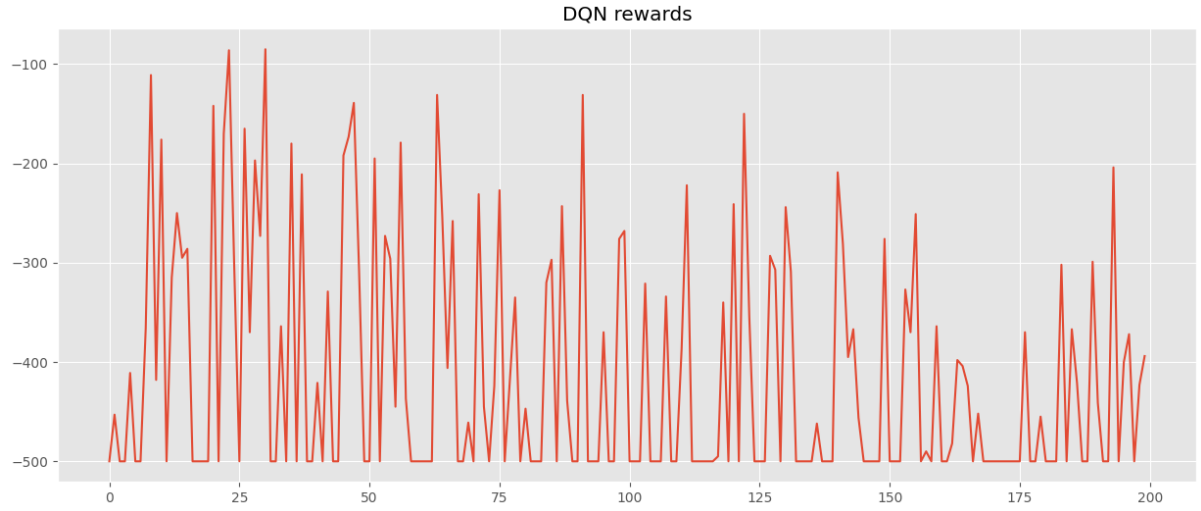
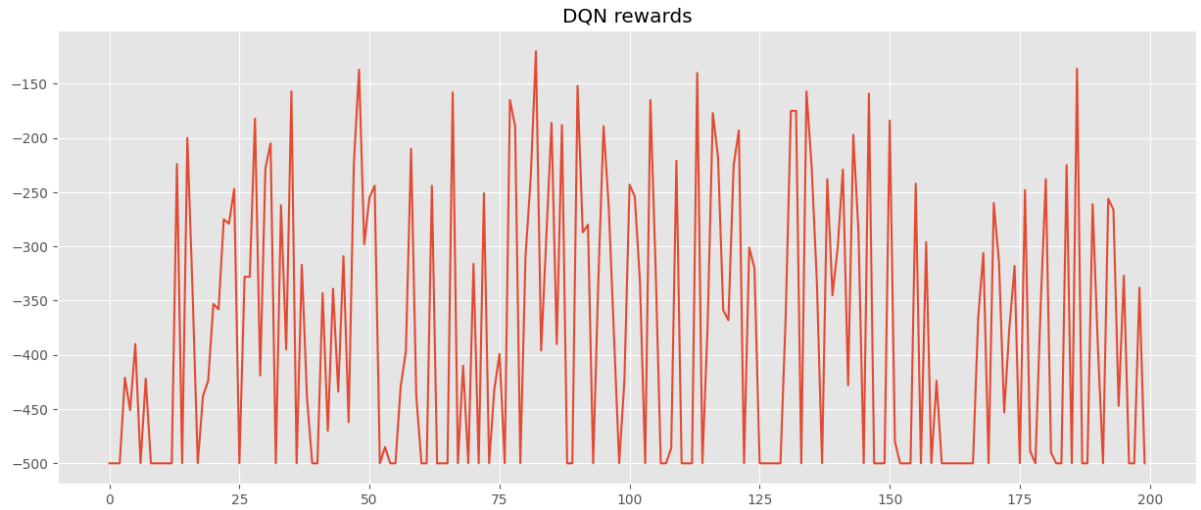
1 DQN for Acrobot-v1

Как я понял, код, написанный на практическом занятии также подходит для домашней работы, так что оставалось только поменять среду и начать подбор гиперпараметров. Сначала я сравнивал как работает алгоритм с различными gamma

