Трапер Максим. СПБГУ. Магистратура 1 курс «Искусственный интеллект и наука о данных»

P.S: очередной подскриптум, как и в любой моей работе). Свои «неправильные» мысли по реализации я оставил (зачеркнул и выделил), чтобы сохранить ход своих мыслей и, если вдруг буду пересматривать эти файлы, не буду нарываться на те же ошибки. Можете их не читать)

Задание №1: реализовать Q-Learning и сравнить его результаты с реализованными ранее алгоритмами: Cross-Entropy, Monte Carlo, SARSA в задаче Taxi-v3. Для сравнения как минимум нужно использовать графики обучения. Причем графики лучше делать относительно количества сгенерированных траекторий.

~~Cross-Entropy берём с параметрами, которые были выбраны в предыдущем ДЗ, как лучшие. q_param=0.4, iteration_n=30, trajectory_n=400, smoothing='Policy', coef_lambda=0.95~~

~~Я сглупил. Изначально хотел брать лучшие параметры из прошлого ДЗ, но не учёл что Cross-Enthropy Method имеет несколько другую природу, т.к. ориентируется на каждом шаге не на одну траекторию, а на множество. В лучших параметрах было выбрано малое число итераций относительно остальных методов (что не позволит адекватно оценить динамику работы метода на графике). Поэтому было принято решение провести некоторую ребалансировку между двумя параметрами – iteration_n и trajectory_n, в сторону увеличения первого и уменьшения второго, т.к. они в некоторой мере взаимозаменяемы.~~

~~Идея не удалась, т.к. при малом количестве траекторий качество обучения падает и упирается в потолок -200. Стоило бы об этом вспомнить.~~

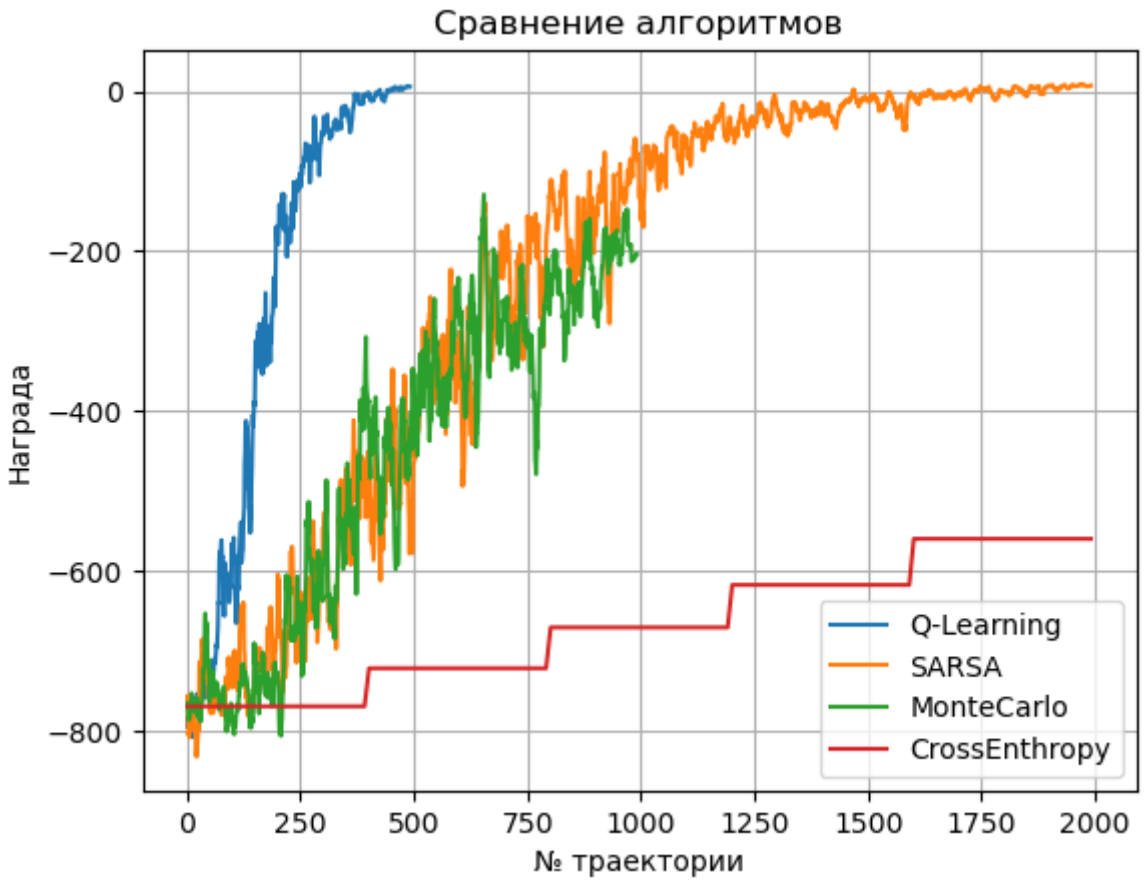
Cross-Entropy берём с параметрами, которые были выбраны в предыдущем ДЗ, как лучшие (на самом деле, обнаружил, что можно было trajectory_n убавить на 100 и ускорить ~ на 15% сходимость). q_param=0.4, iteration_n=30, trajectory_n=400, smoothing='Policy', coef_lambda=0.95. В список mean_total_rewards на каждом шаге итерации записываем результат в количестве trajectory_n.

