### Задание:

Реализуйте любой алгоритм семейства Actor-Critic для произвольной среды.

Программа реализует алгоритм обучения с подкреплением Advantage Actor-Critic (A2C) для решения задачи управления маятником в окружении "CartPole-v1" из библиотеки OpenAI Gym. Основная цель состоит в обучении агента (модели) так, чтобы он мог удерживать маятник в вертикальном положении как можно дольше.

import sys  
import torch  
import gym  
import numpy as np  
import torch.nn as nn  
import torch.optim as optim  
import torch.nn.functional as F  
from torch.autograd import Variable  
import matplotlib.pyplot as plt  
import pandas as pd  
from IPython.display import Image

# Гиперпараметры  
hidden\_size = 256  
learning\_rate = 3e-4  
entropy\_coef = 0.01  
  
# Константы  
GAMMA = 0.99  
num\_steps = 250  
max\_episodes = 2500 # Увеличенное количество эпизодов  
CONST\_ENV\_NAME = "CartPole-v1"  
CONST\_DEVICE = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

# Класс, реализующий модель Actor-Critic  
class ActorCritic(nn.Module):  
 def \_\_init\_\_(self, num\_inputs, num\_actions, hidden\_size, learning\_rate=3e-4):  
 super(ActorCritic, self).\_\_init\_\_()  
  
 self.num\_actions = num\_actions  
 self.critic\_linear1 = nn.Linear(num\_inputs, hidden\_size)  
 self.critic\_linear2 = nn.Linear(hidden\_size, 1)  
  
 self.actor\_linear1 = nn.Linear(num\_inputs, hidden\_size)  
 self.actor\_linear2 = nn.Linear(hidden\_size, num\_actions)  
  
 def forward(self, state):  
 value = F.relu(self.critic\_linear1(state))  
 value = self.critic\_linear2(value)  
  
 policy\_dist = F.relu(self.actor\_linear1(state))  
 policy\_dist = F.softmax(self.actor\_linear2(policy\_dist), dim=1)  
  
 return value, policy\_dist  
  
 def select\_action(self, state):  
 \_, policy\_dist = self.forward(state)  
 action = policy\_dist.multinomial(num\_samples=1).detach()  
 return action[0]

# Функция для реализации алгоритма Advantage Actor-Critic (A2C)  
def a2c(env):  
 num\_inputs = env.observation\_space.shape[0]  
 num\_outputs = env.action\_space.n  
  
 actor\_critic = ActorCritic(num\_inputs, num\_outputs, hidden\_size)  
 ac\_optimizer = optim.Adam(actor\_critic.parameters(), lr=learning\_rate)  
  
 all\_lengths = []  
 average\_lengths = []  
 all\_rewards = []  
 entropy\_term = 0  
  
 for episode in range(max\_episodes):  
 log\_probs = []  
 values = []  
 rewards = []  
  
 state = env.reset()  
 if isinstance(state, tuple):  
 state = state[0]  
 state = torch.tensor(state, dtype=torch.float32).unsqueeze(0)  
 for steps in range(num\_steps):  
 value, policy\_dist = actor\_critic.forward(state)  
 value = value.detach().numpy()[0, 0]  
 dist = policy\_dist.detach().numpy()  
  
 action = np.random.choice(num\_outputs, p=np.squeeze(dist))  
 log\_prob = torch.log(policy\_dist.squeeze(0)[action])  
 entropy = -np.sum(np.mean(dist) \* np.log(dist))  
 result = env.step(action)  
 new\_state = result[0]  
 reward = result[1]  
 done = result[2]  
  
 if isinstance(new\_state, tuple):  
 new\_state = new\_state[0]  
 new\_state = torch.tensor(new\_state, dtype=torch.float32).unsqueeze(0)  
  
 rewards.append(reward)  
 values.append(value)  
 log\_probs.append(log\_prob)  
 entropy\_term += entropy  
 state = new\_state  
  
 if done or steps == num\_steps - 1:  
 Qval, \_ = actor\_critic.forward(new\_state)  
 Qval = Qval.detach().numpy()[0, 0]  
 all\_rewards.append(np.sum(rewards))  
 all\_lengths.append(steps)  
 average\_lengths.append(np.mean(all\_lengths[-10:]))  
 if episode % 10 == 0:  
 sys.stdout.write("episode: {}, reward: {}, total length: {}, average length: {} \n".format(episode, np.sum(rewards), steps, average\_lengths[-1]))  
 break  
  
