Faculty of Technology and Engineering

Chandubhai S. Patel Institute of Technology

Date: / /

Practical Performa

Academic Year	:	2025-26	Semester	:	7th
Course code	:	OCCSE4001	Course name	:	Reinforcement Learning

Practical- No. 6

Aim: To implement the Vanilla Actor-Critic (A2C) algorithm by combining value-based and policy-based methods, and evaluate its performance on the CartPole-v1 environment.

Code:

```
# import gym # Replace gym with gymnasium
import gymnasium as gym # Import gymnasium as gym
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
import matplotlib.pyplot as plt

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

```
/usr/local/lib/python3.12/dist-packages/jupyter_client/session.py:203:
  return datetime.utcnow().replace(tzinfo=utc)
```

```
class Actor(nn.Module):
    def __init__(self, state_dim, action_dim):
        super().__init__()
        self.fc1 = nn.Linear(state_dim, 128)
        self.fc2 = nn.Linear(128, action_dim)

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        return torch.softmax(self.fc2(x), dim=-1)

class Critic(nn.Module):
    def __init__(self, state_dim):
        super().__init__()
        self.fc1 = nn.Linear(state_dim, 128)
        self.fc2 = nn.Linear(128, 1)

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        return self.fc2(x)
```

RL_Prac6.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text ▷ Run all

[3] 0s

```python
class A2CAgent:
    def __init__(self, state_dim, action_dim, lr=1e-3, gamma=0.99):
        self.actor = Actor(state_dim, action_dim).to(device)
        self.critic = Critic(state_dim).to(device)

        self.actor_optimizer = optim.Adam(self.actor.parameters(), lr=lr)
        self.critic_optimizer = optim.Adam(self.critic.parameters(), lr=lr)

        self.gamma = gamma

    def select_action(self, state):
        state = torch.FloatTensor(state).unsqueeze(0).to(device)
        probs = self.actor(state)
        dist = torch.distributions.Categorical(probs)
        action = dist.sample()
        return action.item(), dist.log_prob(action)

    def update(self, state, reward, next_state, done, log_prob):
        state = torch.FloatTensor(state).unsqueeze(0).to(device)
        next_state = torch.FloatTensor(next_state).unsqueeze(0).to(device)

        # Compute TD target & advantage
        td_target = reward + (1 - done) * self.gamma * self.critic(next_state)
        td_error = td_target - self.critic(state)

        # Actor loss (policy gradient with advantage)
        actor_loss = -log_prob * td_error.detach()

        # Critic loss (MSE on value estimate)
        critic_loss = td_error.pow(2)

        # Optimize actor
        self.actor_optimizer.zero_grad()
        actor_loss.backward()
        self.actor_optimizer.step()
```

RL_Prac6.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text ▷ Run all

[3] 0s

```python
        # Optimize critic
        self.critic_optimizer.zero_grad()
        critic_loss.backward()
        self.critic_optimizer.step()
```

[6] 2m

```python
env = gym.make("CartPole-v1")
state_dim = env.observation_space.shape[0]
action_dim = env.action_space.n

agent = A2CAgent(state_dim, action_dim)

MAX_EPISODES = 500
MAX_STEPS = 500
rewards = []

for ep in range(1, MAX_EPISODES+1):
    # state, _ = env.reset() # Modified to unpack the new return value
    state, info = env.reset()
    ep_reward = 0

    for t in range(MAX_STEPS):
        action, log_prob = agent.select_action(state)
        # next_state, reward, terminated, truncated, _ = env.step(action) # Modified to unpack the new return value
        next_state, reward, terminated, truncated, info = env.step(action)
        done = terminated or truncated

        agent.update(state, reward, next_state, done, log_prob)

        state = next_state
        ep_reward += reward
        if done:
            break
```

RL_Prac6.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

[6]

```
    rewards.append(ep_reward)
    print(f"Episode {ep}, Reward: {ep_reward}")
```

Show hidden output

[7]

