2.1 Acrobot-v1

как и в прошлый раз, я решил начать эксперименты с копирования кода практического занятия. Результат был довольно плачевным, награда постоянно была равна -500, самого обучения не происходило, так как метод “update_policy” просто не вызывался.

После этого я решил добавить несколько изменений:

1) Заменить “>” на “>=” в функции get_elite_trajectories, для того чтобы быть уверенным, что метод “update_policy” будет вызван хоть раз;
2) Заменить название “update_policy” на “training_step”, и немного переписать весь класс модели, вытащив оттуда оптимизатор, сделать так, чтобы “training_step” возвращала ошибку (умноженную на 1000), и прописывать loss.backward(), optimizer.step() и optimizer.zero_grad() в функции train (что также позволяет выводить ошибку во время обучения);
3) Добавить q-param scheduling так как на основе первых экспериментов мне показалось, что лучше начинать с достаточно малого порога, постепенно его увеличивая: `q_sched = lambda epoch: max(0.2, (epoch / epochs)*0.95)`
4) Добавить learning rate scheduling, потому что почему бы и нет. Я использовал OneCycleLR.

После этого я решил попробовать обучить модель еще раз. Принимая во внимание опыт предыдущего задания, я решил поставить среднее количество траекторий за эпоху = 100 и обучать на протяжении 50 эпох (а надо было 51). Результат виден на скриншоте ниже (графиков пока нет, ибо смысла их добавлять я пока не вижу)

```
Epoch: [0]              Loss: [1098.61181640625]              Reward: [-493.6]              Q param: [0.2]
Epoch: [2]              Loss: [1098.562744140625]             Reward: [-148.56]             Q param: [0.2]
Epoch: [4]              Loss: [1098.457275390625]             Reward: [-92.32]              Q param: [0.2]
Epoch: [6]              Loss: [1098.4583740234375]            Reward: [-84.28]              Q param: [0.2]
Epoch: [8]              Loss: [1098.451171875]                Reward: [-80.58]              Q param: [0.2]
Epoch: [10]             Loss: [1098.4429931640625]            Reward: [-84.88]              Q param: [0.2]
Epoch: [12]             Loss: [1098.4337158203125]            Reward: [-91.06]              Q param: [0.22799999999999998]
Epoch: [14]             Loss: [1098.4296875]          Reward: [-88.82]              Q param: [0.266]
Epoch: [16]             Loss: [1098.405029296875]             Reward: [-85.58]              Q param: [0.304]
Epoch: [18]             Loss: [1098.3740234375]               Reward: [-83.42]              Q param: [0.34199999999999997]
Epoch: [20]             Loss: [1098.3765869140625]            Reward: [-87.47]              Q param: [0.38]
Epoch: [22]             Loss: [1098.401611328125]             Reward: [-148.97]             Q param: [0.418]
Epoch: [24]             Loss: [1098.5802001953125]            Reward: [-316.91]             Q param: [0.45599999999999996]
Epoch: [26]             Loss: [1098.586181640625]             Reward: [-435.92]             Q param: [0.494]
Epoch: [28]             Loss: [1098.5928955078125]            Reward: [-443.4]              Q param: [0.532]
Epoch: [30]             Loss: [1098.5933837890625]            Reward: [-440.63]             Q param: [0.57]
Epoch: [32]             Loss: [1098.595458984375]             Reward: [-495.4]              Q param: [0.608]
Epoch: [34]             Loss: [1098.595458984375]             Reward: [-496.3]              Q param: [0.646]
Epoch: [36]             Loss: [1098.6124267578125]            Reward: [-500.0]              Q param: [0.6839999999999999]
Epoch: [38]             Loss: [1098.6123046875]               Reward: [-500.0]              Q param: [0.722]
Epoch: [40]             Loss: [1098.612548828125]             Reward: [-500.0]              Q param: [0.76]
Epoch: [42]             Loss: [1098.6124267578125]            Reward: [-500.0]              Q param: [0.7979999999999999]
Epoch: [44]             Loss: [1098.6123046875]               Reward: [-500.0]              Q param: [0.836]
Epoch: [46]             Loss: [1098.6124267578125]            Reward: [-500.0]              Q param: [0.874]
Epoch: [48]             Loss: [1098.6124267578125]            Reward: [-500.0]              Q param: [0.9119999999999999]
Training took 746.4401 secs
```

