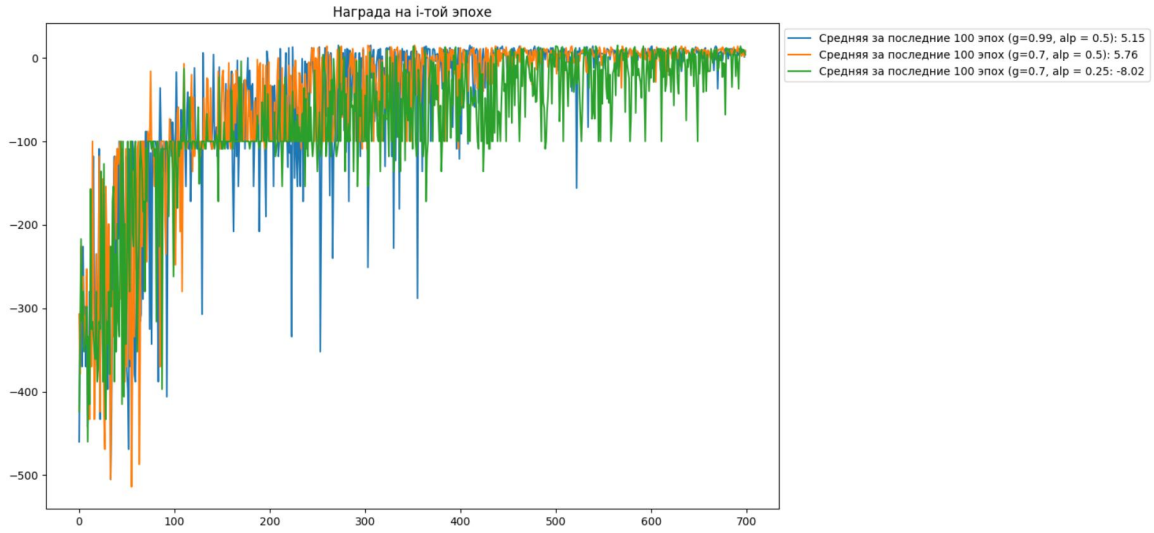
Задание 4.

Task_1.

Я дописал функцию Q-Learninga, изменив только одну строку.

```
qfunction[state][action] += alpha * (
        reward
        + gamma * max(qfunction[next_state])
        - qfunction[state][action]
)
```

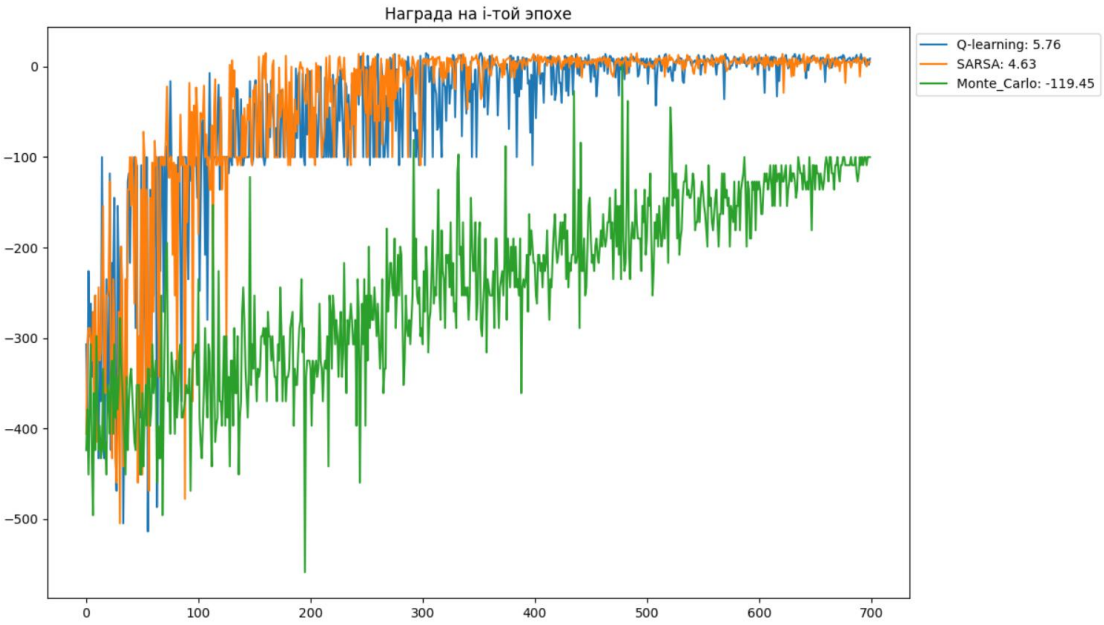
Результаты графиков Q-Learning при разных g and alpha



Наилучшая работа алгоритма была при g =0.7, и al=0.5

При изучение методов Монте-Карло и SARSA при больших значениях Q графики выглядили менее стабильными (меньше резких пиков).

Построим все 3 графика на одном изображение:



Лучше всего отработал метод Q-learninga. Наверное, одна из проблем была неболшое кол-во итераций для модели Monte-Carlo. Не уверен, что модель, вышла на свое «статическое плато». Я думаю еще пару сотен итераций, и модель могла достигнуть более хороших результатов

Task 2.

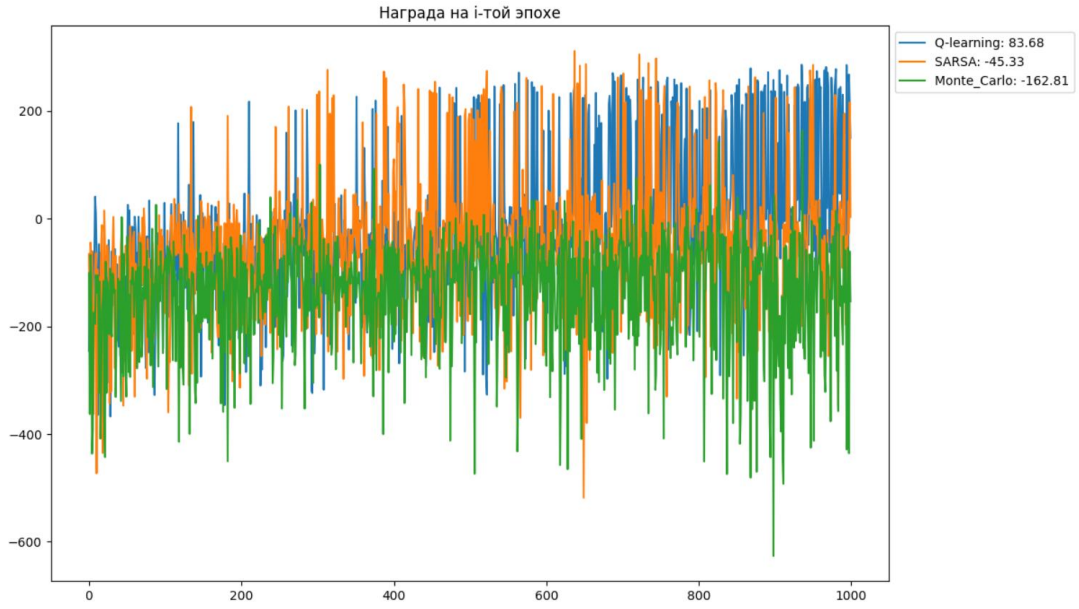
Я выбрал задачу << LunarLander-v2 >>.
Проблема в том, что стандартно, у самолета есть state_n = 8, где 6 значений - флотовые значения из огранничего промежутка.

Я решил, что буду округлять каждое недетриминрованное значение до числа с шагом 0.5 (-3 , -2.5, -2, -1.5, -1, -0.5, 0, 0.5, 1).
Я использовал метод *" state = tuple(np.round(state * 2) / 2) ".*

Я переписал метод наполнения qfunction

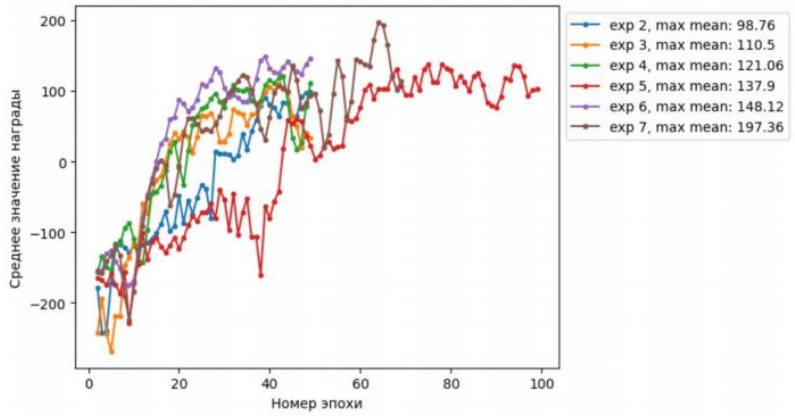
```
total_rewards = []

qfunction = dict()
counter = dict()
for episode in range(episode_n):
    epsilon = 1 - episode / episode_n
    trajectory = {'states': [], 'actions': [], 'rewards': []}

    state = env.reset()
    state = tuple(np.round(state * 2) / 2)  # округлил каждое value до 0.5
    for _ in range(trajectory_len):

        trajectory['states'].append(state)

        if state not in qfunction:
            qfunction[state] = np.zeros(action_n)
            counter[state] = np.zeros(action_n)
        action = get_epsilon_greedy_action(qfunction[state], epsilon, action_n)
        trajectory['actions'].append(action)

        state, reward, done, _ = env.step(action)
        state = tuple(np.round(state * 2) / 2) # округлил каждое value до 0.5

        trajectory['rewards'].append(reward)

        if done:
            break
```

Посмотрим общий график обучение моделей при q = 0.99 и a = 0.5



Как мы видим, средняя наградная получилось наибольшей в случае, когда мы использовали метод Q обучения. Наверно, при изменение кол-во дискретных состояний (брать значения с шагом не 0.5, а более мелким (0.25, 0.1 и т.д.) результаты могли быть существенно другими. Сравним данные результаты с результатами Cross_Entropy обучения из д.з. 2

exp	Q_param	Trajectory_n	Itern_n	Max_len	lr	Hidden_layer(1)	Hidden_layer(2)	Max mean (total_rewards)
1	0.6	50	50	400	0.01	208, 4	-	47.19
2	0.6	50	50	400	0.01	208, 104	104, 4	98.758
3	0.9	100	50	200	0.01	208, 104	104, 4	110.49
4	0.7	100	50	250	0.01	64, 104	104, 4	121
5	0.7	25	100	250	0.01	64,32	32,4	136.43
6	0.8	300	50	300	0.01	128, 64	64,4	148
7	0.9	400	70	500	0.01	128, 64	64,4	197



Если не учитывать кол-во траекторий на каждой из эпох при обучение с помощью нейронных сетей, тогда можно сделать вывод, что награда при эксперементе с самыми неудачными гиперпараметрами на куде, дала лучшие результаты чем Q-learning, Monte_Carlo и SARSA. Наверно, при корректировке кол-во состояний (bin-ы для параметров state_n) данные алгоритмы смогут потягаться с Cross_Entropy обучением.