 # Вычисление значений Q  
 Qvals = np.zeros\_like(values)  
 for t in reversed(range(len(rewards))):  
 Qval = rewards[t] + GAMMA \* Qval  
 Qvals[t] = Qval  
  
 # Обновление модели actor-critic  
 values = torch.FloatTensor(values)  
 Qvals = torch.FloatTensor(Qvals)  
 log\_probs = torch.stack(log\_probs)  
  
 advantage = Qvals - values  
 actor\_loss = (-log\_probs \* advantage).mean()  
 critic\_loss = 0.5 \* advantage.pow(2).mean()  
 ac\_loss = actor\_loss + critic\_loss + entropy\_coef \* entropy\_term  
  
 ac\_optimizer.zero\_grad()  
 ac\_loss.backward()  
 ac\_optimizer.step()  
  
 # Построение графиков результатов  
 smoothed\_rewards = pd.Series(all\_rewards).rolling(10).mean()  
 plt.plot(all\_rewards, label='Награды')  
 plt.plot(smoothed\_rewards, label='Сглаженные награды')  
 plt.xlabel('Эпизод')  
 plt.ylabel('Награда')  
 plt.legend()  
 plt.show()  
  
 plt.plot(all\_lengths, label='Длины эпизодов')  
 plt.plot(average\_lengths, label='Средние длины')  
 plt.xlabel('Эпизод')  
 plt.ylabel('Длина эпизода')  
 plt.legend()  
 plt.show()  
  
 return actor\_critic

# Функция для запуска агента в среде с использованием обученной модели  
def play\_agent(actor\_critic):  
 '''  
 Игра с обученным агентом  
 '''  
 env2 = gym.make(CONST\_ENV\_NAME, render\_mode='human')  
 state = env2.reset()  
 if isinstance(state, tuple):  
 state = state[0]  
 state = torch.tensor(state, dtype=torch.float32, device=CONST\_DEVICE).unsqueeze(0)  
 done = False  
 res = []  
 while not done:  
 action = actor\_critic.select\_action(state)  
 action = action.item()  
 result = env2.step(action)  
 observation = result[0]  
 reward = result[1]  
 done = result[2]  
 truncated = result[3]  
 env2.render()  
  
 res.append((action, reward))  
  
 if done or truncated:  
 done = True  
 else:  
 if isinstance(observation, tuple):  
 observation = observation[0]  
 state = torch.tensor(observation, dtype=torch.float32, device=CONST\_DEVICE).unsqueeze(0)  
  
 print('Данные эпизода: ', res)

if \_\_name\_\_ == "\_\_main\_\_":  
 env = gym.make(CONST\_ENV\_NAME)  
 trained\_actor\_critic = a2c(env)  
  
 # Запуск сессии с обученным агентом  
 play\_agent(trained\_actor\_critic)

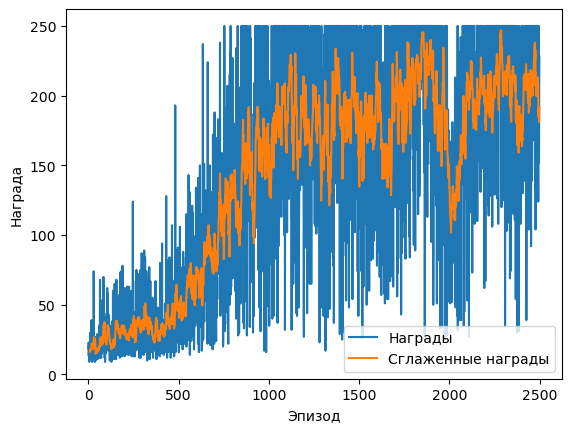
/Users/evseykirichkov/anaconda3/lib/python3.11/site-packages/gym/utils/passive\_env\_checker.py:233: DeprecationWarning: `np.bool8` is a deprecated alias for `np.bool\_`. (Deprecated NumPy 1.24)  
 if not isinstance(terminated, (bool, np.bool8)):

