МИНОБРНАУКИ 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 Волков К.А.

Преподаватель Глазунов С.А.

Санкт-Петербург

2025

СОДЕРЖАНИЕ

ЦЕЛЬ РАБОТЫ

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

ЗАДАНИЕ

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

ВЫПОЛНЕНИЕ РАБОТЫ

1. Исходные данные

Название среды: Mountain Car Continuous.

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.

2. Изменение длины траектории

Использовались следующие значения длины траектории:

$$Max\_steps = 2048$$

Результаты эксперимента представлены на рисунке 1.

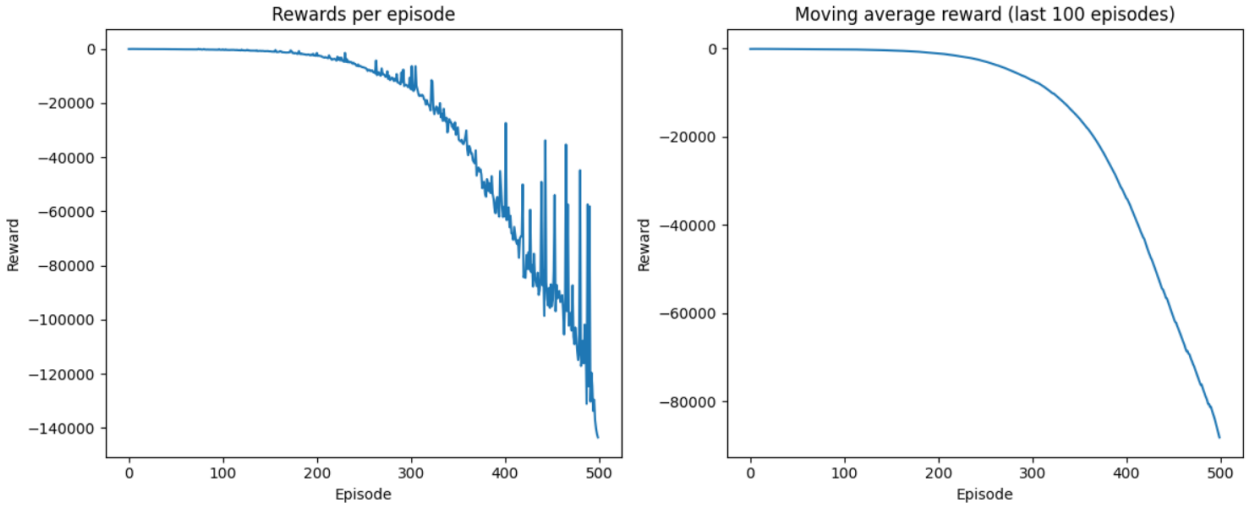

Рисунок 1 – Среднее вознаграждение в зависимости от длины траектории

3. Подбор оптимального коэффициента clip_ratio

В ходе эксперимента использовались следующие значения:

```
def experiment_clip_ratio(env, clip_ratios=[0.1, 0.2, 0.3], episodes=300):
```

Результаты эксперимента представлены на рисунке 2.