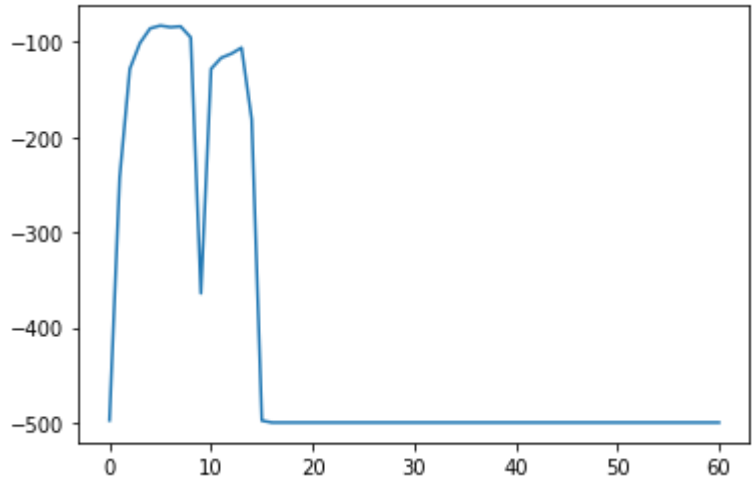
Есть несколько интересных моментов:

1) Ошибка почти не меняется за все время обучения
2) Модель изначально обучалась (эпохи 0-22), а после начала деградировать; не совсем понятно, связано ли это с увеличением q_param (вероятнее всего связано)

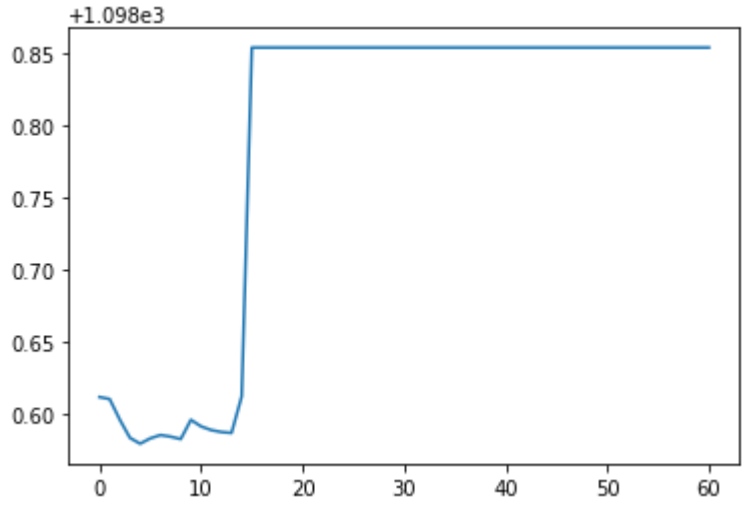
Для того, чтобы протестировать гипотезу, я также решил выводить количество траекторий, признанных элитными. (500 траекторий за эпоху, 61 эпоха)

```
15
Epoch: [0]              Loss: [1098.61181640625]              Reward: [-497.748]            Q param: [0.2]
400
398
Epoch: [2]              Loss: [1098.59619140625]              Reward: [-128.554]            Q param: [0.2]
400
392
Epoch: [4]              Loss: [1098.57958984375]              Reward: [-85.982]             Q param: [0.2]
399
395
Epoch: [6]              Loss: [1098.5855712890625]            Reward: [-84.694]             Q param: [0.2]
388
400
Epoch: [8]              Loss: [1098.582763671875]             Reward: [-95.538]             Q param: [0.2]
258
399
Epoch: [10]             Loss: [1098.591552734375]             Reward: [-128.456]            Q param: [0.2]
399
392
Epoch: [12]             Loss: [1098.587646484375]             Reward: [-112.514]            Q param: [0.2]
397
386
Epoch: [14]             Loss: [1098.6123046875]               Reward: [-181.608]            Q param: [0.21803278688524588]
3
Epoch: [16]             Loss: [1098.8533935546875]            Reward: [-500.0]              Q param: [0.24918032786885247]
Epoch: [18]             Loss: [1098.8533935546875]            Reward: [-500.0]              Q param: [0.280327868852459]
Epoch: [20]             Loss: [1098.8533935546875]            Reward: [-500.0]              Q param: [0.31147540983606553]
Epoch: [22]             Loss: [1098.8533935546875]            Reward: [-500.0]              Q param: [0.34262295081967215]
Epoch: [24]             Loss: [1098.8533935546875]            Reward: [-500.0]              Q param: [0.37377049180327865]
```

Как видно, с увеличением q_param количество элитных траекторий серьезно упало, после чего ошибка стала равна -500, и никогда больше не поднялась
На всякий случай графики:



Средняя награда



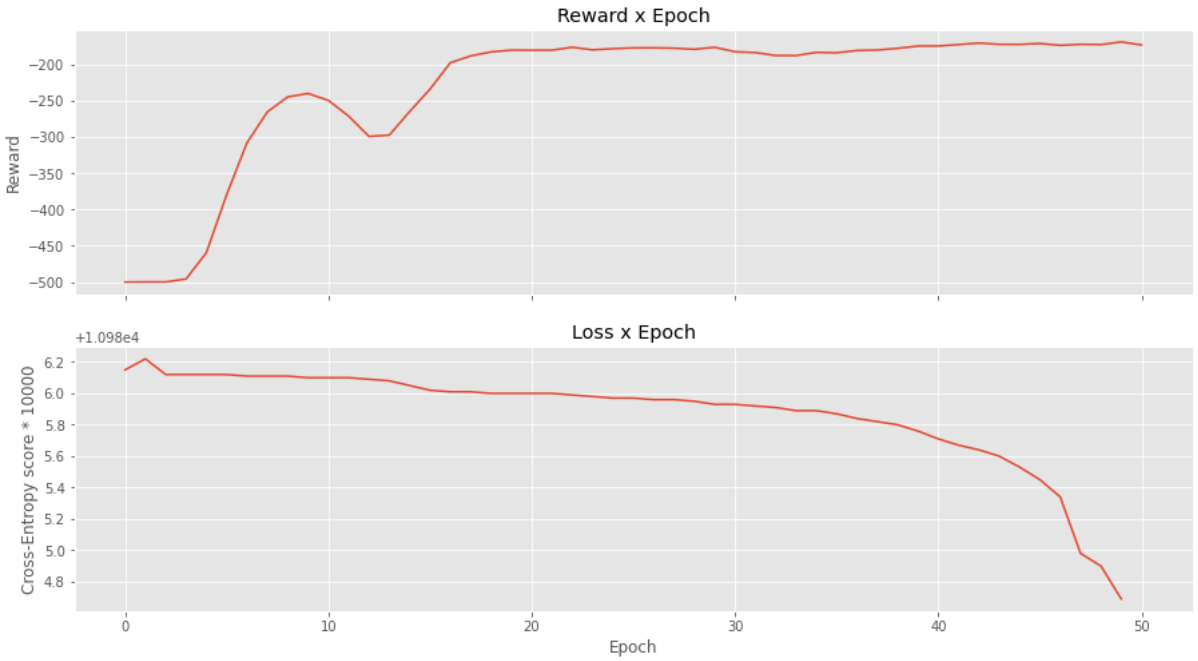
Ошибка (e * 1000)

