بسم الله الرّحمن الرّحیم

تمرین یادگیری تقویتی درس کنترل هوشمند در مکاترونیک

اعضای گروه:

محمدامین محتشمی پور، مرتضی جداری، پویان سلیمانی، محمد میرزایی، وانیا ملک زاهدی

نام استاد:

دکتر ایمان شریفی

بهار ۱۴۰۳

دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

فهرست

۱ سوالات تئوری

۱/۱

Sample Efficiency یا همان بهینگی در نمونه گیری به الگوریتمی گفته می‌شود که از هر نمونه بیشترین بهره را ببرد. یعنی بتواند از هر قطعه از تجربه ای که تولید می‌شود استفاده بهینه ببرد و سریعا سیاست خود را بهبود ببخشد.

برخلاف آن، اگر یک الگوریتم از نمونه هایی که دارد تجربیات درستی به دست نیاورد، آن الگوریتم بهینه نخواهد بود.

یکی از روش‌هایی که برای این امر استفاده شده است، نمونه برداری اهمیتی یا Importance sampling می‌باشد که یک روش مونته کارلو برای ارزیابی ویژگی‌های یک توزیع خاص است.

همچنین از دیگر راه‌های افزایش Sample Efficiency استفاده از رویکرد مبتنی بر Meta-Learning می‌باشد که هدف آن این است که مدل بتواند با داده‌های جدید سریع تر سازگار شود و نیاز به نمونه های کمتری داشته باشد.[۱]

به علاووه یکی دیگر از رویکردهای افزایش Sample Efficiency استفاده از Auxiliary Tasks یا همان وظایف کمکی می‌باشد.[۱]

همچنین یکی دیگر از راه‌های افزایش Sample Efficiency استفاده از بافر تجارت یا Replay Buffer می‌باشد.

۱/۲

بافر تجارت (Replay Buffer) یک تکنیک مهم در یادگیری تقویتی است که به افزایش sample efficiency کمک می کند.

در یادگیری تقویتی، عامل (agent) با تعامل با محیط، تجربیات (transitions) مختلفی را به صورت توالی دنباله‌ای (sequence) کسب می‌کند. این تجربیات شامل وضعیت فعلی، اقدام انجام شده، پاداش دریافتی و وضعیت بعدی هستند.

بافر تجارت یک حافظه است که این تجربیات را ذخیره می‌کند. در طول آموزش، عامل به جای استفاده از تجربیات جدید، از این بافر به صورت تصادفی نمونه‌برداری می‌کند و از آنها برای به‌روزرسانی شبکه عصبی استفاده می‌کند.

این رویکرد به چند دلیل باعث افزایش sample efficiency می‌شود:

۱. تکرار تجربیات: استفاده مکرر از تجربیات ذخیره شده در بافر به جای تجربیات جدید، به عامل کمک می‌کند تا از آنها بیشترین استفاده را ببرد.

۲. شکستن همبستگی: در یادگیری تقویتی، توالی تجربیات ممکن است همبستگی داشته باشند. بافر تجارت با نمونه‌برداری تصادفی از تجربیات، این همبستگی را می‌شکند و به پایداری آموزش کمک می‌کند.

۳. تنوع تجربیات: بافر تجارت می‌تواند تجربیات متنوعی را ذخیره کند. این به عامل کمک می‌کند تا مدل خود را به طور کلی‌تری آموزش دهد.

در مجموع، بافر تجارت یک تکنیک قدرتمند برای افزایش sample efficiency در یادگیری تقویتی است که با ذخیره و استفاده مجدد از تجربیات، به عامل کمک می‌کند تا با داده های کمتری بهتر یاد بگیرد.

۱٫۳

در مقاله‌ی انتخاب شده[۲]، یک بازوی رباتیک UR۵ که در بستر نرم افزار MoJoCo شبیه‌سازی شده است قرار است کنترل شود. در واقع مساله‌ی این مقاله، یادگیری رسیدن بازوی ربات به نقطه‌ی هدف با استفاده از یادگیری تقویتی است و از کنترلر دیگری استفاده نشده است. در واقع مانند تمرین عملی درس، یک محیط ساخته شده است و ربات با استفاده از یادگیری تقویتی، آموزش می‌بیند که چگونه به نقطه‌ی مورد نظر رسیده و شیء را بردارد.

همچنین روشاستفاده شده در این مقاله، model free بوده و هیچ گونه مدلی در آموزش وجود ندارد.

در این مقاله از دو الگوریتم Trust Region Policy Optimization (TRPO) و Deep Q-Network with NAF (DQN-NAF) استفاده شده و نتایج آن گزارش شده است و هدف آن مقایسه‌ی این دو الگوریتم با دو الگوریتم دیگر یعنی Deep Deterministic Policy Gradient (DDPG) و Vanilla Policy Gradient (VPG) می‌باشد. سیاست این الگوریتم بهصورت زیر فرموله شده است و با شبکه عصبی عمیق پارامتر شده است.

$$\pi_\theta(a|s) = \frac{1}{\sqrt{(2\pi)^{n_a} \det \Sigma_\theta(s)}} \exp\left(-\frac{1}{2}||a - \mu_\theta(s)||^2_{\Sigma_\theta^{-1}(s)}\right) \quad (1)$$

حالت های این الگوریتم به صورت زیرمی‌باشد:

$$q = \begin{bmatrix} q_{\text{ur5}} \\ q_{\text{grip}} \end{bmatrix} \in \mathbb{R}^{18} \qquad a = \begin{bmatrix} a_{\text{ur5}} \\ a_{\text{grip}} \end{bmatrix} \in \mathbb{R}^{10}$$

$$q_{ur5} = [q_۱\ q_۲\ q_۳\ q_٤ q_۵\ \ q_٦]^T\ , q_{grip} = [q_{۱۰} \dots q_{۱۳} \quad q_{۲۰} \dots q_{۲۳} \quad q_{۳۰} \dots q_{۳۳}]^T,$$
$$a_{ur5} = [m_۱\ m_۲\ m_۳\ m_٤\ m_۵\ m_٦]^T\ , a_{grip} = [m_{..}\ m_{۱۱}\ m_{۱۲}\ m_{۱۳}]^T$$

که q ها، بردارهای موقعیت گریپر و a ها بردارهای عمل (action) گریپر هستند.