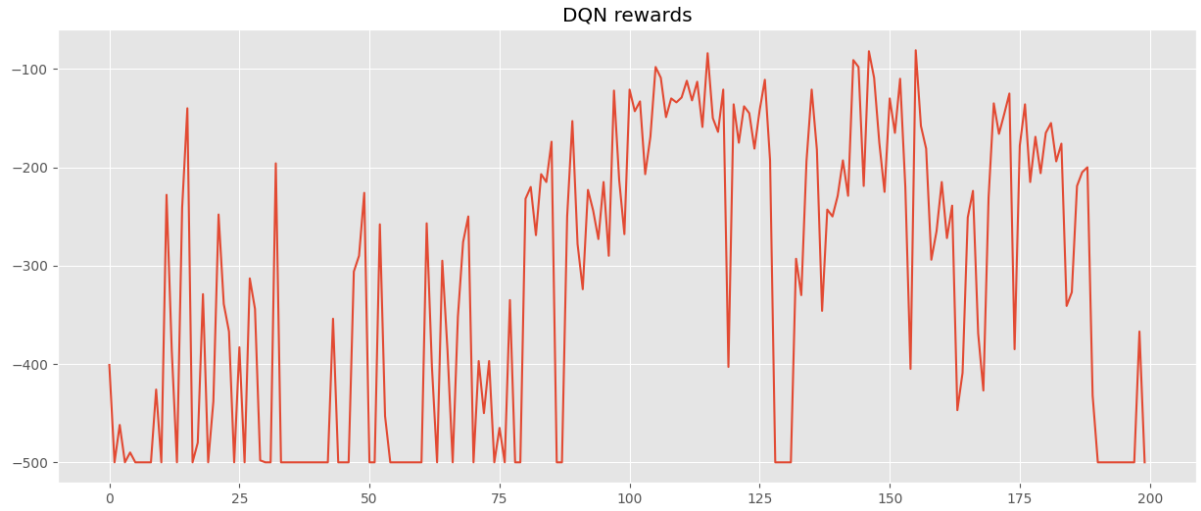
Gamma = 0.85



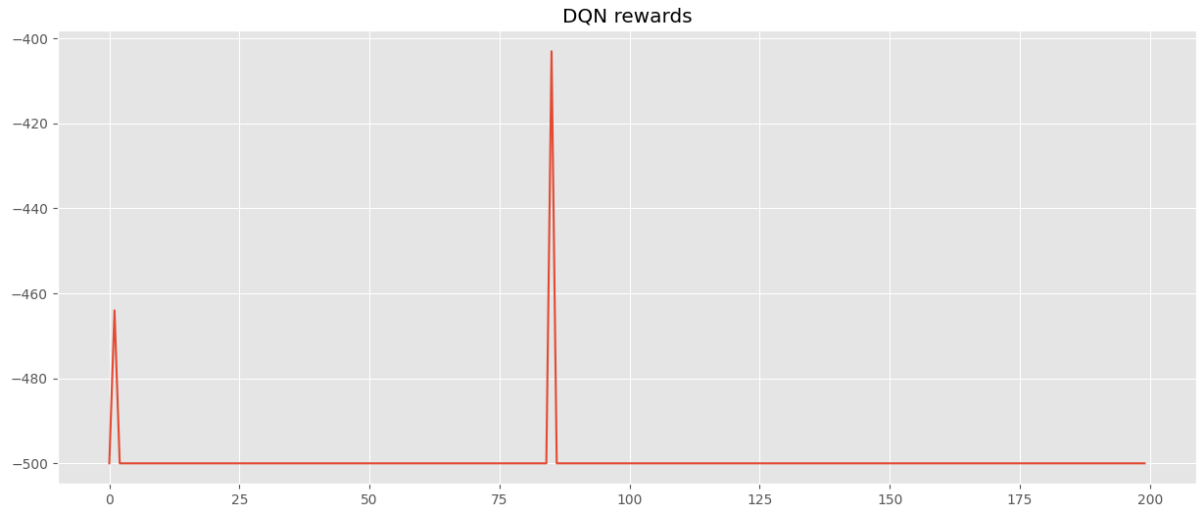
Gamma = 0.9



Gamma = 0.99

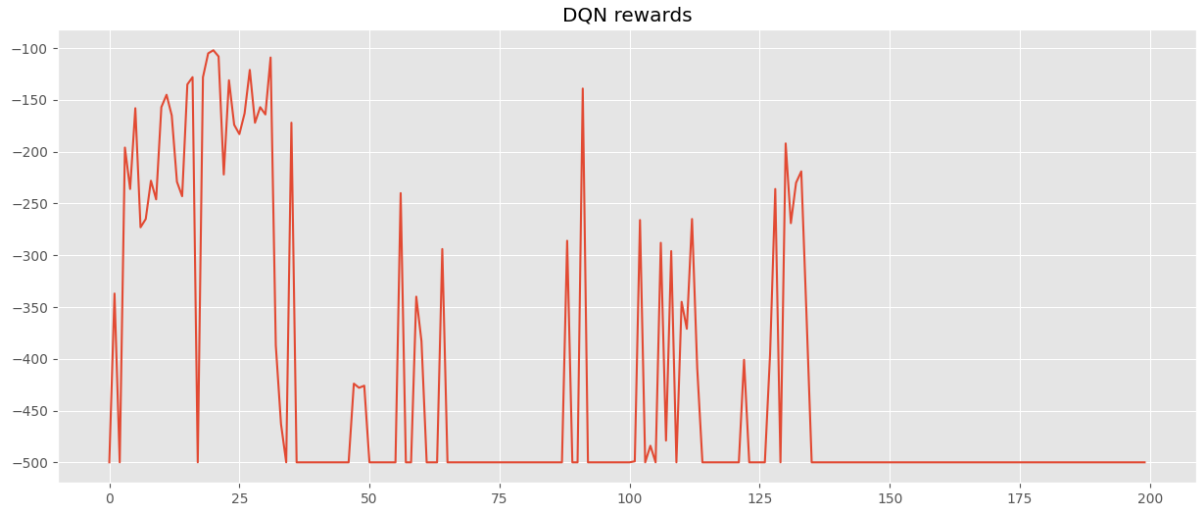


Gamma = 0.999

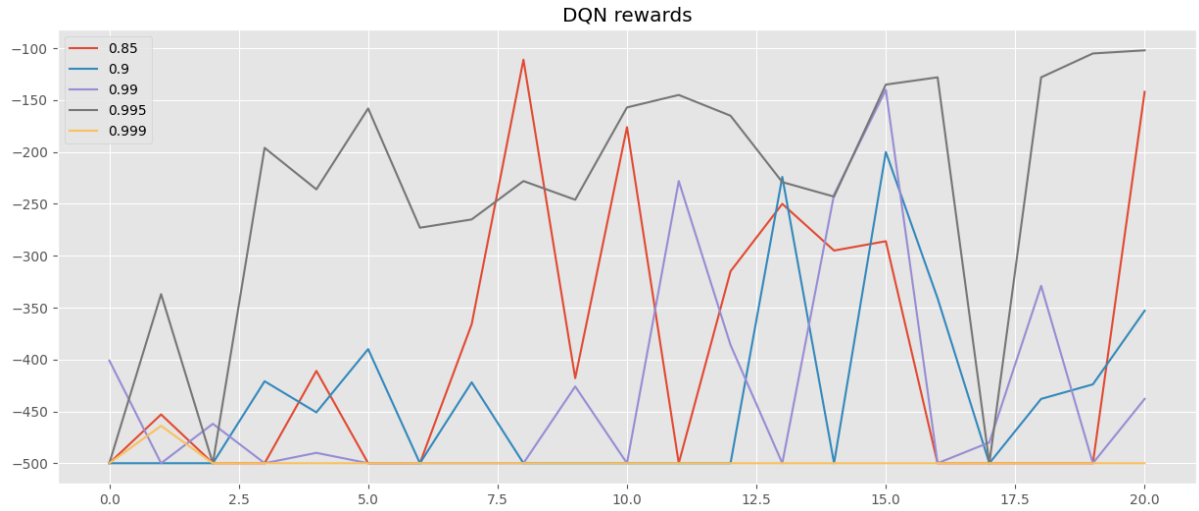


Дальше пытаться увеличивать я пока не стал. Как можно заметить, наилучший результат был показан на gamma = 0.99, и при переходе на 0.999 модель перестала обучаться вообще. Поэтому я сначала решил попробовать обучить с

Gamma = 0.995

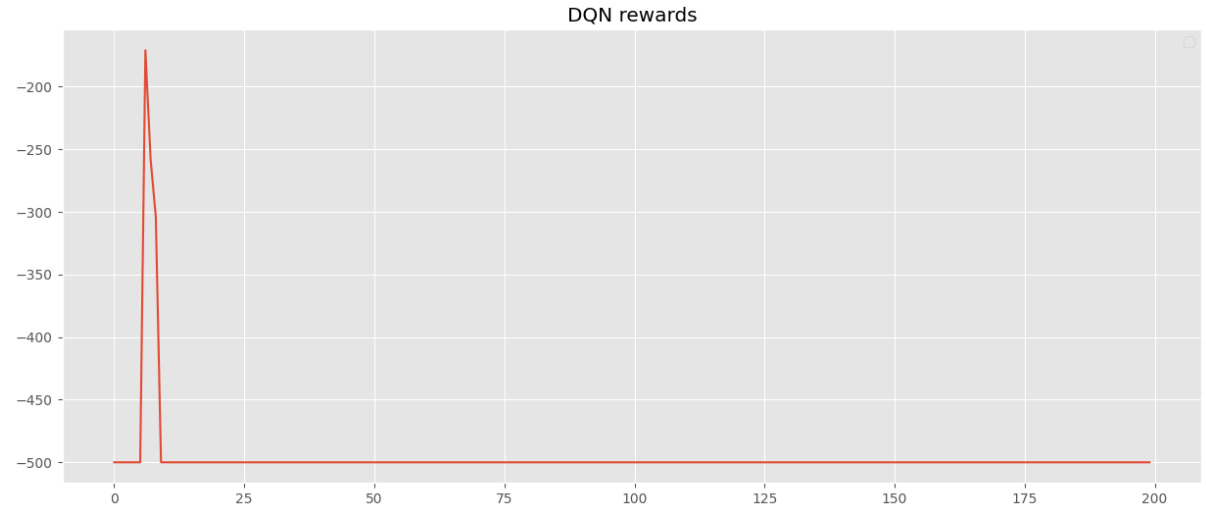


Результат получился неудовлетворительным, и я решил попробовать увеличивать gamma после 10% траекторий. Для начала, я решил посмотреть, какое значение показывает себя лучше на первых 10%:

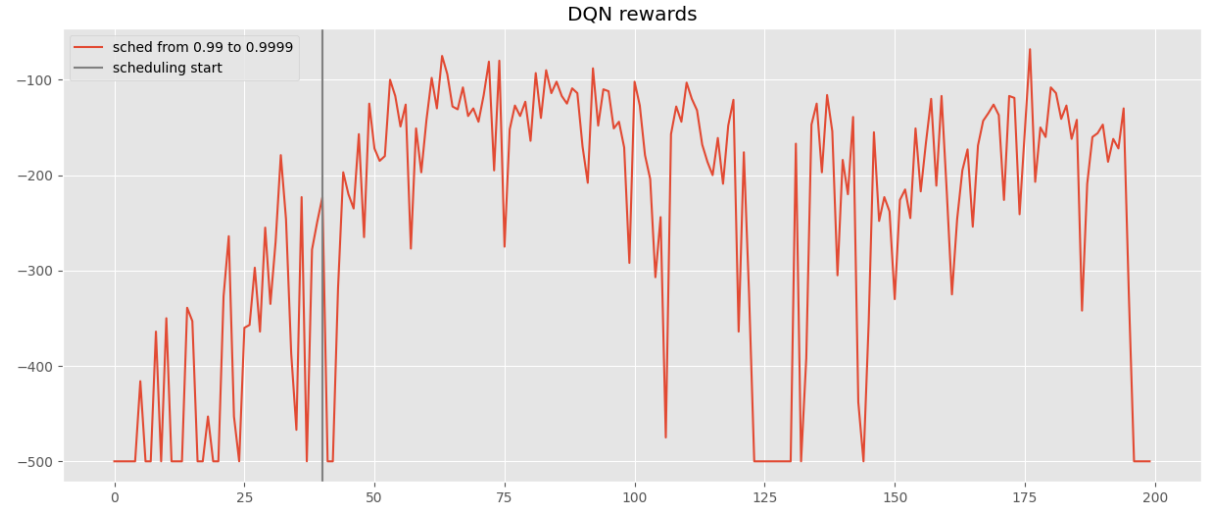


как видно, это значение 0.995. Так как gamma используется только в методе fit() (у меня это training_step()), я решил сделать gamma не свойством класса DQN, а параметром метода. Соответственно, я решил попробовать поставить gamma = 0.994 для первых 20 траекторий, а затем постепенно увеличивать.

