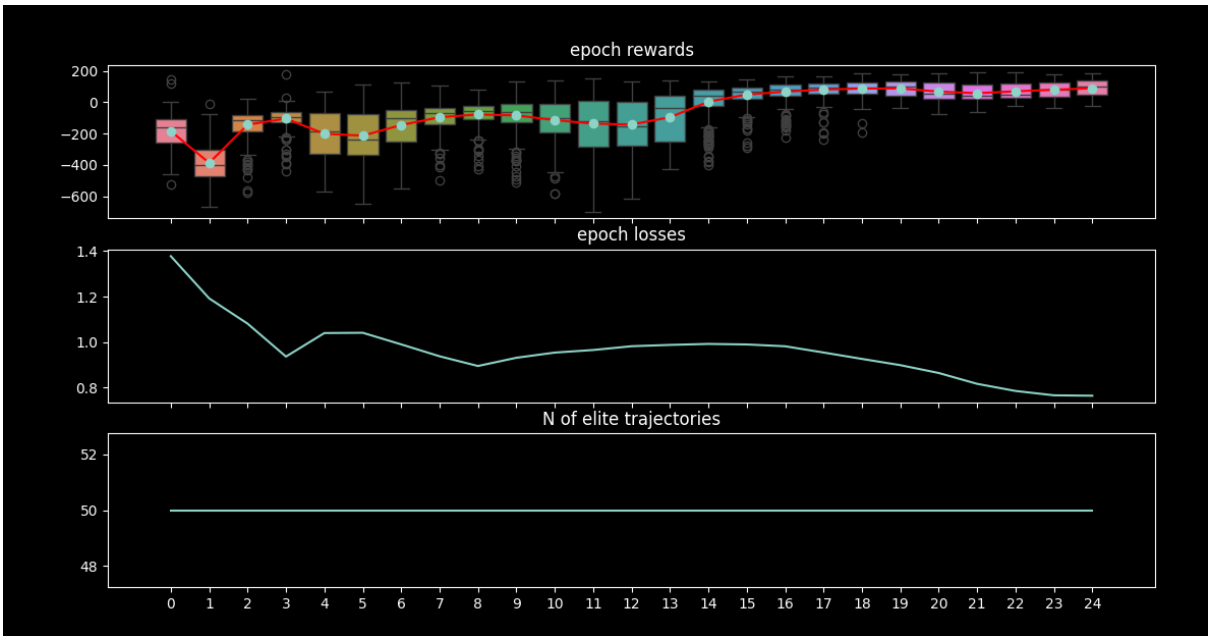
LunarLander

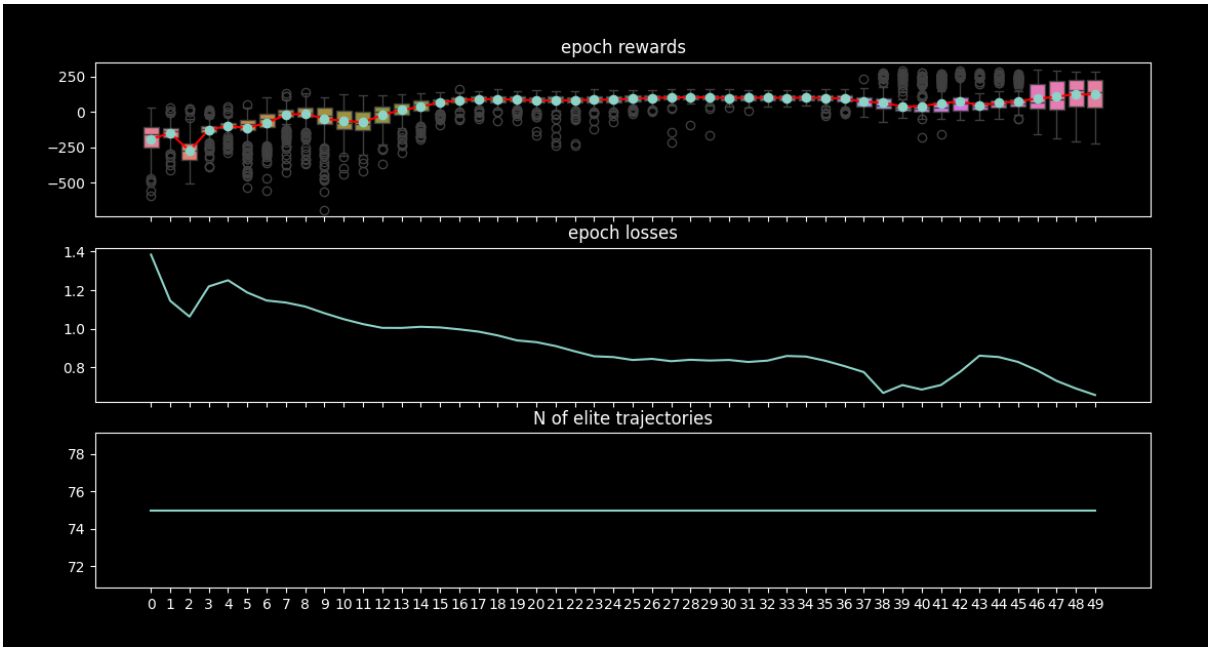
Начал решать задачу с кодом, идентичным тому, что было на практическом занятии. Результаты испытаний:



Last trajectory reward 103.0838076585512

```
n_epochs = 25
traj_per_epoch = 250
trajectory_len = 300
q_param = 0.8
lr = 0.05
```

В целом, как видно, задача решалась почти сразу. но я решил испытать другие гиперпараметр, чтобы найти лучшие решения. Интересно отметить, что проблемы, которая присутствовала в такси - недостаток количества траекторий, попадающих в элитные тут нет. при почти любом значении q_param количество элитных траекторий постоянно.



last_trajectory_reward = 196.4557476783358

```
n_epochs = 50
traj_per_epoch = 250
trajectory_len = 500
q_param = 0.7
lr = 0.05
```

С данными гиперпараметрами задача уже решается лучше, учитывая что идеальная награда = 200. Однако, как видно по графикам, распределение наград внутри эпох все еще довольно высокое, и оно увеличивается к концу обучения.
Полагаю, что это решается увеличением общего времени обучения, больше эпох и больше траекторий за эпоху.

MountainCarContinuous

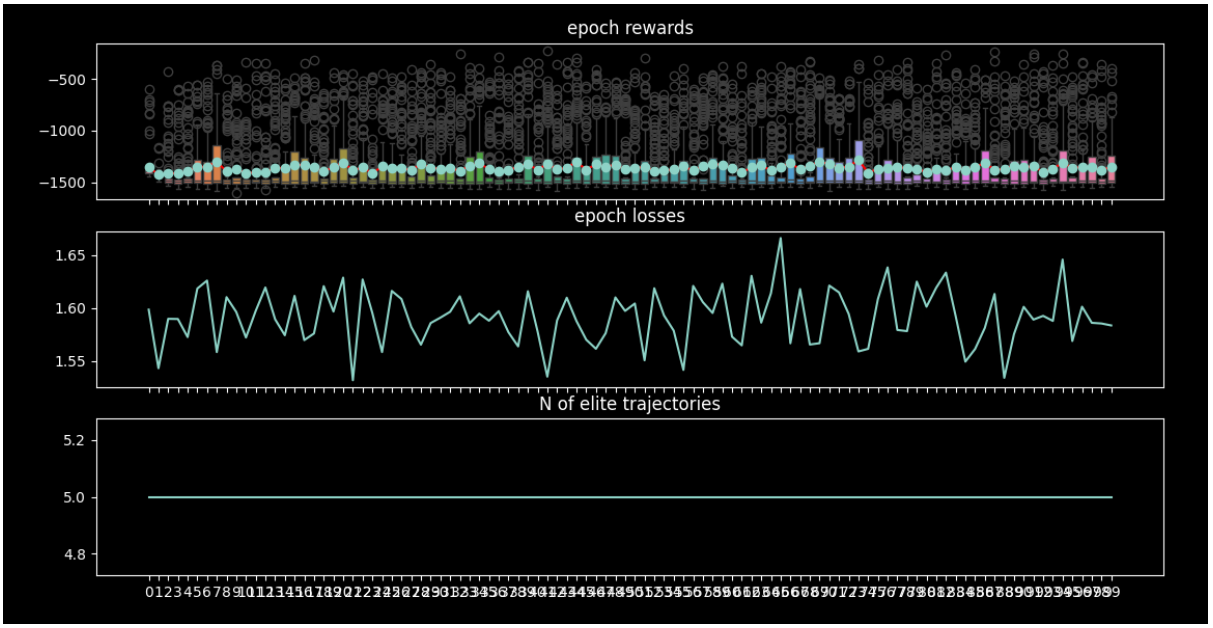
Две проблемы при решении этой задачи:

1. Заставить агента генерировать траектории, которые будут доходить до финиша
2. Сделать так, чтобы в элитные траектории попадали только успешные.