دو الگوریتم بیان شده در مقاله به صورت زیر هستند:

Algorithm 1 Trust Region Policy Optimization (TRPO)

Randomly initialize policy parameters θ_0
for $k = 0, 1, 2, \ldots,$ **do**
  Collect set of N_τ trajectories under policy π_{θ_k}
  Estimate advantages with GAE(λ) and fit $V^{\pi_{\theta_k}}$
  Estimate policy gradient

$$\hat{g}_k = \frac{1}{N_\tau} \sum_{i=1}^{N_\tau} \sum_{t=0}^{T-1} \gamma^t \nabla_\theta \log \pi_\theta(a_{t,i}|s_{t,i})\big|_{\theta=\theta_k} \hat{A}(s_{t,i}, a_{t,i})$$

  Estimate $\hat{H}_k = \nabla^2_\theta D_{KL}(\pi_\theta||\pi_{\theta_k})\Big|_{\theta=\theta_k}$
  Compute $\hat{H}_k^{-1}\hat{g}_k$ with CG algorithm
  Compute policy step $\Delta_k = \sqrt{\frac{2\delta_D}{\hat{g}_k^\top \hat{H}_k^{-1} \hat{g}_k}} \hat{H}_k^{-1}\hat{g}_k$
  for $l = 1, 2, \ldots, L$ **do**
    Compute candidate update $\theta^c = \theta_k + \nu^l \Delta_k$
    if $\mathcal{L}_{\pi_{\theta_k}}(\pi_{\theta^c}) \geq 0$ and $D_{KL}(\pi_{\theta^c}||\pi_{\theta_k}) \leq \delta_D$ **then**
      Accept candidate $\theta_{k+1} = \theta^c$
    end if
  end for
end for

Algorithm 2 Deep Q-Network with NAF (DQN-NAF)

Randomly initialize Q and target Q' with $\theta^{Q'} \leftarrow \theta^Q$
Allocate Replay buffer $\mathscr{R}$
for episode $1, \ldots, N_\tau$ **do**
  for $t = 1, \ldots, T$ **do**
    Execute $a_t = \mu_{\theta^\mu}(s_t)$
    Store in $\mathscr{R}$ transition (s_t, a_t, r_t, s_{t+1})
    for iteration $k = 1, \ldots, K_Q$ **do**
      Sample minibatch of N_b transitions from $\mathscr{R}$
      Set targets $y_i = r_i + \gamma V_{\theta^{V'}}(s_{t+1})$
      Update θ^Q by minimizing loss

$$L(\theta^Q) = \frac{1}{N_b} \sum_{i=1}^{N_b} \Big(y_i - Q_{\theta^Q}(s_i, a_i)\Big)^2$$

      Update target network

$$\theta^{Q'} \leftarrow \xi\theta^Q + (1-\xi)\theta^{Q'}$$

    end for
  end for
end for

این مقاله الگوریتم ها را در دو وظیفه‌ی مختلف بررسی کرده است. یکی رسیدن به نقطه تصادفی و دیگری برداشتن و گذاشتن شیء(peak and place).

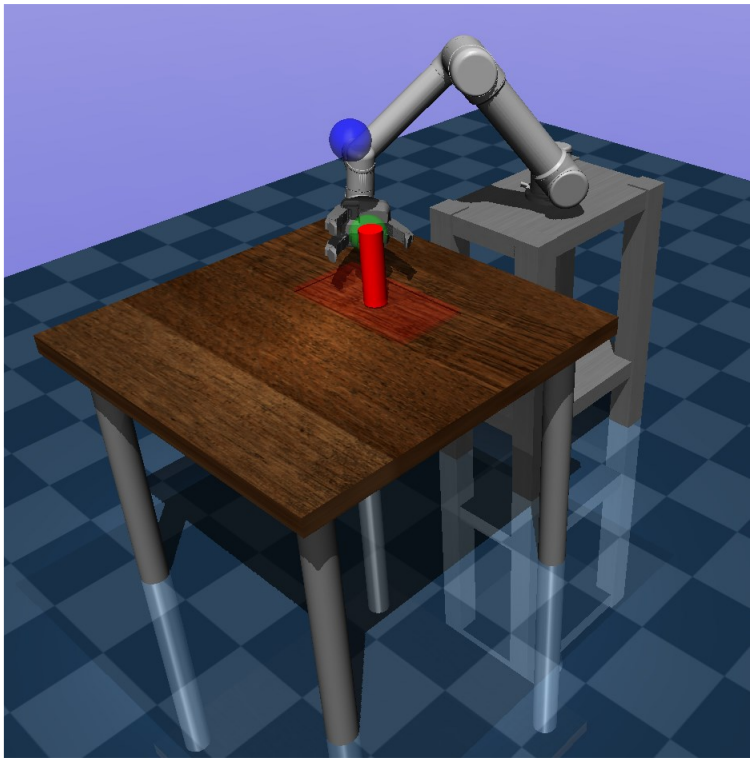
برای رسیدن به نقطه تصادفی، تابع جایزه به صورت زیر تعریف شده است:

$$r_{\mathrm{R}}(s,a) = -||p_{\mathrm{goal}} - p_{\mathrm{ee}}|| - c_a||a||$$

و همچنین برای برداشتن و گذاشتن شیء، تابع جایزه به شکل زیر است:

