



CHARUSAT
CHAROTAR UNIVERSITY OF SCIENCE AND TECHNOLOGY



Faculty of Technology and Engineering

Chandubhai S. Patel Institute of Technology

Date: / /

Practical Performa

Academic Year	:	2025-26	Semester	:	7 th
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:

```

[5]
✓ Os
# 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)

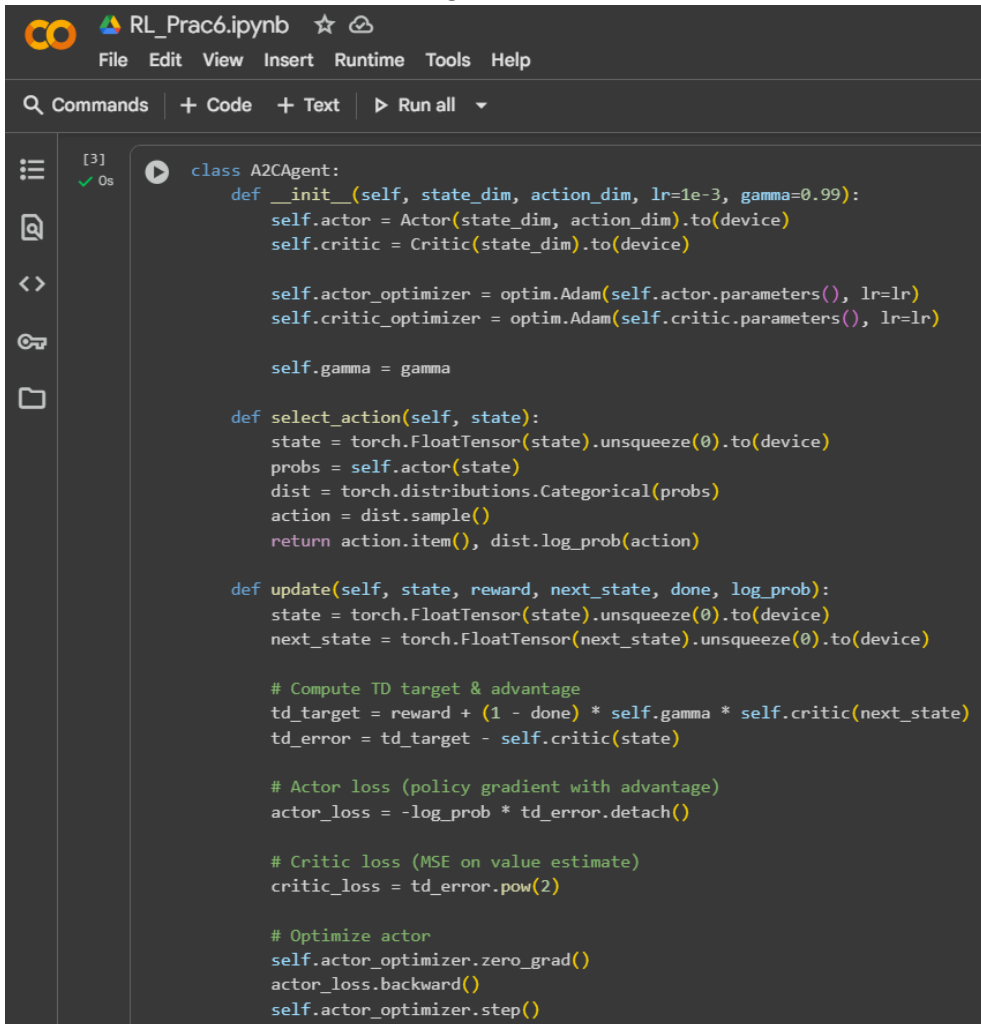
[2]
✓ Os
class Actor(nn.Module):
    def __init__(self, state_dim, action_dim):
        super().__init__()
        self.fc1 = nn.Linear(state_dim, 128)
        self.fc2 = nn.Linear(128, action_dim)

    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)

```



The image shows a Jupyter Notebook interface with the file 'RL_Prac6.ipynb'. The code cell [3] is executed, showing the definition of the A2C Agent class. The class has two main methods: `select_action` and `update`. The `__init__` method initializes the actor and critic networks, optimizers, and the discount factor `gamma`. The `select_action` method takes a state and returns an action based on the actor's output. The `update` method takes a state, reward, next state, and done flag, and updates the actor and critic networks based on the current state and action.

```
[3]
✓ 0s
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)

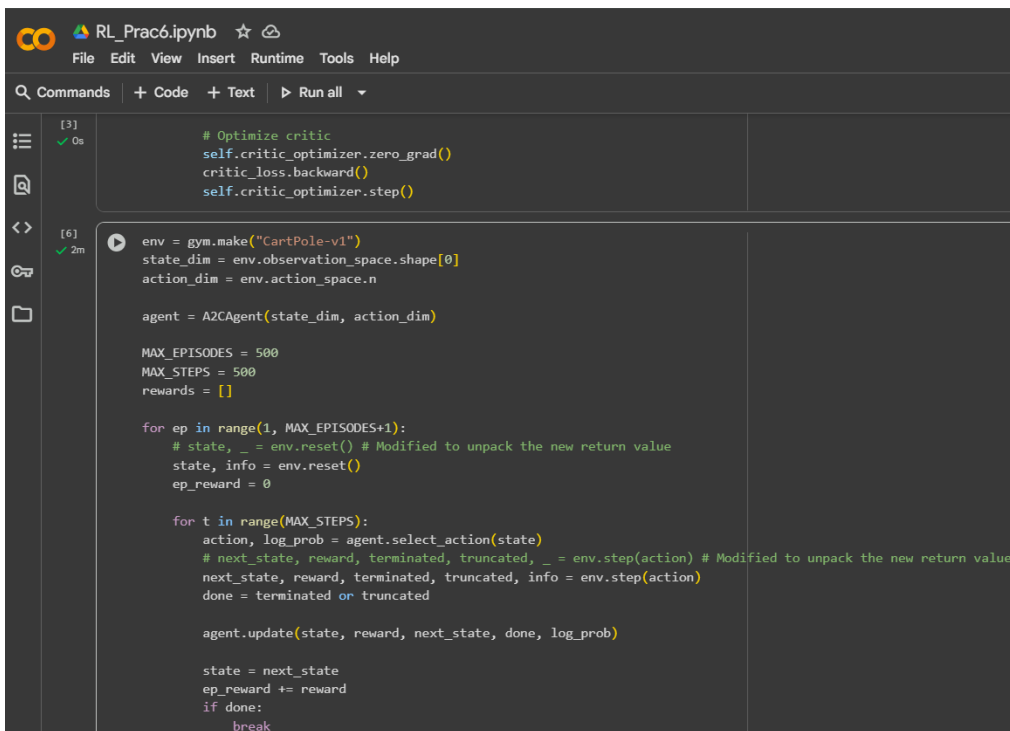
    def update(self, state, reward, next_state, done, log_prob):
        state = torch.FloatTensor(state).unsqueeze(0).to(device)
        next_state = torch.FloatTensor(next_state).unsqueeze(0).to(device)

        # 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()
```