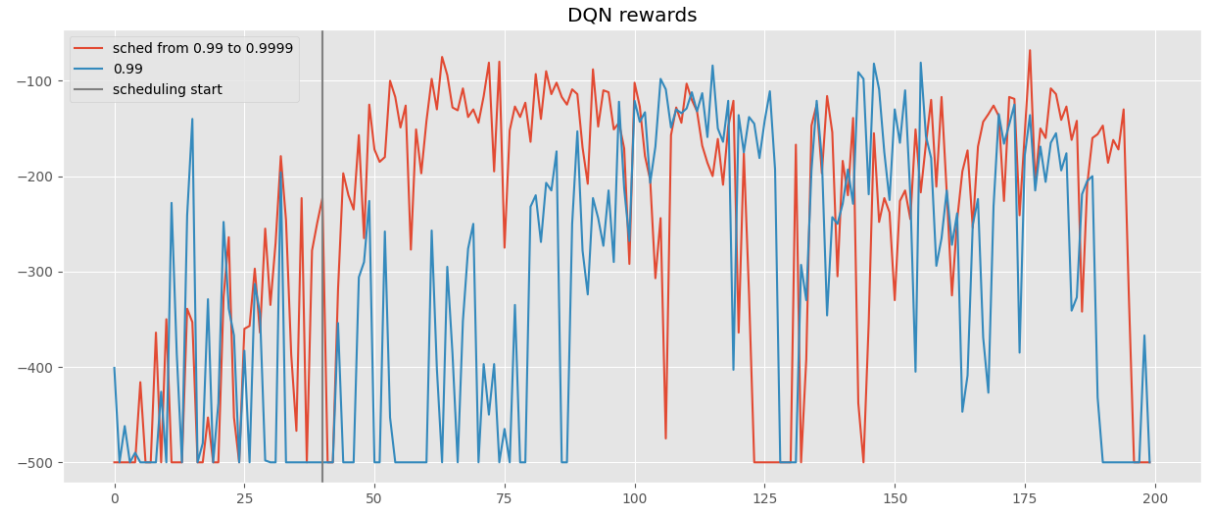
Получилось так себе:



После этого я попробовал увеличивать с 0.99 до 0.9999

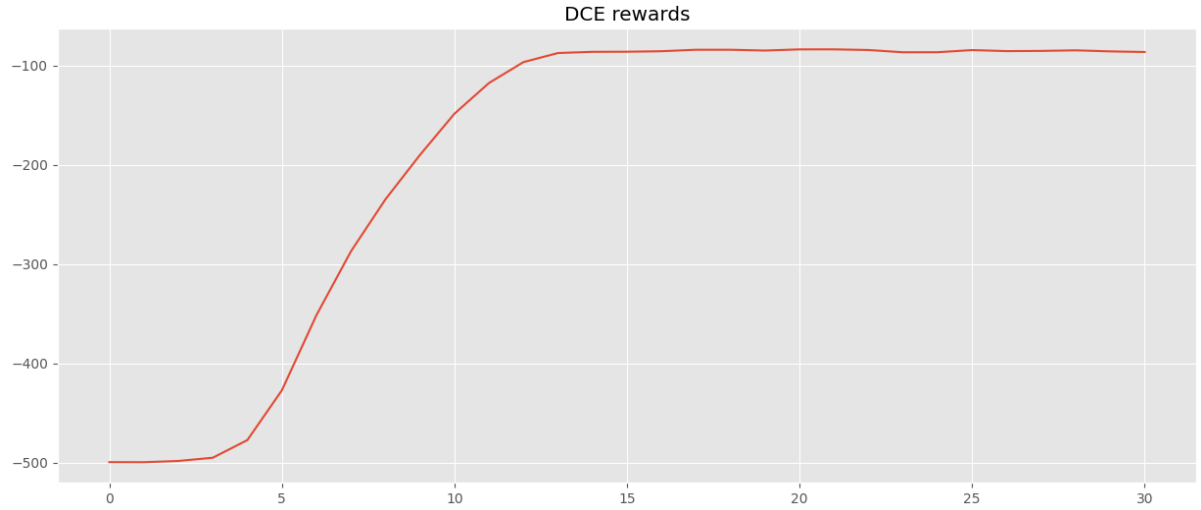


и сравнение с gamma = 0.99



Трудно сказать, что какой то из подходов является более стабильным, однако кажется, что постепенное увеличение дает все таки лучший результат

Сравнение с deep cross-entropy довольно сложное, ибо методы обучения разнятся



DCE занимает больше времени, чем DQN, однако, как видно на графике, результат в разы стабильнее.

2.1 Hard update

После небольшого гугления я понял, что скопировать модель можно с помощью copy.deepcopy(). Я рассматривал вариант писать все вручную, однако зачем.

Функцию train я переписал следующим образом:

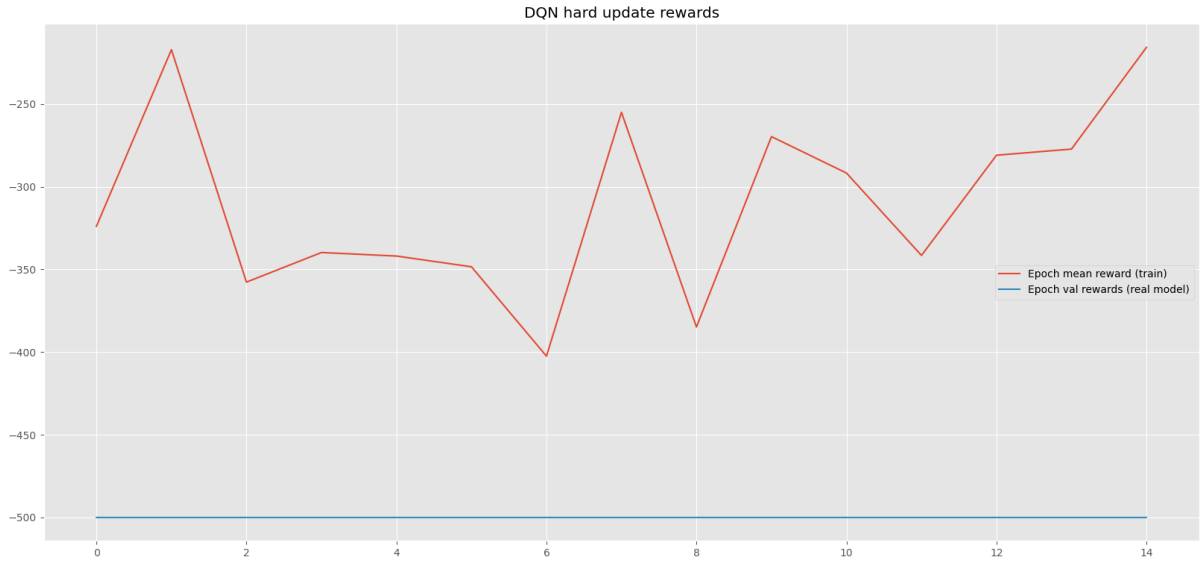
1) Разделение обучения на эпохи
2) В начале каждой эпохи происходит копирование модели:

```
for epoch in range(epochs):

    model_copy = copy.deepcopy(model)
    agent_copy = DQN(model=model_copy, action_n=action_n, batch_size=batch_size, trajectory_n=traj_per_epoch)
```

3) В течении эпохи проходит обучение копии модели
4) В конце эпохи веса перезаписываются в основную модель, и происходит то, что можно назвать валидацией

```
model.load_state_dict(model_copy.state_dict())

state = env.reset()

for _ in range(trajectory_len):
    action = agent.get_action(state)
    state, reward, done, info = env.step(action)
    if done:
        break
```

После первой попытки появилось ощущение, что веса не перезаписываются



Однако, подобная проверка показывала, что веса равны

```
model.load_state_dict(model_copy.state_dict())
for idx, (p1, p2) in enumerate(zip(model.named_parameters(), model_copy.named_parameters())):
    if not p1[0] == p2[0]:
        print('different parameter order for idx {}'.format(idx))
    if not torch.equal(p1[1].data, p2[1].data):
        print('idx {} not equal'.format(idx))
    else:
        print('parameter {} is equal'.format(idx))
```

```
100% 30/30 [01:00<00:00, 2.23s/it]
parameter 0 is equal
parameter 1 is equal
parameter 2 is equal
parameter 3 is equal
parameter 4 is equal
parameter 5 is equal
epoch = 5       mean_epoch_loss = tensor(95575.5859, grad_fn=<DivBackward0>)    mean_epoch_reward = -376.1      val_reward = -500.0
```

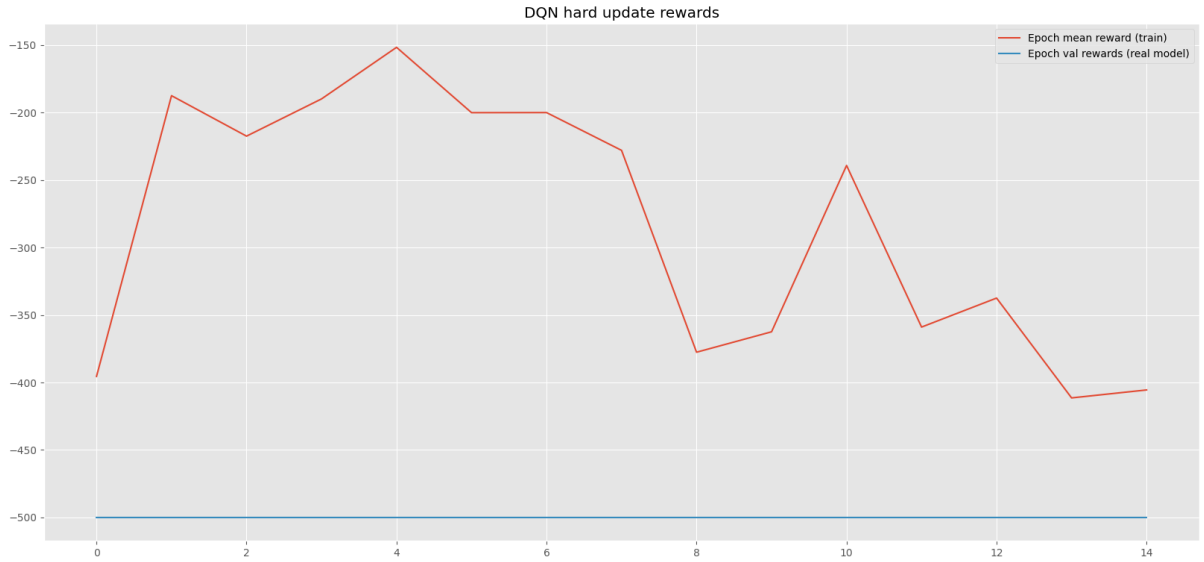
После этого у меня осталось две идеи:

1) инициализировать две модели и два агента, ибо таким образом можно будет вручную посмотреть на веса
2) инициализировать одного агента с двумя моделями (вписать в класс DQN еще одну модель)

