**Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования** 

**«Московский государственный технический университет**

**имени Н.Э. Баумана**

**(национальный исследовательский университет)»**

**(МГТУ им. Н.Э. Баумана)**

**Факультет «Информатика и системы управления»**

**Кафедра ИУ5 «Системы обработки информации и управления»**

Отчет по лабораторной работе №7

по дисциплине «Методы машинного обучения»

по теме «Алгоритмы Actor-Critic»

Выполнил:

студент группы № ИУ5-24М

Никитина К. В.

подпись, дата

Проверил:

подпись, дата

2024 г.

**Задание:**

∙ Реализуйте любой алгоритм семейства Actor-Critic для произвольной среды.

**Текст программы**

Policy.py

import torch.nn as nn

import torch.nn.functional as F

class Policy(nn.Module):

def \_\_init\_\_(self):

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

self.affine1 = nn.Linear(6, 128)

*# actor's layer*

self.action\_head = nn.Linear(128, 3)

*# critic's layer*

self.value\_head = nn.Linear(128, 1)

*# action & reward buffer*

self.saved\_actions = []

self.rewards = []

def forward(self, x):

x = F.relu(self.affine1(x))

*# actor: choses action to take from state s\_t*

*# by returning probability of each action*

action\_prob = F.softmax(self.action\_head(x), dim=-1)

*# critic: evaluates being in the state s\_t*

state\_values = self.value\_head(x)

*# return values for both actor and critic as a tuple of 2 values: # 1. a list with the probability of each action over the action space # 2. the value from state s\_t*

return action\_prob, state\_values

main.py

import gymnasium as gym

import numpy as np

from itertools import count

from collections import namedtuple

import torch

import torch.nn.functional as F

import torch.optim as optim

from torch.distributions import Categorical

from Policy import Policy

import os

os.environ['SDL\_VIDEODRIVER']='dummy'

import pygame

pygame.display.set\_mode((640,480))

*# Cart Pole*

CONST\_ENV\_NAME = 'Acrobot-v1'

env = gym.make(CONST\_ENV\_NAME)

GAMMA = 0.99

SavedAction = namedtuple('SavedAction', ['log\_prob', 'value'])

model = Policy()

optimizer = optim.AdamW(model.parameters(), lr=1e-3)

eps = np.finfo(np.float32).eps.item()

def select\_action(state):

state = torch.from\_numpy(state).float()

probs, state\_value = model(state)

*# create a categorical distribution over the list of probabilities of actions* m = Categorical(probs)

*# and sample an action using the distribution*

action = m.sample()

*# save to action buffer*

model.saved\_actions.append(SavedAction(m.log\_prob(action), state\_value))

*# the action to take (left or right)*

return action.item()

def finish\_episode():

*"""*

*Training code. Calculates actor and critic loss and performs backprop. """*

R = 0

saved\_actions = model.saved\_actions

policy\_losses = [] *# list to save actor (policy) loss*

value\_losses = [] *# list to save critic (value) loss*

returns = [] *# list to save the true values*

*# calculate the true value using rewards returned from the environment* for r in model.rewards[::-1]:

*# calculate the discounted value*

R = r + GAMMA \* R

returns.insert(0, R)

returns = torch.tensor(returns)

returns = (returns - returns.mean()) / (returns.std() + eps)

for (log\_prob, value), R in zip(saved\_actions, returns):

advantage = R - value.item()

*# calculate actor (policy) loss*

policy\_losses.append(-log\_prob \* advantage)

*# calculate critic (value) loss using L1 smooth loss*

value\_losses.append(F.smooth\_l1\_loss(value, torch.tensor([R])))

*# reset gradients*

optimizer.zero\_grad()

*# sum up all the values of policy\_losses and value\_losses*

loss = torch.stack(policy\_losses).sum() + torch.stack(value\_losses).sum()

*# perform backprop*

loss.backward()

optimizer.step()

*# reset rewards and action buffer*

del model.rewards[:]

del model.saved\_actions[:]

def main():

running\_reward = -500

*# run infinitely many episodes*

for i\_episode in count(1):

*# print(running\_reward)*

*# reset environment and episode reward*

state, \_ = env.reset()

ep\_reward = 0

*# for each episode, only run 9999 steps so that we don't # infinite loop while learning*

for t in range(1, 99999):

*# select action from policy*

action = select\_action(state)

*# take the action*

state, reward, done, truncated, \_ = env.step(action)

model.rewards.append(reward)

ep\_reward += reward

if done or truncated:

break

print(ep\_reward)

*# update cumulative reward*

running\_reward = 0.05 \* ep\_reward + (1 - 0.05) \* running\_reward *# perform backprop*

finish\_episode()