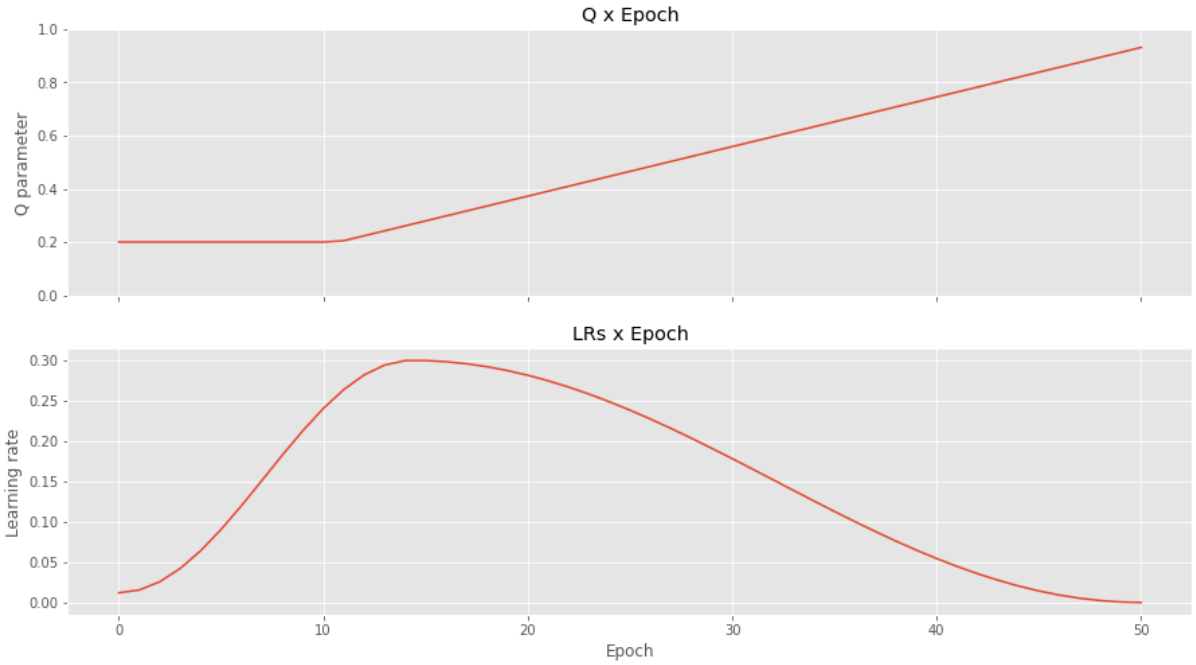
На третьей попытке я решил:

- вернуть строгое неравенство в функцию get_elite_trajectories()
- немного переписть саму сеть, изменив количество нейронов:

```
self.net = nn.Sequential(
           nn.Linear(self.state_dim, 128),
           nn.ReLU(True),
           nn.Linear(128, 256),
           nn.ReLU(True),
           nn.Linear(256, 128),
           nn.ReLU(True),
           nn.Linear(128, self.action_n),
           nn.Softmax(dim=0))
```

После этого я тренировал сеть при 800 траекториях за эпоху на протяжении 51 эпохи.
Ниже представлены графики наград, ошибки * 10000, q_param и learning rates.
Как видно, награда доходит до -200, что как минимум значит, что задача решена.





Что также интересно, и стоит заметить - в начале, на генерацию 800 траекторий уходило приблизительно по 3 минуты:

```
100% 800/800 [03:03<00:00, 4.40it/s]
N of elite_trajectories [3]
Epoch [1]     Reward [-499.79]     Loss [10986.15]      Q param [0.2]    Last lr [0.01546]
100% 800/800 [03:09<00:00, 4.47it/s]
N of elite_trajectories [1]
Epoch [2]     Reward [-499.84]     Loss [10986.22]      Q param [0.2]    Last lr [0.02568]
100% 800/800 [03:01<00:00, 4.22it/s]
N of elite_trajectories [45]
Epoch [3]     Reward [-495.76]     Loss [10986.12]      Q param [0.2]    Last lr [0.04216]
100% 800/800 [02:50<00:00, 3.49it/s]
```

Однако, ближе к завершению обучения на это требовалось уже всего чуть больше минуты:

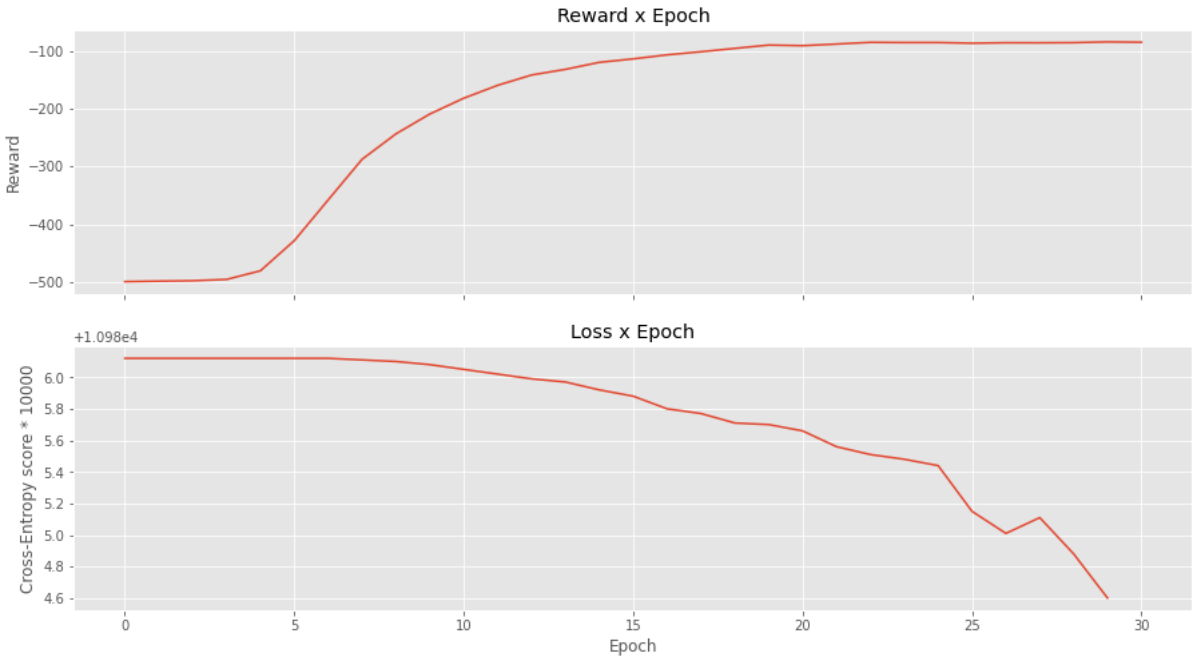
```
Epoch [47]    Reward [-171.9]      Loss [10985.34]      Q param [0.87549]    Last lr [0.0052]
100% 800/800 [01:06<00:00, 12.04it/s]
N of elite_trajectories [67]
Epoch [48]    Reward [-172.31]     Loss [10984.98]      Q param [0.89412]    Last lr [0.00232]
100% 800/800 [01:19<00:00, 12.99it/s]
N of elite_trajectories [65]
Epoch [49]    Reward [-168.66]     Loss [10984.9]   Q param [0.91275]    Last lr [0.00058]
100% 800/800 [01:08<00:00, 11.70it/s]
N of elite_trajectories [54]
Epoch [50]    Reward [-172.74]     Loss [10984.69]      Q param [0.93137]    Last lr [0.0]
Training took 4519.2679 secs
```

Моя гипотеза заключалась в том, что основное время программа тратит на генерацию траекторий, а не на собственно обучение. Хотя это вполне можно посчитать: 50 эпох, на каждую уходило как минимум больше минуты, то есть генерация целиком заняла около 60 минут, в то время как вся программа заняла примерно 75.

После этого я примерно сутки пытался подключить multiprocessing для генерации траекторий, и у меня не получилось.

На основе графиков выше видно, что награда выходит на плато после примерно 30 эпохи, что подсказывает, что 30 эпох может быть уже достаточно.

