ВМИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ

по лабораторной работе №2

по дисциплине «Обучение с подкреплением»

Тема: Реализация PPO для среды MountainCarContinuous-v0

Студент гр. 0310 _____ Хвостунов М. М.

Преподаватель _____ Глазунов С. А.

Санкт-Петербург

2025

Цель работы.

Написать алгоритм PPO для обучения агента в среде MountainCarContinuous-v0.

Постановка задачи.

1. Изменить длину траектории (steps);
2. Подобрать оптимальный коэффициент clip_ratio;
3. Добавить нормализацию преимуществ;
4. Сравните обучение при разных количествах эпох.

Выполнение задач.

Среда - Mountain Car Continuous

Пространство действий представляет собой одно значение: сила толчка, варьирующаяся от −1 до 1, применяющаяся к машинке.

Награды:

• Равна −0.1 × action2 за каждое действие, чтобы машинка не использовала слишком большие толчки;

• Если машинка достигает конца траектории, то к награде добавляется +100.

Начальное состояние: положение машинки устанавливается случайным образом в диапазоне от −0.6 до −0.4 на основе равномерного распределения.

Окончание эпизода:

• Если машинка достигает флажка (верхней части горки), то эпизод завершается (если позиция машинки больше или равна 0.45);

• Если количество эпизодов равно 999.

Изменение длины траектории.

Использовались следующие значения длины траектории:

$$steps = \{500, 1000, 2000\} \quad (1)$$

Результаты эксперимента представлены на рисунке (Рисунок 1).

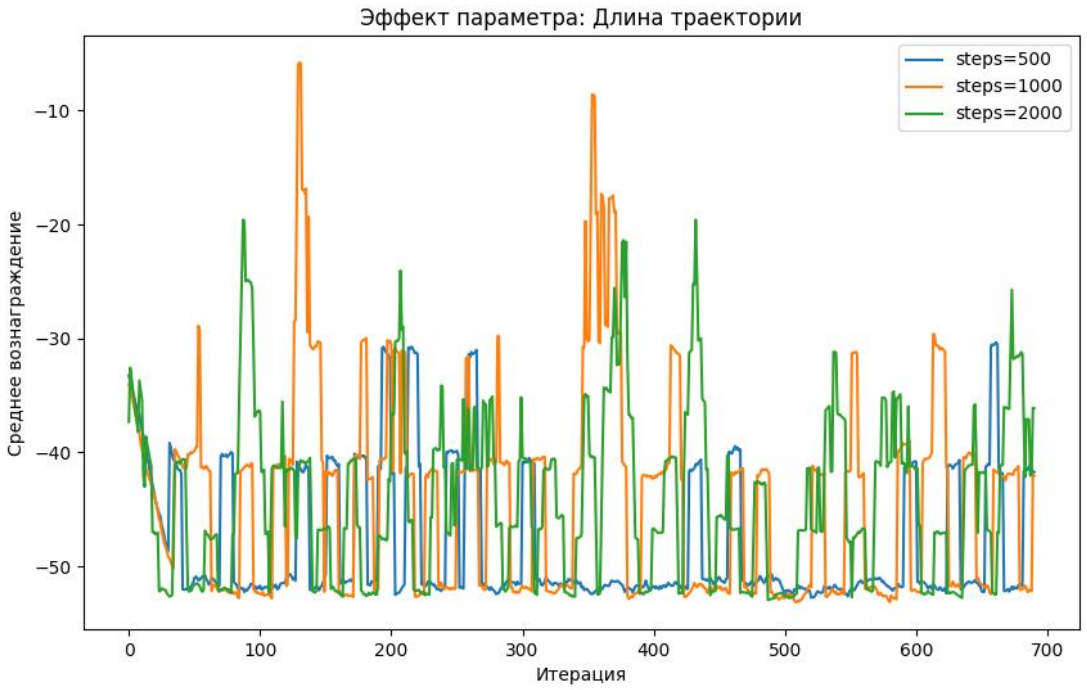

Рисунок 1 – Различные длины траекторий

По результатам анализа графиков можно сделать несколько выводов:

- Значение длины траектории равное 1000 оказалось самым оптимальным по той причине, что именно оно дало большее количество пиков повышения среднего вознаграждения;
- Малое (500) и большое (2000) значения длины траектории привело к ухудшению качества обучения модели. Хоть и не одна из вариаций не смогла достигнуть терминального состояния, эти две вариации значений дали меньший средний показатель вознаграждения;
- Ни одна из вариаций количества шагов не дала финального результата (состояния), что может говорить о недостаточной продолжительности экспериментов, либо о недостатках архитектуры нейронной сети.