Так как все методы, кроме CEM, очень шумные, применяем перед выводом графика скользящее окно = 10.

Параметры алгоритмов:

1. CrossEntropy - q_param=0.4, iteration_n=16, trajectory_n=400, smoothing='Policy', coef_lambda=0.95
2. QLearning - episode_n=500, noisy_episode_n=400, t_max=1000, gamma=0.999, alpha=0.5
3. SARSA - episode_n=2000, trajectory_len=1000, gamma=0.999, alpha=0.5
4. MonteCarlo - episode_n=2000, trajectory_len=1000, gamma=0.99

Посмотрим три основных алгоритма «поближе», обрезав CEM на 2000 траектории.

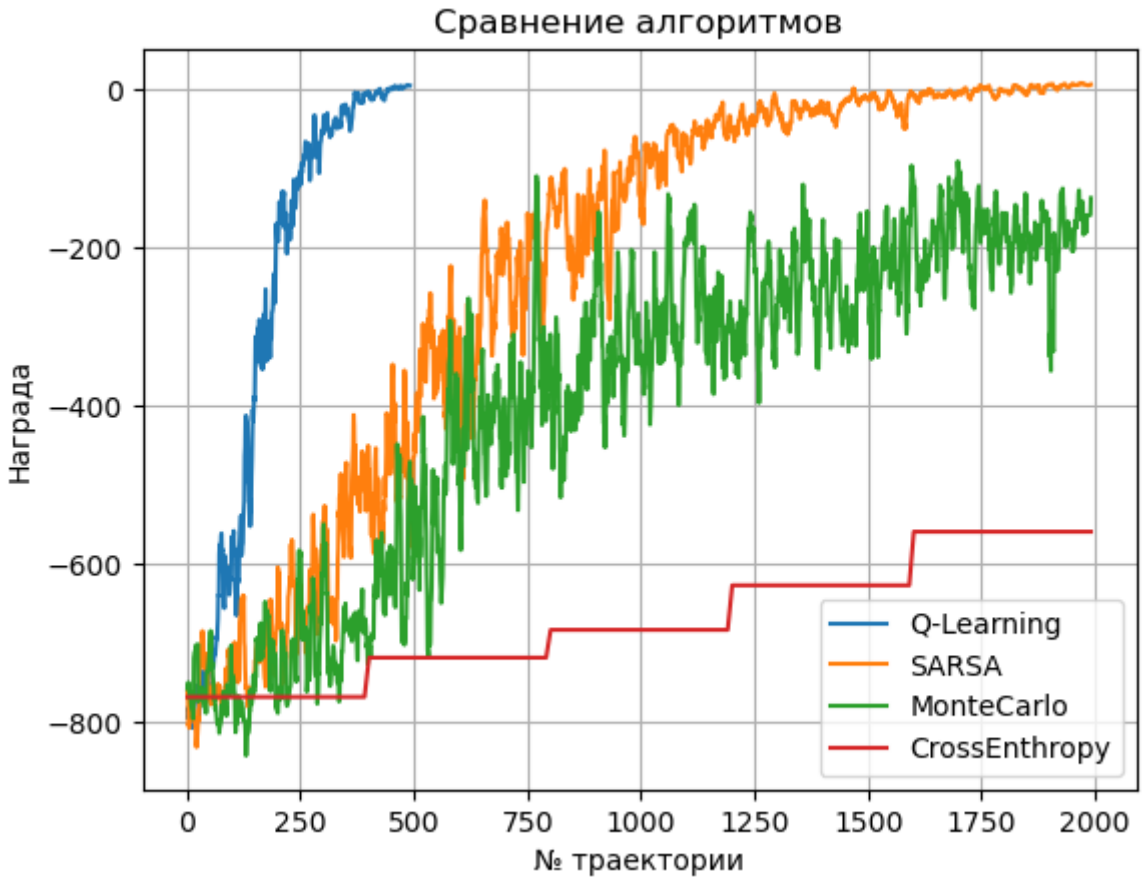



```
CrossEntropy_total_rewards
✓ 1m 0.5s

QLearning_total_rewards
✓ 3.7s

SARSA_total_rewards
✓ 11.8s

MonteCarlo_total_rewards
✓ 12.9s
```

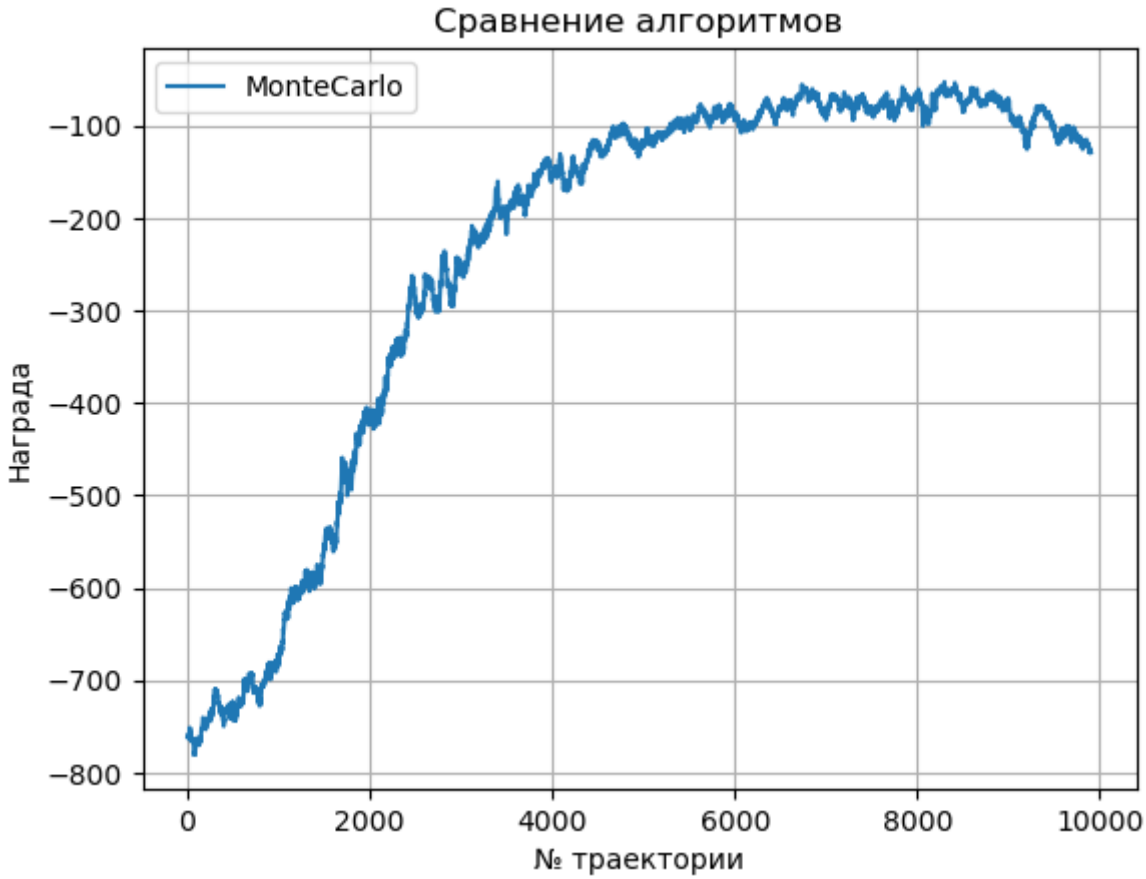
В общем, в данном случае, безоговорочным лидером среди 4-ех алгоритмов является QLearning. Ему и траекторий понадобилось меньше, и кратно меньше времени на вычисления. Но надо учитывать, что скорее всего такое качество достигается в симбиозе алгоритма и специфичности решаемой задачи (дискретное пространство, отсутствие наказаний).

