МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЁТ

по лабораторной работе № 2 по дисциплине

«Обучение с подкреплением»

Тема: Реализация PPO для среды MountainCarContinuous-v0

Студентка гр. 0310 Шкода М. А.

Преподаватель Глазунов С.А.

Санкт-Петербург
2025

Цель работы

Реализация PPO для среды MountainCarContinuous-v0 и сравнительный анализ параметров.

Задание

1. Измените длину траектории (steps).
2. Подберите оптимальный коэффициент clip_ratio.
3. Добавьте нормализацию преимуществ.
4. Сравните обучение при разных количествах эпох.

Ход работы

Для выполнения работы были реализованы классы: Actor, он отвечает за стратегию выбора действий. С помощью метода act происходит выбор действия. Код класса представлен ниже. Весь код приведен в приложении А.

```
class Actor(nn.Module):
    def __init__(self, state_dim, action_dim):
        super().__init__()
        self.shared = nn.Sequential(
            nn.Linear(state_dim, 64),
            nn.Tanh(),
            nn.Linear(64, 64),
            nn.Tanh()
        )
        self.mean = nn.Linear(64, action_dim)
        self.log_std = nn.Parameter(torch.zeros(action_dim))

    def forward(self, x):
        x = self.shared(x)
        return self.mean(x), self.log_std.exp()
```

```
    def get_dist(self, state):
        mean, std = self.forward(state)
        return Normal(mean, std)

    def act(self, state):
        state = torch.tensor(state, dtype=torch.float32).to(device)
        dist = self.get_dist(state)
        action = dist.sample()
        return action.cpu().numpy(),
dist.log_prob(action).sum().item()
```

Класс Critic реализует оценку текущего состояния. Он стремится уменьшить MSE (ошибка между предсказанным и фактическим результатом) на каждой итерации.

```
critic_loss = (critic(s).squeeze() - ret).pow(2).mean()
```

На рисунках 1.1 – 1.2 показаны графики наград и потерь при изменении длины траектории по значениям 512, 1024, 2048. При длине 2048 величина награды стабильно растёт и уже до 100 итерации подходит к досрочному завершению алгоритма. При этом функция потерь не стремится к нулю, что может быть связано с резкой политикой обновлений.

Результаты для 512 и 1024 имеют стремящиеся к нулю потери и достигают к 150 итерации приблизительно одинаковых наград. При 512 в начале награды больше, но всё ещё неположительные. Рост при этом значении длины медленный и не достигает хороших наград, поэтому можно считать длину 2048 оптимальной. В дальнейшем для сравнительного анализа будет использоваться она.