Подбор оптимального коэффициента clip_ratio.

Использовались следующие значения длины траектории:

$$\textbf{clip\_ratio} = \{\ 0.15, 0.25, 0.35\} \quad (2)$$

Результаты эксперимента представлены на рисунке (Рисунок 2).

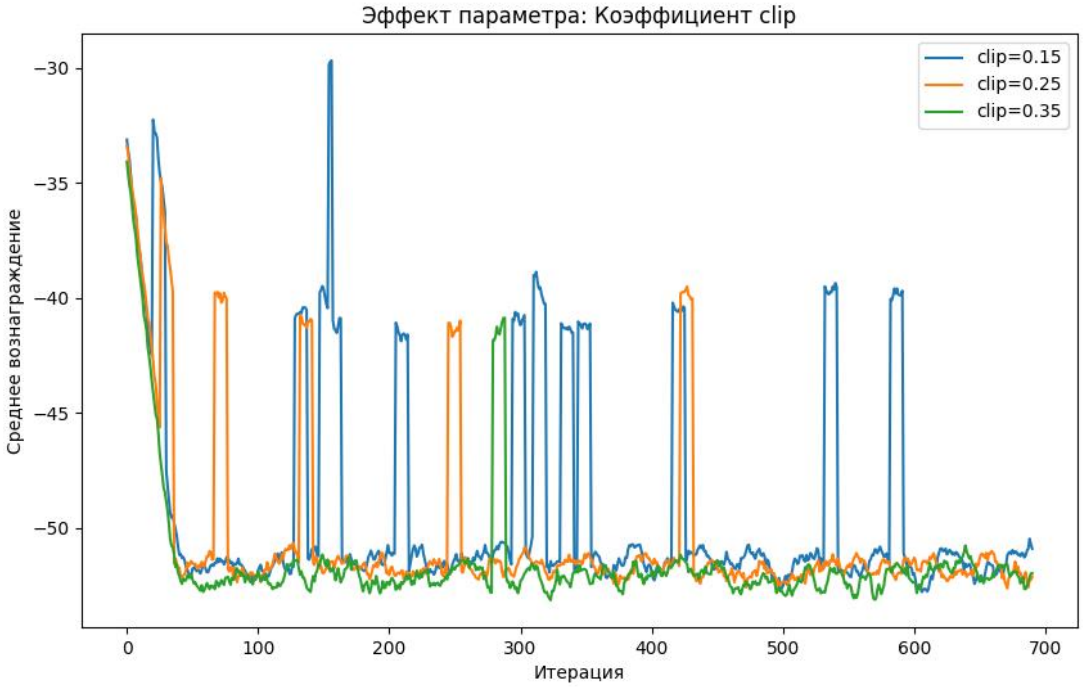

Рисунок 2 – Различные значения clip_ratio

По итогам анализа графиков можно сделать несколько выводов:

- Значения clip_ratio равное 0.15 и 0.25 оказались наиболее удачными. На данных значениях, модель показывала всплески повышения среднего вознаграждения.
- Увеличение значения clip_ratio привело к ухудшению качества предсказания модели, из-за чего результат оказался наименее стабильным и даже и не давал всплесков повышения вознаграждения;
- Ни одна из вариаций значения clip_ratio не дала финального результата (состояния), что может говорить о недостаточной

продолжительности экспериментов, либо о недостатках архитектуры нейронной сети.

Сравнение обучения при разных количествах эпох.

Использовались следующие значения длины траектории:

$$\textbf{epochs} = \{8, 12, 20\} \quad (3)$$

Результаты эксперимента представлены на рисунке (Рисунок 3).

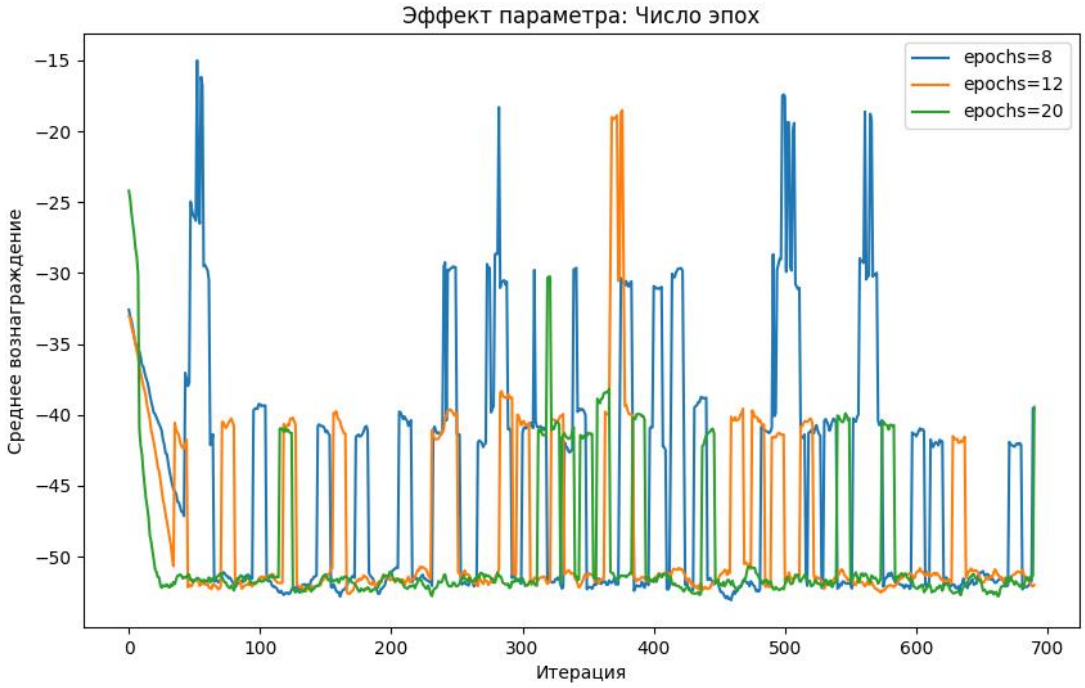

Рисунок 3 – Различное количество эпох

По итогам анализа графиков можно сделать несколько выводов:

- У значений 12 и 20 получились схожие результаты.
- Слишком большое значение (20) дало наихудший результат и модель в данном случае не давала повышения вознаграждения выше 30;
- Ни одна из вариаций значения количства эпох не дала финального результата (состояния), что может говорить о

недостаточной продолжительности экспериментов, либо о недостатках архитектуры нейронной сети.

Добавление нормализации преимуществ.

После нормализации получились графики, представленные ниже (Рисунки 4-6).