Последний экспиремент, код которого будет загружен, я провел без особых изменений кода: (30 эпох, 800 траекторий за эпоху, возрастающий q_param). Результат заметно улучшился, награда возросла до -80, что, как я понимаю, вполне достойный результат:

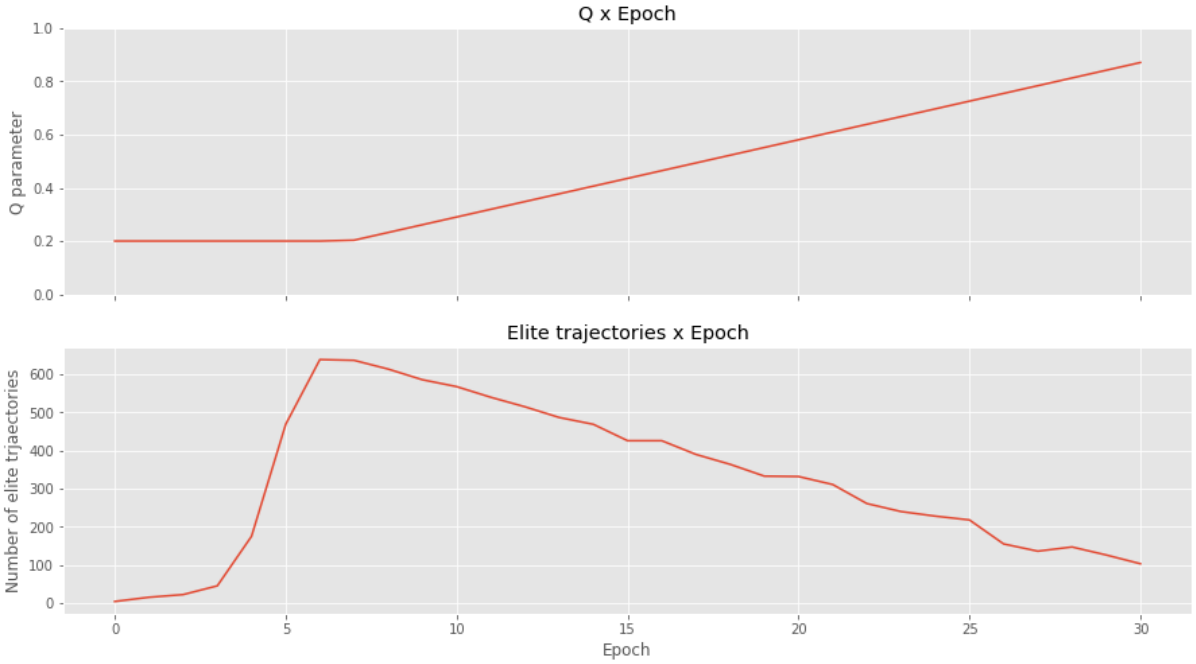


Результат последней эпохи:

```
%%time
last_trajectory = get_trajectory(env, agent, trajectory_len=500)
print(last_trajectory['reward'])
```

```
-65.0
CPU times: user 65.7 ms, sys: 2.92 ms, total: 68.6 ms
Wall time: 161 ms
```

Также я решил исследовать то, сколько элитных траекторий есть за эпоху и каким образом увелечение q_param сказывается на этом количестве:



Результат вполне ожидаем и не противоречит логике, что хорошо. Ради интереса, я также решил проверить коэффициент корреляции Пирсона (подходит в данном случае, так как оба массива - continuous values) предварительно нормализовав данные, что на самом деле вроде уже вшито в функцию pearsonr из scipy.stats

```
Pearson correlation for q_params and elite traj numbers: test statistic and p-value
(-0.360623644640199, 0.04626262179019945)
```

Как видно, p-value = 0.04, что меньше, чем 0.05 и сам коэффициент равен -0.36, что позволяет сделать вывод о присутствии отрицательной корреляции.

Однако, когда ты начинаешь заниматься такими вещами, становится трудно остановиться и поэтому я решил также проверить корреляцию q_param со всем остальным. Простейший способ сделать это - построить корреляционную матрицу:

```
import pandas as pd
import seaborn as sns

history_df = pd.DataFrame(history)

history_df = (history_df - history_df.mean()) / history_df.std()

corr_mat = history_df.corr()
```

2.2 MountainCarContinuous

Собственно, никакие из моделей, построенных ранее, не работали, ибо лосс постоянно был = 0 и обучения не происходило.

Как я понял, чтобы реализовать метод кросс-энтропии для данной задачи, нужно переписать всю модель таким образом, чтобы вручную сохранять и перезаписывать веса и байасы элитных траекторий.

До дедлайна я этого сделать не успеваю, возможно добавлю в папку позже.