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)

Курс «Методы машинного обучения»

Отчет по лабораторной работе №7:
«Алгоритмы Actor-Critic»

Выполнила:

студентка группы ИУ5-24М

Мащенко Е. И.

Проверил:

Балашов А.М.

2023

Цель работы

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

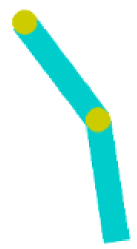
Задание

Реализуйте любой алгоритм семейства Actor-Critic для произвольной среды.

Выполнение работы

Для реализации алгоритма Actor-Critic была выбрана среда обучения с подкреплением Acrobot из библиотеки Gym.

Среда Acrobot состоит из двух звеньев, соединенных в цепь, один конец которой закреплен. Соединение между двумя звеньями приводится в действие. Цель состоит в том, чтобы приложить крутящий момент к приводимому в действие шарниру, чтобы поднять свободный конец цепи выше заданной высоты, начиная с исходного состояния:



Пространство действий представляет собой крутящий момент, приложенный к приводимому в действие соединению между двумя звеньями:

Num	Action	Unit
0	apply -1 torque to the actuated joint	torque (N m)
1	apply 0 torque to the actuated joint	torque (N m)
2	apply 1 torque to the actuated joint	torque (N m)

Пространство состояний представляет собой матрицу ndarray (6,), которая предоставляет информацию о двух углах соединения при вращении, а также об их угловых скоростях:

Num	Observation	Min	Max
0	Cosine of `theta1`	-1	1
1	Sine of `theta1`	-1	1
2	Cosine of `theta2`	-1	1
3	Sine of `theta2`	-1	1
4	Angular velocity of `theta1`	~ -12.567 (-4 * pi)	~ 12.567 (4 * pi)
5	Angular velocity of `theta2`	~ -28.274 (-9 * pi)	~ 28.274 (9 * pi)

Лабораторная работа №7

В [1]:
```
! 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
```

```
WARNING: You are using pip version 20.1.1; however, version 23.1.2 is available.
You should consider upgrading via the 'c:\users\user\appdata\local\programs\python\python38\python.exe -m pip install --upgr
ade pip' command.
```

```
Requirement already satisfied: gymnasium in c:\users\user\appdata\local\programs\python\python38\lib\site-packages (0.28.1)
Requirement already satisfied: farama-notifications>=0.0.1 in c:\users\user\appdata\local\programs\python\python38\lib\site-
packages (from gymnasium) (0.0.4)
Requirement already satisfied: jax-jumpy>=1.0.0 in c:\users\user\appdata\local\programs\python\python38\lib\site-packages (f
rom gymnasium) (1.0.0)
Requirement already satisfied: importlib-metadata>=4.8.0; python_version < "3.10" in c:\users\user\appdata\local\programs\py
thon\python38\lib\site-packages (from gymnasium) (6.6.0)
Requirement already satisfied: numpy>=1.21.0 in c:\users\user\appdata\local\programs\python\python38\lib\site-packages (from
gymnasium) (1.24.3)
Requirement already satisfied: cloudpickle>=1.2.0 in c:\users\user\appdata\local\programs\python\python38\lib\site-packages
(from gymnasium) (1.3.0)
Requirement already satisfied: typing-extensions>=4.3.0 in c:\users\user\appdata\local\programs\python\python38\lib\site-pac
kages (from gymnasium) (4.6.3)
Requirement already satisfied: zipp>=0.5 in c:\users\user\appdata\local\programs\python\python38\lib\site-packages (from imp
ortlib-metadata>=4.8.0; python_version < "3.10"->gymnasium) (3.15.0)
```

В [2]:
```
!pip install pygame

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

```
Requirement already satisfied: pygame in c:\users\user\appdata\local\programs\python\python38\lib\site-packages (2.4.0)
```

```
WARNING: You are using pip version 20.1.1; however, version 23.1.2 is available.
You should consider upgrading via the 'c:\users\user\appdata\local\programs\python\python38\python.exe -m pip install --upgr
ade pip' command.
```

Out[2]:
```
<Surface(640x480x32 SW)>
```

В [3]:

```
# Cart Pole
CONST_ENV_NAME = 'Acrobot-v1'
env = gym.make(CONST_ENV_NAME)
GAMMA = 0.99
SavedAction = namedtuple('SavedAction', ['log_prob', 'value'])
```

В [4]:

```
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
```

В [5]:

```
model = Policy()
optimizer = optim.AdamW(model.parameters(), lr=1e-3)
eps = np.finfo(np.float32).eps.item()
```

В [6]:

```
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()
```

В [7]:

```
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)
```

```python
    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[:]
```

В [8]:

```python
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
```

```
Episode 2870	Last reward: -203.00	Average reward: -209.34
-173.0
-151.0
-244.0
-161.0
-150.0
-172.0
-220.0
-365.0
-177.0
-181.0
Episode 2880	Last reward: -181.00	Average reward: -206.40
-139.0
-126.0
Solved! Running reward is now -199.18003893689894 and the last episode runs to 127 time steps!
```