Тагиров Али, ИТМО

Задание 6.1

Попробовал два варианта:

1. Использовал для расчета Advantage формулу без использования returns, но при этом оставил расчет Loss2 прежним, как в теории

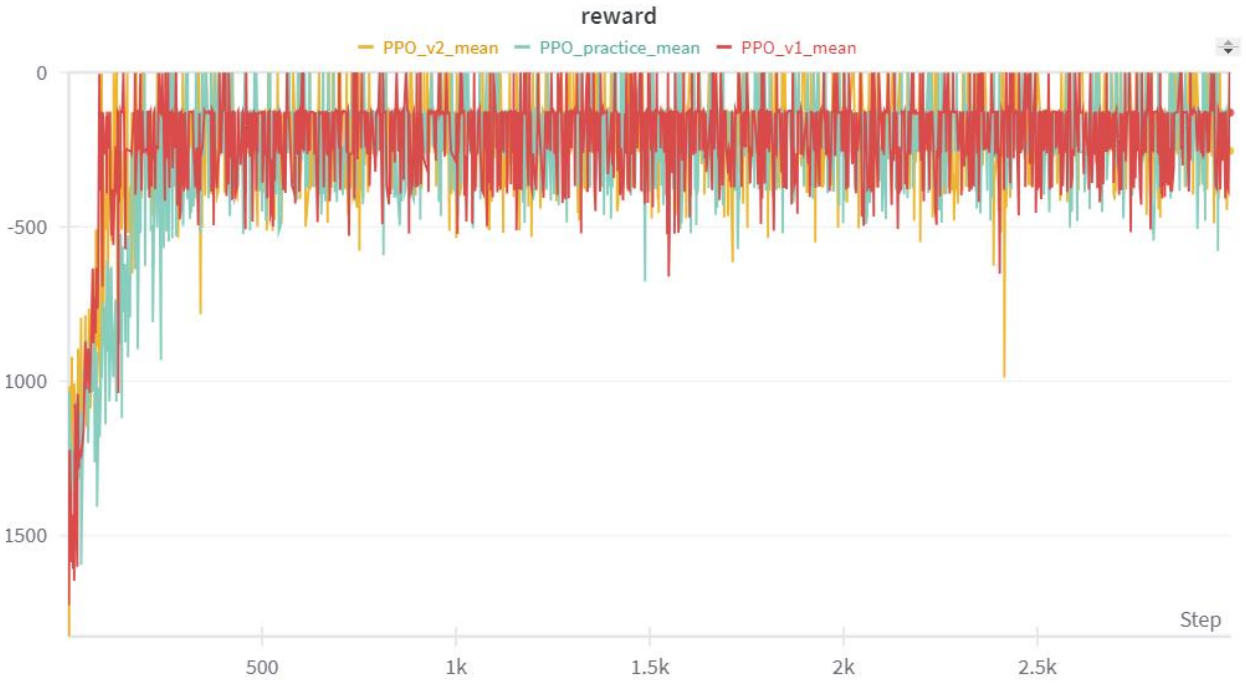
$$Loss_2(\theta) = \frac{1}{T}\sum_{t=0}^{T-1}\left(V^{\theta}(S_t) - G_t\right)^2$$

$$A^{\theta}(S_t, A_t) = R_t + \gamma V^{\theta}(S_{t+1}) - V^{\theta}(S_t),$$

2. Полностью избавился от returns и считал Loss2 и Advantage одной формулой

```
b_advantage = b_rewards + self.gamma \
            * self.v_model(b_next_states) - self.v_model(b_states)
```

Результаты обучения PPO разными способами:



Каждый способ был обучен 3 раза и взят усредненный результат обучения. На графике видно, что при использовании “новых” способов обучения скорость обучения выросла, при этом первый способ показывает более стабильные результаты.