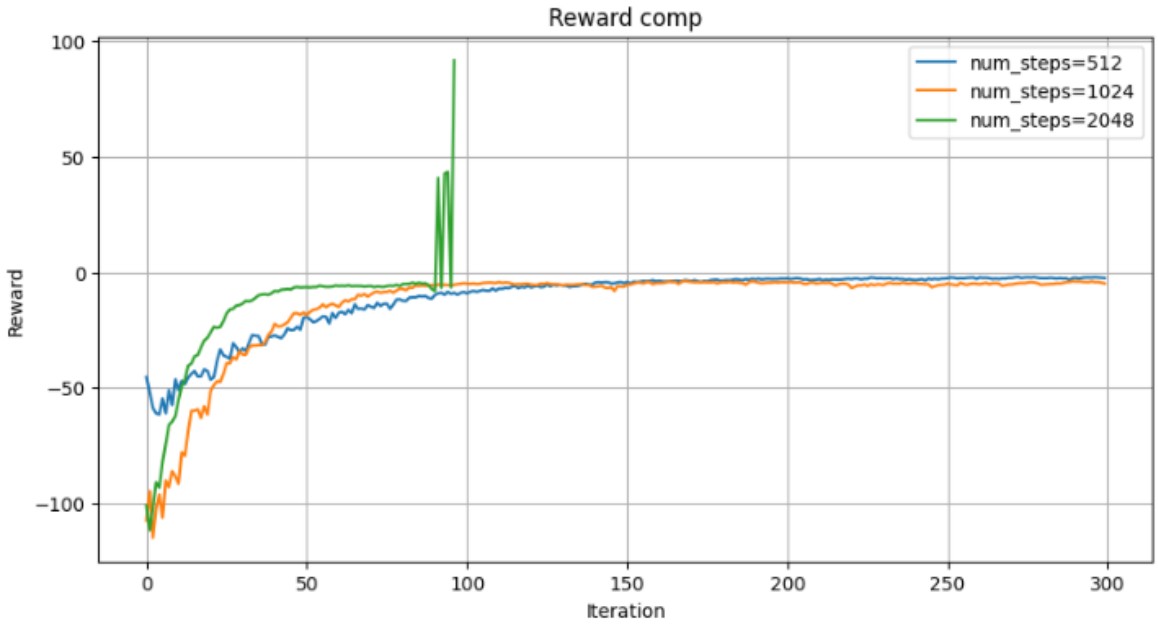

Рисунок 1.1 – График наград при изменении длины траектории

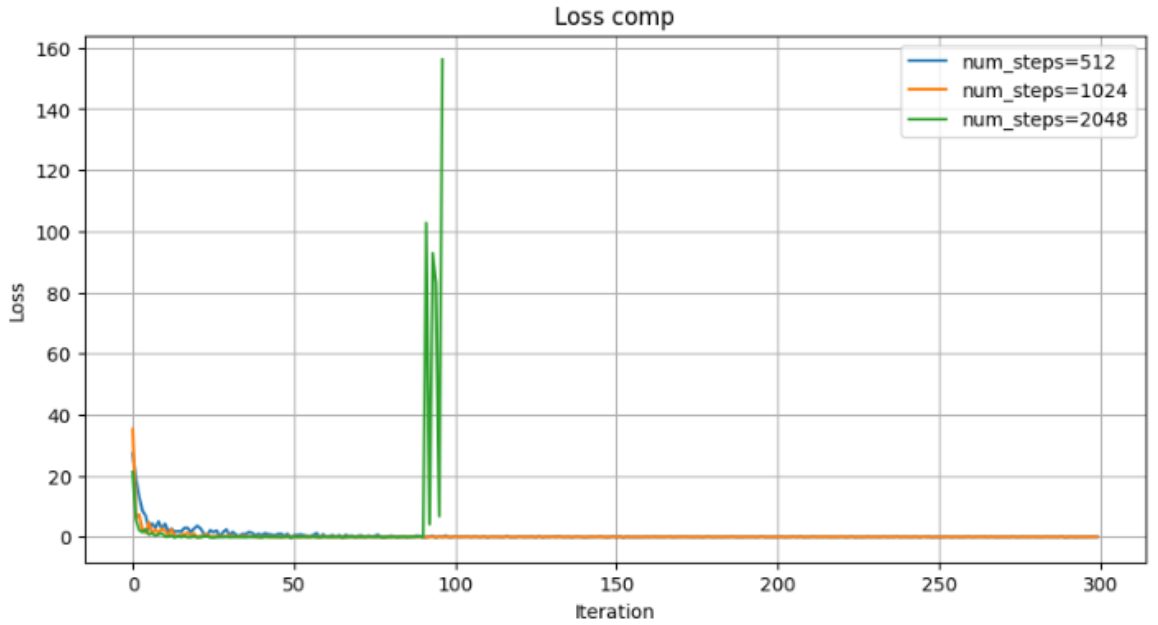

Рисунок 1.2 – График потерь при изменении длины траектории

На рисунках 2.1 – 2.2 показаны графики наград и потерь при изменении параметра clip_ratio 0.1, 0.2, 0.5. Данный параметр отвечает за ограничение изменение политики. При небольших значениях 0.1 и 0.2 скорость обучения высокая, при этом быстрый рост награды отражается на графике потерь.. При 0.5 обучение медленное. Данную величину

параметра нельзя назвать стабильной. Такое большое значение clip_ratio даёт широкий диапазон при обновлении шагов, что не сказывается положительно на обучении. Например, при другом seed со значением clip_ratio 0.5 награда не превышает ноль, что видно на рисунке 2.3. При этом показатели с коэффициентами 0.1 и 0.2 даже улучшились. В связи с этим предпочтительнее использовать параметры 0.1 – 0.2.

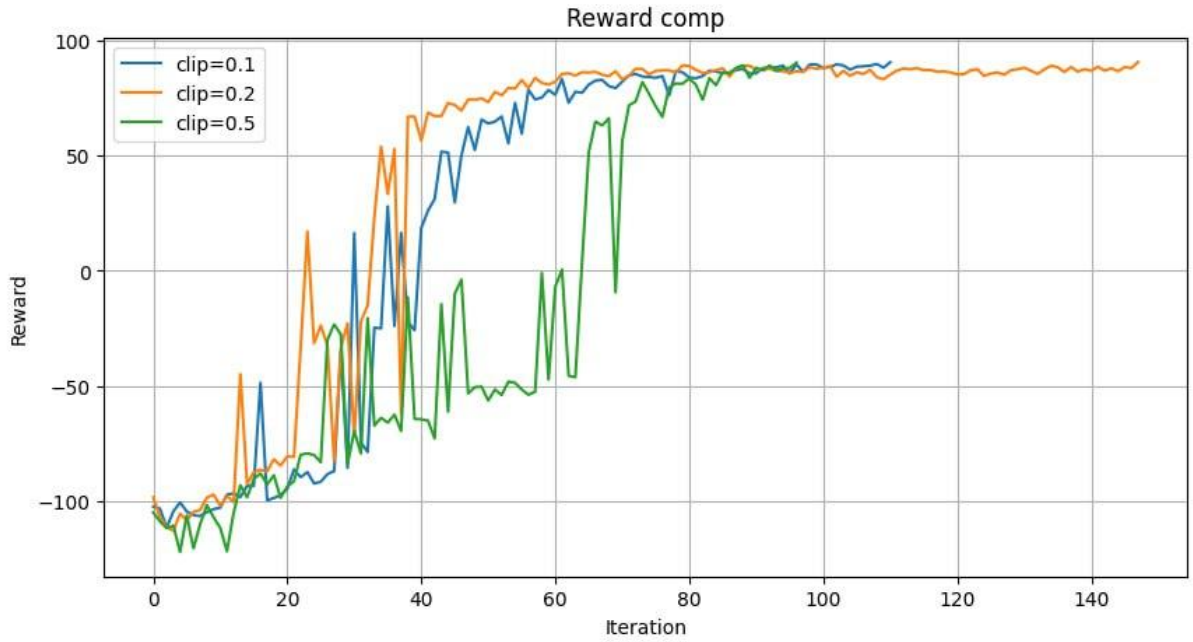

Рисунок 2.1 – График наград при изменении clip_ratio

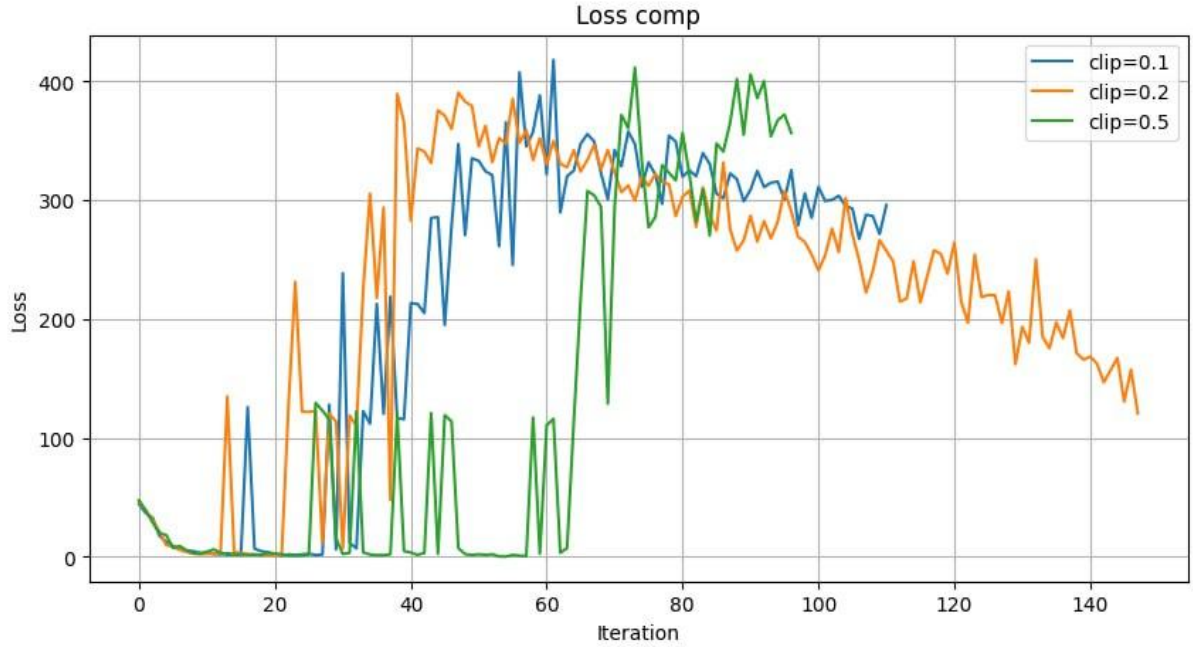

Рисунок 2.2 – График потерь при изменении clip_ratio

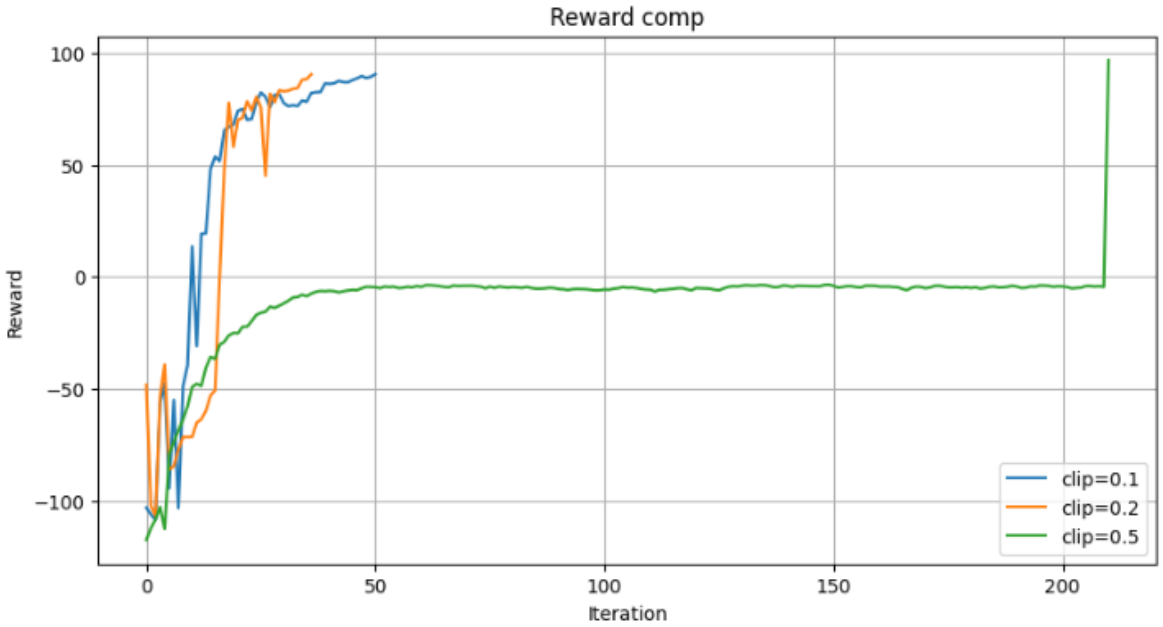

Рисунок 2.3– График наград при изменении clip_ratio

На рисунках 3.1 – 3.2 показаны графики наград и потерь при разных количествах эпох 10, 30 и 40. Как и ожидалось, скорость обучения повышается с увеличением количества эпох, так как этот параметр отвечает за количество «проходов» по одним и тем же данным и чем он выше, тем тщательнее происходит обучение. Так, при количестве эпох 30 обучение происходит быстро и достигается нужное значение награды (было установлено 90) до 50ой итерации. Для 10 эпох график растёт нестабильно, но всё же возрастает и до 150ой итерации также доходит до досрочного завершения. В то время как при 40 эпохах величина награды не превосходит 0 и имеется медленный рост. Это может быть связано с переобучением, излишняя «тщательность» может негативно отразиться на результатах обучения.