The image shows a Jupyter Notebook interface with the file 'RL_Prac6.ipynb'. The code cell [6] is executed, showing the training loop for the A2C Agent. The loop consists of `MAX_EPISODES` episodes, each with `MAX_STEPS` steps. In each episode, the environment is reset, and the agent is trained for `MAX_STEPS` steps. The agent's action is selected based on the current state, and the environment is stepped forward. The agent's state, reward, next state, and done flag are updated, and the agent's update method is called. The episode's reward is accumulated, and the episode is reset when it is done.

```
[6]
✓ 2m
# Optimize critic
self.critic_optimizer.zero_grad()
critic_loss.backward()
self.critic_optimizer.step()

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

agent = A2CAgent(state_dim, action_dim)

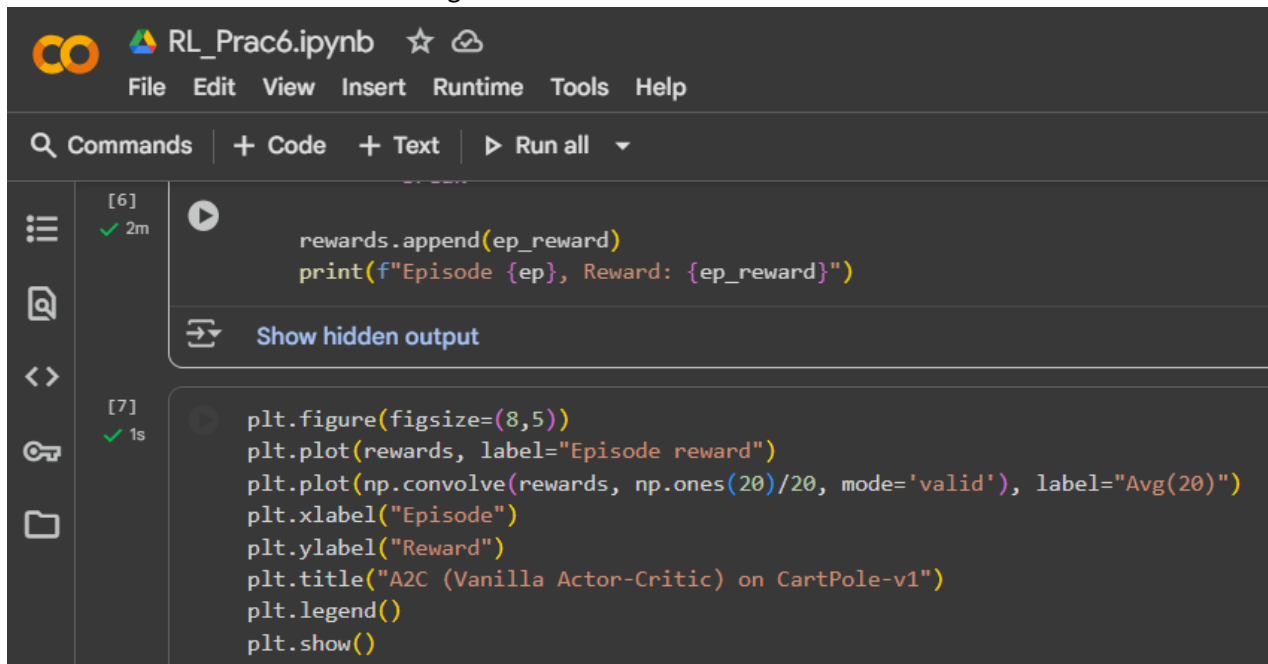
MAX_EPISODES = 500
MAX_STEPS = 500
rewards = []

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
```

