Задание # 1 Алгоритм Кросс-Энтропии для конечного пространства действий, обученный решать Acrobot-v1

Среда Acrobot основана на работе Саттона «Обобщение в обучении с подкреплением: успешные примеры использования разреженного грубого кодирования» и книге Саттона и Барто . Система состоит из двух звеньев, соединенных линейно в цепь, при этом один конец цепи закреплен. Срабатывает соединение между двумя звеньями. Цель состоит в том, чтобы приложить крутящие моменты к приводному шарниру, чтобы повернуть свободный конец линейной цепи выше заданной высоты, начиная с исходного состояния подвешивания вниз

Дано:

Пространство действий = 3

Размерность состояний = 6

Я проводил много экспериментов добавляя слои для нейронной сети, но это по большей части делало даже хуже. Если оценивать по метрикам Поэтому я остановился на одном с 128 нейронами:

```
self.network = nn.Sequential(
    nn.Linear(self.state_dim, 128),
    nn.ReLU(),
    nn.Linear(128, self.action_n)
)
```

Реализация метода обучения:

```python
def fit(self, env: Env, epochs: int, q_param: float, trajectory_n: int, trajectory_len: int, debug: bool = False):
    logs = []
    for epoch in range(epochs):
        trajectories = [self.get_trajectory(env=env, trajectory_len=trajectory_len) for _ in range(trajectory_n)]

        total_reward = [trajectory['total_reward'] for trajectory in trajectories]
        info = {
            "iteration:": epoch,
            "mean_total_reward": np.mean(total_reward),
            "max_total_reward": np.max(total_reward),   You, Today • Uncommitted changes
        }
        logs.append(info)
        if debug:
            print(info)

        elite_trajectories = self.get_elite_trajectories(trajectories=trajectories, q_param=q_param)

        if len(elite_trajectories) > 0:
            self.update_policy(elite_trajectories)
    return {
        "q_param": q_param,
        "trajectory_n": trajectory_n,
        "epochs": epochs,
        "trajectory_len": trajectory_len,
        "info": logs,
    }
```

Код обновления политик:

```python
def update_policy(self, elite_trajectories):
    elite_states = []
    elite_actions = []
    for trajectory in elite_trajectories:
        elite_states.extend(trajectory['states'])
        elite_actions.extend(trajectory['actions'])
    elite_states = torch.from_numpy(np.array(elite_states))
    elite_actions = torch.from_numpy(np.array(elite_actions))

    loss = self.loss(self.forward(elite_states), elite_actions)
    loss.backward()
    self.optimizer.step()   You, 4 minutes ago • Uncommitted ch
    self.optimizer.zero_grad()
```

В финальной версии, я не использую такое сглаживания для вычисления action_prob, так как алгоритм не сходился.

```python
def get_action(self, state, epoch: int, n_epochs: int):
    state = torch.FloatTensor(state)
    logits = self.forward(state)
    action_prob = self.softmax(logits).detach().numpy()
    action_prob = (epoch / n_epochs) * action_prob + (n_epochs - epoch / n_epochs) * self.uniform
    action_prob = action_prob / np.sum(action_prob)
    action = np.random.choice(self.action_n, p=action_prob)
    return action
```

Поэтому решил оставить следующий метод get_action:

```python
2 usages (1 dynamic)
def get_action(self, state):
    state = torch.FloatTensor(state)
    logits = self.forward(state)
    action_prob = self.softmax(logits).detach().numpy()
    action = np.random.choice(self.action_n, p=action_prob)
    return action
```

Рассмотрим сводную таблицу результатов экспериментов, обратите внимание, что значения взяты на последней итерации mean_total_reward, max_total_reward:

	q_param	trajectory_n	epochs	trajectory_len	mean_total_reward	max_total_reward
0	0.9	100	100	300	-81.170	-62.0
1	0.9	100	100	600	-85.220	-63.0
2	0.9	200	100	300	-85.935	-61.0
3	0.9	200	100	600	-87.145	-60.0
4	0.6	100	100	300	-88.060	-63.0
5	0.6	100	100	600	-84.370	-63.0
6	0.6	200	100	300	-80.840	-62.0
7	0.6	200	100	600	-79.615	-63.0
8	0.4	100	100	300	-86.180	-64.0
9	0.4	100	100	600	-81.960	-64.0
10	0.4	200	100	300	-81.625	-62.0
11	0.4	200	100	600	-89.815	-62.0