CEM всё ещё способна находить решение задачи, что было выявлено ранее (ДЗ №1), но делает это кратно дольше Q-Learning и SARSA.

Попробуем увеличить количество эпизодов в MonteCarlo до 2000.



Ситуация с MonteCarlo улучшилась, но слабо. Вероятно, при такой динамике он скоро выйдет на плато. Попробуем увеличить кол-во итераций до 10000. Всё ради интереса.



Ну всё же нет, на плато алгоритм вышел примерно на 6-7к траекториях и потом очень плавно награда начала падать. Понятно, что нужно оптимизировать значения других гиперпараметров. Ещё, судя по следующему заданию, поможет оптимизация подбора значения эпсилон на каждом шаге. Дальше посмотрим :)

~~P.S: к слову, странно метод работает. При episode_n = 2000 награда достигает -200, а при episode_n = 10000 награда на промежуточном шаге в 2000 достигает награды лишь в -400...~~

Тот факт, что для сходных результатов по награде при большем episode_n нужно сделать больше шагов, является ещё одним доказательством того, что значения эпсилон вычисляются не по самой оптимальной формуле.

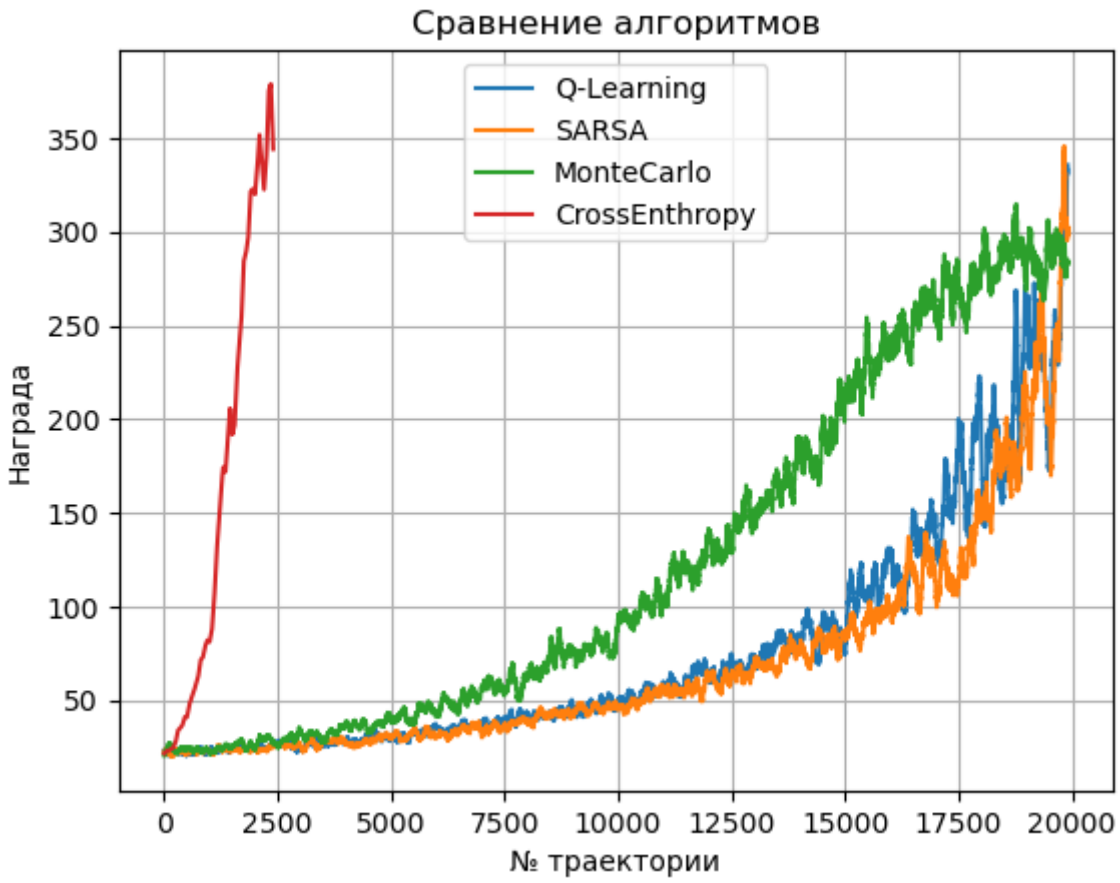
Задание №2: дискретизировать (можно использовать numpy.round()) пространство состояний и обучить Агента решать CartPole-v1 методами Monte-Carlo, SARSA и Q-Learning. Сравнить результаты этих алгоритмов и реализованного ранее алгоритма Deep Cross-Entropy на графиках.

