**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра МОЭВМ**

**ОТЧЕТ**

**по практической работе №2**

**по дисциплине «Обучение с подкреплением»**

**Тема: Реализация PPO для среды MountainCarContinuous-v0**

| Студент гр. 0306 |  | Кумаритов А.О. |
| --- | --- | --- |
| Преподаватель |  | Глазунов. С.А. |

Санкт-Петербург

2025

**Задание:**

Реализовать PPO для среды MountainCarContinuous-v0.

**Задания для эксперимента:**

Измените длину траектории (steps).

Подберите оптимальный коэффициент clip\_ratio.

Добавьте нормализацию преимуществ.

Сравните обучение при разных количествах эпох.

**Описание среды:**

Action space состоит из числа от -1 до 1 умноженного на мощность 0.0015, которое представляет силу, приложенную к машине.

Observation space состоит из 2 чисел:

Car position - позиция машины по оси X, значения от -1.2 до 0.6

Car velocity - скорость машины, значение -0.07 до 0.07

Rewards негативный равен -0.1 \* , при достижении цели добавляется +100 к негативной награде.

Starting state - position присваивается случайное значение от -0.6 до -0.4, velocity присваивается 0.

Конец эпизода в двух случаях:

Car position больше или равен 0.45

Длительность эпизода равна 999

**Описание алгоритма:**

Базовое описание алгоритма представлено на рисунке 1.

|  |
| --- |
| Рис. 1 - алгоритм Proximal Policy Optimization (PPO) |

В реализации следующие параметры:

GAMMA - коэффициент дисконтирования

GAE\_LAMBDA - параметр Generalized Advantage Estimation

ENTROPY\_COEFFICIENT - коэффициент энтропии

CLIP\_RATIO - коэффициент обрезки

STEPS - длина траектории

ITERATIONS - количество итераций обучения

N\_EPOCH - количество эпох

BATCH\_SIZE - размер батча

LR - скорость обучения оптимизаторов

WITH\_NORMALIZATION - флаг нормализации преимуществ

В реализации используется несколько классов:

Agent - класс, реализующий инициализацию агента и все необходимые для обучения методы.

Actor, Critic - две нейросети. Actor рассчитывает действие, а Critic оценивает состояние. Их конфигурация представлена на рисунке 2 и 3 соответственно:

|  |
| --- |
| Рис. 2 - конфигурация нейронной сети Actor |

|  |
| --- |
| Рис. 3 - конфигурация нейронной сети Critic |

**Выполнение экспериментов.**

**Изменение длины траектории (steps):**

В рамках эксперимента по сравнению влияния различных значений параметра steps на обучение алгоритма PPO было проведено три запуска с параметром steps равным 1024, 2048, 4096. Этот параметр определяет длину траектории или количество шагов агента до обновления. Результат представлен на рисунке 4:

|  |
| --- |
| Рис. 4 - влияние различного значения steps |

Лучший результат обучения показала длина траектории равная 4096. Стабильность получения награды достигалась к 60-ой итерации. Среднее значение длины траектории в 2048 показало средний результат. Стабильность награды достигалась к 200-ой итерации. Из-за недостатка статистики для обновления политики длина траектории 1024 показала худший результат. Стабильного значения награды достигнуто не было. В данных условиях использование большей длины траектории улучшает процесс обучения, но увеличивает вычислительные затраты.

**Подбор оптимального коэффициента clip\_ratio:**

В рамках эксперимента по сравнению влияния различных значений параметра clip\_ratio на обучение алгоритма PPO было проведено три запуска с параметром clip\_ratio равным 0.1, 0.2, 0.3. Этот параметр определяет резкость изменения политики. Результат представлен на рисунке 5:

|  |
| --- |
| Рис. 5 - влияние различных значений clip\_ratio |

Лучший результат обучения показало значение clip\_ratio равное 0.2. Стабильно высокие награды достигались к 200-ой итерации. Самое большое значение clip\_ratio в 0.3 показало сравнимый, но менее стабильный результат. Запуск с clip\_ratio равной 0.1 показал худший и нестабильный результат обучения. Это говорит о важности баланса между скоростью обучения и стабильностью.

**Добавление нормализации преимуществ:**

В рамках эксперимента по сравнению влияния нормализации преимуществ на обучение алгоритма PPO было проведено два запуска с включённой и выключенной нормализацией преимуществ. Результат представлен на рисунке 6:

|  |
| --- |
| Рис. 6 - влияние нормализации |

Запуск с нормализацией не дал ощутимых преимуществ на старте обучения и замедлил процесс. Без нормализации выход к высоким наградам был к 180-ой итерации, в то время как для запуска с нормализацией к 225-ой. После выхода к высоким наградам запуск со нормализацией показывает большую стабильность, чем запуск без нормализации. Таким образом нормализация может замедлить достижение высоких наград, но способствует повышению стабильности.

**Сравнение обучения при разных количествах эпох:**

В рамках эксперимента по сравнению влияния различных значений параметра n\_epoch на обучение алгоритма PPO было проведено три запуска с параметром n\_epoch равным 5, 10, 20. Этот параметр определяет количество эпох для итерации. Результат представлен на рисунке 7:

|  |
| --- |
| Рис. 7 - влияние различных значений n\_epoch |

Лучший результат обучения показало значение n\_epoch равное 20. Стабильно высокие награды достигались к 75-ой итерации. Значение n\_epoch равное 10 достигло стабильно высоких наград к 225-ой итерации. А значение n\_epoch равное 5 показало худший результат, не достигнув стабильно высоких наград за 300 итераций. В данных условиях использование большего значения n\_epoch улучшает процесс обучения, но увеличивает вычислительные затраты.

**Выводы.**

Был реализован PPO для среды MountainCarContinuous-v0. Были проведены исследования при различных значениях steps, clip\_ratio, нормализации, n\_epoch. Оптимальным для высоких и стабильных результатов обучения, исходя из полученных данных, является запуск с steps = 4096, clip\_ratio = 0.2, включённой нормализацией, n\_epoch = 20.

**ПРИЛОЖЕНИЕ А.**

**ИСХОДНЫЙ КОД ПРОГРАММЫ.**

Исходный код main.py

import gymnasium as gym

import matplotlib.pyplot as plt

import numpy as np

import torch

from agent import Agent

def run\_experiment(steps, clip\_ratio, n\_epoch, with\_normalization, seed=42):

env = gym.make("MountainCarContinuous-v0")

torch.manual\_seed(seed)

env.reset(seed=seed)

np.random.seed(seed)

agent = Agent(

env=env,

gamma=0.99,

gae\_lambda=0.95,

entropy\_coefficient=0.4,

clip\_ratio=clip\_ratio,

steps=steps,

iterations=300,

n\_epoch=n\_epoch,

batch\_size=32,

lr=3e-4,

with\_normalization=with\_normalization

)

agent.train()

return agent.scores

def run\_and\_plot(param\_values, param\_name, train\_kwargs, filename\_prefix):

plt.figure(figsize=(10, 6))

color\_cycle = plt.rcParams['axes.prop\_cycle'].by\_key()['color']

for i, val in enumerate(param\_values):

print(f"Обучение при {param\_name} = {val}")

kwargs = train\_kwargs(val)

scores = run\_experiment(

steps=kwargs.get("steps", 2048),

clip\_ratio=kwargs.get("clip\_ratio", 0.2),

n\_epoch=kwargs.get("n\_epoch", 10),

with\_normalization=kwargs.get("with\_normalization", True)

)

color = color\_cycle[i % len(color\_cycle)]

plt.plot(range(len(scores)), scores, linestyle='--', alpha=0.3,

color=color, label=f"{param\_name}={val} (raw)")

smoothed = np.convolve(scores, np.ones(10) / 10, mode='valid')

plt.plot(range(len(smoothed)), smoothed, linestyle='-', color=color,

label=f"{param\_name}={val} (smoothed)")

plt.title(f"PPO: сравнение по параметру {param\_name}")

plt.xlabel("Итерация")

plt.ylabel("Счёт")

plt.legend()

plt.grid(True)

plt.tight\_layout()

plt.savefig(f"{filename\_prefix}\_{param\_name}.png")

def different\_steps():

steps = [1024, 2048, 4096]

run\_and\_plot(steps, "steps", lambda st: {"steps": st}, "results")

def different\_clip\_ratio():

clip\_ratio = [0.1, 0.2, 0.3]

run\_and\_plot(clip\_ratio, "clip\_ratio", lambda cr: {"clip\_ratio": cr}, "results")

def different\_normalization():

with\_normalization = [False, True]

run\_and\_plot(with\_normalization, "with\_normalization", lambda wn: {"with\_normalization": wn}, "results")

def different\_n\_epoch():

n\_epoch = [5, 10, 20]

run\_and\_plot(n\_epoch, "n\_epoch", lambda ne: {"n\_epoch": ne}, "results")

def main():

different\_steps()

different\_clip\_ratio()

different\_normalization()

different\_n\_epoch()

if \_\_name\_\_ == "\_\_main\_\_":

main()

Исходный код agent.py

import numpy as np

import torch

import torch.optim as optim

from actor import Actor

from critic import Critic

class Agent:

def \_\_init\_\_(

self,

env,

gamma,

gae\_lambda,

entropy\_coefficient,

clip\_ratio,

steps,

iterations,

n\_epoch,

batch\_size,

lr,

with\_normalization

):

self.env = env

self.n\_observations = env.observation\_space.shape[0]

self.n\_actions = env.action\_space.shape[0]

self.gamma = gamma

self.gae\_lambda = gae\_lambda

self.entropy\_coefficient = entropy\_coefficient

self.clip\_ratio = clip\_ratio

self.steps = steps

self.iterations = iterations

self.n\_epoch = n\_epoch

self.batch\_size = batch\_size

self.with\_normalization = with\_normalization

self.actor = Actor(self.n\_observations, self.n\_actions)

self.critic = Critic(self.n\_observations)

self.actor\_optimizer = optim.Adam(self.actor.parameters(), lr=lr)

self.critic\_optimizer = optim.Adam(self.critic.parameters(), lr=lr)

self.actor\_loss\_history = []

self.critic\_loss\_history = []

self.scores = []

def get\_action(self, state):

state = torch.as\_tensor(state, dtype=torch.float32)

action, \_ = self.actor.get\_action(state)

return action.item()

def make\_action(self, action):

clipped\_action = np.clip(action, -1, 1)

next\_state, reward, terminated, truncated, \_ = self.env.step(clipped\_action)

is\_terminal = terminated or truncated

return next\_state, reward, is\_terminal

def train(self):

for \_ in range(self.iterations):

states, actions, rewards, values, is\_terminals, log\_probs, episode\_rewards = self.get\_trajectories()

returns, advantages = self.get\_returns\_and\_advantages(rewards, values, is\_terminals)

self.update\_net\_weights(states, actions, log\_probs, returns, advantages)

if episode\_rewards:

avg\_score = np.mean(episode\_rewards)

self.scores.append(avg\_score)

self.env.close()

def update\_net\_weights(self, states, actions, old\_log\_probs, returns, advantages):

self.actor.train()

self.critic.train()

actor\_losses, critic\_losses = [], []

states = np.array(states)

actions = np.array(actions)

old\_log\_probs = np.array(old\_log\_probs)

for \_ in range(self.n\_epoch):

indices = np.random.permutation(len(states))

for start in range(0, len(states), self.batch\_size):

end = start + self.batch\_size

batch\_indices = indices[start:end]

batch\_states = torch.as\_tensor(states[batch\_indices], dtype=torch.float32)

batch\_actions = torch.as\_tensor(actions[batch\_indices], dtype=torch.float32)

batch\_old\_log\_probs = torch.as\_tensor(old\_log\_probs[batch\_indices], dtype=torch.float32)

batch\_returns = returns[batch\_indices]

batch\_advantages = advantages[batch\_indices]

dist = self.actor.get\_dist(batch\_states)

cur\_log\_probs = dist.log\_prob(batch\_actions).sum(dim=-1)

ratio = torch.exp(cur\_log\_probs - batch\_old\_log\_probs)

entropy = dist.entropy().mean()

batch\_advantages = batch\_advantages.detach()

loss = batch\_advantages \* ratio

clipped\_loss = (

torch.clamp(ratio, 1. - self.clip\_ratio, 1. + self.clip\_ratio)