первая идея, которая выглядела вот так

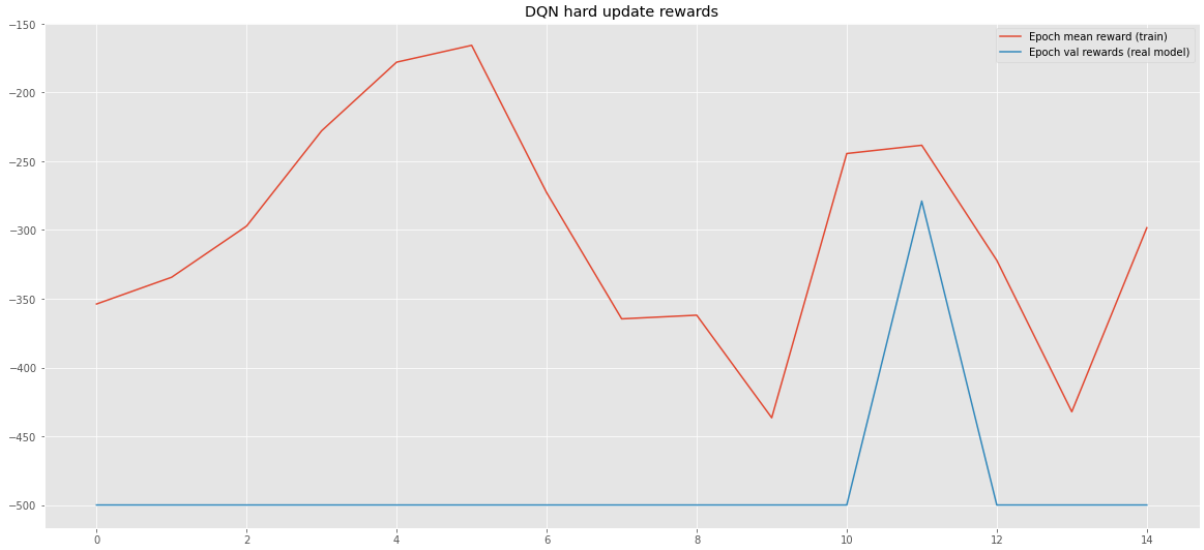
```
target_model = NN(state_dim=state_dim, action_n=action_n)
target_agent = DQN(model=target_model, action_n=action_n, batch_size=batch_size, trajectory_n=traj_per_epoch)

training_model = NN(state_dim=state_dim, action_n=action_n)
training_agent = DQN(model=training_model, action_n=action_n, batch_size=batch_size, trajectory_n=traj_per_epoch)
```

не привела к изменениям в результате

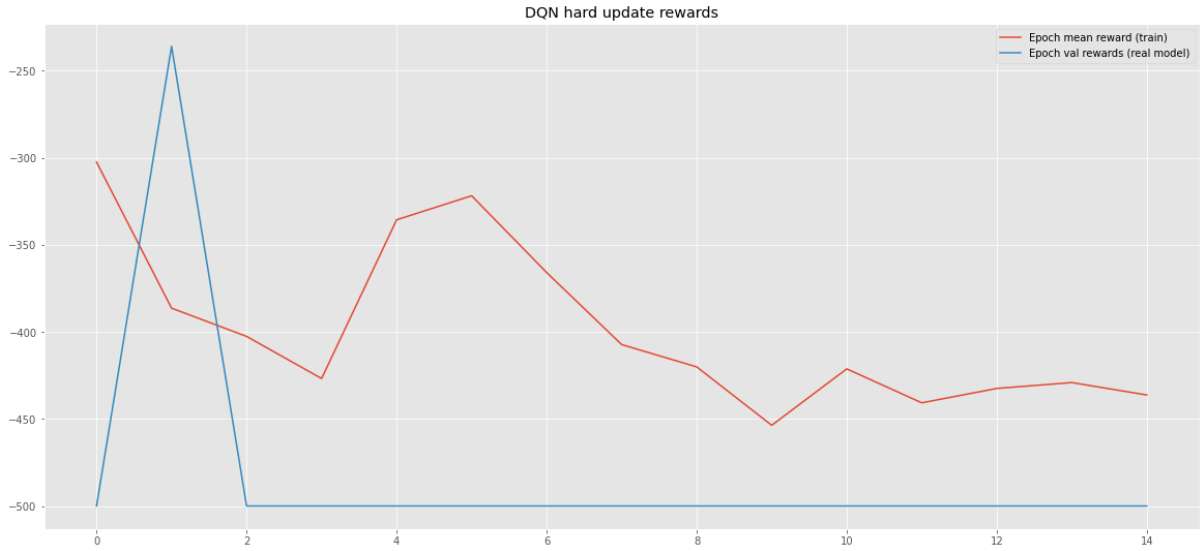


Но она дала нечто другое: теперь оба агента - глобальные переменные и можно было посмотреть, что не работает. Как я понял, ничего не работает, ибо награда с траектории, полученной с помощью training_agent так же была -500

Для реализации второй идеи я просто убрал model из свойств самого класса DQN и сделал model параметром передаваемым в функции get_action и training_step,
График наград:



Сложно объяснить, почему target_model показала такой результат на 11 эпохе. Так же я не очень сильно понимаю из-за чего на шаге валидации награда постоянно -500 (возможно стоит записывать не последние веса модели, на которой обучаем, я средние за эпоху). Также я не могу сказать является ли то что я делал неудаче, ибо модель как-то обучается. Однако я также решил сохранять модель в конце каждой эпохи, и после обучения награда при использовании каждой из этих моделей была -500.
После этого я попробовал увеличить количество траекторий за эпоху (с 30 до 100), результат был следующим