Честно, я выбрал CartPole-v1 т.к. она оказалась легче всех для методов. Все остальные мне даже не поддались и награды, в основном, были минимальными. Например, где-то размерность состояния была большой (Acrobot = 6).

Собственно, сначала я подобрал оптимальные параметры под каждый метод для задачи (опустил это, чтобы не городить кучу текста). В DeepCrossEnthropy оставил непрерывное пространство, т.к. так он лучше себя всего покажет.

Лучшие параметры:

1. QLearning - episode_n=20000, t_max=500, gamma=0.999, alpha=0.5, levels=10
2. SARSA_total_rewards - episode_n=20000, trajectory_len=500, gamma=0.999, alpha=0.3
3. MonteCarlo - episode_n=20000, trajectory_len=500, gamma=0.99
4. DeepCrossEnthropy - iteration_n=50, epsilon=0.3, hidden_size=128, lr=1e-2, trajectory_n=50, trajectory_len=300, q=0.8



Использовалось скользящее окно = 100, чтобы снизить шум на графике.

На самом деле, до этого были результаты получше у части методов, награда чуть больше 400. Но понятно, это всё ввиду стохастичности процессов.

Все три метода показали последовательный рост награды с каждой новой траекторией. Это конечно плюс, хотя судя по методу Монте-Карло, возможно уже было достигнуто плато в качестве.