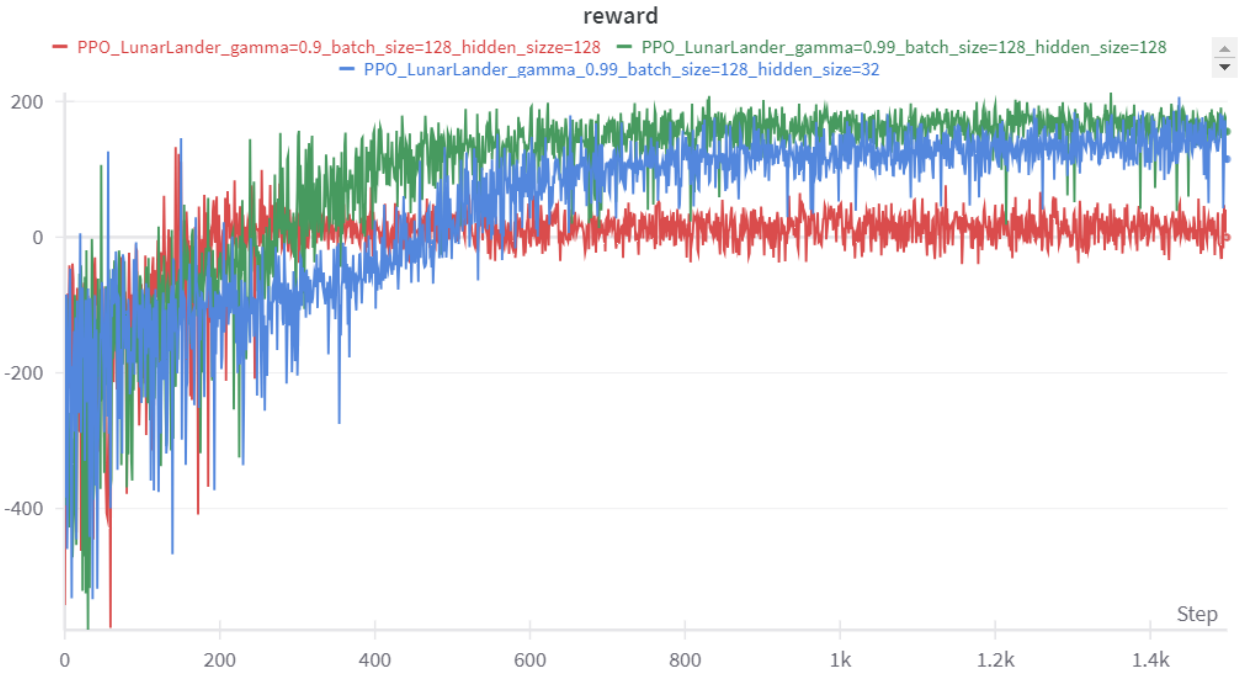
Задание 6.2

В случае LunaLander, так как пространство действий двумерное я разбил выход с pi_model следующим образом:

```
def get_action(self, state):
    x = self.pi_model(torch.FloatTensor(state))
    mean, log_std = x[:2], x[2:]
    dist = Normal(mean, torch.exp(log_std))
    action = dist.sample()
    return action.numpy()
```

При этом универсальнее будет разделить pi_model на несколько слоев (слой для mean и log_std принимающие на вход, выход со скрытого слоя)

Посмотри на кривые обучения:



Преодолеть результат больше 100, получилось только после увеличения gamma до 0.99, увеличение размера скрытого состояния также улучшает результаты модели, но не так сильно.

Задание 6.3

Написать PPO для работы в средах с конечным пространствам действий и решить Acrobot.

Для решения задания использовал Categorical, в который можно передать либо вероятности, либо логиты, но при этом при одних и тех же параметрах модель лучше обучается при передаче логитов

```
def get_action(self, state):
    logits = self.pi_model(torch.FloatTensor(state))
    dist = Categorical(logits=logits)
    action = dist.sample()
    return action.numpy()
```

График обучения выглядит следующим образом: