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

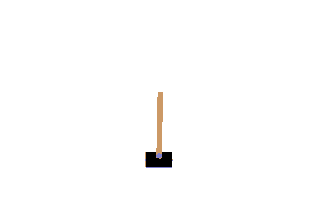
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## **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

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)

return\_list = rl\_utils.train\_on\_policy\_agent(env, agent, num\_episodes)

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

## 

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)

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))

plt.plot(episodes\_list, mv\_return)

plt.xlabel('Episodes')

plt.ylabel('Returns')

plt.title('Actor-Critic on {}'.format(env\_name))

plt.show()