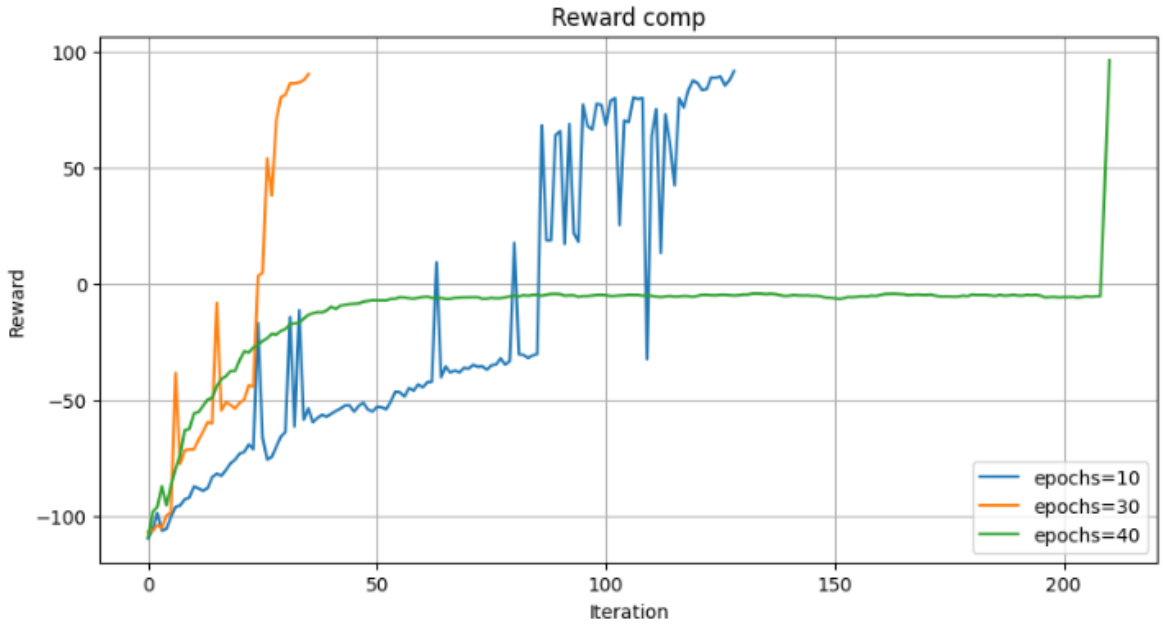

Рисунок 3.1 – График наград при разных количествах эпох

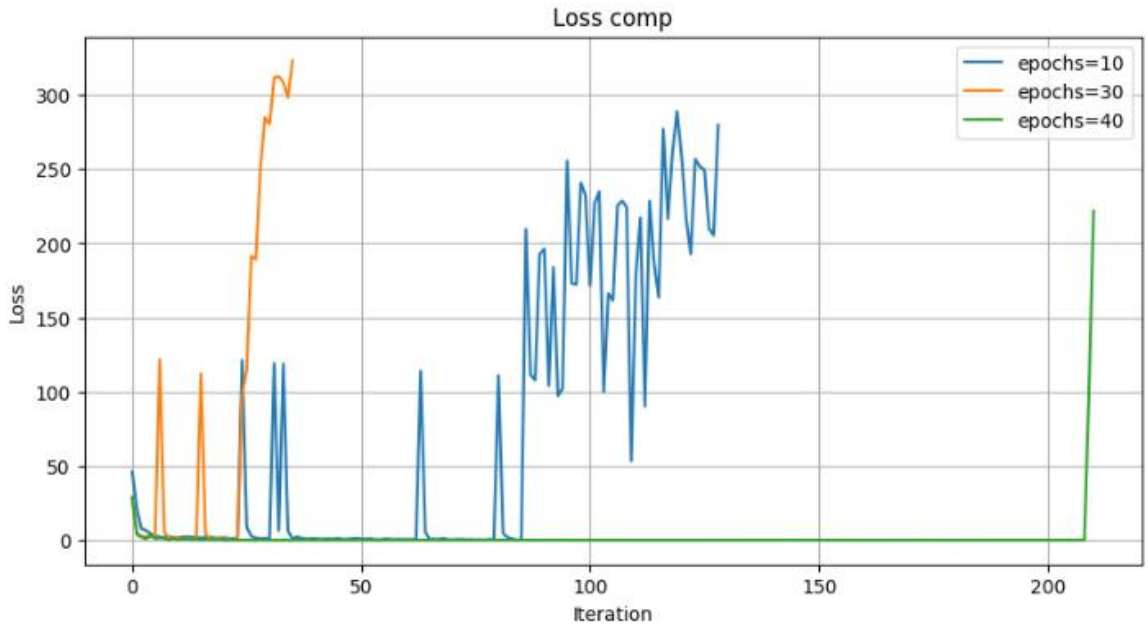

Рисунок 3.2 – График потерь при разных количествах эпох

На рисунках 4.1 – 6.2 показаны результаты после нормализации преимуществ. После нормализации обучение стало стабильнее, однако даже при длине траектории 2048 или 10-30 эпохах не достигает высоких значений награды..

Однако для коэффициента clip_ratio удается получить более стабильные результаты для 0.1. Награда превышает 75, что на фоне остальных нулевых значений является отличным результатом.

Подобное влияние нормализации может быть связано с тем, что при нормализации вклад награды становится меньше, ощущается не так существенно, в связи с чем агент не приходит к нужной стратегии. Нормализация проводилась по всей траектории, не учитывая батчи и не исключая возможные выбросы.

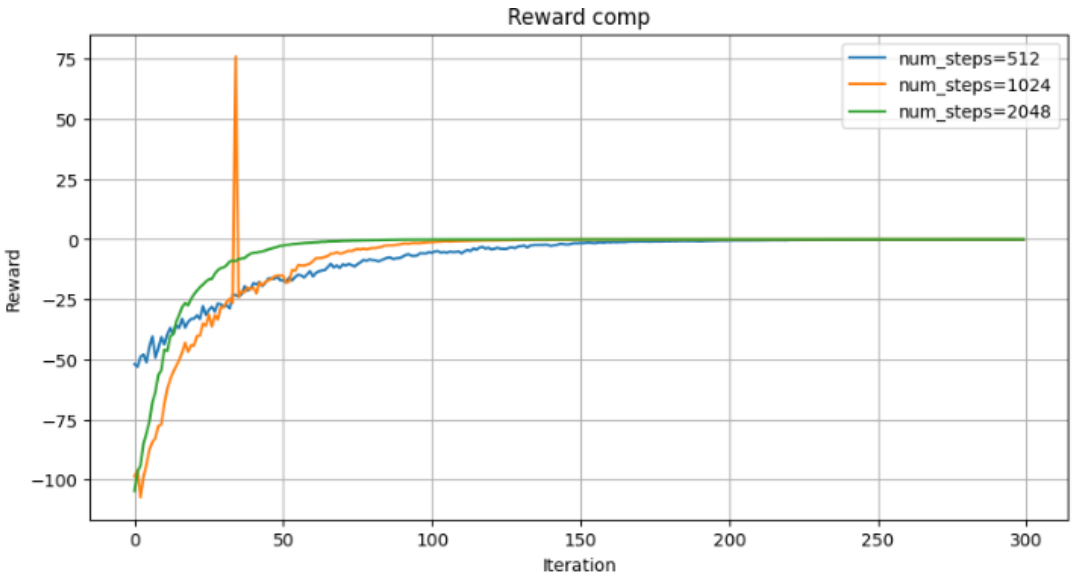

Рисунок 4.1 – График наград при изменении длины траектории после нормализации

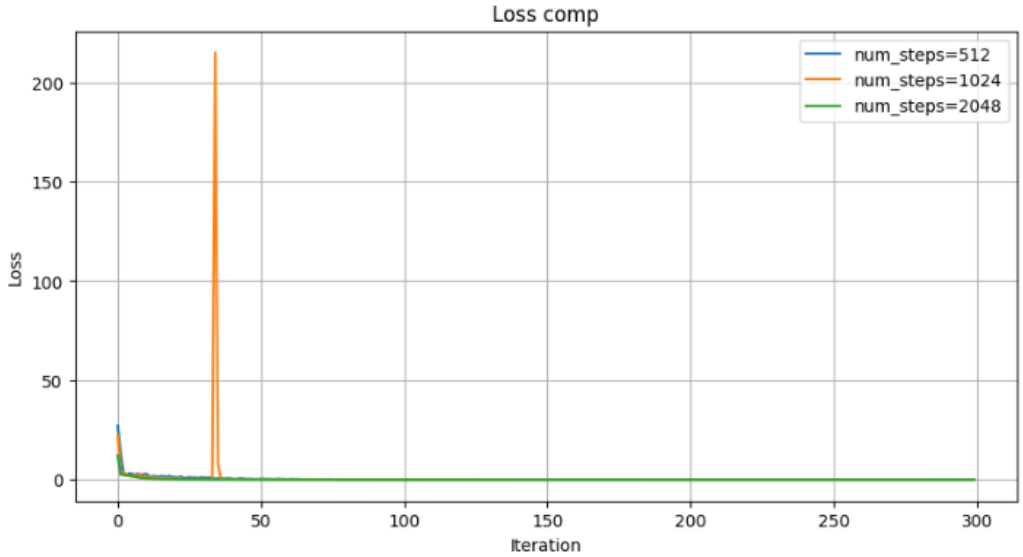

Рисунок 4.2 – График потерь при изменении длины траекториия после нормализации

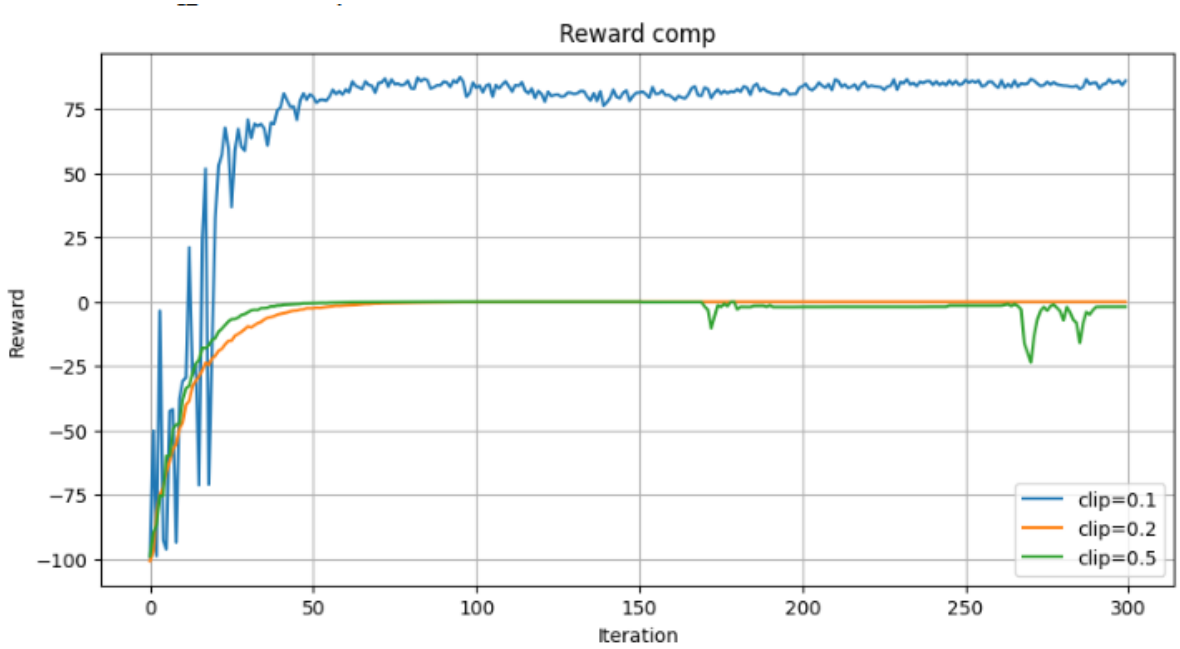

Рисунок 5.1 – График наград при изменении clip_ratio после нормализации

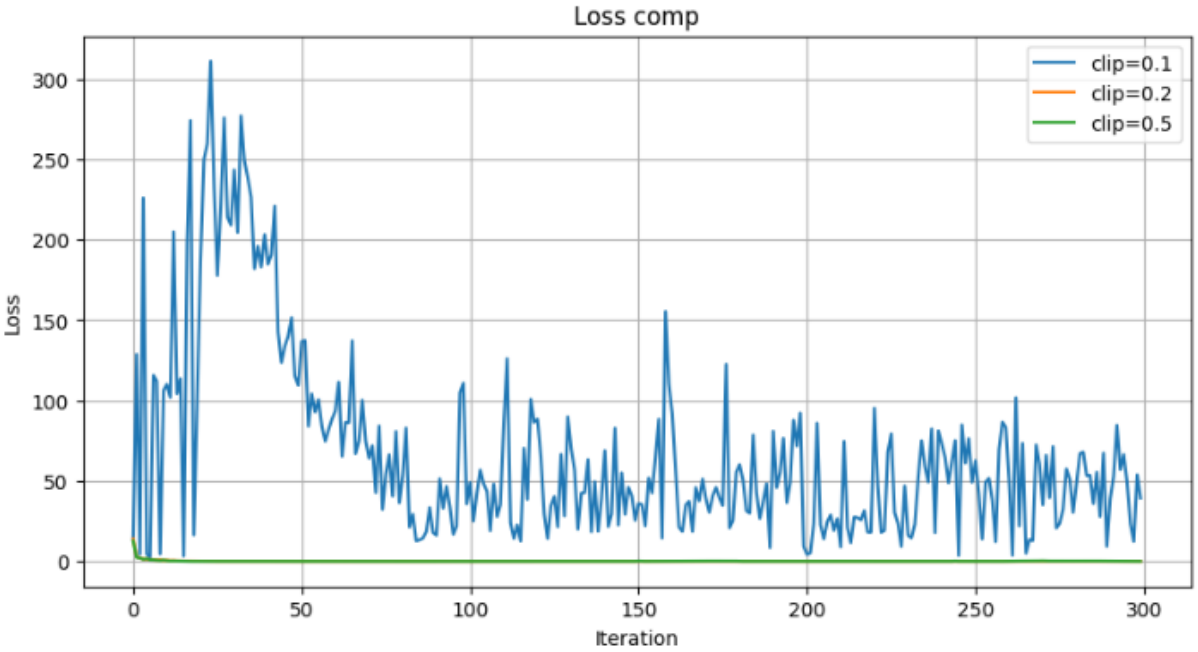

Рисунок 5.2 – График потерь при изменении clip_ratio после нормализации

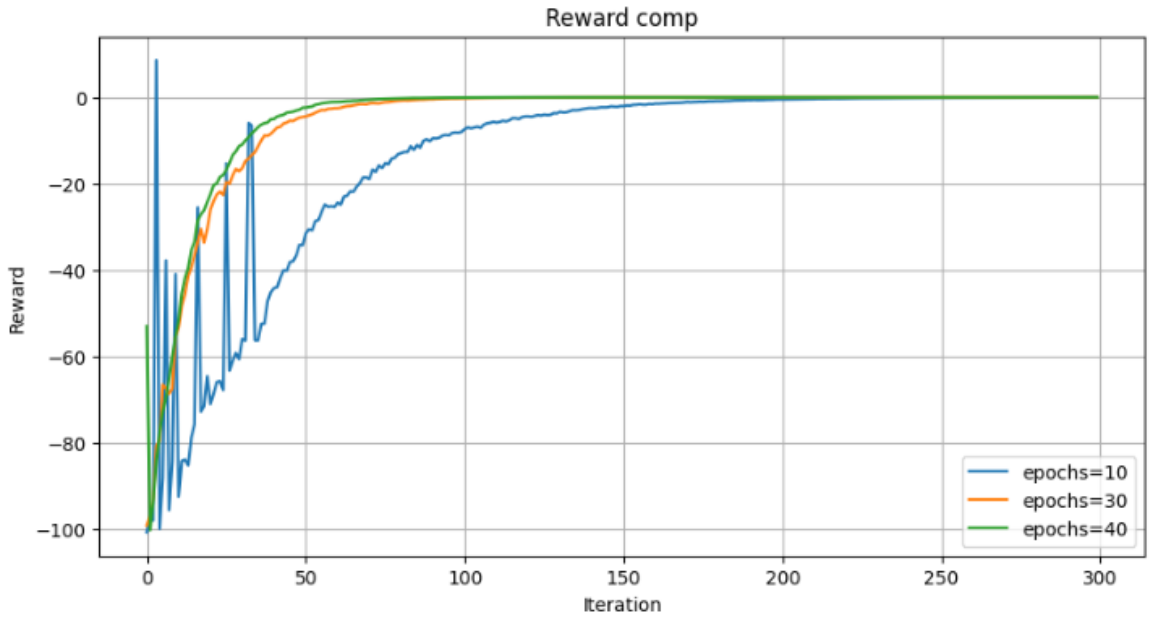

Рисунок 6.1 – График наград при разных количествах эпох после нормализации

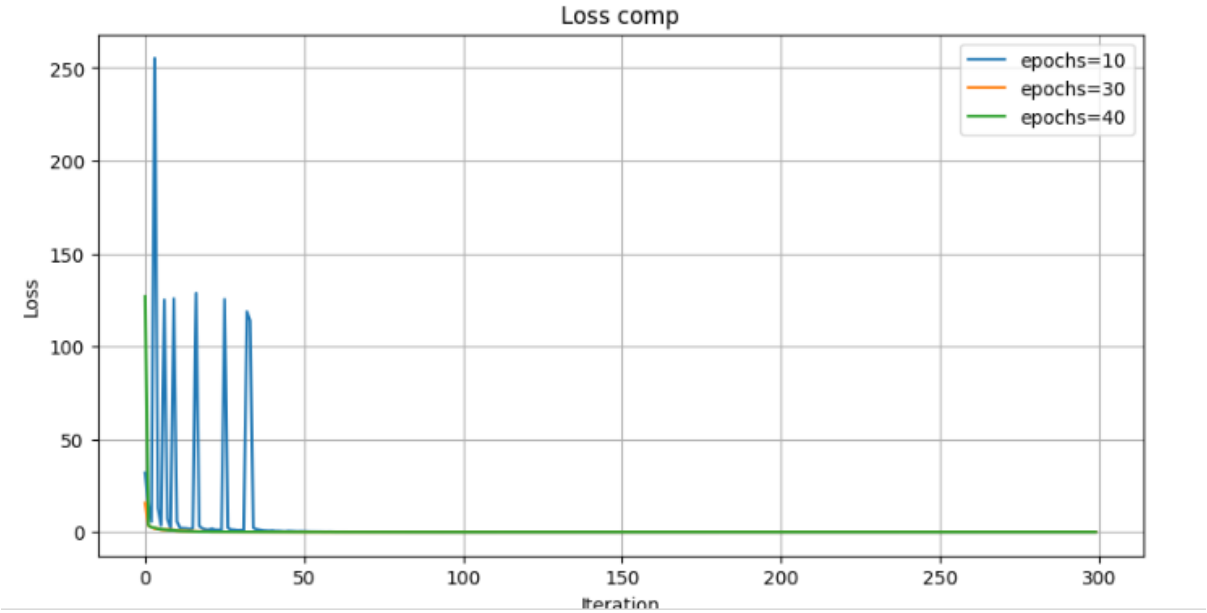

Рисунок 6.2 – График потерь при разных количествах эпох после нормализации

Заключение

В результате выполнения данной работы был реализован алгоритм PPO и проведен анализ влияния различных параметров на обучение. Наилучшей комбинацией параметров стали значения: длина траектории = 2048, количество эпох = 30, коэффициент clip_ratio = 0.1/0.2. При данных параметрах было достигнуто максимальное значение награды и алгоритм был завершен досрочно до 50 итерации. При других значениях награда не превышала нуля, что могло быть связано как с переобучением в связи с слишком углубленной работой над конкретными данными, так и с недостаточно строгой политикой в случае широкого ограничения на шаг обновления.

Добавление нормализации негативно сказалось на изменении длины траектории и количества эпох, замедлив обучение и ограничив величину награды нулем. Причиной этому может послужить сниженная значимость преимуществ после нормализации. Ожидаемой сходимости и стабильности для данных параметров не возникло.

Приложение А

```
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 import Normal

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
env_name = "MountainCarContinuous-v0"
#параметры по умолчанию
num_iterations = 300
num_steps = 2048 #длина траектории
ppo_epochs = 10
mini_batch_size = 256
gamma = 0.99
clip_ratio = 0.2
value_coef = 0.5
entropy_coef = 0.01
lr = 3e-4


class Actor(nn.Module):
    def __init__(self, state_dim, action_dim):
        super().__init__()
        self.shared = nn.Sequential(
            nn.Linear(state_dim, 64),
            nn.Tanh(),
            nn.Linear(64, 64),
            nn.Tanh()
        )
        self.mean = nn.Linear(64, action_dim)
        self.log_std = nn.Parameter(torch.zeros(action_dim))

    def forward(self, x):
        x = self.shared(x)
        return self.mean(x), self.log_std.exp()
```

```
    def get_dist(self, state):
        mean, std = self.forward(state)
        return Normal(mean, std)

    def act(self, state):
        state = torch.tensor(state, dtype=torch.float32).to(device)
        dist = self.get_dist(state)
        action = dist.sample()
        return action.cpu().numpy(), dist.log_prob(action).sum().item()


class Critic(nn.Module):
    def __init__(self, state_dim):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(state_dim, 64),
            nn.Tanh(),
            nn.Linear(64, 64),
            nn.Tanh(),
            nn.Linear(64, 1)
        )

    def forward(self, x):
        return self.net(x)


def collect_trajectories(env, policy, steps):
    states, actions, log_probs, rewards, dones = [], [], [], [], []
    state, _ = env.reset(seed=1)
    for _ in range(steps):
        action, log_prob = policy.act(state)
        next_state, reward, terminated, truncated, _ = env.step(action)
        done = terminated or truncated

        states.append(state)
        actions.append(action)
        log_probs.append(log_prob)
        rewards.append(reward)
        dones.append(done)

        state = next_state
```

```
        if done:
            state, _ = env.reset()

    return map(np.array, (states, actions, log_probs, rewards, dones))


def compute_advantages(rewards, dones, values, norm):
    returns, advantages = [], []
    R = 0
    for r, d, v in zip(reversed(rewards), reversed(dones),
reversed(values)):
        R = r + gamma * R * (1 - d)
        returns.insert(0, R)
        advantages.insert(0, R - v)
    returns, advantages = np.array(returns), np.array(advantages)
    #нормализация преимуществ
    if norm:
      advantages = (advantages - advantages.mean()) / (advantages.std() +
1e-8)
    return returns, advantages


def train(num_steps, clip_ratio, ppo_epochs, norm):
    env = gym.make(env_name)
    actor = Actor(2, 1).to(device)
    critic = Critic(2).to(device)

    opt_actor = optim.Adam(actor.parameters(), lr=lr)
    opt_critic = optim.Adam(critic.parameters(), lr=lr)

    avg_rewards = []
    total_losses = []

    for iteration in range(num_iterations):
        states, actions, log_probs, rewards, dones =
collect_trajectories(env, actor, num_steps)
        states_tensor = torch.tensor(states,
dtype=torch.float32).to(device)
        actions_tensor = torch.tensor(actions,
dtype=torch.float32).to(device)
```

```
        old_log_probs_tensor = torch.tensor(log_probs,
dtype=torch.float32).to(device)
        values = critic(states_tensor).squeeze().detach().cpu().numpy()

        returns, advantages = compute_advantages(rewards, dones, values,
norm)
        returns = torch.tensor(returns, dtype=torch.float32).to(device)
        advantages = torch.tensor(advantages,
dtype=torch.float32).to(device)

        iter_losses = []
        for _ in range(ppo_epochs):
            idxs = np.arange(len(states))
            np.random.shuffle(idxs)
            for start in range(0, len(states), mini_batch_size):
                end = start + mini_batch_size
                batch_idx = idxs[start:end]

                s = states_tensor[batch_idx]
                a = actions_tensor[batch_idx]
                old_logp = old_log_probs_tensor[batch_idx]
                ret = returns[batch_idx]
                adv = advantages[batch_idx]

                dist = actor.get_dist(s)
                new_logp = dist.log_prob(a).sum(dim=-1)
                ratio = torch.exp(new_logp - old_logp)

                actor_loss = -torch.min(ratio * adv, torch.clamp(ratio, 1 -
clip_ratio, 1 + clip_ratio) * adv).mean()
                entropy = dist.entropy().mean()
                critic_loss = (critic(s).squeeze() - ret).pow(2).mean()

                loss = actor_loss + value_coef * critic_loss - entropy_coef
* entropy
                iter_losses.append(loss.item())

                opt_actor.zero_grad()
                opt_critic.zero_grad()
                loss.backward()
                opt_actor.step()
```

```
                opt_critic.step()

        avg_reward = sum(rewards) / (sum(dones) if sum(dones) > 0 else 1)
        avg_rewards.append(avg_reward)
        total_losses.append(np.mean(iter_losses))

        if iteration%25 == 0:
          print(f"iteration {iteration}: avg_reward: {avg_reward:.2f},
loss: {total_losses[-1]:.4f}")

        if avg_reward >= 90:
            break


    torch.save(actor.state_dict(),
f"ppo_actor_steps{num_steps}_clip{clip_ratio}_epochs{ppo_epochs}.pth")
    return avg_rewards, total_losses

def plot_results(rewards_dict, losses_dict):
    plt.figure(figsize=(10, 5))
    for name, rewards in rewards_dict.items():
        plt.plot(rewards, label=name)
    plt.title('Reward comp')
    plt.xlabel('Iteration')
    plt.ylabel('Reward')
    plt.legend()
    plt.grid(True)
    plt.show()

    plt.figure(figsize=(10, 5))
    for name, losses in losses_dict.items():
        plt.plot(losses, label=name)
    plt.title('Loss comp')
    plt.xlabel('Iteration')
    plt.ylabel('Loss')
    plt.legend()
    plt.grid(True)
    plt.show()

cr_rewards = {}
cr_losses = {}
```

```
for cr in [0.1, 0.2, 0.5]:
  avg_rewards, total_losses = train(num_steps=2048, clip_ratio=cr,
ppo_epochs=40, norm = False)
  cr_rewards[f"clip={cr}"] = avg_rewards
  cr_losses[f"clip={cr}"] = total_losses

plot_results(cr_rewards, cr_losses)

cr_rewards_tr = {}
cr_losses_tr = {}

for cr in [0.1, 0.2, 0.5]:
  avg_rewards, total_losses = train(num_steps=2048, clip_ratio=cr,
ppo_epochs=40, norm = True)
  cr_rewards_tr[f"clip={cr}"] = avg_rewards
  cr_losses_tr[f"clip={cr}"] = total_losses

plot_results(cr_rewards_tr, cr_losses_tr)

epochs_rewards = {}
epochs_losses = {}

for ep in [10, 30, 40]:
  avg_rewards, total_losses = train(num_steps=2048, clip_ratio=0.2,
ppo_epochs=ep, norm = False)
  epochs_rewards[f"epochs={ep}"] = avg_rewards
  epochs_losses[f"epochs={ep}"] = total_losses

plot_results(epochs_rewards, epochs_losses)

epochs_rewards = {}
epochs_losses = {}

for ep in [10, 30, 40]:
  avg_rewards, total_losses = train(num_steps=2048, clip_ratio=0.2,
ppo_epochs=ep, norm = True)
  epochs_rewards[f"epochs={ep}"] = avg_rewards
  epochs_losses[f"epochs={ep}"] = total_losses

plot_results(epochs_rewards, epochs_losses)
```

```
step_rewards = {}
step_losses = {}

for st in [512, 1024, 2048]:
  avg_rewards, total_losses = train(num_steps=st, clip_ratio=0.2,
ppo_epochs=40, norm = False)
  step_rewards[f"num_steps={st}"] = avg_rewards
  step_losses[f"num_steps={st}"] = total_losses

plot_results(step_rewards, step_losses)

step_rewards = {}
step_losses = {}

for st in [512, 1024, 2048]:
  avg_rewards, total_losses = train(num_steps=st, clip_ratio=0.2,
ppo_epochs=40, norm = True)
  step_rewards[f"num_steps={st}"] = avg_rewards
  step_losses[f"num_steps={st}"] = total_losses

plot_results(step_rewards, step_losses)
```