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, ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ И СИСТЕМЫ УПРАВЛЕНИЯ

КАФЕДРА ________СИСТЕМЫ ОБРАБОТКИ ИНФОРМАЦИИ И УПРАВЛЕНИЯ (ИУ5)_________

ОТЧЕТ

Лабораторная работа №7
«Алгоритмы Actor-Critic»

по курсу «Методы машинного обучения»

ИСПОЛНИТЕЛЬ:	Савченко Г. А.
	ФИО
группа ИУ5-21М	______________________
	подпись
	"__"_________2023 г.
ПРЕПОДАВАТЕЛЬ:	Гапанюк Ю.Е.
	ФИО
	______________________
	подпись
	"__"_________2023 г.

Москва - 2023

1. Задание

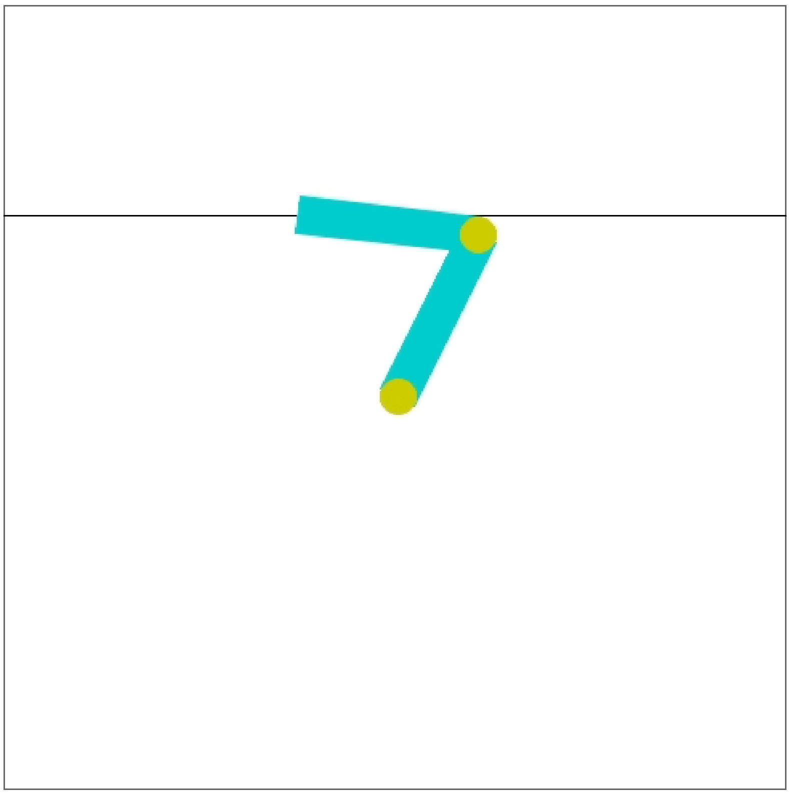
- Реализуйте любой алгоритм семейства Actor-Critic для произвольной среды.

2. Текст программы

```python
#!/usr/bin/env python

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

# 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()

```

```python
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()
```

3. Экранные формы с примерами выполнения программы



```
-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
-410.0
-500.0
-292.0
```

```
Episode 30      Last reward: -292.00    Average reward: -485.54
-232.0
-335.0
-500.0
-500.0
-500.0
-500.0
-500.0
-500.0
-461.0
-500.0
Episode 40      Last reward: -500.00    Average reward: -475.57
-500.0
-500.0
-500.0
-424.0
-362.0
-391.0
-350.0
-500.0
-257.0
-500.0
Episode 50      Last reward: -500.00    Average reward: -454.83
-500.0
-384.0
-500.0
-481.0
-500.0
-292.0
-500.0
-342.0
-428.0
-468.0
Episode 60      Last reward: -468.00    Average reward: -447.79
-500.0
-329.0
-472.0
-500.0
-412.0
-299.0
-500.0
-500.0
-284.0
-455.0
Episode 70      Last reward: -455.00    Average reward: -437.99
-241.0
-333.0
-500.0
-315.0
-339.0
-292.0
-258.0
-408.0
-273.0
-297.0
Episode 80      Last reward: -297.00    Average reward: -392.21
-451.0
-387.0
-268.0
-317.0
-255.0
```

```
-223.0
-352.0
-267.0
-258.0
-242.0
Episode 90      Last reward: -242.00    Average reward: -353.33
-208.0
-273.0
-283.0
-165.0
-231.0
-195.0
-237.0
-306.0
-213.0
-267.0
Episode 100     Last reward: -267.00    Average reward: -307.42
-242.0
-149.0
-236.0
-258.0
-196.0
-204.0
-152.0
-366.0
-251.0
-285.0
Episode 110     Last reward: -285.00    Average reward: -279.45
-210.0
-143.0
-185.0
-231.0
-142.0
-253.0
-251.0
-322.0
-160.0
-162.0
Episode 120     Last reward: -162.00    Average reward: -250.42
-313.0
-205.0
-186.0
-162.0
-186.0
-186.0
-217.0
-124.0
-171.0
-208.0
Episode 130     Last reward: -208.00    Average reward: -227.24
-214.0
-207.0
-149.0
-125.0
-245.0
-204.0
-175.0
-243.0
-135.0
-459.0
Episode 140     Last reward: -459.00    Average reward: -225.08
```

```
-175.0
-163.0
-212.0
-177.0
-166.0
-157.0
-175.0
-140.0
-156.0
-197.0
Episode 150     Last reward: -197.00    Average reward: -203.49
-241.0
-193.0
-154.0
-156.0
Solved! Running reward is now -199.8982282342606 and the last episode runs to 157 time steps!
```