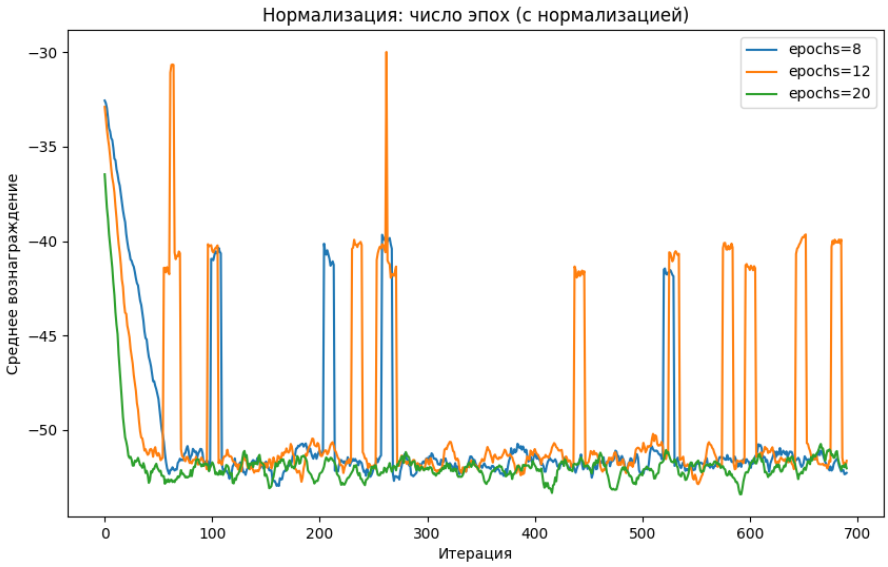

Рисунок 4 – Нормализация с разным числом эпох

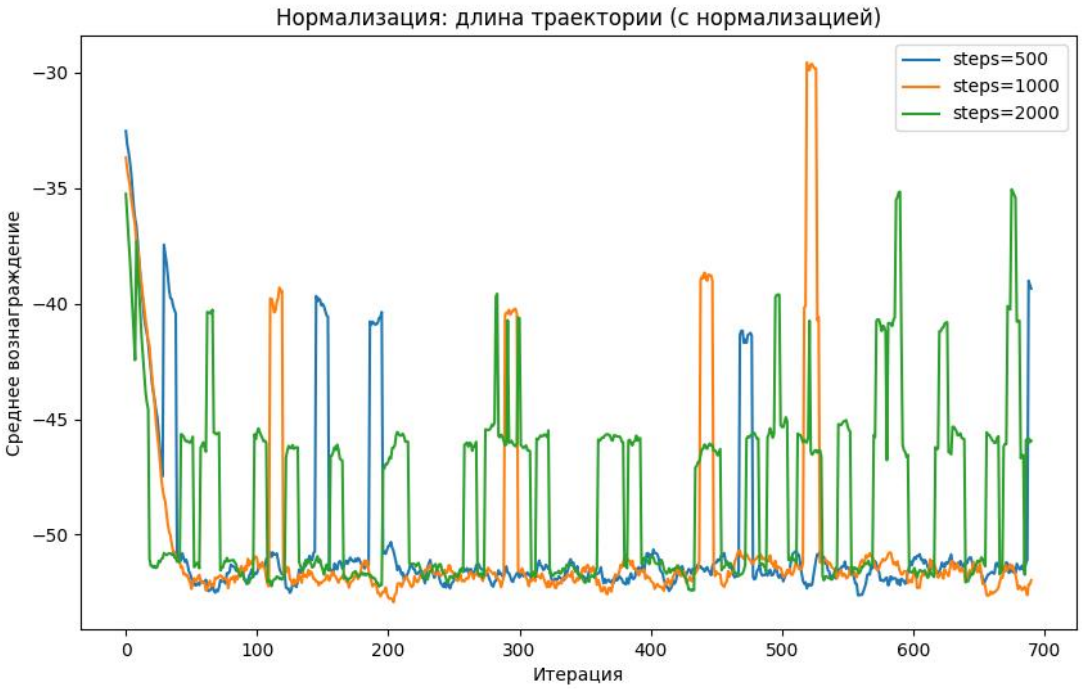

Рисунок 5 – Нормализация с разной длиной траектории

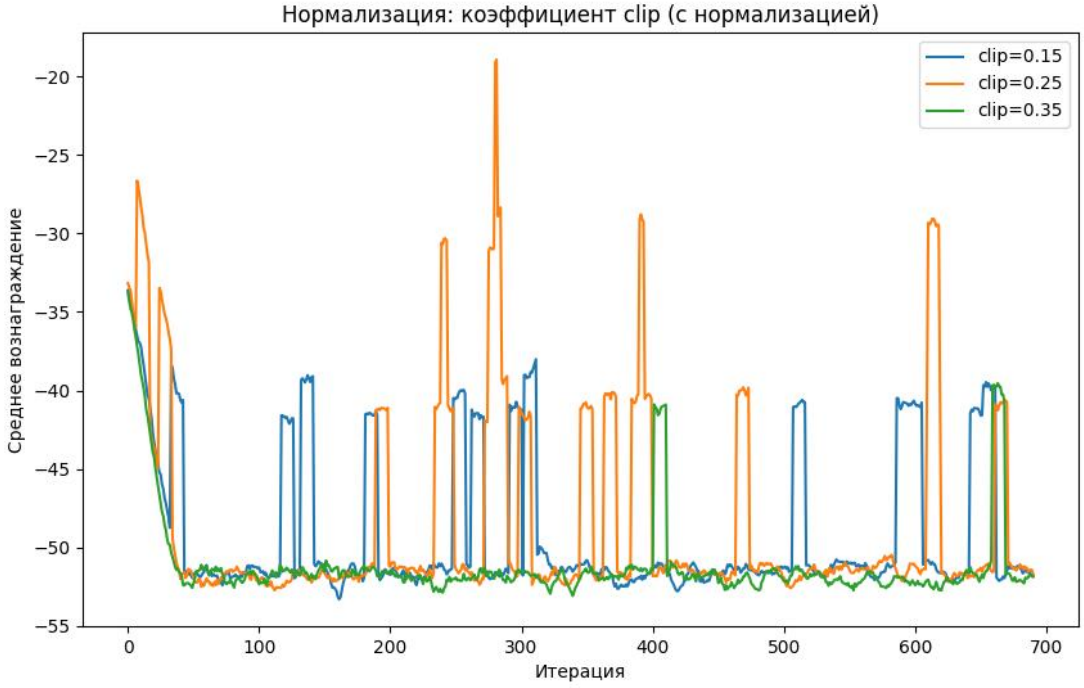

Рисунок 6 – Нормализация с разным числом clip

По полученным графика можно сделать следующие выводы:

• Длина траектории steps

• Результаты с нормализованными преимуществами получились совершенно иными.;

• steps = 500 дало самый худший результат.

• steps = 1000 результат средний, тенденция развития схожа с логарифмической функцией;

• steps = 2000 наилучший результат, тенденция получения лучшего значения вознаграждения стала значительно лучше.

• Коэффициент clip_ratio

• Среднее значение коэффициента по-прежнему дает хороший результат;

• При этом значения 0.15, 0.35 дают практически одинаковые результаты..

• Количество эпох epochs

• Среднее количество эпох (12), дало лучшую модель;

• Соответственно, модель со значением эпох равным 8 и 20 дает худший результат.

Заключение.

В ходе лабораторной работы был реализован алгоритм PPO для обучения агента в среде MountainCarContinuous-v0. Проведенные эксперименты позволили сделать следующие выводы.

1. Оптимальная длина траектории составляет 1000 шагов, так как она обеспечивает более частое стремление вознаграждения к повышению.
2. Коэффициент clip_ratio = 0.25 или 0.15 оказались наиболее подходящими, обеспечивая баланс между стабильностью и скоростью обучения.

3. Увеличение количества эпох положительно влияет на качество обучения, при этом значение 12 эпох показало наилучший результат, а 20 показало чуть менее хороший результат.
4. Нормализация преимуществ улучшает обучение, смещая среднее вознаграждение ближе к нулю. После нормализации длинные траектории (2000 шагов) стали более предпочтительными, а влияние коэффициента clip_ratio стабилизировалось.

Стоит также отметить, что ни одна из вариаций моделей, которые были использованы для построения не достигли терминальных значений, что может говорить о недостаточной сложности и комплексности модели, реализованной в коде, однако финальная версия модели все же дала возможность проанализировать влияния различных параметров на результаты работы модели.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

```
import os
import gymnasium as gym
import numpy as np
import matplotlib.pyplot as plt
import torch
import torch.nn as nn
import torch.optim as optim
from torch.distributions.normal import Normal
from tqdm import tqdm

# Настройки
ENV ID = "MountainCarContinuous-v0"
DEVICE = torch.device("cuda" if torch.cuda.is available() else "cpu")
SEED = 123

# Гиперпараметры (дефолтные)
LR = 2.5e-4
BATCH_SIZE = 256
GAMMA = 0.98
ENTROPY_WEIGHT = 0.02
VALUE_WEIGHT = 0.4
DEFAULT_CLIP = 0.25
DEFAULT_EPOCHS = 12
DEFAULT_STEPS = 900
ITERATIONS = 700

# Варианты экспериментов
CLIPS = [0.15, 0.25, 0.35]
TRAJ LENGTHS = [500, 1000, 2000]
EPOCH OPTIONS = [8, 12, 20]
NORMALIZE_OPTIONS = [True, False]

torch.manual seed(SEED)
np.random.seed(SEED)

class PolicyNetwork(nn.Module):
```

```
    def __init__(self, input_dim, output_dim, size=64):
        super().__init__()
        self.model = nn.Sequential(
            nn.Linear(input_dim, size), nn.Tanh(),
            nn.Linear(size, size), nn.Tanh(),
            nn.Linear(size, size), nn.Tanh(),
            nn.ReLU()
        )
        self.mean = nn.Linear(size, output dim)
        self.log_std = nn.Parameter(torch.full((output_dim,), -0.5))

    def forward(self, x):
        features = self.model(x)
        mean = self.mean(features)
        std = torch.clamp(torch.exp(self.log_std), 1e-3, 1.0)
        return Normal(mean, std)

class ValueNetwork(nn.Module):
    def   init  (self, input dim, size=64):
        super().__init__()
        self.critic = nn.Sequential(
            nn.Linear(input_dim, size), nn.ReLU(),
            nn.Linear(size, size), nn.ReLU(),
            nn.Linear(size, 1)
        )

    def forward(self, x):
        return self.critic(x).squeeze(-1)

def rollout(env, model, steps):
    state, _ = env.reset()
    data = []
    ep rewards = []
    ep return = 0
    steps_taken = 0

    while steps taken < steps or len(ep rewards) == 0:
        st_tensor = torch.tensor(state,
dtype=torch.float32).unsqueeze(0).to(DEVICE)
        with torch.no_grad():
```

```
            dist = model(st_tensor)
            action = dist.sample()
            logprob = dist.log_prob(action).sum(dim=-1)
        clipped_action = torch.clamp(action, -1.0, 1.0)
        next_state, reward, done, trunc, _ = env.step(clipped_ac-
tion.cpu().numpy().flatten())

        data.append((state, clipped_action.squeeze().cpu().numpy(), re-
ward, logprob.item(), done))
        ep_return += reward
        state = next state
        steps_taken += 1

        if done or trunc:
            ep_rewards.append(ep_return)
            ep return = 0
            state,   = env.reset()

    avg return = np.mean(ep rewards)
    return data, avg_return

def calc_advantages(data, val_model, gamma=GAMMA, normalize=True):
    states = torch.tensor(np.array([d[0] for d in data]),
dtype=torch.float32).to(DEVICE)
    rewards = [d[2] for d in data]
    dones = [d[4] for d in data]
    with torch.no_grad():
        values = val_model(states).cpu().numpy()
    returns, advs = [], []
    R = 0
    for r, d, v in zip(reversed(rewards), reversed(dones), reversed(val-
ues)):
        if d: R = 0
        R = r + gamma * R
        returns.insert(0, R)
        advs.insert(0, R - v)
    returns = torch.tensor(returns, dtype=torch.float32).to(DEVICE)
    advantages = torch.tensor(advs, dtype=torch.float32).to(DEVICE)
    if normalize:
```

```
        advantages = (advantages - advantages.mean()) / (advantages.std()
+ 1e-8)
    return returns, advantages

def ppo_update(actor, critic, data, returns, advantages, epochs, clip):
    optimizer_pi = optim.Adam(actor.parameters(), lr=LR)
    optimizer_v = optim.Adam(critic.parameters(), lr=LR)
    states = torch.tensor([d[0] for d in data],