Интересно, что награда, отличная от -500 стала появляться, когда я перешел в колаб. Я попытался увеличить количество траекторий до 250, но сессия крашнулась на 5 эпохе из-за того, что закончилась оперативка. Я попытался запустить на gpu, но там на эпоху уходит около 10 минут (при 250 траекториях за эпоху), так что графики с наградами добавить я вряд ли успею. Судя по первым эпохам, проблема никуда не исчезла, val reward = -500.

```
100% 250/250 [06:53<00:00, 1.41s/it]
parameter 0 is equal
parameter 1 is equal
parameter 2 is equal
parameter 3 is equal
parameter 4 is equal
parameter 5 is equal
epoch = 0     mean_epoch_loss = tensor(8957.9873, device='cuda:0', grad_fn=<DivBackward0>)    mean_epoch_reward = -199.284      val_reward = -500.0
100% 250/250 [11:17<00:00, 2.15s/it]
parameter 0 is equal
parameter 1 is equal
parameter 2 is equal
parameter 3 is equal
parameter 4 is equal
parameter 5 is equal
epoch = 1     mean_epoch_loss = tensor(68785.3750, device='cuda:0', grad_fn=<DivBackward0>)    mean_epoch_reward = -324.48   val_reward = -500.0
100% 250/250 [11:05<00:00, 2.12s/it]
parameter 0 is equal
parameter 1 is equal
parameter 2 is equal
parameter 3 is equal
parameter 4 is equal
parameter 5 is equal
epoch = 2     mean_epoch_loss = tensor(11911.8027, device='cuda:0', grad_fn=<DivBackward0>)    mean_epoch_reward = -320.58   val_reward = -500.0
38% 96/250 [03:56<07:15, 2.83s/it]
```

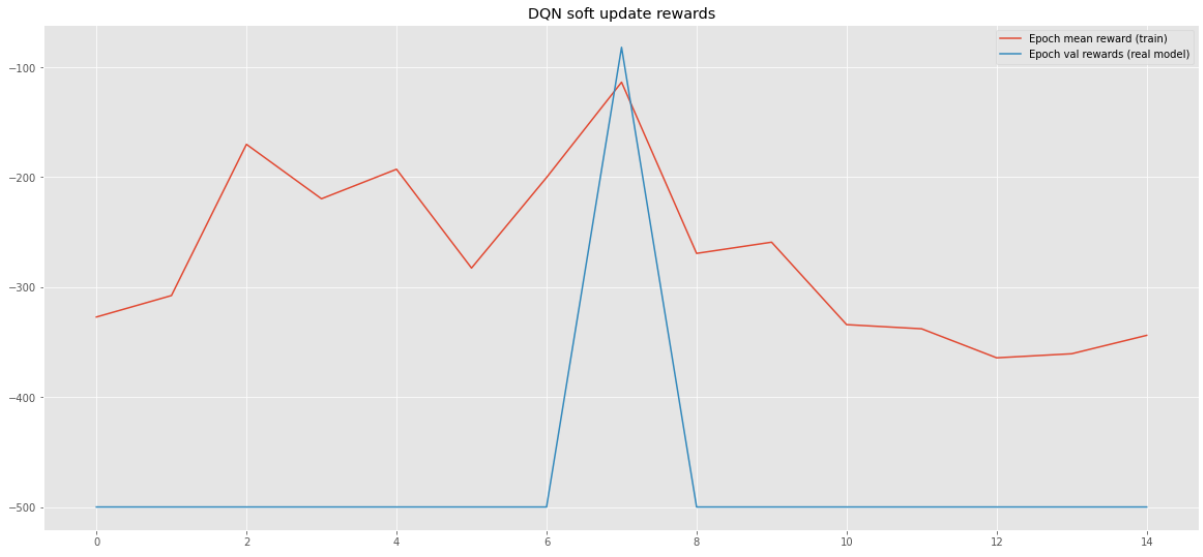
Вероятнее всего, я упускаю что-то на техническом уровне.

2.2 Soft update

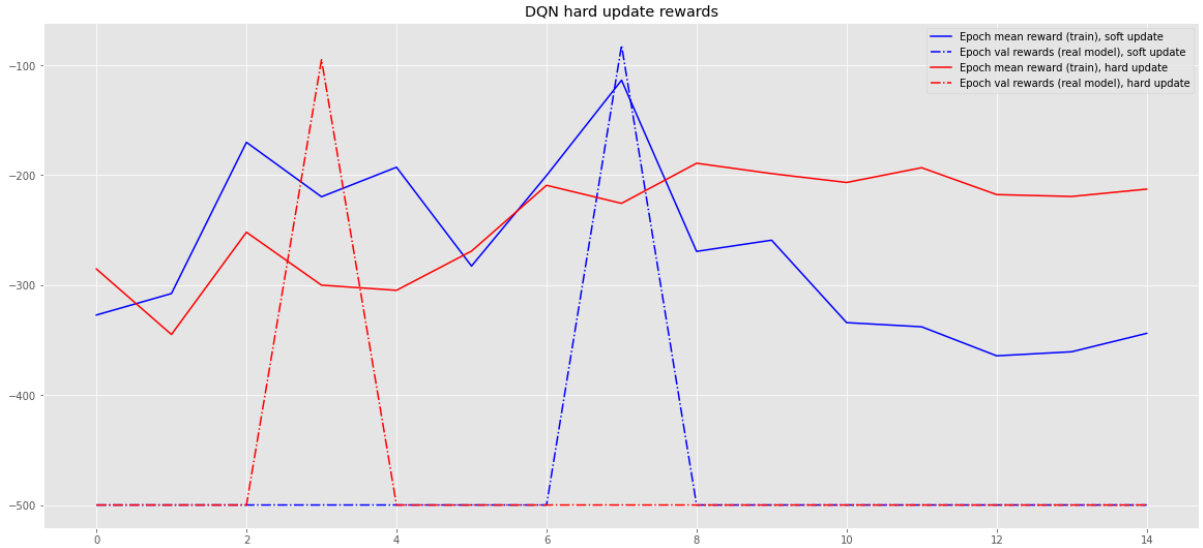
Не уверен, что есть смысл переходить к Soft update, однако почему бы не попробовать. Как я понял исходя из лекции концепт должен быть примерно таким:

```python
checkpoint = training_model.state_dict()
real_model = target_model.state_dict()
tau = 0.8

for p_f, p_r in zip(checkpoint, real_model):
    real_model[p_r] = (checkpoint[p_f] * tau) + (real_model[p_r] * (1 - tau))


target_model.load_state_dict(real_model)
```