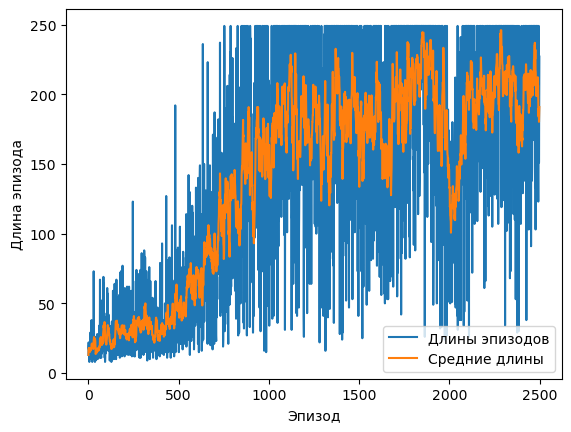
episode: 0, reward: 14.0, total length: 13, average length: 13.0   
episode: 10, reward: 30.0, total length: 29, average length: 16.9   
episode: 20, reward: 32.0, total length: 31, average length: 21.2   
episode: 30, reward: 43.0, total length: 42, average length: 24.6   
episode: 40, reward: 10.0, total length: 9, average length: 13.7   
episode: 50, reward: 17.0, total length: 16, average length: 17.8   
episode: 60, reward: 32.0, total length: 31, average length: 18.7   
episode: 70, reward: 13.0, total length: 12, average length: 21.7   
episode: 80, reward: 18.0, total length: 17, average length: 22.6   
episode: 90, reward: 12.0, total length: 11, average length: 36.0   
episode: 100, reward: 29.0, total length: 28, average length: 22.1   
episode: 110, reward: 17.0, total length: 16, average length: 31.5   
episode: 120, reward: 26.0, total length: 25, average length: 23.6   
episode: 130, reward: 38.0, total length: 37, average length: 19.7   
episode: 140, reward: 16.0, total length: 15, average length: 20.3   
episode: 150, reward: 51.0, total length: 50, average length: 32.0   
episode: 160, reward: 25.0, total length: 24, average length: 31.9   
episode: 170, reward: 57.0, total length: 56, average length: 31.7   
episode: 180, reward: 74.0, total length: 73, average length: 30.0   
episode: 190, reward: 35.0, total length: 34, average length: 30.4   
episode: 200, reward: 13.0, total length: 12, average length: 27.6   
episode: 210, reward: 29.0, total length: 28, average length: 26.6   
episode: 220, reward: 14.0, total length: 13, average length: 25.7

…

episode: 2200, reward: 67.0, total length: 66, average length: 179.0   
episode: 2210, reward: 173.0, total length: 172, average length: 198.7   
episode: 2220, reward: 250.0, total length: 249, average length: 198.5   
episode: 2230, reward: 128.0, total length: 127, average length: 199.5   
episode: 2240, reward: 211.0, total length: 210, average length: 185.5   
episode: 2250, reward: 232.0, total length: 231, average length: 209.8   
episode: 2260, reward: 250.0, total length: 249, average length: 220.4   
episode: 2270, reward: 250.0, total length: 249, average length: 206.4   
episode: 2280, reward: 250.0, total length: 249, average length: 218.9   
episode: 2290, reward: 250.0, total length: 249, average length: 234.0   
episode: 2300, reward: 250.0, total length: 249, average length: 205.7   
episode: 2310, reward: 250.0, total length: 249, average length: 200.5   
episode: 2320, reward: 186.0, total length: 185, average length: 214.5   
episode: 2330, reward: 250.0, total length: 249, average length: 226.1   
episode: 2340, reward: 250.0, total length: 249, average length: 195.3

episode: 2350, reward: 250.0, total length: 249, average length: 216.2   
episode: 2360, reward: 207.0, total length: 206, average length: 209.7   
episode: 2370, reward: 248.0, total length: 247, average length: 185.7   
episode: 2380, reward: 242.0, total length: 241, average length: 172.8   
episode: 2390, reward: 202.0, total length: 201, average length: 182.0   
episode: 2400, reward: 250.0, total length: 249, average length: 171.5   
episode: 2410, reward: 250.0, total length: 249, average length: 188.8   
episode: 2420, reward: 250.0, total length: 249, average length: 208.0   
episode: 2430, reward: 249.0, total length: 248, average length: 210.1   
episode: 2440, reward: 104.0, total length: 103, average length: 192.2   
episode: 2450, reward: 250.0, total length: 249, average length: 217.1   
episode: 2460, reward: 250.0, total length: 249, average length: 203.2   
episode: 2470, reward: 250.0, total length: 249, average length: 224.6   
episode: 2480, reward: 178.0, total length: 177, average length: 208.3   
episode: 2490, reward: 250.0, total length: 249, average length: 212.2