Но есть то, что мне очень не нравится по сравнению с DeepCEM. Не смотря на огромное окно для сглаживания шумов на графике, результаты всё равно не самые стабильные. Всё это говорит о нестабильности обучения под задачу с использованием этих методов. Т.е. тяжело в реальной работе положиться на какую-то отдельно обученную политику данными методами.

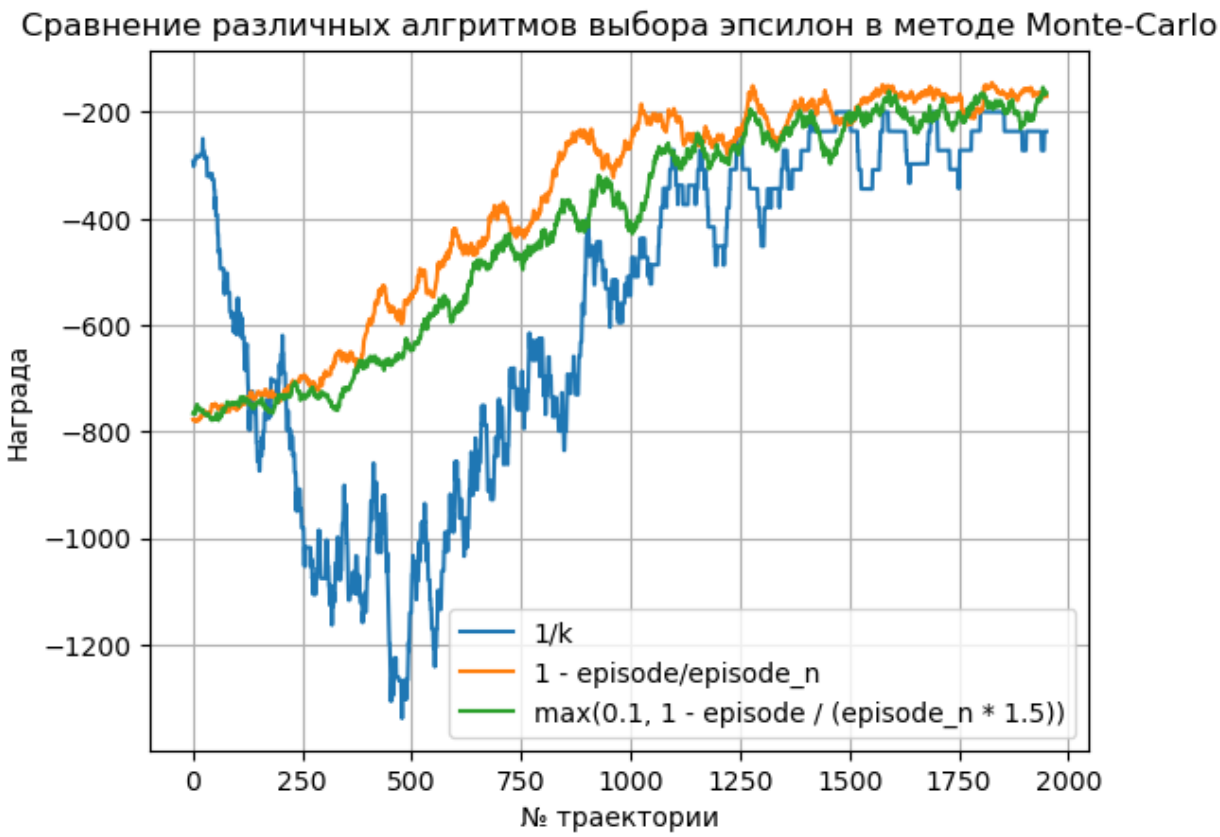
Общий вывод – новые методы текущего занятия – SARSA, QLearning и Monte-Carlo конечно можно адаптировать для использования в задачах с непрерывными пространствами, но всё же есть методы (DCEM, etc), гораздо лучше работающие в таких задачах.