\* batch\_advantages

)

actor\_loss = (

-torch.mean(torch.min(loss, clipped\_loss))

- entropy \* self.entropy\_coefficient

)

cur\_value = self.critic(batch\_states)

critic\_loss = (batch\_returns - cur\_value).pow(2).mean()

self.actor\_optimizer.zero\_grad()

actor\_loss.backward()

self.actor\_optimizer.step()

self.critic\_optimizer.zero\_grad()

critic\_loss.backward()

self.critic\_optimizer.step()

actor\_losses.append(actor\_loss.item())

critic\_losses.append(critic\_loss.item())

avg\_actor\_loss = sum(actor\_losses) / len(actor\_losses)

avg\_critic\_loss = sum(critic\_losses) / len(critic\_losses)

self.actor\_loss\_history.append(avg\_actor\_loss)

self.critic\_loss\_history.append(avg\_critic\_loss)

def get\_returns\_and\_advantages(self, rewards, values, is\_terminals):

gae = 0

returns, advantages = [], []

for i in reversed(range(len(rewards))):

if is\_terminals[i]:

next\_value = 0.0

delta = rewards[i] + self.gamma \* (values[i + 1] if i < len(rewards) - 1 else 0) - values[i]

gae = delta + self.gamma \* self.gae\_lambda \* gae

returns.insert(0, (gae + values[i]).detach().clone().float())

advantages.insert(0, gae.detach().clone().float())

returns = torch.tensor(returns)

advantages = torch.tensor(advantages)

if self.with\_normalization:

advantages = (advantages - advantages.mean()) / (advantages.std() + 1e-8)

return returns, advantages

def get\_trajectories(self):

states, actions, rewards, values, is\_terminals, episode\_rewards, log\_probs = [], [], [], [], [], [], []

current\_reward = 0

state, \_ = self.env.reset()

for \_ in range(self.steps):

state\_tensor = torch.as\_tensor(np.array(state), dtype=torch.float32).unsqueeze(0)

with torch.no\_grad():

action\_tensor, log\_prob = self.actor.get\_action(state\_tensor)

value = self.critic(state\_tensor)

action = np.array([action\_tensor.item()])

next\_state, reward, is\_terminal = self.make\_action(action)

states.append(state)

actions.append(action)

log\_probs.append(log\_prob.item())

rewards.append(reward)

is\_terminals.append(is\_terminal)

values.append(value.squeeze())

current\_reward += reward

state = next\_state

if is\_terminal:

episode\_rewards.append(current\_reward)

current\_reward = 0

state, \_ = self.env.reset()

return states, actions, rewards, values, is\_terminals, log\_probs, episode\_rewards

Исходный код actor.py

import torch

import torch.nn as nn

from torch.distributions import Normal

class Actor(nn.Module):

def \_\_init\_\_(self, n\_observations, n\_actions, hidden\_size=256):

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

self.log\_std = nn.Parameter(torch.zeros(n\_actions))

self.model = nn.Sequential(

nn.Linear(n\_observations, hidden\_size),

nn.Tanh(),

nn.Linear(hidden\_size, hidden\_size // 2),

nn.Tanh(),

nn.Linear(hidden\_size // 2, n\_actions)

)

def forward(self, x):

return self.model(x)

def get\_dist(self, x):

mean = self.forward(x)

std = torch.exp(self.log\_std).clamp(1e-6, 1)

return Normal(mean, std)

def get\_action(self, x):

x = torch.as\_tensor(x, dtype=torch.float32)

dist\_x = self.get\_dist(x)

action = dist\_x.rsample()

log\_prob = dist\_x.log\_prob(action).sum(-1)

return action, log\_prob

Исходный код critic.py

import torch.nn as nn

class Critic(nn.Module):

def \_\_init\_\_(self, n\_observations, hidden\_size=128):

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

self.model = nn.Sequential(

nn.Linear(n\_observations, hidden\_size),

nn.Tanh(),

nn.Linear(hidden\_size, hidden\_size // 2),

nn.Tanh(),

nn.Linear(hidden\_size // 2, 1)

)

def forward(self, x):

return self.model(x)