И то и то является проблемой так как в большинстве случаев агент может спокойно сходиться к награде = 0 и застрявать там. Для того чтобы добиться генерации траекторий, которые будут достигать финиша, нужно добавлять много шума.
Для того, чтобы в элитные траектории попадали только “финиширующие” я пытался каким то образом либо модифицировать сам отбор элитных траекторий, либо функцию награды.
Модификация отбора траекторий выглядела как:
`q_value = max(q_value, 0)`
Логика следующая: так как при генерации траектории начисляется только негативная награда, траектории с наградой больше 0 будут только в тех, которые финишировали.
Недостаток идеи: финиширующие траектории могут появиться даже если награда меньше 0, так что для того, чтобы метод работал нужно очень много рандомных

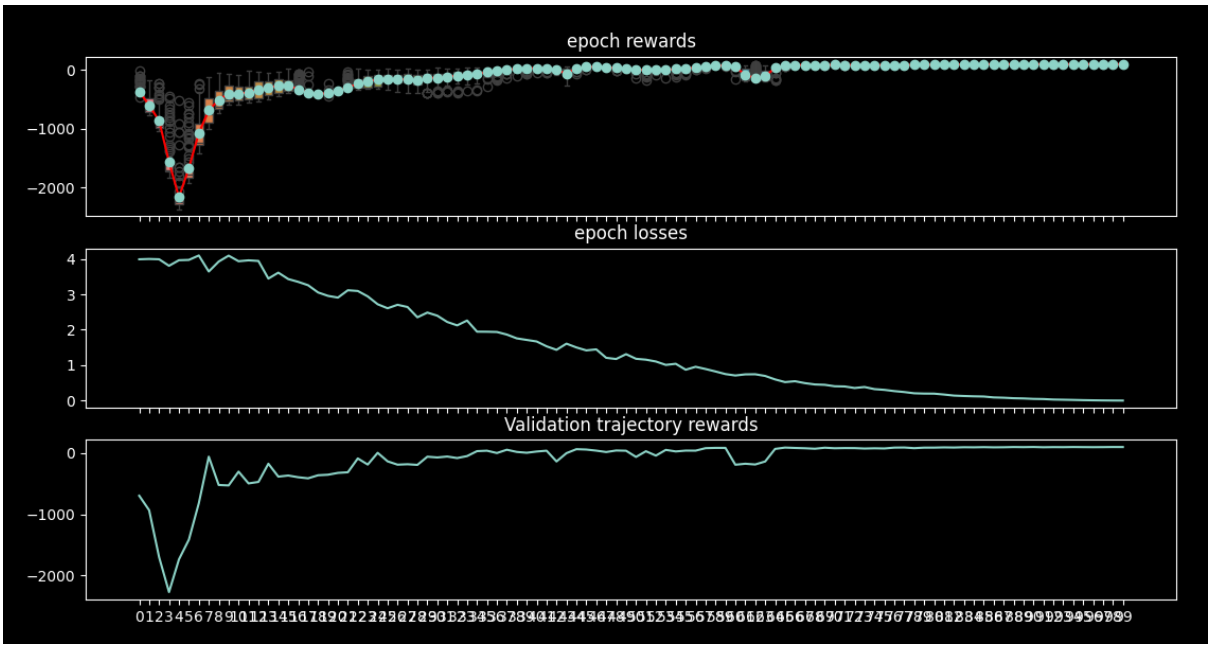
траекторий. У меня этот метод по итогу не сработал, так как я посчитал, что сидеть и ждать очень долго это не лучший вариант решения проблемы.

Модификация функции награды выглядила так:
```python3
def get_trajectory(env, agent, trajectory_len):
        …
        total_reward = 0
        for _ in range(trajectory_len):
                …
                state, reward, done, _ = env.step(action)
                total_reward += reward
                …
        total_reward -= len(states)
        return trajectory
```
то есть отнимаем от награды длину траектории.
Логика в том, что “финиширующие” траектории в подавляющем большинстве случаев будут короче, чем те, которые были терминированы средой. Вот график для такого способа:



По итогу с данным способом у меня задача не решилась.