*# log results*

if i\_episode % 10 == 0:

print(f"Episode {i\_episode}\tLast reward: {ep\_reward:.2f}\tAverage reward: {running\_reward:.2f}")

*# check if we have "solved" the cart pole problem*

if running\_reward > env.spec.reward\_threshold \* 2:

print(f"Solved! Running reward is now {running\_reward} and the last episode runs to {t} time steps!")

break

env2 = gym.make(CONST\_ENV\_NAME, render\_mode='human')

*# reset environment and episode reward*

state, \_ = env2.reset()

ep\_reward = 0

*# for each episode, only run 9999 steps so that we don't*

*# infinite loop while learning*

for t in range(1, 10000):

*# select action from policy*

action = select\_action(state)

*# take the action*

state, reward, done, \_, \_ = env2.step(action)

model.rewards.append(reward)

ep\_reward += reward

if done:

break

if \_\_name\_\_ == '\_\_main\_\_':

main()

**Экранные формы**

C:\Users\Pes\_Tick\PycharmProjects\Laba\_7\Scripts\python.exe C:/Users/Pes\_Tick/Documents/GitHub/MMO/Laba\_7/main.py -500.0

-500.0

-500.0

-500.0

-500.0

-500.0

-500.0

-500.0

-500.0

-500.0

Episode 10 Last reward: -500.00 Average reward: -500.00 -500.0

-500.0

-500.0

-500.0

-500.0

-500.0

-500.0

-500.0

-500.0

-500.0

Episode 20 Last reward: -500.00 Average reward: -500.00 -500.0

-500.0

-500.0

-500.0

-500.0

-500.0

-500.0

-500.0

-500.0

-500.0

Episode 30 Last reward: -500.00 Average reward: -500.00 -500.0

-500.0

-500.0

-500.0

-500.0

-500.0

-500.0

-500.0

-500.0

-500.0

Episode 40 Last reward: -500.00 Average reward: -500.00 -500.0

-500.0

-500.0

-500.0

-500.0

-500.0

-500.0

-500.0

-500.0

-500.0

Episode 50 Last reward: -500.00 Average reward: -500.00 -500.0

-500.0

-500.0

-500.0

-500.0

-500.0

-500.0

-500.0

-500.0

-500.0

Episode 60 Last reward: -500.00 Average reward: -500.00 -500.0

-500.0

-500.0

-500.0

-500.0

-500.0

-500.0

-500.0

-500.0

-500.0

Episode 70 Last reward: -500.00 Average reward: -500.00 -500.0

-500.0

-500.0

-500.0

-500.0

-500.0

-500.0

-500.0

-500.0

-500.0

Episode 80 Last reward: -500.00 Average reward: -500.00 -500.0

-500.0

-500.0

-500.0

-500.0

-500.0

-500.0

-500.0

-500.0

-500.0

Episode 90 Last reward: -500.00 Average reward: -500.00 -474.0

-500.0

-500.0

-500.0

-500.0

-500.0

-500.0

-500.0

-500.0

-369.0

Episode 100 Last reward: -369.00 Average reward: -492.63 -500.0

-500.0

-500.0

-414.0

-369.0

-500.0

-500.0

-500.0

-500.0

-500.0

Episode 110 Last reward: -500.00 Average reward: -487.36 -500.0

-500.0

-500.0

-364.0

-500.0

-500.0

-443.0

-500.0

-463.0

-500.0

Episode 120 Last reward: -500.00 Average reward: -483.23 -352.0

-481.0

-500.0

-500.0

-500.0

-389.0

-458.0

-387.0

-394.0

-389.0

Episode 130 Last reward: -389.00 Average reward: -462.66 -246.0

-326.0

-306.0

-325.0

-297.0

-268.0

-247.0

-280.0

-218.0

-476.0

Episode 140 Last reward: -476.00 Average reward: -397.99 -251.0

-397.0

-217.0

-247.0

-223.0

-196.0

-223.0

-233.0

-191.0

-208.0

Episode 150 Last reward: -208.00 Average reward: -332.18

-265.0

-212.0

-208.0

-192.0

-259.0

-188.0

-168.0

-183.0

-213.0

-188.0

Episode 160 Last reward: -188.00 Average reward: -281.25

-230.0

-210.0

-153.0

-212.0

-190.0

-183.0

-200.0

-206.0

-182.0

-167.0

Episode 170 Last reward: -167.00 Average reward: -245.41

-147.0

-171.0

-152.0

-159.0

-175.0

-200.0

-156.0

-179.0

-165.0

-142.0

Episode 180 Last reward: -142.00 Average reward: -213.01

-200.0

-200.0

-123.0

-185.0

-158.0

-184.0

-147.0

-171.0

Solved! Running reward is now -198.55073115939416 and the last episode runs to 172 time steps! Process finished with exit code 0