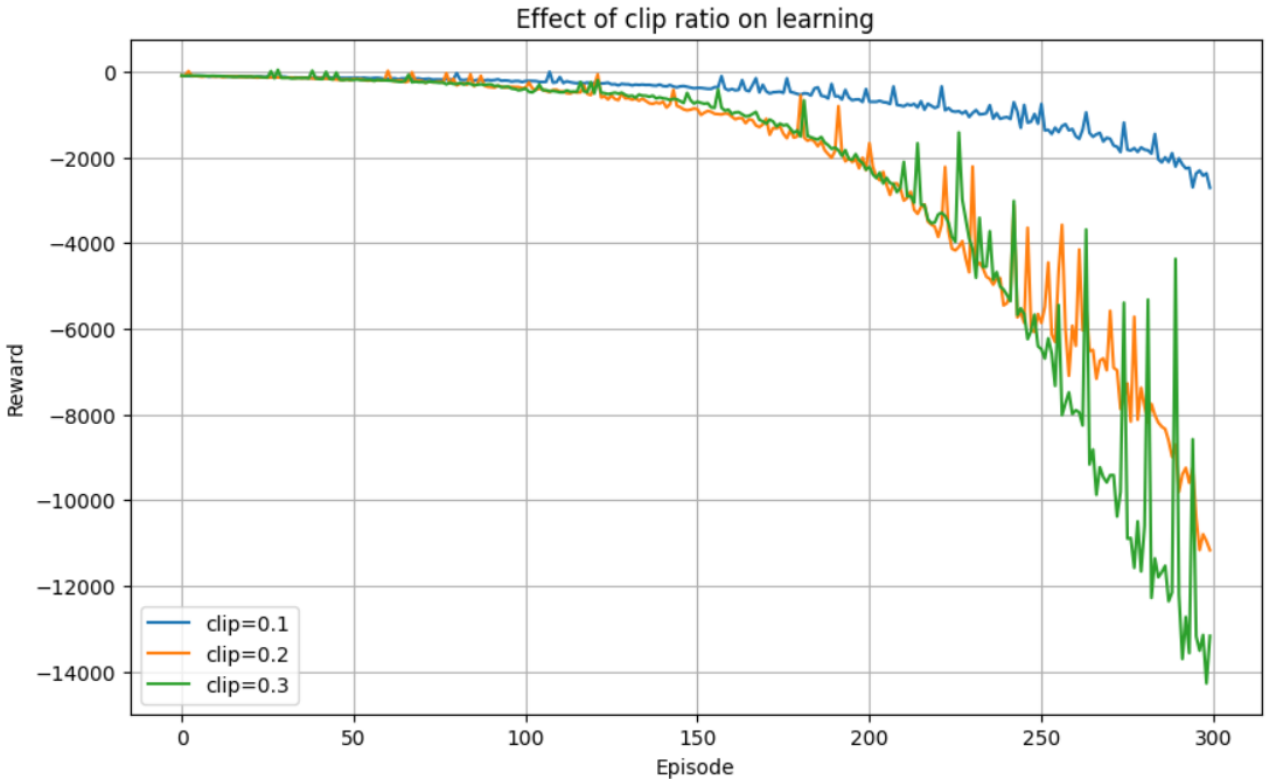

Рисунок 2 – Среднее вознаграждение в зависимости от коэффициента clip_ratio

4. Сравнение обучения при разных количествах эпох

Для проведения эксперимента использовались следующие значения:

```
def experiment_epochs(env, epochs_list=[5, 10, 20], episodes=300):
```

Результаты эксперимента представлены на рисунке 3.

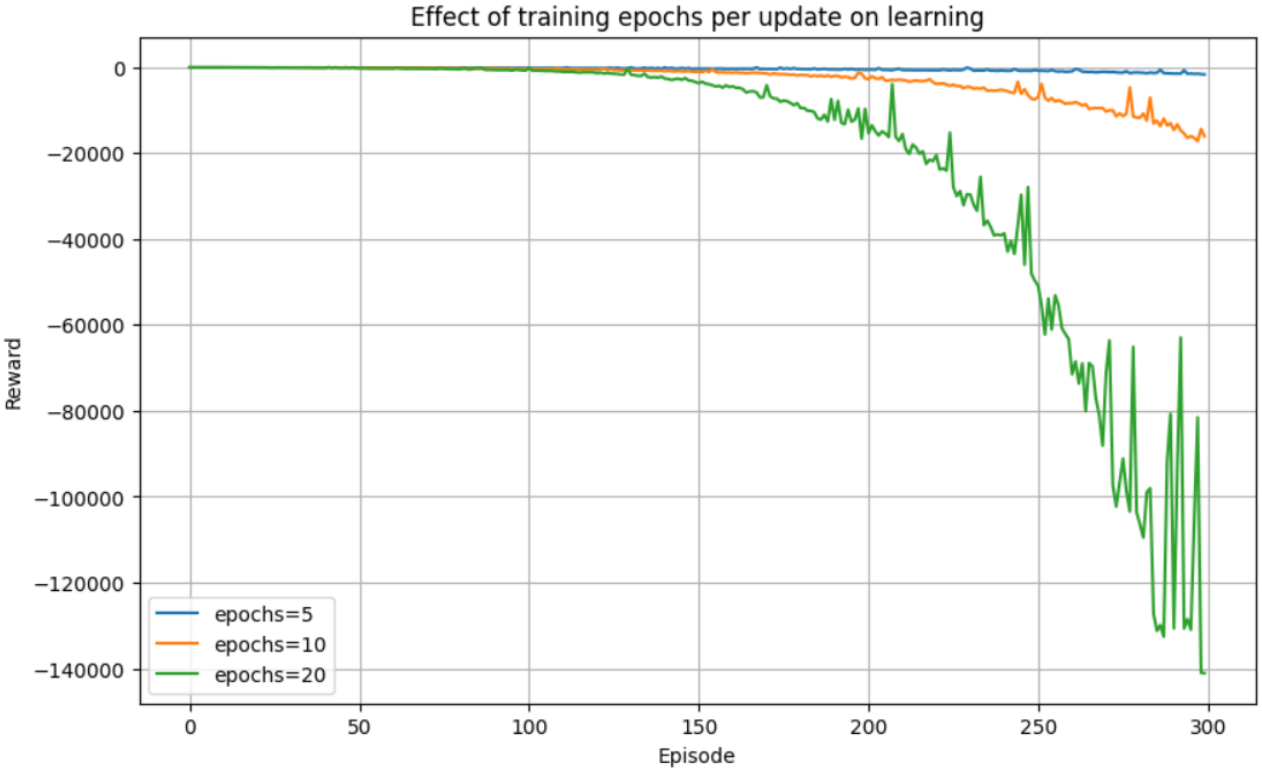

Рисунок 3 – Среднее вознаграждение в зависимости от количества эпох

По полученным результатам можно сделать следующие выводы:

- У значений 5 и 10 получились схожие результаты
- Можно предположить, что оптимальное значение количества эпох может быть немногим больше значения 20.

ВЫВОДЫ

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

Таким образом, оптимизация параметров и добавление нормализации преимуществ позволили улучшить качество обучения агента.

ПРИЛОЖЕНИЕ А

Исходный код

```
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
import gymnasium as gym
from torch.distributions import Normal
import matplotlib.pyplot as plt
from collections import deque
import random

device = torch.device("cuda" if torch.cuda.is_available() else
"cpu")

class ActorCritic(nn.Module):
def __init__(self, state_dim, action_dim, hidden_size=64):
super(ActorCritic, self).__init__()

self.shared_layers = nn.Sequential(
nn.Linear(state_dim, hidden_size),
nn.Tanh(),
nn.Linear(hidden_size, hidden_size),
nn.Tanh()
)

self.actor_mean = nn.Sequential(
nn.Linear(hidden_size, action_dim),
nn.Tanh()
)
self.actor_logstd = nn.Parameter(torch.zeros(1, action_dim))

self.critic = nn.Linear(hidden_size, 1)

def forward(self, state):
shared = self.shared_layers(state)
return self.actor_mean(shared), self.critic(shared)

def act(self, state):
with torch.no_grad():
state = torch.FloatTensor(state).unsqueeze(0).to(device)
mean, value = self.forward(state)
dist = Normal(mean, self.actor_logstd.exp())
action = dist.sample()
log_prob = dist.log_prob(action)
```

```
return action.cpu().numpy()[0], log_prob.cpu().numpy()[0],
value.cpu().numpy()[0]

class PPO:
def __init__(self, state_dim, action_dim,
lr=3e-4,
gamma=0.99,
gae_lambda=0.95,
clip_ratio=0.2,
train_epochs=10,
batch_size=64,
entropy_coef=0.01):

self.policy = ActorCritic(state_dim, action_dim).to(device)
self.optimizer = optim.Adam(self.policy.parameters(), lr=lr)

self.gamma = gamma
self.gae_lambda = gae_lambda
self.clip_ratio = clip_ratio
self.train_epochs = train_epochs
self.batch_size = batch_size
self.entropy_coef = entropy_coef

self.old_policy = ActorCritic(state_dim, action_dim).to(device)
self.old_policy.load_state_dict(self.policy.state_dict())

def update(self, states, actions, log_probs_old, returns,
advantages):
states = torch.FloatTensor(np.array(states)).to(device)
actions = torch.FloatTensor(np.array(actions)).to(device)
log_probs_old =
torch.FloatTensor(np.array(log_probs_old)).to(device)
returns = torch.FloatTensor(np.array(returns)).to(device)
advantages = torch.FloatTensor(np.array(advantages)).to(device)

dataset = torch.utils.data.TensorDataset(states, actions,
log_probs_old, returns, advantages)
loader = torch.utils.data.DataLoader(dataset,
batch_size=self.batch_size, shuffle=True)

for _ in range(self.train_epochs):
for batch in loader:
batch_states, batch_actions, batch_log_probs_old, batch_returns,
batch_advantages = batch

means, values = self.policy(batch_states)
dist = Normal(means, self.policy.actor_logstd.exp())
log_probs = dist.log_prob(batch_actions)
```

```
entropy = dist.entropy().mean()

ratio = (log_probs - batch_log_probs_old).exp()

surr1 = ratio * batch_advantages
surr2 = torch.clamp(ratio, 1.0 - self.clip_ratio, 1.0 +
self.clip_ratio) * batch_advantages
actor_loss = -torch.min(surr1, surr2).mean() - self.entropy_coef *
entropy

critic_loss = 0.5 * (batch_returns - values).pow(2).mean()

loss = actor_loss + critic_loss

self.optimizer.zero_grad()
loss.backward()
self.optimizer.step()

self.old_policy.load_state_dict(self.policy.state_dict())

def compute_gae(self, rewards, values, dones):
advantages = np.zeros_like(rewards)
last_advantage = 0

for t in reversed(range(len(rewards))):
if t == len(rewards) - 1:
next_value = 0
next_non_terminal = 1.0 - dones[t]
else:
next_value = values[t+1]
next_non_terminal = 1.0 - dones[t]

delta = rewards[t] + self.gamma * next_value * next_non_terminal -
values[t]
advantages[t] = delta + self.gamma * self.gae_lambda *
next_non_terminal * last_advantage
last_advantage = advantages[t]

returns = advantages + values
return advantages, returns

def train(env, agent, max_steps=2048, episodes=1000,
normalize_advantages=True):
episode_rewards = []