После этого я решил больше поиграться с гиперпараметрами, с обычной функцией награды и обычным отбором элитных траекторий.
По итогу сошлось:




```
epoch = 84, epoch_mean_reward = 91.93, elite_trajectory_n = 5, loss = 0.12, val_reward = 91.87, agent.exploration_rate = 0.17777777777777[755/4
q_value = 95.58586089571544
epoch = 85, epoch_mean_reward = 92.6, elite_trajectory_n = 5, loss = 0.11, val_reward = 94.82, agent.exploration_rate = 0.16666666666666818
q_value = 95.89737178371573
epoch = 86, epoch_mean_reward = 93.1, elite_trajectory_n = 5, loss = 0.09, val_reward = 91.67, agent.exploration_rate = 0.15555555555555708
q_value = 95.94185023614948
epoch = 87, epoch_mean_reward = 93.68, elite_trajectory_n = 5, loss = 0.08, val_reward = 93.06, agent.exploration_rate = 0.14444444444444599
q_value = 96.61748041720865
epoch = 88, epoch_mean_reward = 93.9, elite_trajectory_n = 5, loss = 0.07, val_reward = 96.53, agent.exploration_rate = 0.13333333333333489
q_value = 96.52413001723589
epoch = 89, epoch_mean_reward = 94.32, elite_trajectory_n = 5, loss = 0.06, val_reward = 94.84, agent.exploration_rate = 0.12222222222222377
q_value = 96.78905235308824
epoch = 90, epoch_mean_reward = 94.48, elite_trajectory_n = 5, loss = 0.05, val_reward = 97.21, agent.exploration_rate = 0.11111111111111266
q_value = 96.97610099388486
epoch = 91, epoch_mean_reward = 94.62, elite_trajectory_n = 5, loss = 0.04, val_reward = 93.11, agent.exploration_rate = 0.10000000000000155
q_value = 97.0879306838016
epoch = 92, epoch_mean_reward = 94.83, elite_trajectory_n = 5, loss = 0.03, val_reward = 95.37, agent.exploration_rate = 0.08888888888889043
q_value = 97.15347834257001
epoch = 93, epoch_mean_reward = 95.11, elite_trajectory_n = 5, loss = 0.02, val_reward = 94.58, agent.exploration_rate = 0.07777777777777932
q_value = 97.39392919138523
epoch = 94, epoch_mean_reward = 95.33, elite_trajectory_n = 5, loss = 0.02, val_reward = 96.71, agent.exploration_rate = 0.0666666666666682
q_value = 97.54524148089259
epoch = 95, epoch_mean_reward = 95.41, elite_trajectory_n = 5, loss = 0.01, val_reward = 95.44, agent.exploration_rate = 0.05555555555555709
q_value = 97.56574349942224
epoch = 96, epoch_mean_reward = 95.41, elite_trajectory_n = 5, loss = 0.01, val_reward = 94.39, agent.exploration_rate = 0.04444444444444598
q_value = 97.38454287705815
epoch = 97, epoch_mean_reward = 95.74, elite_trajectory_n = 5, loss = 0.0, val_reward = 95.37, agent.exploration_rate = 0.033333333333334866
q_value = 97.74386551005303
epoch = 98, epoch_mean_reward = 95.95, elite_trajectory_n = 5, loss = 0.0, val_reward = 96.98, agent.exploration_rate = 0.022222222222223753
q_value = 97.78148516526782
epoch = 99, epoch_mean_reward = 95.98, elite_trajectory_n = 5, loss = 0.0, val_reward = 96.45, agent.exploration_rate = 0.011111111111112642
```

```
n_epochs = 100
traj_per_epoch = 100
trajectory_len = 1000
q_param = 0.95
lr = 0.05
mu, sigma = 0, 2
decrease_after = 10
```

Графика для того, как я уменьшал шум, к сожалению, нет, но я уменьшал его равномерно линейно после определенной траектории, сам шум - `np.random.normal(mu, sigma)`

```
decrease_rate = agent.exploration_rate / (n_epochs - decrease_after)
for epoch in range(n_epochs):
    if epoch > decrease_after:
        agent.exploration_rate -= decrease_rate
```

Структура сети:

```
self.net = nn.Sequential(
        nn.Linear(self.state_dim, 128),
        nn.ReLU(True),
        nn.Linear(128, self.action_dim),
        )
self.loss_f = nn.MSELoss()
self.exploration_rate = 1
```

Обучение сети не заняло много времени, около 20-25 минут всего. Возможно можно решить быстрее.

Вывод

В целом, появилось ощущение, что метод довольно сильно зависит от количества генерируемых траекторий, и довольно нужно подгонять под условия определенной среды. Тут появляется вопрос о том, являются ли такие вещи как модификация функции награды или модификация отбора траекторий для определенной среды нечестной. Понятно, что с точки зрения необходимости предоставить решение - любые способы хороши, однако конечная цель вероятно не решение определенной задачи, а изучение и понимание недостатков алгоритма кросс-энтропии, которые являются ровно такими, как было обозначено в первой лекции: сильная зависимость от случайности. И по итогу приходится генерировать очень много траекторий, для того чтобы в сгенерериованных смогли оказаться оптимальные. В целом я достаточно рад, что у меня задачи по итогу решились без дополнительных хаков среды.