Тагиров Али, ИТМО

Задание 5.1

Обучить Агента решать Acrobot-v1, MountainCar-v0, или LunarLander-v2 (одну на выбор) методом DQN. Найти оптимальные гиперпараметры. Сравнить с алгоритмом Deep Cross-Entropy на графиках.

Выбрал агента Acrobot-v1

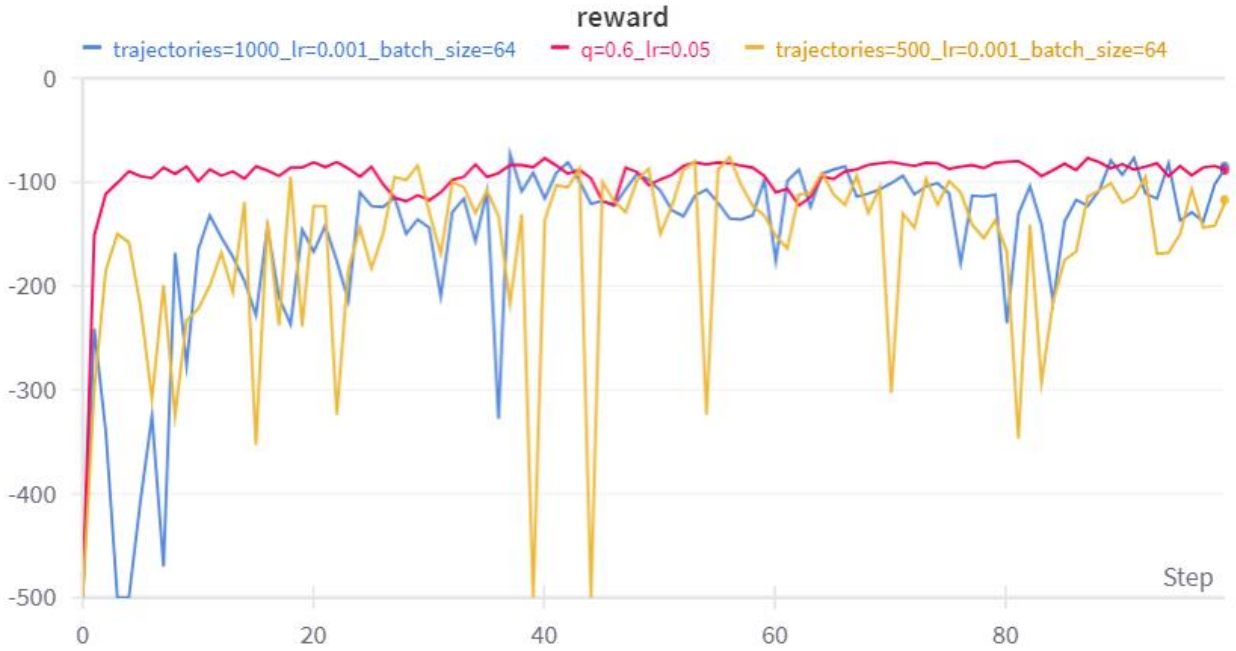
Были исследованы результаты со следующими гиперпараметрами:

Сиды зафиксированы

- Количество эпизодов: 100
- количество траекторий: [300, 500, 1000]
- learning rate: [0.001, 0.01, 0.05]
- batch size: [32, 64, 128]

Метод DQN достаточно быстро обучается, но при этом имеет сильный разброс. По графику видно, что наименьший разброс показали модели с learning rate = 0.001. Также увеличение кол-ва траекторий до 1000 сильно уменьшило разброс модели, что видно на следующем графике.

Сравним с алгоритм Deep Cross-Entropy:



Алгоритм DQN работает гораздо быстрее Deep Cross-Entropy, но при этом Deep Cross-Entropy быстрее сходится и не имеет такого большого разброса

Задание 5.2

Реализовать и сравнить (на выбранной ранее среде) друг с другом и с обычным DQN следующие его модификации:

DQN с Hard Target Update

```
targets = rewards + self.gamma * (1 - dones) * torch.max(self.target_q_function(next_states), dim=1).values
q_values = self.q_function(states)[torch.arange(self.batch_size), actions]
```

```
        if (step + 1) % self.target_upd_freq:
            self.update_target_q_function()

1 usage
def update_target_q_function(self):
    self.target_q_function.load_state_dict(self.q_function.state_dict())
```

Посмотрим, как поведет модификация DQN с Hard Target Update при изменении частоты обновления (каждые 50, 100, 200 итераций) target_q_function:

Модели отработали примерно одинаково, чем большая частота обновлений, тем быстрее модель обучается