dtype=torch.float32).to(DEVICE)
    actions = torch.tensor(np.array([d[1] for d in data]),
dtype=torch.float32).to(DEVICE)
    old_logp = torch.tensor([d[3] for d in data],
dtype=torch.float32).to(DEVICE)

    for _ in range(epochs):
        for i in range(0, len(states), BATCH SIZE):
            idx = slice(i, i + BATCH SIZE)
            dist = actor(states[idx])
            new logp = dist.log prob(actions[idx]).sum(-1)
            ratio = torch.exp(new_logp - old_logp[idx])
            clipped = torch.clamp(ratio, 1 - clip, 1 + clip) * ad-
vantages[idx]
            pi_loss = -torch.min(ratio * advantages[idx], clipped).mean()
            value_pred = critic(states[idx])
            v_loss = ((returns[idx] - value_pred) ** 2).mean()
            entropy = dist.entropy().mean()
            loss = pi_loss + VALUE_WEIGHT * v_loss - ENTROPY_WEIGHT * en-
tropy

            optimizer_pi.zero_grad()
            optimizer_v.zero_grad()
            loss.backward()
            optimizer pi.step()
            optimizer v.step()

def run variation(title, param name, values):
    env = gym.make(ENV ID)
    results = {}
    for val in values:
        actor = PolicyNetwork(2, 1).to(DEVICE)
```

```
        critic = ValueNetwork(2).to(DEVICE)
        config = {
            "clip": DEFAULT_CLIP,
            "epochs": DEFAULT_EPOCHS,
            "steps": DEFAULT_STEPS,
            "normalize": True
        }
        config[param_name] = val
        rewards = []
        for _ in range(ITERATIONS):
            batch, avg r = rollout(env, actor, config["steps"])
            rets, advs = calc_advantages(batch, critic, normalize=con-
fig["normalize"])
            ppo update(actor, critic, batch, rets, advs, con-
fig["epochs"], config["clip"])
            rewards.append(avg r)
        results[f"{param name}={val}"] = rewards

    plt.figure(figsize=(10, 6))
    def smooth(x, k=10):
        return np.convolve(x, np.ones(k)/k, mode='valid')

    for label, rew in results.items():
        plt.plot(smooth(rew), label=label)
    plt.xlabel("Итерация")
    plt.ylabel("Среднее вознаграждение")
    plt.title(f"Эффект параметра: {title}")
    plt.legend()
    os.makedirs("fig_custom", exist_ok=True)
    plt.savefig(f"fig_custom/ppo_{param_name}.png")
    print(f"График сохранён: fig_custom/ppo_{param_name}.png")

def main():
    # Без нормализации для трёх экспериментов
    run_variation("Длина траектории", "steps", TRAJ_LENGTHS)
    run variation("Коэффициент clip", "clip", CLIPS)
    run variation("Число эпох", "epochs", EPOCH OPTIONS)

    # С нормализацией преимуществ для всех тех же параметров
    for name, param, values in [
```

```
        ("Нормализация: длина траектории", "steps", TRAJ_LENGTHS),
        ("Нормализация: коэффициент clip", "clip", CLIPS),
        ("Нормализация: число эпох", "epochs", EPOCH_OPTIONS)
    ]:
        env = gym.make(ENV_ID)
        results = {}
        for val in values:
            actor = PolicyNetwork(2, 1).to(DEVICE)
            critic = ValueNetwork(2).to(DEVICE)
            rewards = []
            for   in range(ITERATIONS):
                batch, avg_r = rollout(env, actor, val if param ==
"steps" else DEFAULT STEPS)
                rets, advs = calc advantages(batch, critic, normal-
ize=True)
                ppo update(actor, critic, batch, rets, advs,
                        val if param == "epochs" else DEFAULT EPOCHS,
                        val if param == "clip" else DEFAULT_CLIP)
                rewards.append(avg r)
            results[f"{param}={val}"] = rewards

        plt.figure(figsize=(10, 6))
        def smooth(x, k=10):
            return np.convolve(x, np.ones(k)/k, mode='valid')

        for label, rew in results.items():
            plt.plot(smooth(rew), label=label)
        plt.xlabel("Итерация")
        plt.ylabel("Среднее вознаграждение")
        plt.title(f"{name} (с нормализацией)")
        plt.legend()
        os.makedirs("fig_custom", exist_ok=True)
        filename = f"fig custom/ppo norm {param}.png"
        plt.savefig(filename)
        print(f"График сохранён: {filename}")

if   name   == "  main  ":
    main()
```