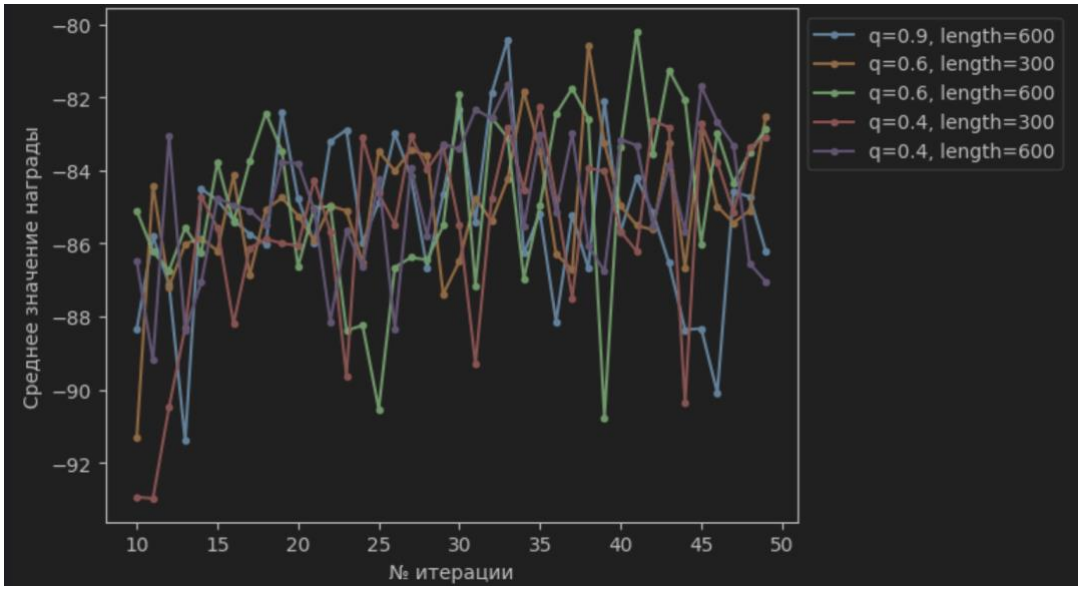
Лучшие гиперпараметры выбирал по максимальному mean_total_reward:

	q_param	trajectory_n	epochs	trajectory_len	mean_total_reward	max_total_reward
7	0.6	200	100	600	-79.615	-63.0

Обратим внимание, что сходимость достигнуто еще на 40 эпохе обучения, mean_total_reward будет находится около -80 с небольшими отклонениями:

```
{'iteration:': 40, 'mean_total_reward': -82.63, 'max_total_reward': -63.0}
{'iteration:': 41, 'mean_total_reward': -86.425, 'max_total_reward': -63.0}
{'iteration:': 42, 'mean_total_reward': -83.805, 'max_total_reward': -63.0}
{'iteration:': 43, 'mean_total_reward': -83.9, 'max_total_reward': -63.0}
{'iteration:': 44, 'mean_total_reward': -84.205, 'max_total_reward': -64.0}
{'iteration:': 45, 'mean_total_reward': -86.185, 'max_total_reward': -63.0}
{'iteration:': 46, 'mean_total_reward': -82.765, 'max_total_reward': -63.0}
{'iteration:': 47, 'mean_total_reward': -84.27, 'max_total_reward': -63.0}
{'iteration:': 48, 'mean_total_reward': -85.495, 'max_total_reward': -63.0}
{'iteration:': 49, 'mean_total_reward': -80.04, 'max_total_reward': -63.0}
{'iteration:': 50, 'mean_total_reward': -81.745, 'max_total_reward': -63.0}
{'iteration:': 51, 'mean_total_reward': -84.915, 'max_total_reward': -63.0}
{'iteration:': 52, 'mean_total_reward': -81.93, 'max_total_reward': -64.0}
{'iteration:': 53, 'mean_total_reward': -86.295, 'max_total_reward': -63.0}
{'iteration:': 54, 'mean_total_reward': -82.79, 'max_total_reward': -63.0}
{'iteration:': 55, 'mean_total_reward': -82.345, 'max_total_reward': -63.0}
{'iteration:': 56, 'mean_total_reward': -82.425, 'max_total_reward': -63.0}
{'iteration:': 57, 'mean_total_reward': -81.155, 'max_total_reward': -63.0}
{'iteration:': 58, 'mean_total_reward': -83.505, 'max_total_reward': -63.0}
{'iteration:': 59, 'mean_total_reward': -82.63, 'max_total_reward': -63.0}
```

Что касается графиков mean_total_reward на каждой итерации. Какого-либо различие тяжело найти, лишь для некоторых параметров нужно больше эпох для обучения:



Аналогично первому домашнему заданию для сходимости алгоритма нужно выбирать q – значение не слишком большим, так как множество траекторий не достигли хороших результатов. Однако Алгоритм сошелся, что не может ни радовать.

Задание # 2.1 Алгоритм Кросс-Энтропии для конечного пространства действий, обученный решать MountainCarContinuous-v0

Дано:

Пространство действий = (-1 ;1)

Размерность действий = 1

Размерность состояний = 2

Максимальное кол-во действий = 999

В данном задании у нас нет, конкретного действия мы можем выбрать как действие любое число из множества (-1 ;1). Что касается позиции машины от (-inf; 0) машина находится на слева, если от (0, inf) то она находится справа. Как действие мы можем передать либое число, но оно клипанется до (-1 ;1) и возведется в степень 0,0015

Награда

Отрицательное вознаграждение в размере -0,1 * действие 2 получается на каждом временном шаге для наказания за совершение действий большого масштаба. Если горная машина достигает цели, к отрицательной награде за этот период времени добавляется положительная награда +100.

Реализация нейронной сети была выбрана функция активации Tanh так как она выдает значения от (-1;1), что нам и нужно для пространства действий и лосс функцией была взята MSE:

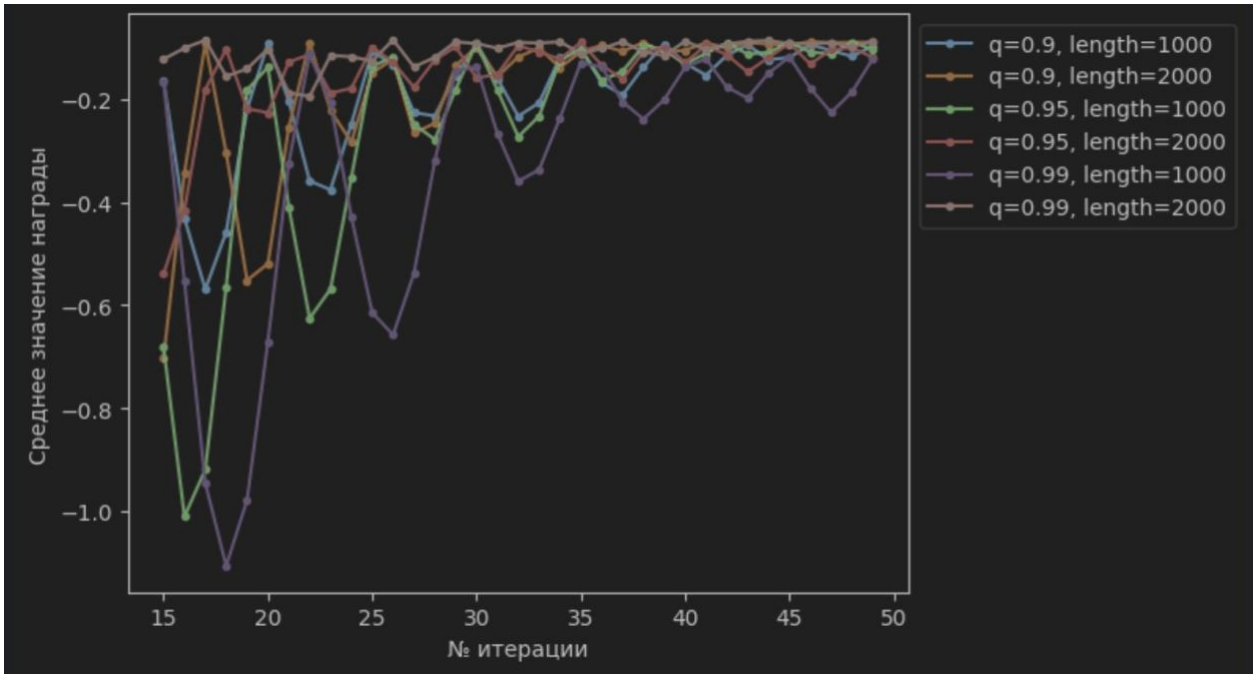
```
self.network = nn.Sequential(
    nn.Linear(self.state_dim, 128),
    nn.ReLU(),
    nn.Linear(128, 1)
)

self.activation = nn.Tanh()
self.optimizer = torch.optim.Adam(self.parameters(), lr=0.01)
self.loss = nn.MSELoss()
```

В методе get_action произошли изменения был добавлен шум из равномерного распределения:

```
2 usages (1 dynamic)
def get_action(self, state):
    state = torch.FloatTensor(state)
    logits = self.forward(state) + torch.FloatTensor(np.random.uniform(-0.05, 0.05, 1))
    return self.activation(logits).detach().numpy()    You, A minute ago • Uncommitted chang
```

Однако, к сожалению, алгоритму не получатся вытолкнуть машину на вершину, она катается с маленькой амплитудой внизу горки:

	q_param	trajectory_n	epochs	trajectory_len	mean_total_reward	max_total_reward
0	0.90	500	50	1000	-0.096743	-0.081456
1	0.90	500	50	2000	-0.089431	-0.077653
2	0.95	500	50	1000	-0.102779	-0.089485
3	0.95	500	50	2000	-0.118972	-0.097980
4	0.99	500	50	1000	-0.121035	-0.081422
5	0.99	500	50	2000	-0.087041	-0.078708



Я старался брать Q большим, и длину траекторий = 1000 и 2000 и генерировал 500 траекторий чтобы найти именно ту стратегию, которая вытолкнет машину наверх и за нее зацепиться в дальнейших шагах.

Вывод: Данную задачу тяжело решить данным алгоритмом Машина в конце концов остается внизу горы.