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 работы №7

по дисциплине «Методы машинного обучения»

по теме «Алгоритмы Actor-Critic»

Выполнил:
студент группы № ИУ5-21М
Торжков М.С.
подпись, дата

Проверила:
Балашов А.М.
подпись, дата

2023 г.

Задание.

Реализуйте любой алгоритм семейства Actor-Critic для произвольной среды

```
! pip install gymnasium
import gymnasium as gym
import numpy as np
from itertools import count
from collections import namedtuple

import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.distributions import Categorical
```

```
Looking in indexes: https://pypi.org/simple, https://us-
python.pkg.dev/colab-wheels/public/simple/
Collecting gymnasium
  Downloading gymnasium-0.28.1-py3-none-any.whl (925 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 925.5/925.5 kB 19.6 MB/s eta
0:00:00
ent already satisfied: numpy>=1.21.0 in
/usr/local/lib/python3.10/dist-packages (from gymnasium) (1.22.4)
Collecting jax-jumpy>=1.0.0 (from gymnasium)
  Downloading jax_jumpy-1.0.0-py3-none-any.whl (20 kB)
Requirement already satisfied: cloudpickle>=1.2.0 in
/usr/local/lib/python3.10/dist-packages (from gymnasium) (2.2.1)
Requirement already satisfied: typing-extensions>=4.3.0 in
/usr/local/lib/python3.10/dist-packages (from gymnasium) (4.5.0)
Collecting farama-notifications>=0.0.1 (from gymnasium)
  Downloading Farama_Notifications-0.0.4-py3-none-any.whl (2.5 kB)
Installing collected packages: farama-notifications, jax-jumpy,
gymnasium
Successfully installed farama-notifications-0.0.4 gymnasium-0.28.1
jax-jumpy-1.0.0
```

```
!pip install pygame

import os
os.environ['SDL_VIDEODRIVER']='dummy'
import pygame
pygame.display.set_mode((640,480))
```

```
Looking in indexes: https://pypi.org/simple, https://us-
python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: pygame in
/usr/local/lib/python3.10/dist-packages (2.3.0)
```

```
<Surface(640x480x32 SW)>
```

```
# Cart Pole
CONST_ENV_NAME = 'Acrobot-v1'
env = gym.make(CONST_ENV_NAME)
```

```
GAMMA = 0.99
SavedAction = namedtuple('SavedAction', ['log_prob', 'value'])

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

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[:]

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
```

```
-500.0
-500.0
-500.0
-500.0
-500.0
-500.0
-500.0
-500.0
-461.0
-500.0
Episode 10 Last reward: -500.00  Average reward: -498.15
-500.0
-500.0
-500.0
-500.0
-500.0
-500.0
-500.0
-479.0
-500.0
-500.0
Episode 20 Last reward: -500.00  Average reward: -497.94
-340.0
-500.0
-500.0
-425.0
-400.0
-448.0
-500.0
-500.0
-500.0
-500.0
Episode 30 Last reward: -500.00  Average reward: -484.98
-478.0
-471.0
-499.0
-500.0
-408.0
-472.0
-427.0
-483.0
-500.0
-500.0
Episode 40 Last reward: -500.00  Average reward: -480.72
-500.0
-361.0
-500.0
```

```
-500.0
-500.0
-333.0
-340.0
-299.0
-342.0
-377.0
Episode 50 Last reward: -377.00  Average reward: -447.46
-347.0
-482.0
-344.0
-259.0
-311.0
-304.0
-500.0
-326.0
-263.0
-295.0
Episode 60 Last reward: -295.00  Average reward: -404.17
-184.0
-201.0
-328.0
-300.0
-199.0
-410.0
-288.0
-368.0
-339.0
-300.0
Episode 70 Last reward: -300.00  Average reward: -361.39
-292.0
-274.0
-332.0
-312.0
-233.0
-222.0
-363.0
-238.0
-285.0
-252.0
Episode 80 Last reward: -252.00  Average reward: -328.22
-226.0
-259.0
-354.0
-217.0
-500.0
-197.0
-181.0
-251.0
-198.0
```

```
-220.0
Episode 90 Last reward: -220.00  Average reward: -299.42
-244.0
-195.0
-233.0
-165.0
-191.0
-170.0
-202.0
-229.0
-218.0
-187.0
Episode 100     Last reward: -187.00  Average reward: -260.65
-166.0
-153.0
-203.0
-120.0
-218.0
-176.0
-249.0
-171.0
-255.0
-131.0
Episode 110     Last reward: -131.00  Average reward: -230.52
-226.0
-222.0
-176.0
-182.0
-137.0
-158.0
-197.0
-217.0
-185.0
-154.0
Episode 120     Last reward: -154.00  Average reward: -211.80
-183.0
-173.0
-153.0
-153.0
-116.0
Solved! Running reward is now -198.72727598097 and the last episode
runs to 117 time steps!
```