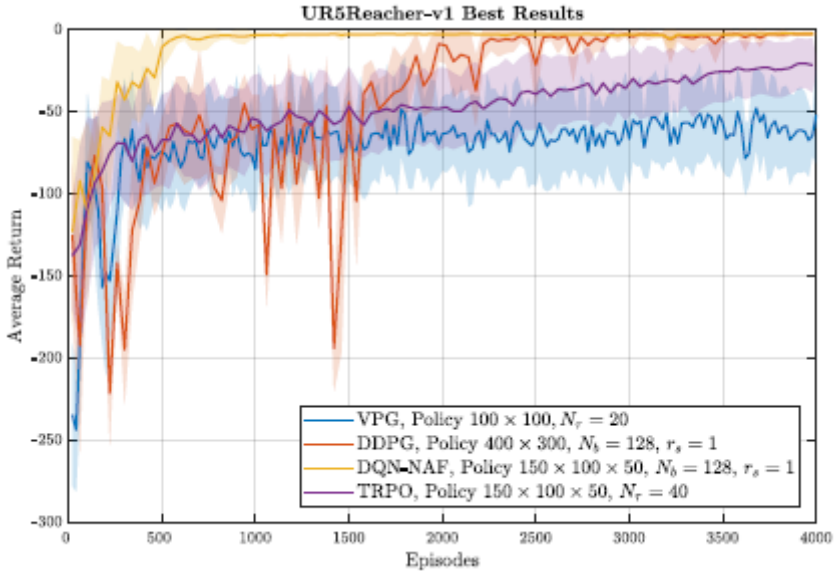
$$r_{\mathrm{P}}(s,a) = \frac{1}{1 + \sum_{j=1}^{3} c_j d_j} - c_a||a||$$

محیط شبیه‌سازی به شکل زیر است:



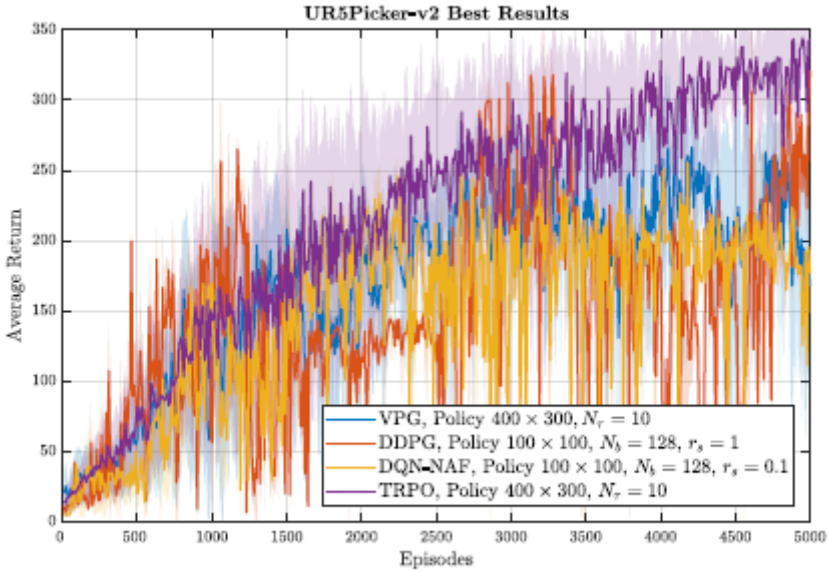
نتایج به دست آمده برای دو الگوریتم پیشنهاد شده و همچنین دو الگوریتم مورد مقایسه در حالت رسیدن به نقطه دلخواه به شکل زیر بیان شده است:

Algorithm	Episodes Req.	Final Avg Return	Max Return
VPG	260	-60.90 ± 24.70	-47.65 ± 24.71
DDPG	2860	-2.83 ± 2.57	-2.13 ± 1.21
DQN-NAF	700	-2.36 ± 1.23	-2.18 ± 1.04
TRPO	2360	-22.87 ± 16.27	-20.27 ± 14.83



همچنین این نتایج برای حالت برداشتن شیء به صورت زیر است:

Algorithm	Episodes Req.	Final Avg Return	Max Return
VPG	740	187.11 ± 70.63	$273.95 \pm 18,21$
DDPG	4950	257.02 ± 12.59	321.24 ± 14.37
DQN-NAF	1980	173.08 ± 9.38	255.39 ± 5.73
TRPO	4980	324.97 ± 43.35	344.01 ± 17.74



[۱] Towards Sample Efficient Reinforcement Learning; Yang Yu

[۲] Robotic Arm Control and Task Training through Deep Reinforcement Learning; Andrea Franceschetti, Elisa Tosello, Nicola Castaman, and Stefano Ghidoni

۲ پروژه عملی

۲٫۱ مقدمه

در این پروژه قرار است یک ربات سه درجه آزادی صفحه‌ای با استفاده از مدل یادگیری تقویتی، برای رسیدن به یک نقطه‌ی خاص آموزش ببیند.