Я добавил код выше в функцию train, однако это не решило проблему:



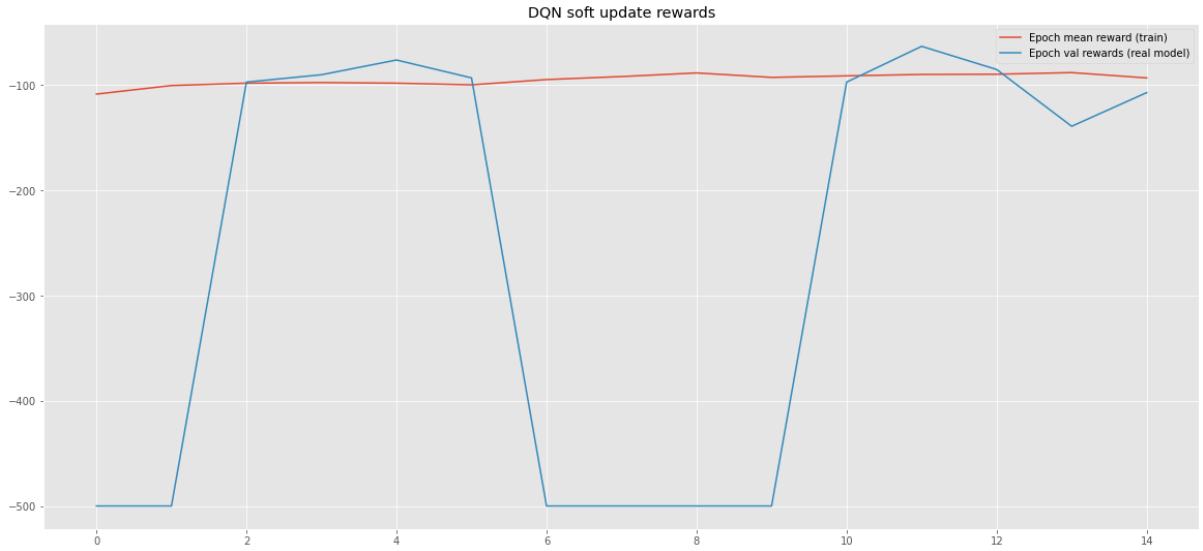
И сравнение hard update с soft update:



Сильных улучшений замечено не было
Тут везде по 50 траекторий за эпоху

Возможно стоит добавить больше траекторий, или больше эпох.
Я решил пока что не переходить к dqn, сначала хочется разобраться с текущей проблемой.

Спустя некоторое время, я решил вместо нескольких функций train написать одну, в которой можно уточнять метод обучения (hard update, soft update или double dqn). На удивление, после этого график обучения стал выглядеть вот так



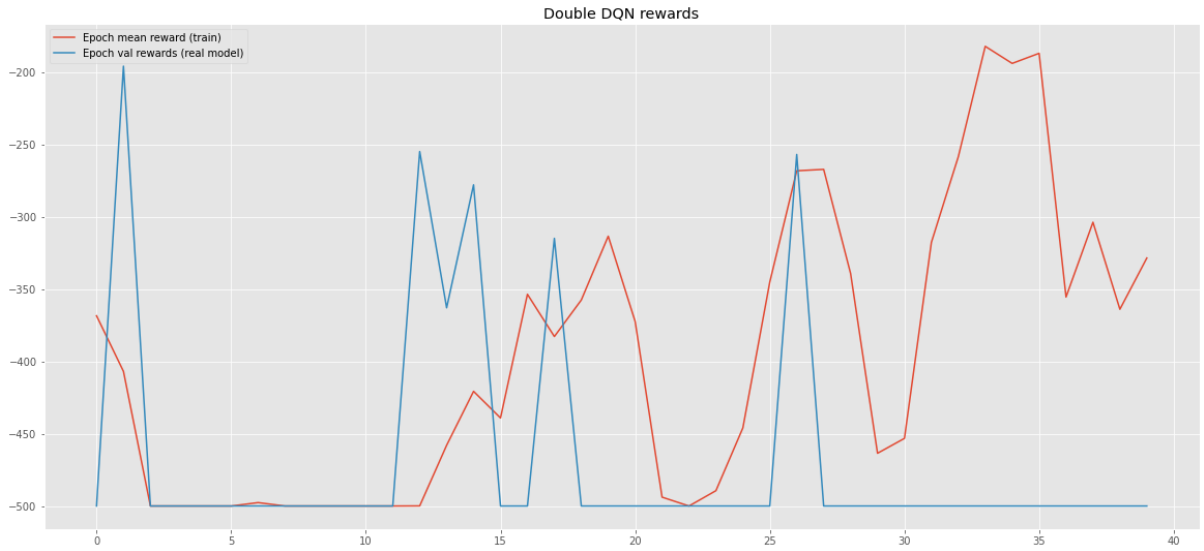
Не совсем понятно, почему это решило проблему, но вроде работает лучше, чем было.

2.3 Double DQN

Для double dqn, как я понял, нужно немного поменять функцию training step в классе dqn.

```
q = training_model(states)

if not double:
    q_next = training_model(next_states)
else:
    q_next = target_model(next_states)
```

также я подумал что имеет смысл обновлять веса на target model почаще, и поставил 40 эпох с 50 траекториями за эпоху вместо 15 и 150 соответственно. График наград выглядит вот так



Почему модель обучается не так хорошо, пока что остается загадкой. Я добавил третий файл .py со всеми внесенными изменениями.