МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
им. Н.Э. Баумана

Факультет «Информатика и системы управления»
Кафедра «Систем обработки информации и управления»

ОТЧЕТ

Лабораторная работа №__7__
по дисциплине «Методы машинного обучения»

Тема: «Алгоритмы Actor-Critic.»

ИСПОЛНИТЕЛЬ: Лу Сяои
ФИО

группа ИУ5И-22М
____
подпись

"10" __Июнь__ 2023 г.

ПРЕПОДАВАТЕЛЬ: ______________
ФИО

____
подпись

"__"________2023 г.

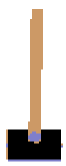
Москва - 2023

описание задания

Реализуйте любой алгоритм семейства Actor-Critic для произвольной среды.

текст программы и экранные формы с примерами выполнения программы.

Я выбрала среду **CartPole.**



В среде со стержнем находится небольшой автомобиль, и задача интеллектуального тела - удерживать стержень на автомобиле в вертикальном положении, перемещая его влево и вправо; игра заканчивается, если стержень наклоняется слишком сильно, или если автомобиль слишком сильно отклоняется от своего начального положения влево и вправо, или если время сохранения достигает 200 кадров. Состояние интеллектуального тела - это вектор размерности 4, каждое измерение которого непрерывно, а действия дискретны, с пространством действий размера 2.

За каждый кадр, удерживаемый в игре, интеллектуальное тело получает вознаграждение в виде 1 балла. Чем дольше время удержания, тем выше итоговый балл, а наивысший балл достигается при удержании в течение 200 кадров.

Observation

Type: Box(4)

Num	Observation	Min	Max
0	Cart Position	-2.4	2.4
1	Cart Velocity	-Inf	Inf
2	Pole Angle	~ -0.418 rad (-24°)	~ 0.418 rad (24°)
3	Pole Velocity At Tip	-Inf	Inf

Actions

Type: Discrete(2)

Num	Action
0	Push cart to the left
1	Push cart to the right

Note: The amount the velocity is reduced or increased is not fixed as it depends on the angle the pole is pointing. This is because the center of gravity of the pole increases the amount of energy needed to move the cart underneath it

Reward

Reward is 1 for every step taken, including the termination step. The threshold is 475 for v1.

Starting State

All observations are assigned a uniform random value between ±0.05.

Episode Termination

Pole Angle is more than ±12°

Cart Position is more than ±2.4 (center of the cart reaches the edge of the display)

Episode length is greater than 200 (500 for v1).

Solved Requirements

Considered solved when the average reward is greater than or equal to 195.0 over 100 consecutive trials.

```
import gym
import torch
import torch.nn.functional as F
import numpy as np
import matplotlib.pyplot as plt
import rl_utils
```

```
class PolicyNet(torch.nn.Module):
    def __init__(self, state_dim, hidden_dim, action_dim):
        super(PolicyNet, self).__init__()
        self.fc1 = torch.nn.Linear(state_dim, hidden_dim)
        self.fc2 = torch.nn.Linear(hidden_dim, action_dim)

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

```
class ValueNet(torch.nn.Module):
    def __init__(self, state_dim, hidden_dim):
        super(ValueNet, self).__init__()
        self.fc1 = torch.nn.Linear(state_dim, hidden_dim)
        self.fc2 = torch.nn.Linear(hidden_dim, 1)

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

```
class ActorCritic:
    def __init__(self, state_dim, hidden_dim, action_dim, actor_lr,
critic_lr,
                 gamma, device):
        # 策略网络
        self.actor = PolicyNet(state_dim, hidden_dim,
action_dim).to(device)
        self.critic = ValueNet(state_dim, hidden_dim).to(device)  # 价
值网络
        # 策略网络优化器
        self.actor_optimizer =
torch.optim.Adam(self.actor.parameters(),
                                                lr=actor_lr)
        self.critic_optimizer =
torch.optim.Adam(self.critic.parameters(),
```

```
                                                   lr=critic_lr)  # 价值
网络优化器
        self.gamma = gamma
        self.device = device

    def take_action(self, state):
        state = torch.tensor([state],
dtype=torch.float).to(self.device)
        probs = self.actor(state)
        action_dist = torch.distributions.Categorical(probs)
        action = action_dist.sample()
        return action.item()
```

```
    def update(self, transition_dict):
        states = torch.tensor(transition_dict['states'],
                              dtype=torch.float).to(self.device)
        actions = torch.tensor(transition_dict['actions']).view(-1,
1).to(
            self.device)
        rewards = torch.tensor(transition_dict['rewards'],
                               dtype=torch.float).view(-1,
1).to(self.device)
        next_states = torch.tensor(transition_dict['next_states'],
                                   dtype=torch.float).to(self.device)
        dones = torch.tensor(transition_dict['dones'],
                             dtype=torch.float).view(-1,
1).to(self.device)
```

```
        # 时序差分目标
        td_target = rewards + self.gamma * self.critic(next_states) *
(1 -

 dones)
        td_delta = td_target - self.critic(states)  # 时序差分误差
        log_probs = torch.log(self.actor(states).gather(1, actions))
        actor_loss = torch.mean(-log_probs * td_delta.detach())
        # 均方误差损失函数
        critic_loss = torch.mean(
            F.mse_loss(self.critic(states), td_target.detach()))
        self.actor_optimizer.zero_grad()
        self.critic_optimizer.zero_grad()
        actor_loss.backward()  # 计算策略网络的梯度
        critic_loss.backward()  # 计算价值网络的梯度
        self.actor_optimizer.step()  # 更新策略网络的参数
        self.critic_optimizer.step()  # 更新价值网络的参数
```

```
actor_lr = 1e-3
critic_lr = 1e-2
num_episodes = 1000
hidden_dim = 128
```

```python
gamma = 0.98
device = torch.device("cuda") if torch.cuda.is_available() else
torch.device(
    "cpu")

env_name = 'CartPole-v0'
env = gym.make(env_name)
env.seed(0)
torch.manual_seed(0)
state_dim = env.observation_space.shape[0]
action_dim = env.action_space.n
agent = ActorCritic(state_dim, hidden_dim, action_dim, actor_lr,
critic_lr,
                    gamma, device)
```

```python
return_list = rl_utils.train_on_policy_agent(env, agent, num_episodes)
```

```python
def train_on_policy_agent(env, agent, num_episodes):
    return_list = []
    for i in range(10):
        with tqdm(total=int(num_episodes/10), desc='Iteration %d' % i) as pbar:
            for i_episode in range(int(num_episodes/10)):
                episode_return = 0
                transition_dict = {'states': [], 'actions': [], 'next_states': [],
'rewards': [], 'dones': []}
                state = env.reset()
                done = False
                while not done:
                    action = agent.take_action(state)
                    next_state, reward, done, _ = env.step(action)
                    transition_dict['states'].append(state)
                    transition_dict['actions'].append(action)
                    transition_dict['next_states'].append(next_state)
                    transition_dict['rewards'].append(reward)
                    transition_dict['dones'].append(done)
                    state = next_state
                    episode_return += reward
                return_list.append(episode_return)
                agent.update(transition_dict)
                if (i_episode+1) % 10 == 0:
                    pbar.set_postfix({'episode': '%d' % (num_episodes/10 * i +
i_episode+1), 'return': '%.3f' % np.mean(return_list[-10:])})
                pbar.update(1)
    return return_list
```

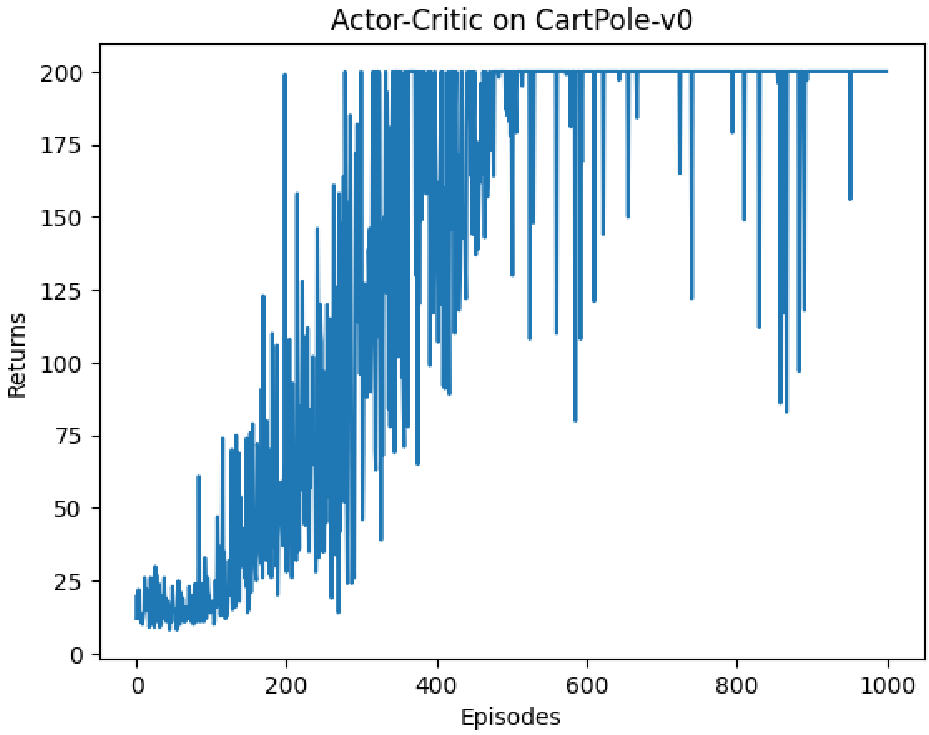
```
Iteration 0:   0%|          | 0/100 [00:00<?, ?it/s]<ipython-input-8-fac1243a855b>:16: UserWarning: Creating a tensor from a list
  state = torch.tensor([state], dtype=torch.float).to(self.device)
Iteration 0: 100%|██████████| 100/100 [00:02<00:00, 37.41it/s, episode=100, return=19.300]
Iteration 1: 100%|██████████| 100/100 [00:03<00:00, 30.10it/s, episode=200, return=68.500]
Iteration 2: 100%|██████████| 100/100 [00:04<00:00, 21.11it/s, episode=300, return=126.000]
Iteration 3: 100%|██████████| 100/100 [00:07<00:00, 12.70it/s, episode=400, return=157.700]
Iteration 4: 100%|██████████| 100/100 [00:08<00:00, 11.20it/s, episode=500, return=193.300]
Iteration 5: 100%|██████████| 100/100 [00:09<00:00, 10.58it/s, episode=600, return=187.700]
Iteration 6: 100%|██████████| 100/100 [00:10<00:00,  9.88it/s, episode=700, return=200.000]
Iteration 7: 100%|██████████| 100/100 [00:10<00:00,  9.99it/s, episode=800, return=197.900]
Iteration 8: 100%|██████████| 100/100 [00:09<00:00, 10.05it/s, episode=900, return=191.500]
Iteration 9: 100%|██████████| 100/100 [00:10<00:00,  9.55it/s, episode=1000, return=200.000]
```

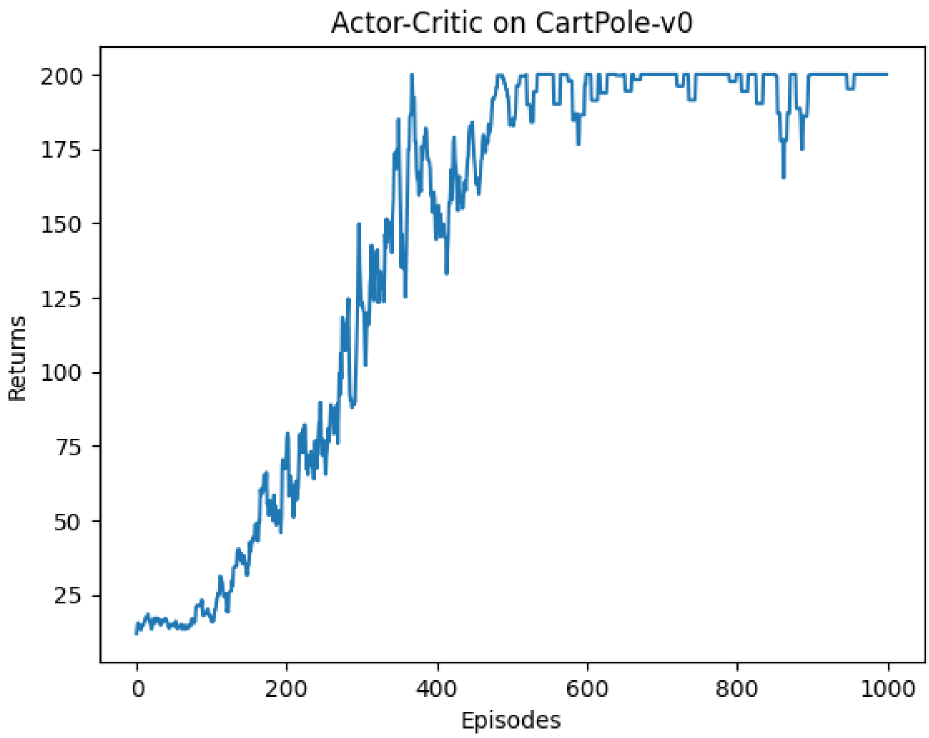
```python
episodes_list = list(range(len(return_list)))
plt.plot(episodes_list, return_list)
plt.xlabel('Episodes')
plt.ylabel('Returns')
plt.title('Actor-Critic on {}'.format(env_name))
plt.show()

mv_return = rl_utils.moving_average(return_list, 9)
```

```python
def moving_average(a, window_size):
    cumulative_sum = np.cumsum(np.insert(a, 0, 0))
    middle = (cumulative_sum[window_size:] - cumulative_sum[:-window_size]) /
window_size
    r = np.arange(1, window_size-1, 2)
    begin = np.cumsum(a[:window_size-1])[::2] / r
    end = (np.cumsum(a[:-window_size:-1])[::2] / r)[::-1]
    return np.concatenate((begin, middle, end))
```

```python
plt.plot(episodes_list, mv_return)
plt.xlabel('Episodes')
plt.ylabel('Returns')
plt.title('Actor-Critic on {}'.format(env_name))
plt.show()
```


Actor-Critic on CartPole-v0
Returns
Episodes
0
25
50
75
100
125
150
175
200
0
200
400
600
800
1000


Actor-Critic on CartPole-v0
Returns
Episodes
25
50
75
100
125
150
175
200
0
200
400
600
800
1000