Московский государственный технический университет им. Н.Э. Баумана
Факультет «Головной учебно-исследовательский и методический центр
профессиональной реабилитации лиц с ограниченными возможностями здоровья (инвалидов)»
Кафедра «Системы обработки информации и управления»

Лабораторная работа 7

по дисциплине «Методы машинного обучения в АСОИУ»

" Алгоритмы Actor-Critic "

СТУДЕНТ:
студент группы ИУ5Ц-21М
Москалик А.А.

ПРЕПОДАВАТЕЛЬ:
Гапанюк Ю.Е.

Москва, 2024

Цель лабораторной работы: ознакомление с базовыми методами обучения с подкреплением на основе алгоритмов Actor-Critic.

Задание:

Реализуйте любой алгоритм семейства Actor-Critic для произвольной среды.

Ход работы

1: Импорт библиотек и определение классов Actor и Critic

Actor: Используется для выборки действий в среде на основе текущего состояния. Это помогает агенту выбирать действия, которые максимизируют ожидаемую награду.

Critic: Оценивает, насколько хорошее состояние агент находится в данный момент, что позволяет корректировать действия агента для достижения более высокой награды.

```python
import gym
import torch
import torch.nn as nn
import torch.optim as optim
import numpy as np
from torch.distributions import Categorical

# Определение классов Actor и Critic
class Actor(nn.Module):
    def __init__(self, state_size, action_size):
        super(Actor, self).__init__()
        self.network = nn.Sequential(
            nn.Linear(state_size, 128),
            nn.ReLU(),
            nn.Linear(128, action_size),
            nn.Softmax(dim=-1)
        )

    def forward(self, state):
        return self.network(state)

class Critic(nn.Module):
    def __init__(self, state_size):
        super(Critic, self).__init__()
        self.network = nn.Sequential(
            nn.Linear(state_size, 128),
            nn.ReLU(),
            nn.Linear(128, 1)
        )

    def forward(self, state):
        return self.network(state)
```

2: Инициализация среды и агентов

Gym создает среду CartPole-v1, где задача заключается в том, чтобы удерживать полюс в вертикальном положении на движущейся тележке.

```
# Инициализация среды Gym
env = gym.make('CartPole-v1')

# Получение размеров состояния и действий
state_size = env.observation_space.shape[0]
action_size = env.action_space.n

# Инициализация агентов и оптимизаторов
actor = Actor(state_size, action_size)
critic = Critic(state_size)

actor_optimizer = optim.Adam(actor.parameters(), lr=0.01)
critic_optimizer = optim.Adam(critic.parameters(), lr=0.01)
```

3: Функция выполнения эпизода

Эта функция run_episode выполняет один эпизод взаимодействия агента с средой, используя текущую политику (Actor) для выбора действий и Critic для оценки состояния. Результаты взаимодействия используются для последующего обучения.

```
# Функция для выполнения одного эпизода
def run_episode(env, actor, critic):
    state = env.reset()[0]  # получение первого элемента состояния
    rewards = []
    log_probs = []
    values = []
    done = False

    while not done:
        state = torch.FloatTensor(state).unsqueeze(0)
        probs = actor(state)
        dist = Categorical(probs)
        action = dist.sample()

        log_prob = dist.log_prob(action)
        value = critic(state)

        state, reward, done, _, _ = env.step(action.item())

        rewards.append(reward)
        log_probs.append(log_prob)
        values.append(value)

    return rewards, log_probs, values
```

4: Функция обновления политики

Эта функция update_policy обновляет параметры нейронных сетей Actor и Critic на основе данных, собранных во время одного эпизода. Она использует метод обучения с подкреплением, называемый Advantage Actor-Critic (A2C).

```
# Функция обновления политики
def update_policy(rewards, log_probs, values, actor_optimizer, critic_optimizer, gamma=0.99):
    G = 0
    returns = []

    for reward in reversed(rewards):
        G = reward + gamma * G
        returns.insert(0, G)

    returns = torch.tensor(returns)
    log_probs = torch.cat(log_probs)
    values = torch.cat(values).squeeze()

    advantage = returns - values

    actor_loss = -(log_probs * advantage.detach()).mean()
    critic_loss = advantage.pow(2).mean()

    actor_optimizer.zero_grad()
    critic_optimizer.zero_grad()

    actor_loss.backward()
    critic_loss.backward()

    actor_optimizer.step()
    critic_optimizer.step()
```

5: Запуск обучения агента

Этот код запускает процесс обучения агента в среде CartPole-v1, используя алгоритм Actor-Critic. Обучение проводится на протяжении заданного количества эпизодов.

```
# Запуск обучения агента
num_episodes = 1000  # количество эпизодов для обучения

for i in range(num_episodes):
    rewards, log_probs, values = run_episode(env, actor, critic)
    update_policy(rewards, log_probs, values, actor_optimizer, critic_optimizer)

    if (i+1) % 100 == 0:
        print(f"Episode {i+1}: Total reward = {sum(rewards)}")
```

```
Episode 100: Total reward = 118.0
```

Вывод:

Ответ "Episode 100: Total reward = 118.0" означает, что в 100-м эпизоде ваш агент получил общую награду 118.0. В контексте задачи CartPole, где цель заключается в том, чтобы держать полюс сбалансированным на тележке как можно дольше, общая награда является суммой всех наград, полученных за один эпизод. Каждая временная отметка, на которой полюс остается сбалансированным, приносит агенту награду 1.0. Следовательно, сумма наград напрямую указывает на количество шагов, в течение которых агенту удавалось удерживать полюс в вертикальном положении.

Для среды CartPole задача считается "решенной", когда среднее значение общей награды за 100 последовательных эпизодов достигает 195 баллов или выше. Таким образом, результат в 118 баллов за один эпизод показывает, что ваш агент уже демонстрирует неплохую производительность, но еще не достиг критерия успешного выполнения задачи.

Если агент стабильно набирает выше 195 баллов в среднем по 100 эпизодам, это будет означать, что он научился эффективно решать задачу. Для достижения этой цели может потребоваться дальнейшая настройка гиперпараметров, таких как скорость обучения или архитектура нейронных сетей, а также увеличение количества эпизодов обучения.