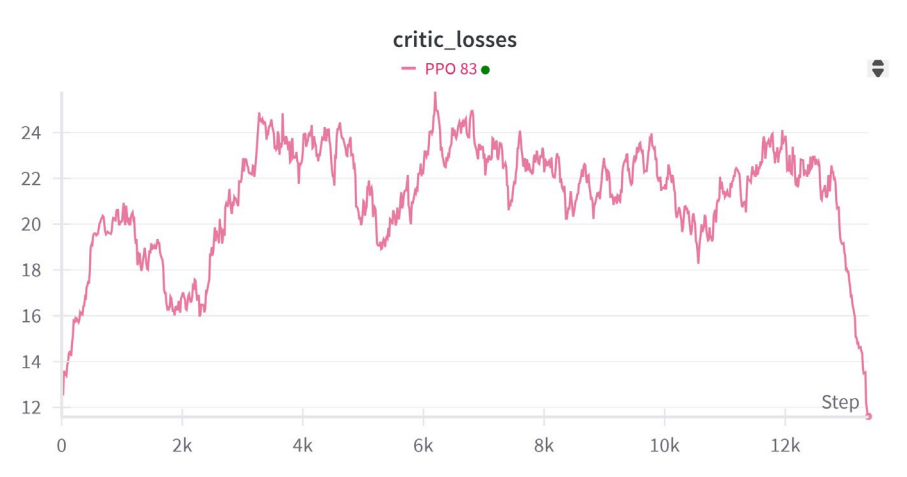
```python
def calculate_advantage_and_returns(self, reward_arr, val_arr, done_arr, next_val):
    advantage = np.zeros_like(reward_arr, dtype=np.float32)
    running_gae = 0

    for t in reversed(range(len(reward_arr))):
        if t == len(reward_arr) - 1:
            done = done_arr[t]
            next_value = next_val
        else:
            done = done_arr[t + 1]
            next_value = val_arr[t + 1]

        delta = reward_arr[t] + self.gamma * next_value * (1 - done) - val_arr[t]
        running_gae = delta + self.gamma * self.lmbda * (1 - done) * running_gae
        advantage[t] = running_gae

    returns = advantage + val_arr

    # Convert to tensors and move to the appropriate device.
    advantage = T.tensor(advantage).to(self.actor.device)
    returns = T.tensor(returns,dtype=T.float32).to(self.actor.device)

    #region Logging for debugging and monitoring.···
    return advantage, returns
```

- חישוב GAE כמו חישוב ה G נעשה מהסוף להתחלה, ומחזיר מערך של A לכל t.
- כיוון שאנו מחשבים n-step אנחנו מוסיפים value(s) ובודקים אם לא הגענו למצב סופי שאז v(s) = 0.
- בנוסף, הפונקציה מקבלת next_val שהינו ערך המצב לאחר המצב האחרון שדגמנו, כדי שנוכל לחשב את δ של המצב האחרון שדגמנו.

תוצאות הרצה 12000 משחקים







PPOG – PPO Gilad Version

- אני מצרף וריאציה של PPO המבצעת את החישוב של ה Actor_loss בצורה ישירה ולא מתחכמת.
- הפונקציה בודקת אם קיימת חריגה ב r ומבצעת את חישוב ה loss בהתאם:
  - אם r אינו חורג מהתחום המותר – חישוב ה Loss יהיה החישוב הרגיל.
  - אם r חורג מהתחום המותר ועדכון הפרמטרים היה בכיוון הנכון – אנחנו נקבע loss=0 כך שהנגזרת תהיה 0, ולא יתבצע כל עדכון.
  - אם r חורג מהתחום המותר ועדכון הפרמטרים לא היה בכיוון הנכון – אנחנו נעדכן את הפרמטרים כדי לתקן את הטעות שעשינו בעדכונים הקודמים. חישוב ה Loss יהיה שווה לחישוב הרגיל.

PPOG - PPO Gilad Version

קריית החינוך פארק המדע
בית לערכים למצוינות ולחדשנות

- השינוי היחידי נעשה בחישוב ה Actor_loss כמופיע בקוד למטה.
- אין הבדך משמעותי בין הגרסאות.

```python
# Compute the current policy distribution and associated log probabilities
dist = self.actor(states)
new_log_probs = dist.log_prob(actions)
critic_value = self.critic(states).squeeze(-1)

# Compute the probability ratio
ratio = T.exp(new_log_probs - old_log_probs)
# Build the condition:
#   Condition 1: r is outside the trusted region
cond1 = (ratio < (1 - self.clip_epsilon)) | (ratio > (1 + self.clip_epsilon))
#   Condition 2: the update is in the "correct" direction:
#                For A > 0, we expect r > 1; for A < 0, we expect r < 1.
cond2 = ((batch_advantage > 0) & (ratio > 1)) | ((batch_advantage < 0) & (ratio < 1))
# Combined condition
condition = cond1 & cond2

# Compute the loss:
#   If the condition is True, set loss = 0.
#   Otherwise, use the standard actor-critic loss A * ln(pi(s,a)).
raw_actor_loss = -batch_advantage * new_log_probs
actor_loss = (T.where(condition, T.zeros_like(batch_advantage), raw_actor_loss)).mean()
```