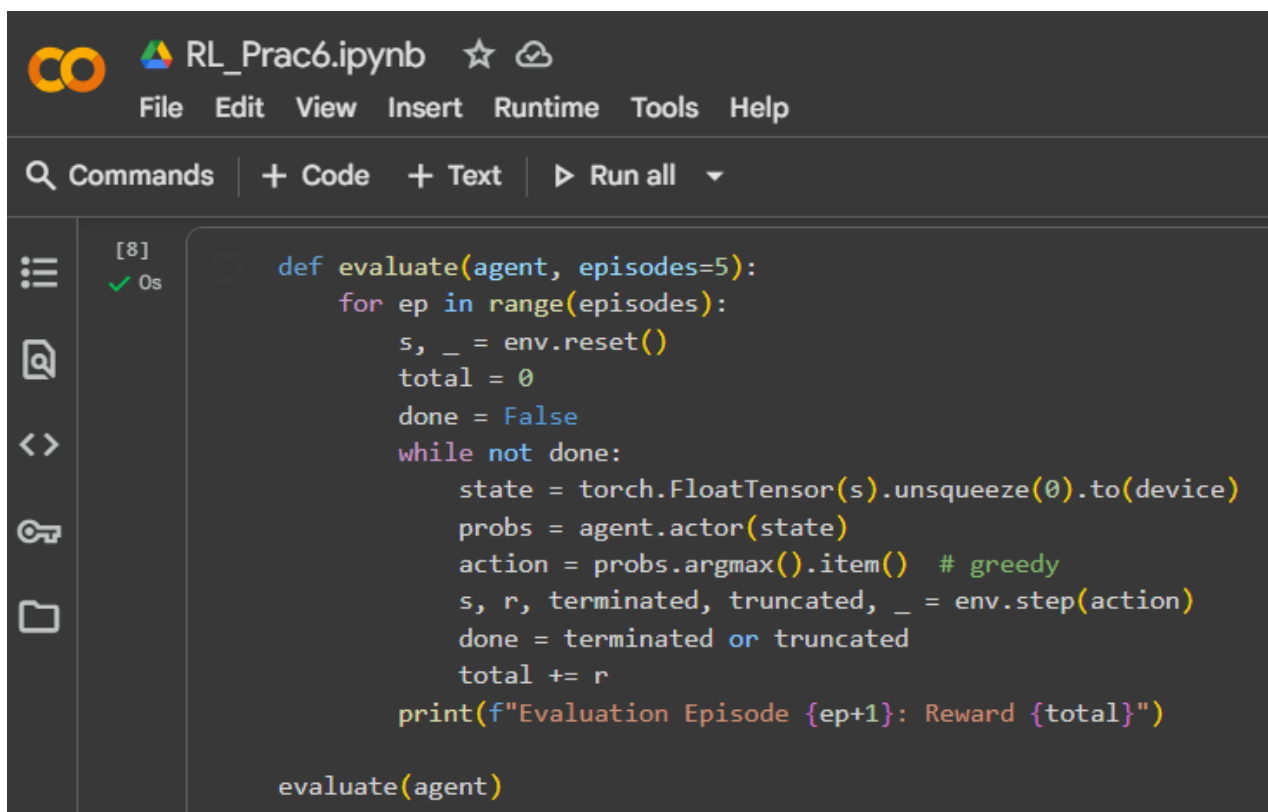


The image shows a Jupyter Notebook interface with a dark theme. The top bar includes the Colab logo, the filename 'RL_Prac6.ipynb', and a star icon. Below the bar is a menu with 'File', 'Edit', 'View', 'Insert', 'Runtime', 'Tools', and 'Help'. A search bar and command palette are visible, showing 'Commands', '+ Code', '+ Text', and 'Run all'. The notebook has two cells. Cell [6] contains code to append episode rewards and print them. Cell [7] contains code to plot the rewards and their average over 20 episodes. The left sidebar shows icons for the table of contents, search, navigation, and file explorer.

```
[6]
✓ 2m
rewards.append(ep_reward)
print(f"Episode {ep}, Reward: {ep_reward}")

Show hidden output

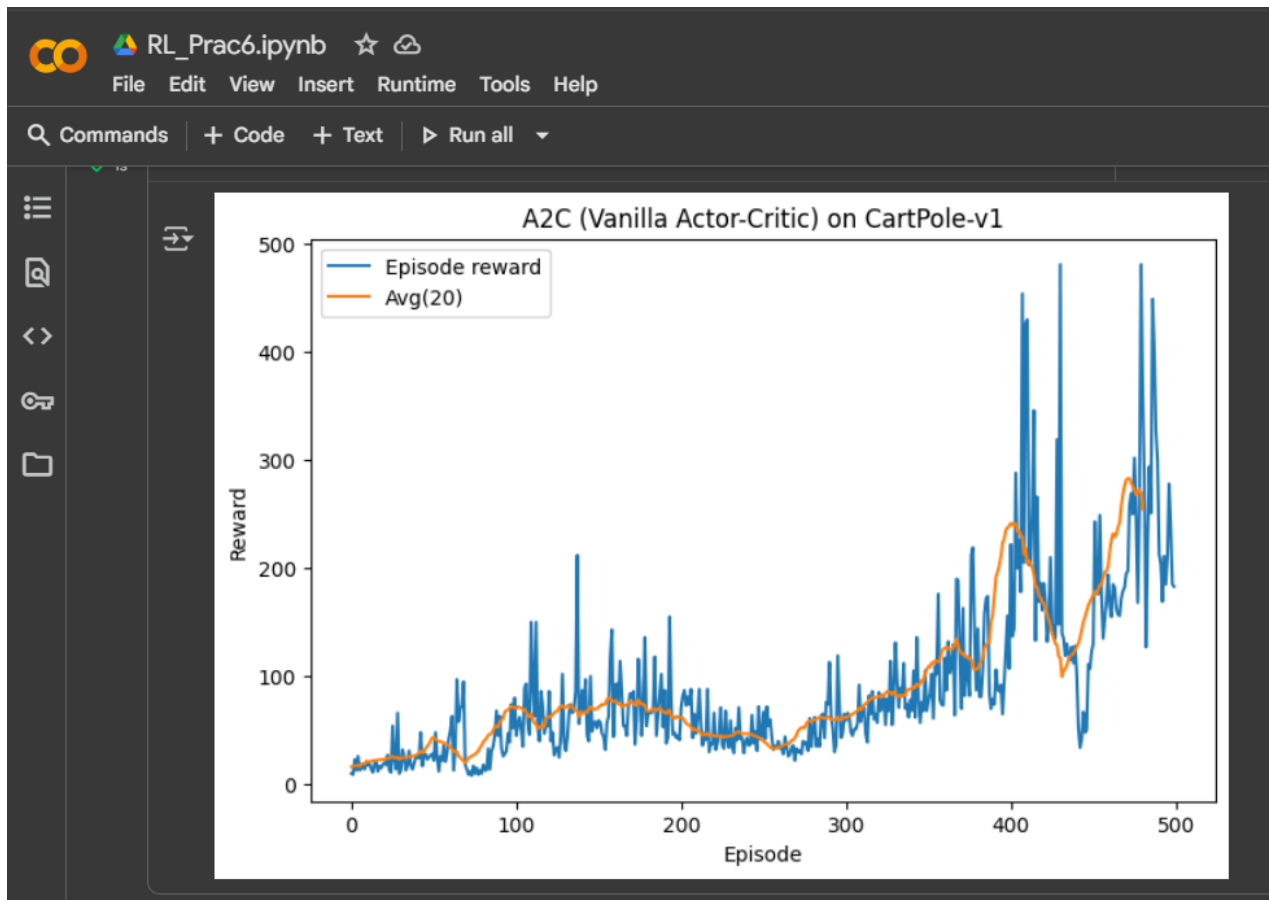
[7]
✓ 1s
plt.figure(figsize=(8,5))
plt.plot(rewards, label="Episode reward")
plt.plot(np.convolve(rewards, np.ones(20)/20, mode='valid'), label="Avg(20)")
plt.xlabel("Episode")
plt.ylabel("Reward")
plt.title("A2C (Vanilla Actor-Critic) on CartPole-v1")
plt.legend()
plt.show()
```



The image shows a Jupyter Notebook interface with a dark theme. The top bar includes the Colab logo, the filename 'RL_Prac6.ipynb', and a star icon. Below the bar is a menu with 'File', 'Edit', 'View', 'Insert', 'Runtime', 'Tools', and 'Help'. A search bar and command palette are visible, showing 'Commands', '+ Code', '+ Text', and 'Run all'. The notebook has one cell [8] containing code to evaluate the agent by running a fixed number of episodes and printing the total reward for each episode. The left sidebar shows icons for the table of contents, search, navigation, and file explorer.

```
[8]
✓ 0s
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:

```
total += r
print(f"Evaluation Episode {ep+1}: Reward {total}")

evaluate(agent)
```

```
Evaluation Episode 1: Reward 205.0
Evaluation Episode 2: Reward 191.0
Evaluation Episode 3: Reward 213.0
Evaluation Episode 4: Reward 184.0
Evaluation Episode 5: Reward 183.0
```

Grade/Marks

(____ / 10)

Sign of Lab Teacher with Date