****Московский государственный технический университет им. Н.Э. Баумана

Кафедра «Системы обработки информации и управления»

Лабораторная работа №7

по дисциплине

«Методы машинного обучения»

на тему

# **«Алгоритмы Actor-Critic»**

Выполнил:

студент группы ИУ5-22М

Джин Шуо

Москва — 2024 г.

**1. Цель лабораторной работы**

Oзнакомление с базовыми методами обучения с подкреплением на основе алгоритмов Actor-Critic.

**2. Задание**

Реализуйте любой алгоритм семейства Actor-Critic для произвольной среды.

**3. текст программы**

import gym

import torch

import torch.nn as nn

import torch.optim as optim

import numpy as np

class Actor(nn.Module):

def \_\_init\_\_(self, state\_dim, action\_dim):

super(Actor, self).\_\_init\_\_()

self.fc1 = nn.Linear(state\_dim, 128)

self.fc2 = nn.Linear(128, 128)

self.fc3 = nn.Linear(128, action\_dim)

self.softmax = nn.Softmax(dim=-1)

def forward(self, x):

x = torch.relu(self.fc1(x))

x = torch.relu(self.fc2(x))

x = self.fc3(x)

return self.softmax(x)

class Critic(nn.Module):

def \_\_init\_\_(self, state\_dim):

super(Critic, self).\_\_init\_\_()

self.fc1 = nn.Linear(state\_dim, 128)

self.fc2 = nn.Linear(128, 128)

self.fc3 = nn.Linear(128, 1)

def forward(self, x):

x = torch.relu(self.fc1(x))

x = torch.relu(self.fc2(x))

x = self.fc3(x)

return x

env = gym.make('CartPole-v1')

state\_dim = env.observation\_space.shape[0]

action\_dim = env.action\_space.n

actor = Actor(state\_dim, action\_dim)

critic = Critic(state\_dim)

actor\_optimizer = optim.Adam(actor.parameters(), lr=1e-4)

critic\_optimizer = optim.Adam(critic.parameters(), lr=1e-3)

gamma = 0.99

num\_episodes = 500

for episode in range(num\_episodes):

state = env.reset()

episode\_reward = 0

while True:

if len(state) != state\_dim:

print(f"State length mismatch: expected {state\_dim}, got {len(state)}")

break

state\_tensor = torch.FloatTensor(state).unsqueeze(0)

action\_probs = actor(state\_tensor)

action = np.random.choice(action\_dim, p=action\_probs.detach().numpy()[0])

next\_state, reward, done, \_ = env.step(action)

next\_state\_tensor = torch.FloatTensor(next\_state).unsqueeze(0)

value = critic(state\_tensor)

next\_value = critic(next\_state\_tensor)

target = reward + gamma \* next\_value \* (1 - int(done))

advantage = target - value

critic\_loss = advantage.pow(2).mean()

critic\_optimizer.zero\_grad()

critic\_loss.backward()

critic\_optimizer.step()

log\_prob = torch.log(action\_probs[0, action])

actor\_loss = -log\_prob \* advantage.detach()

actor\_optimizer.zero\_grad()

actor\_loss.backward()

actor\_optimizer.step()

state = next\_state

episode\_reward += reward

if done:

break

if (episode + 1) % 10 == 0:

print(f'Episode {episode + 1}, Reward: {episode\_reward}')

env.close()

1. **экранные формы с примерами выполнения программы.**



**Список литературы**

[1] https://github.com/ugapanyuk/courses\_current/wiki/LAB\_MMO\_RL\_PG