Задание №3: придумать стратегию для выбора epsilon позволяющую агенту наилучшим образом решать Taxi-v3 алгоритмом Monte Carlo.

Сначала я взял два метода: из лекции и предложенный на практике, как некоторый бейзлайн. Они будут фигурировать на всех графиках.

Затем я проверил методы подбора эпсилон, предложенные ChatGPT в течении работы над прошлыми заданиями.

Max(0.1, 1 – episode/(episode_n * 1.5)). Логика простая – просто более медленное линейное снижение эпсилон с ростом номера эпизода.



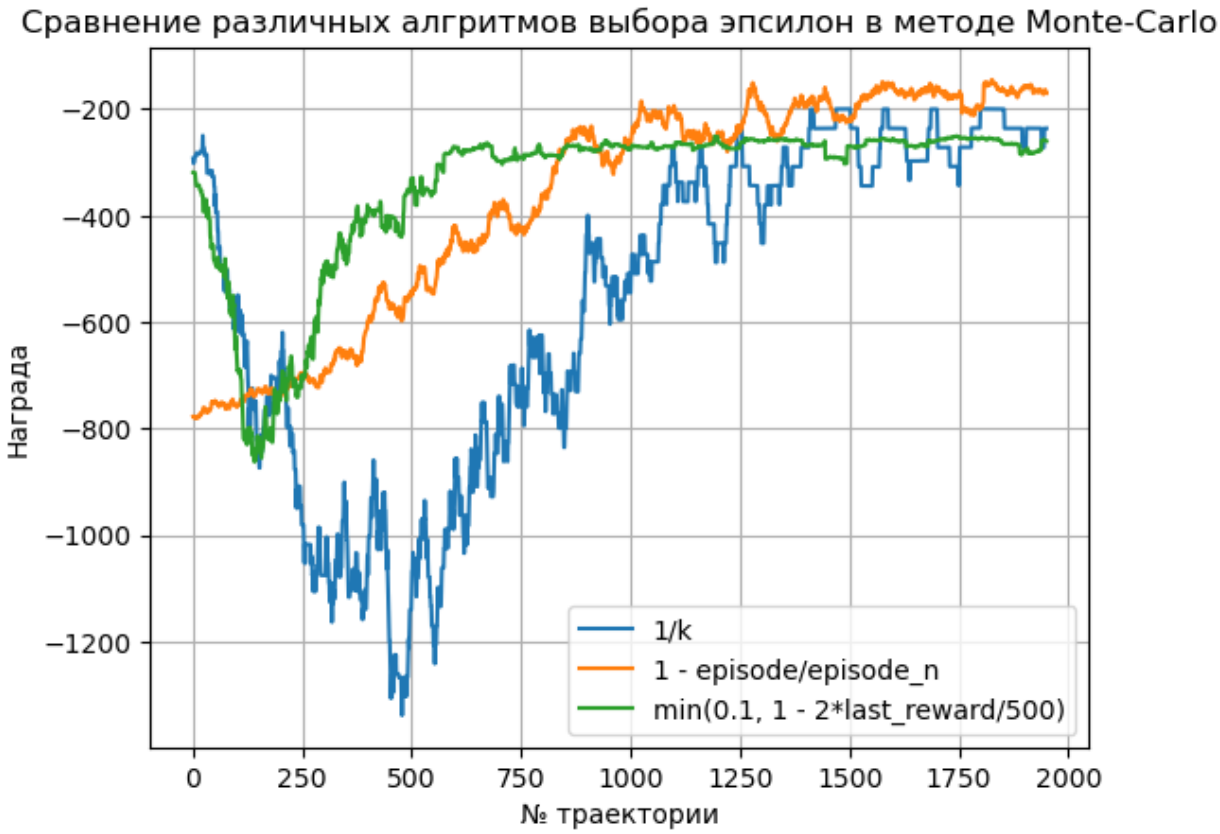
Лучше не стало.