DQN с Soft Target Update

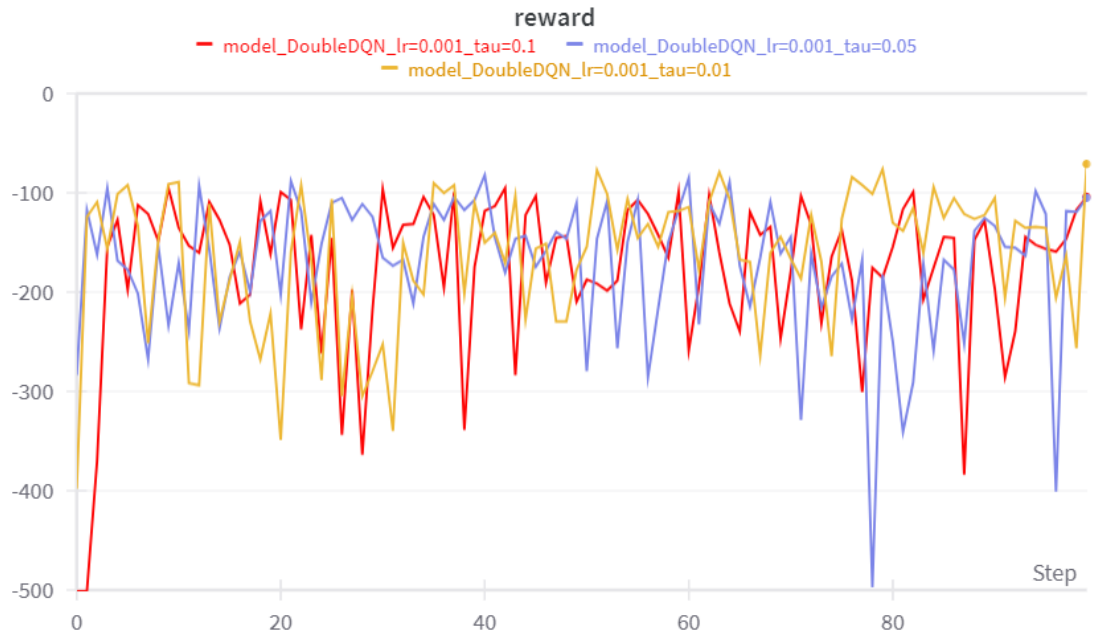
```
def update_target_q_function(self):
    for param, target_param in zip(self.q_function.parameters(), self.target_q_function.parameters()):
        target_param.data.copy_(self.tau * param.data + (1 - self.tau) * target_param.data)
```

Посмотрим, как поведет модификация DQN с Hard Target Update при изменении tau: [0.001, 0.05, 0.1]:

Double DQN

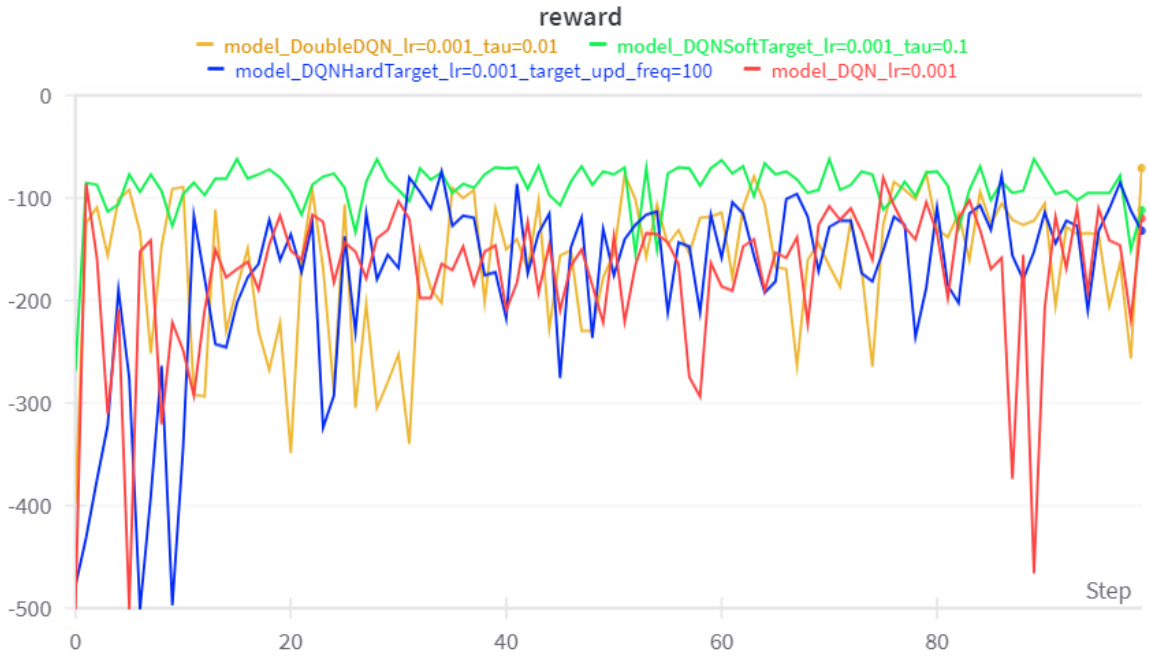
```
target_q_argmax_actions = torch.argmax(self.target_q_function(next_states), dim=1)
targets = rewards + self.gamma * (1 - dones) \
          * self.q_function(next_states)[torch.arange(self.batch_size), target_q_argmax_actions]
```

Также посмотрим на модель Double DQN при изменении tau:



На графики видно, что результаты модели имеют большой разброс, модель с tau = 0.01, показала себя немного лучше. Возможно нужно попробовать более меньшее значения коэффициента tau

Сравним модификации с друг другом и обычным DQN



Лучше всего себя показала модификация DQN с Soft Target, хуже всего обычный DQN, который даже на последних эпизодах обучения показывает минимальные значения rewards. Модификация Double DQN также, как обычные DQN имеет большой разброс rewards, но при этом он не показывает таких же экстремальных изменений как обычный DQN

Вывод

Модификации DQN благодаря фиксированию/сглаживанию Q позволяют стабилизировать обычный DQN, уменьшить разброс.