۲٫۲ ساخت محیط

برای ساخت محیط یادگیری، از کتابخانه‌ی gymnasium استفاده شده است. با کمک این کتابخانه، یک کلاس جدیدبه نام ThreeLinkRobotEnvکه از سوپر کلاس gym.Env ارث بری شده است، ساخته شده است و توابع لازم آنها یعنی __init__، reset، step و render فراخوانی شده اند. در تابع step مقادیر مربوط به پاداش قرار داده شده است. پاداش این ربات با استفاده از تابع $\frac{۲۰}{x}$ داده شده است تا با سریع تر به نتیجه برسد. همچنین دینامیک سیستم نیز در این قسمت تعریف شده است. تابعrender نیز برای نمایش نتیجه به صورت شهودی تعریف شده است. کدهای این بخش به شکل زیر هستند:

```
class ThreeLinkRobotEnv(gym.Env):
    def __init__(self , render_mode = 'human'):
        super(ThreeLinkRobotEnv, self).__init__()

        # Define action and observation space
        self.action_space = spaces.Box(low=-۰,۱, high=۰,۱, shape=(۳,),
dtype=np.float۳۲)
        self.observation_space = spaces.Box(low=-np.pi, high=np.pi,
shape=(٦,), dtype=np.float۳۲)

        # Define robot parameters
        self.l۱, self.l۲, self.l۳ = ۱,۰, ۱,۰, ۱,۰
        self.target = np.array([۱,٥, ۱,٥])  # Target position
        self.state = None

        # Reset the environment
        self.reset()

    def reset(self):
        # Set initial state close to the origin
        self.current_step = ۰
        initial_theta = np.random.uniform(low=-np.pi/٤, high=np.pi/٤,
size=۳)
        self.state = np.concatenate([initial_theta, np.zeros(۳)])
        return self.state

    def step(self, action):

        self.current_step += ۱
        theta۱, theta۲, theta۳ = self.state[:۳] + action
        x = (self.l۱ * np.cos(theta۱) + self.l۲ * np.cos(theta۱ +
theta۲) + self.l۳ * np.cos(theta۱ + theta۲ + theta۳))
        y = (self.l۱ * np.sin(theta۱) + self.l۲ * np.sin(theta۱ +
theta۲) + self.l۳ * np.sin(theta۱ + theta۲ + theta۳))

        distance_to_target = np.linalg.norm(self.target - np.array([x,
y]))
        reward = ۲۰/distance_to_target  # Negative distance to
encourage reaching the target
```

```
        # Penalty for large actions to promote smooth movements
        reward -= np.sum(np.abs(action)) * ۰,۳

        self.state = np.array([theta۱, theta۲, theta۳, x, y,
distance_to_target])

        done = distance_to_target < ۰,۱ or self.current_step >= ۲۰۰

        return self.state, reward, done, {}

    def render(self, render_mode="human"):
        theta۱, theta۲, theta۳ = self.state[:۳]
        x۱, y۱ = self.l۱ * np.cos(theta۱), self.l۱ * np.sin(theta۱)
        x۲, y۲ = x۱ + self.l۲ * np.cos(theta۱ + theta۲), y۱ + self.l۲ *
np.sin(theta۱ + theta۲)
        x۳, y۳ = x۲ + self.l۳ * np.cos(theta۱ + theta۲ + theta۳), y۲ +
self.l۳ * np.sin(theta۱ + theta۲ + theta۳)

        plt.figure()
        plt.plot([۰, x۱], [۰, y۱], 'ro-')
        plt.plot([x۱, x۲], [y۱, y۲], 'go-')
        plt.plot([x۲, x۳], [y۲, y۳], 'bo-')
        plt.plot(self.target[۰], self.target[۱], 'kx')
        plt.xlim(-۳, ۳)
        plt.ylim(-۳, ۳)
        plt.show()
```

۲٫۳ یادگیری محیط

برای یادگیری محیط از الگوریتم یادگیری تقویتی PPO استفاده شده است. محیط با استفاده از این الگوریتم ۵ بار اجرا شده است. هر اپیزود نیز نمیتواند بیش از ۲۰۰ حرکت انجام دهد تا سرعت یادگیری بیشتر شود. برای این کار از کدهای زیر استفاده شده است:

```
env = TimeLimit(env, max_episode_steps=۲۰۰)
env = make_vec_env(lambda: env, n_envs=۱)
```

همچنین برای رسم نمودارها نیاز به مقادیر جایزه‌ها داریم. برای این کار، یک کلاس call back با نامRewardCallback تعریف شده است و یک شیء از آن به تابع learn داده شده است. به صورت زیر:

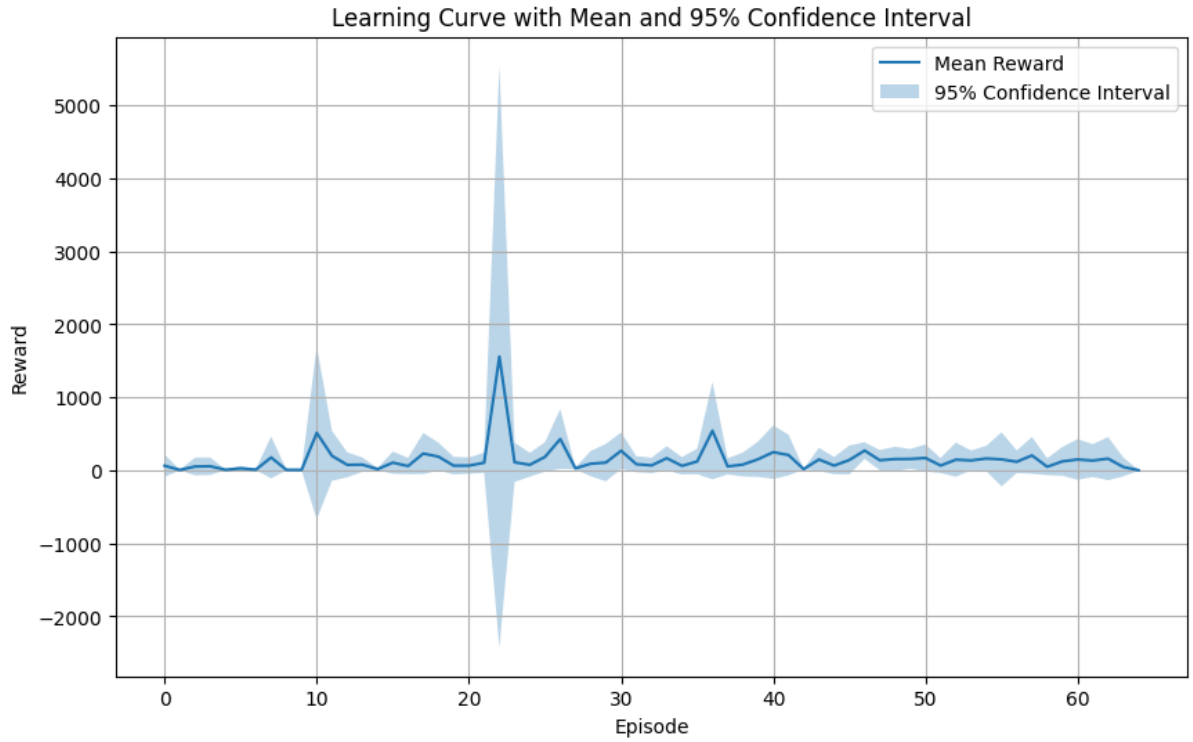
```
class RewardCallback(BaseCallback):
    def __init__(self, verbose=۰):
        super(RewardCallback, self).__init__(verbose)
        self.episode_rewards = []

    def _on_step(self) -> bool:
        if self.locals['dones'][۰]:
```

```
            self.episode_rewards.append(self.locals['rewards'][۰])
        return True
reward_callback = RewardCallback()
    callback = [reward_callback]
    model.learn(total_timesteps=۱۰۰۰۰, callback=callback)
    all_rewards.append(reward_callback.episode_rewards)
```

کدهای مربوط به بخش یادگیری نیز به صورت زیر هستند:

```
env = ThreeLinkRobotEnv()
env = TimeLimit(env, max_episode_steps=۲۰۰)
env = make_vec_env(lambda: env, n_envs=۱)

# Define the number of runs
num_runs = ۵
all_rewards = []

# Run the training and collect rewards
for run in range(num_runs):
    model = PPO('MlpPolicy', env, verbose=۱)

    reward_callback = RewardCallback()
    callback = [reward_callback]
    model.learn(total_timesteps=۱۰۰۰۰, callback=callback)
    all_rewards.append(reward_callback.episode_rewards)

# Print the collected rewards
for run in range(num_runs):
    print(f"Run {run+۱} rewards: {all_rewards[run]}")

# Save the model
model.save("ppo_planar۳dof")
```

نمودار Learning Curve with Mean and ۹۵٪ Confidence Interval برای یادگیری به صورت زیر به دست می‌آید:

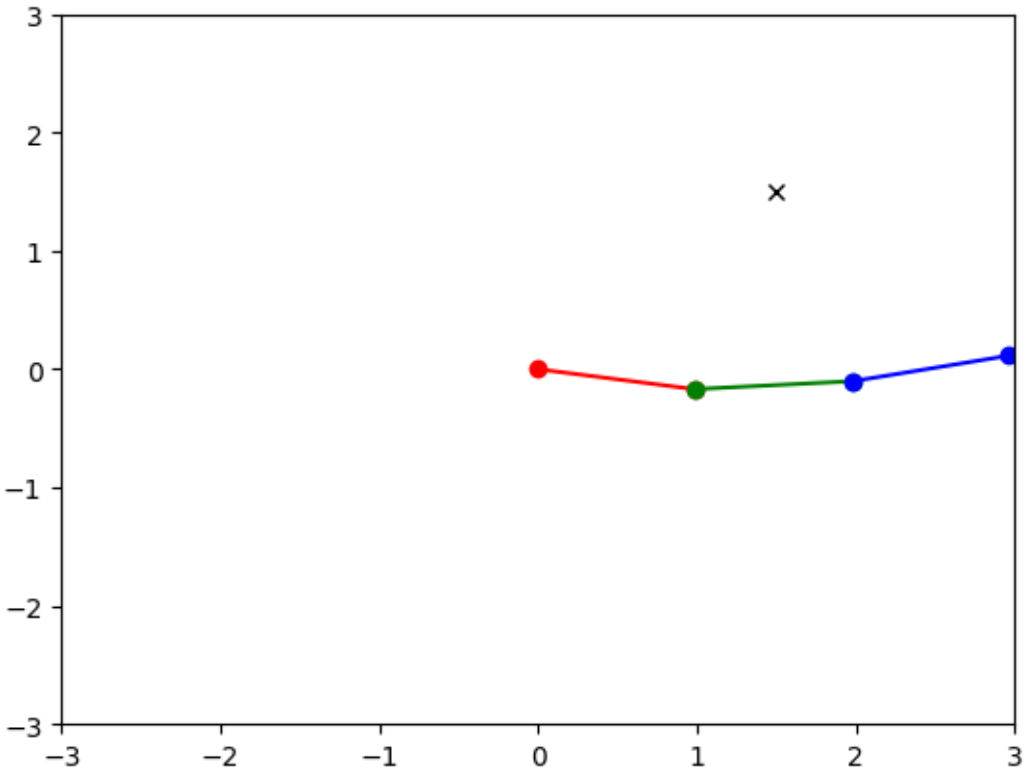


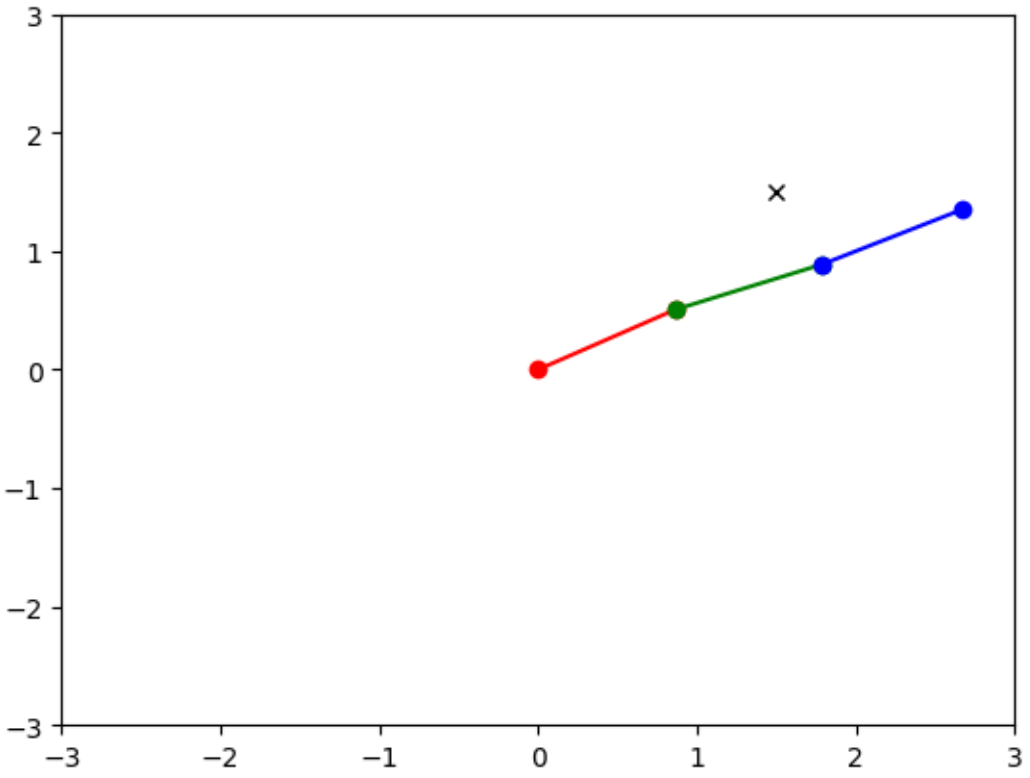
۲٫٤ رسم مدل

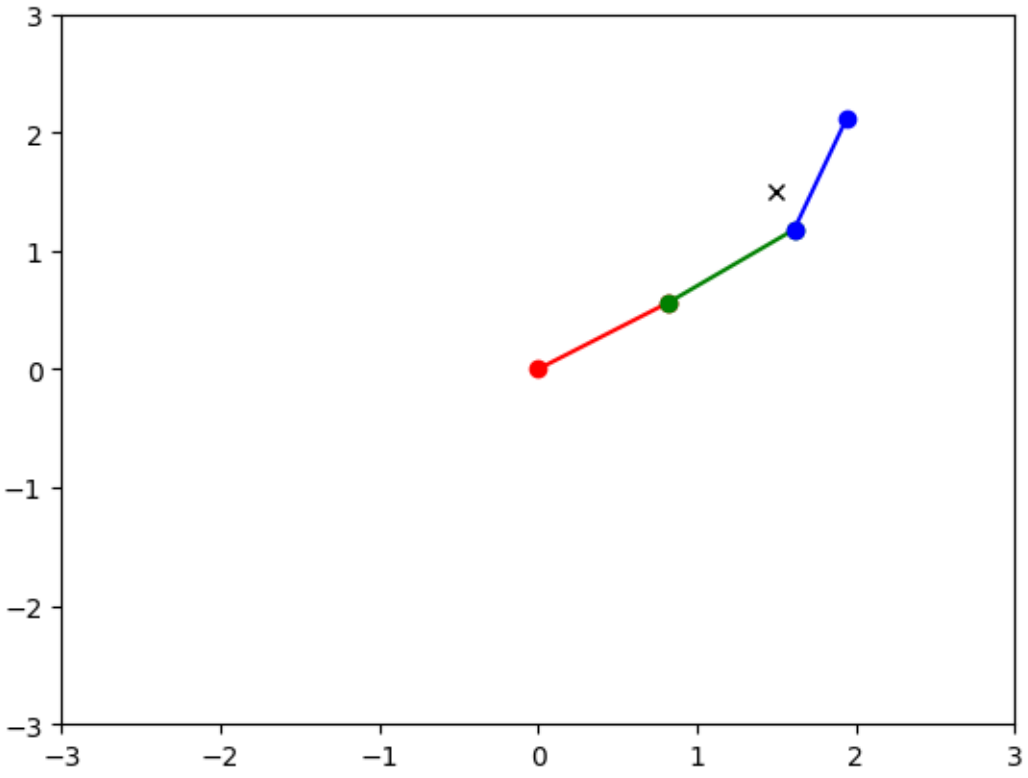
پس ازیادگیری مدل، نمونه‌ی اجرا شده‌ی آن روی ربات مذکور را با کد زیر نمایش می‌دهیم:

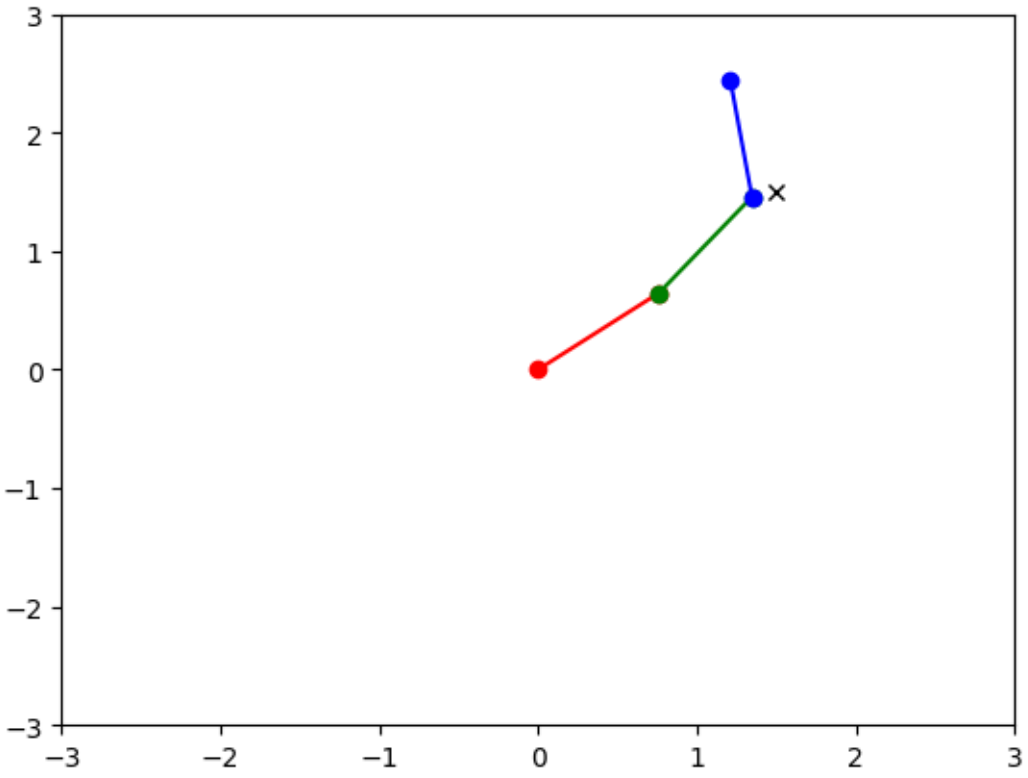
```
model = PPO.load("ppo_planar۳dof")


env = ThreeLinkRobotEnv()

# Test the trained model
obs = env.reset()
for i in range(۱۰۰۰):
    action, _states = model.predict(obs)
    obs, rewards, dones, _ = env.step(action)
    env.render()

    # Stop the training if the target is reached
    if dones:
        print("Target reached!")
        break
```

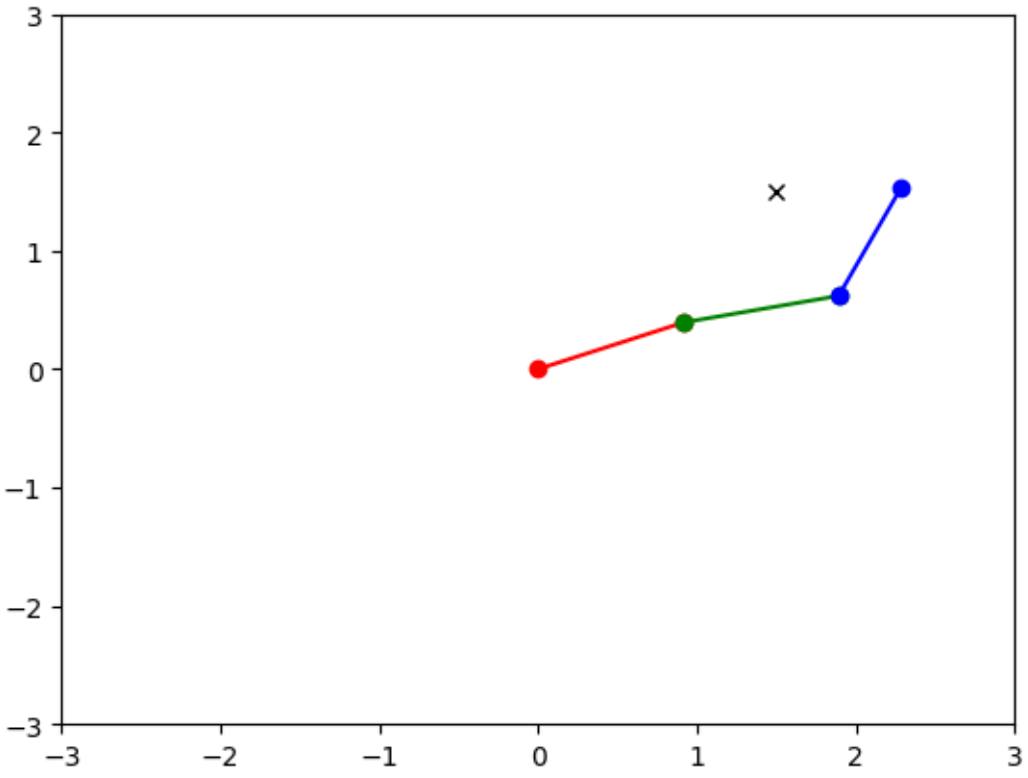
برخی از تصاویر به عنوان نمونه در زیر آورده خواهند شد:

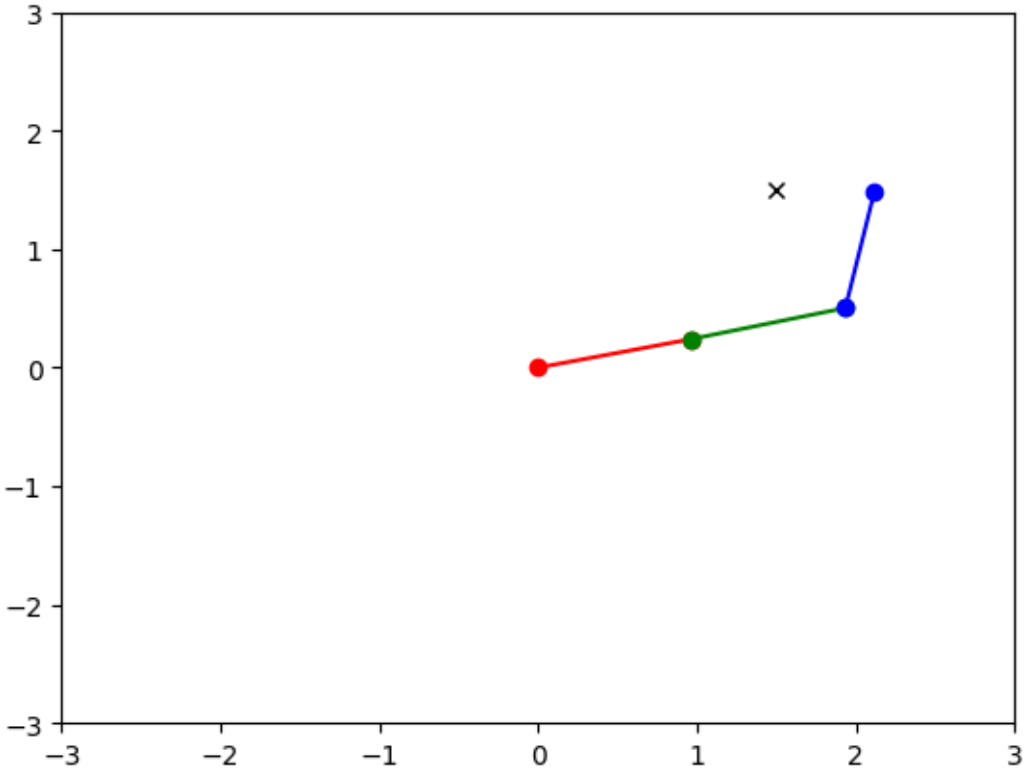

3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3

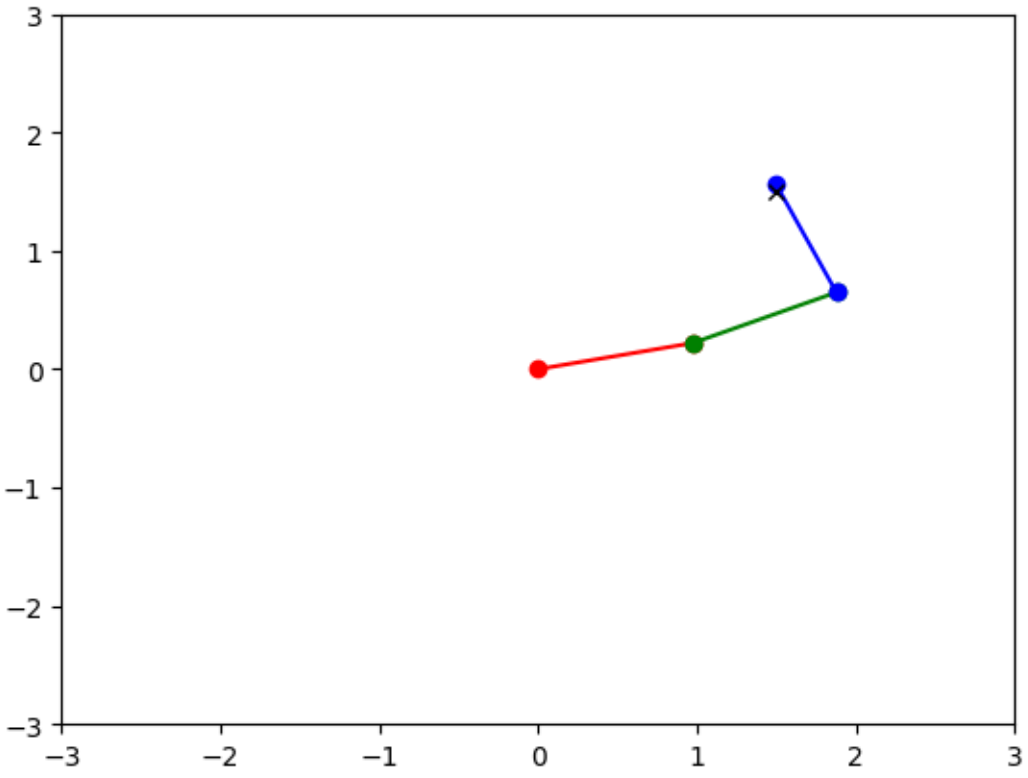

3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3


3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3


3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3


3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3


3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3


3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3