Данные эпизода: [(0, 1.0), (1, 1.0), (0, 1.0), (1, 1.0), (1, 1.0), (0, 1.0), (1, 1.0), (0, 1.0), (1, 1.0), (1, 1.0), (0, 1.0), (0, 1.0), (1, 1.0), (1, 1.0), (0, 1.0), (1, 1.0), (1, 1.0), (0, 1.0), (1, 1.0), (0, 1.0), (0, 1.0), (1, 1.0), (0, 1.0), (0, 1.0), (1, 1.0), (0, 1.0), (0, 1.0), (1, 1.0), (1, 1.0), (0, 1.0), (1, 1.0), (0, 1.0), (0, 1.0), (1, 1.0), (0, 1.0), (1, 1.0), (0, 1.0), (1, 1.0), (1, 1.0), (1, 1.0), (0, 1.0), (0, 1.0), (1, 1.0), (0, 1.0), (1, 1.0), (0, 1.0), (0, 1.0), (0, 1.0), (1, 1.0), (1, 1.0), (1, 1.0), (0, 1.0), (0, 1.0), (1, 1.0), (0, 1.0), (1, 1.0), (0, 1.0), (1, 1.0), (0, 1.0), (1, 1.0), (0, 1.0), (1, 1.0), (1, 1.0), (0, 1.0), (1, 1.0), (1, 1.0), (0, 1.0), (0, 1.0), (0, 1.0), (1, 1.0), (0, 1.0), (0, 1.0), (0, 1.0), (0, 1.0), (1, 1.0), (1, 1.0), (1, 1.0), (1, 1.0), (0, 1.0), (0, 1.0), (1, 1.0), (0, 1.0), (0, 1.0), (1, 1.0), (1, 1.0), (0, 1.0), (0, 1.0), (1, 1.0), (1, 1.0), (0, 1.0), (0, 1.0), (0, 1.0), (1, 1.0), (1, 1.0), (0, 1.0), (0, 1.0), (1, 1.0), (0, 1.0), (1, 1.0), (0, 1.0), (0, 1.0), (0, 1.0), (0, 1.0), (1, 1.0), (1, 1.0), (1, 1.0), (1, 1.0), (0, 1.0), (0, 1.0), (0, 1.0), (1, 1.0), (1, 1.0), (0, 1.0), (0, 1.0), (0, 1.0), (0, 1.0), (1, 1.0), (1, 1.0), (0, 1.0), (1, 1.0), (1, 1.0), (0, 1.0), (1, 1.0), (0, 1.0), (0, 1.0), (1, 1.0), (0, 1.0), (1, 1.0), (0, 1.0), (1, 1.0), (1, 1.0), (0, 1.0), (1, 1.0), (0, 1.0), (1, 1.0), (0, 1.0), (1, 1.0), (0, 1.0), (0, 1.0), (1, 1.0), (0, 1.0), (0, 1.0), (1, 1.0), (1, 1.0), (1, 1.0), (0, 1.0), (0, 1.0), (1, 1.0), (0, 1.0), (1, 1.0), (1, 1.0), (0, 1.0), (1, 1.0), (1, 1.0), (0, 1.0), (1, 1.0), (0, 1.0), (1, 1.0), (0, 1.0), (0, 1.0), (0, 1.0), (1, 1.0), (1, 1.0), (0, 1.0), (0, 1.0), (1, 1.0), (0, 1.0), (0, 1.0), (1, 1.0), (0, 1.0), (1, 1.0), (1, 1.0), (1, 1.0), (0, 1.0), (1, 1.0), (1, 1.0), (0, 1.0), (0, 1.0), (0, 1.0), (1, 1.0), (1, 1.0), (0, 1.0), (1, 1.0), (0, 1.0), (0, 1.0), (1, 1.0), (1, 1.0), (1, 1.0), (1, 1.0), (0, 1.0), (0, 1.0), (1, 1.0), (0, 1.0), (1, 1.0), (1, 1.0), (0, 1.0), (0, 1.0), (1, 1.0), (1, 1.0), (1, 1.0), (0, 1.0), (0, 1.0), (1, 1.0), (0, 1.0), (1, 1.0), (1, 1.0), (1, 1.0), (0, 1.0), (0, 1.0), (1, 1.0), (1, 1.0), (0, 1.0), (0, 1.0), (0, 1.0), (0, 1.0), (1, 1.0), (1, 1.0), (0, 1.0), (1, 1.0), (1, 1.0), (0, 1.0), (0, 1.0), (1, 1.0), (0, 1.0), (1, 1.0), (0, 1.0), (0, 1.0), (1, 1.0), (0, 1.0)]

image\_path = "data/a2c\_result.png"  
display(Image(filename=image\_path, width=500, height=500))