```
plt.figure(figsize=(8,5))
plt.plot(rewards, label="Episode reward")
plt.plot(np.convolve(rewards, np.ones(20)/20, mode='valid'), label="Avg(20)")
plt.xlabel("Episode")
plt.ylabel("Reward")
plt.title("A2C (Vanilla Actor-Critic) on CartPole-v1")
plt.legend()
plt.show()
```

RL_Prac6.ipynb

File Edit View Insert Runtime Tools Help

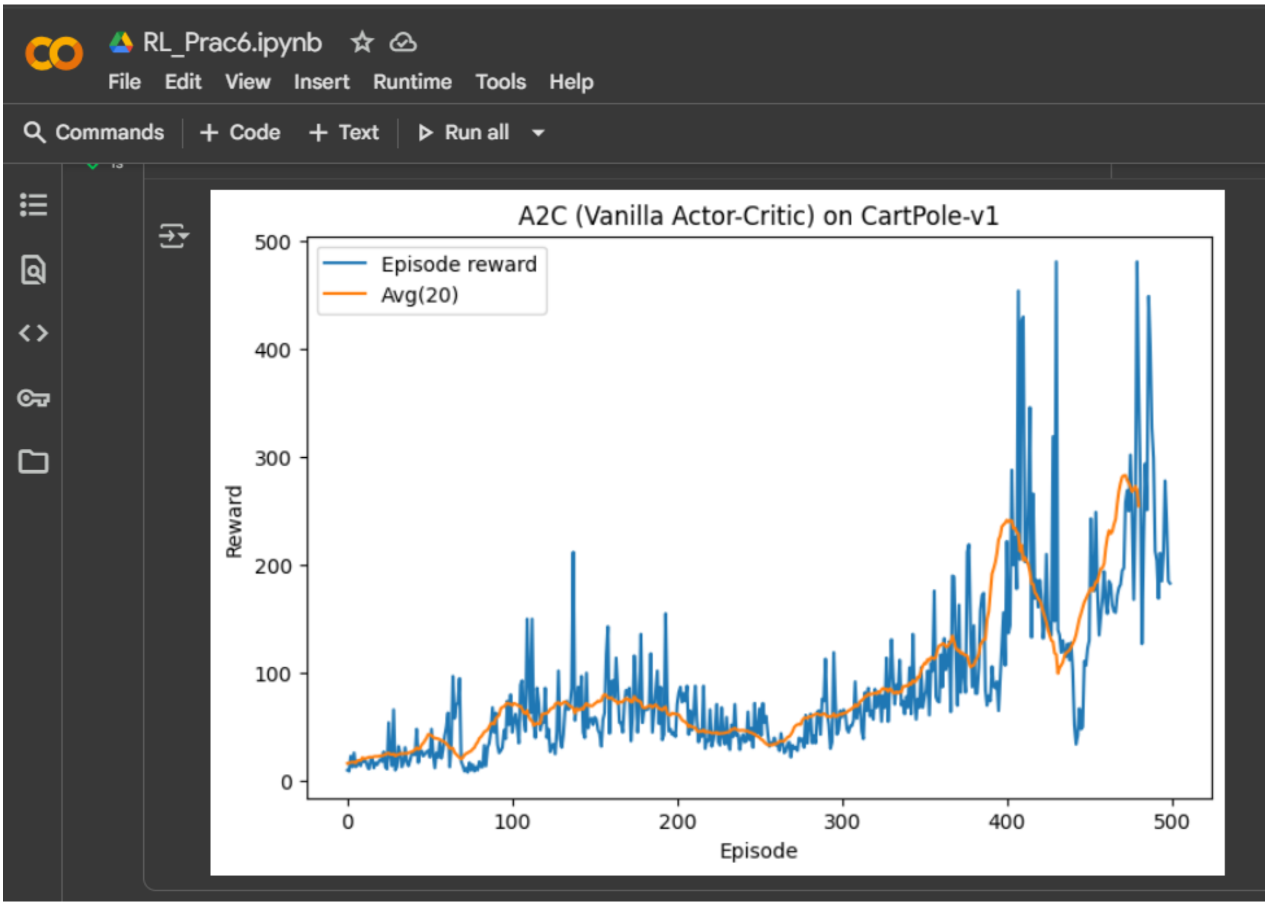
Commands + Code + Text Run all

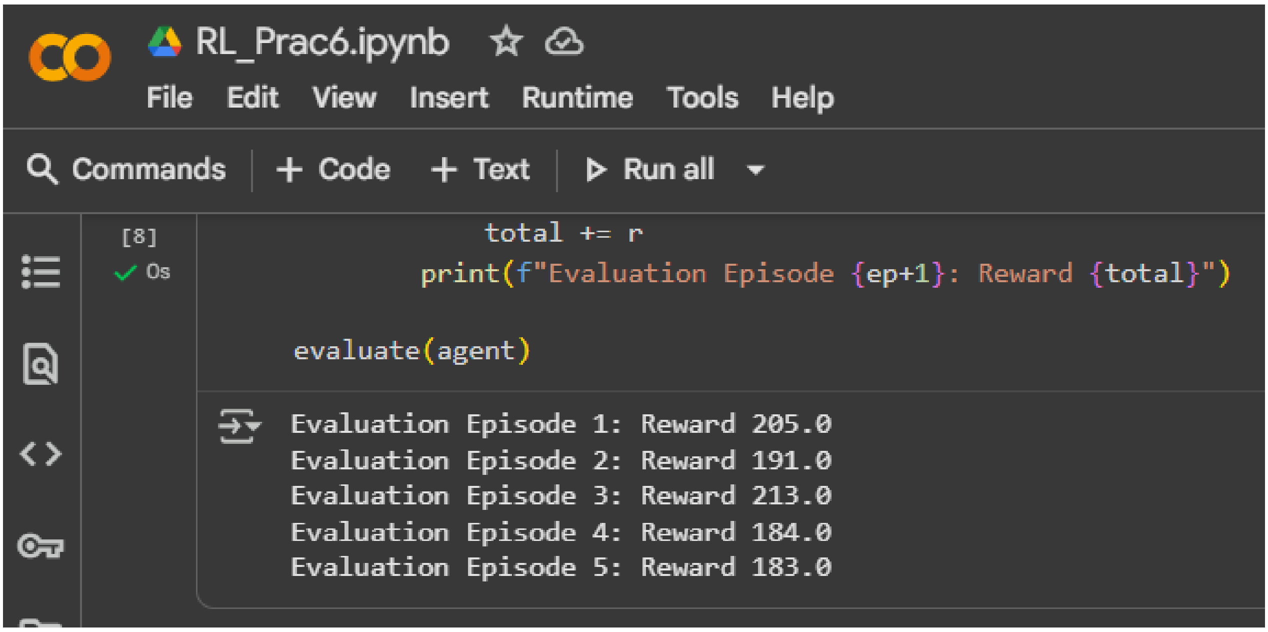
[8]

```
def evaluate(agent, episodes=5):
    for ep in range(episodes):
        s, _ = env.reset()
        total = 0
        done = False
        while not done:
            state = torch.FloatTensor(s).unsqueeze(0).to(device)
            probs = agent.actor(state)
            action = probs.argmax().item()  # greedy
            s, r, terminated, truncated, _ = env.step(action)
            done = terminated or truncated
            total += r
        print(f"Evaluation Episode {ep+1}: Reward {total}")

evaluate(agent)
```

Output:





Grade/Marks **Sign of Lab Teacher with Date**

(_____ / 10)