moving_avg_rewards = []
best_reward = -np.inf

for episode in range(episodes):
```

```
states = []
actions = []
rewards = []
dones = []
values = []
log_probs = []

state, _ = env.reset()
episode_reward = 0

for _ in range(max_steps):
action, log_prob, value = agent.old_policy.act(state)
next_state, reward, terminated, truncated, _ = env.step(action)
done = terminated or truncated

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

state = next_state
episode_reward += reward

if done:
break

values = np.array(values)
advantages, returns = agent.compute_gae(rewards, values, dones)

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

agent.update(states, actions, log_probs, returns, advantages)

episode_rewards.append(episode_reward)
moving_avg = np.mean(episode_rewards[-100:])
moving_avg_rewards.append(moving_avg)

if episode % 10 == 0:
print(f"Episode: {episode}, Reward: {episode_reward}, Moving Avg:
{moving_avg:.1f}")

if moving_avg > best_reward and episode >= 100:
best_reward = moving_avg
torch.save(agent.policy.state_dict(), "best_model.pth")
```

```
return episode_rewards, moving_avg_rewards

def plot_results(rewards, moving_avg):
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(rewards)
plt.title('Rewards per episode')
plt.xlabel('Episode')
plt.ylabel('Reward')

plt.subplot(1, 2, 2)
plt.plot(moving_avg)
plt.title('Moving average reward (last 100 episodes)')
plt.xlabel('Episode')
plt.ylabel('Reward')

plt.tight_layout()
plt.show()

def test(env, agent, episodes=10, render=True):
policy = agent.policy
policy.eval()

for episode in range(episodes):
state, _ = env.reset()
done = False
total_reward = 0

while not done:
if render:
env.render()
with torch.no_grad():
action, _, _ = policy.act(state)
next_state, reward, terminated, truncated, _ = env.step(action)
done = terminated or truncated
state = next_state
total_reward += reward

print(f"Test Episode: {episode+1}, Reward: {total_reward}")

def experiment_clip_ratio(env, clip_ratios=[0.1, 0.2, 0.3],
episodes=300):
results = {}

for clip in clip_ratios:
print(f"\nRunning experiment with clip_ratio={clip}")
```

```
agent = PPO(
state_dim=env.observation_space.shape[0],
action_dim=env.action_space.shape[0],
clip_ratio=clip
)
rewards, _ = train(env, agent, episodes=episodes)
results[clip] = rewards

plt.figure(figsize=(10, 6))
for clip, rewards in results.items():
plt.plot(rewards, label=f"clip={clip}")
plt.title('Effect of clip ratio on learning')
plt.xlabel('Episode')
plt.ylabel('Reward')
plt.legend()
plt.grid()
plt.show()

return results

def experiment_epochs(env, epochs_list=[5, 10, 20], episodes=300):
results = {}

for epochs in epochs_list:
print(f"\nRunning experiment with train_epochs={epochs}")
agent = PPO(
state_dim=env.observation_space.shape[0],
action_dim=env.action_space.shape[0],
train_epochs=epochs
)
rewards, _ = train(env, agent, episodes=episodes)
results[epochs] = rewards

plt.figure(figsize=(10, 6))
for epochs, rewards in results.items():
plt.plot(rewards, label=f"epochs={epochs}")
plt.title('Effect of training epochs per update on learning')
plt.xlabel('Episode')
plt.ylabel('Reward')
plt.legend()
plt.grid()
plt.show()

return results

def main():
env = gym.make("MountainCarContinuous-v0")
state_dim = env.observation_space.shape[0]
```

```
action_dim = env.action_space.shape[0]

agent = PPO(
state_dim=state_dim,
action_dim=action_dim,
lr=3e-4,
gamma=0.99,
gae_lambda=0.95,
clip_ratio=0.2,
train_epochs=10,
batch_size=64,
entropy_coef=0.01
)

print("Starting training...")
rewards, moving_avg = train(env, agent, max_steps=2048,
episodes=500, normalize_advantages=True)
plot_results(rewards, moving_avg)

print("\nTesting the agent...")
test(env, agent, episodes=5)

print("\nRunning clip ratio experiment...")
clip_results = experiment_clip_ratio(env)

print("\nRunning training epochs experiment...")
epochs_results = experiment_epochs(env)

env.close()

if __name__ == "__main__":
main()
```