Далее я попробовал применить что-то отдалённо схожее с экспоненциальным сглаживание. Epsilon = 0.1 + (1 – 0.1) * e^(-0.01*episode)

Ну лучше всё ещё не стало))

Следующая функция уже содержала в себе некоторую семантику. Логика – нужно ориентироваться на награду, т.е. при стремлении |Reward| -> max, epsilon -> min,

Epsilon = min(0.1, 1 - 2*last_reward/500)



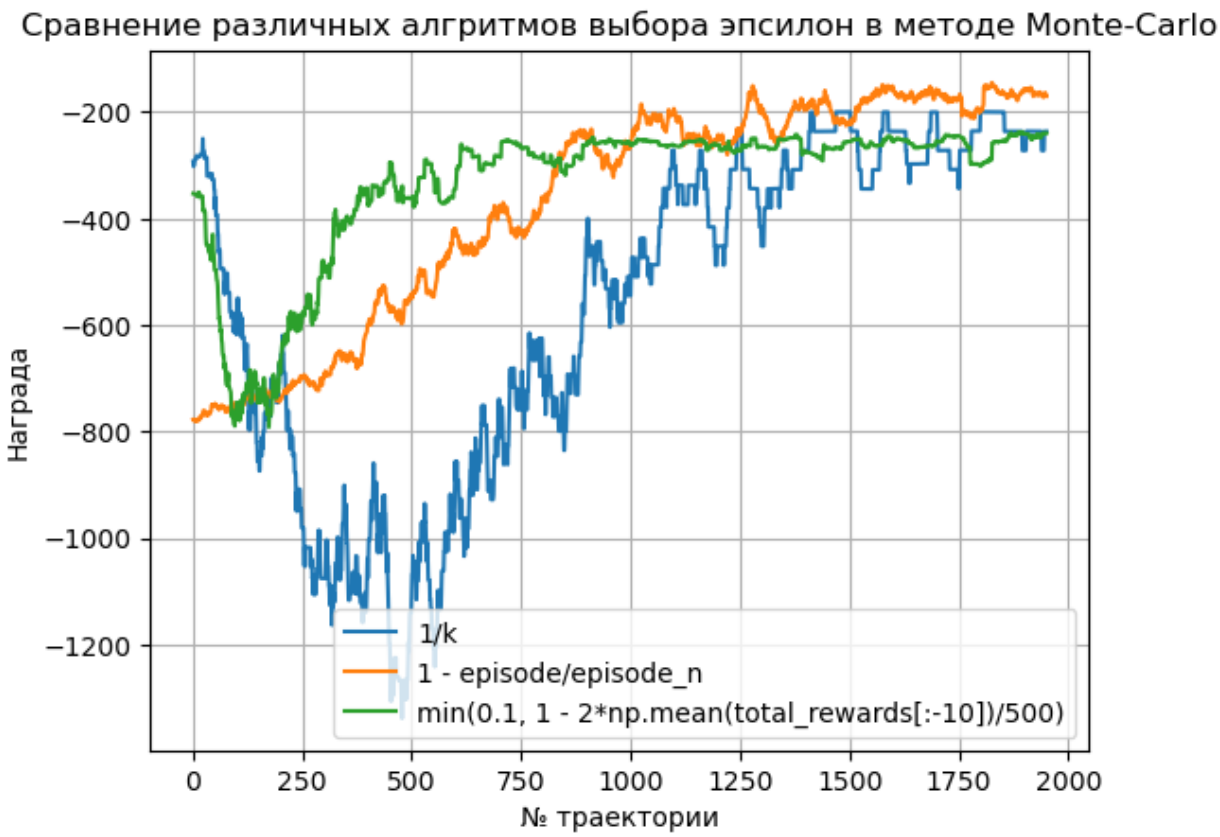
Всё ещё не лучше..

Далее я попросил помочь ChatGPT допилить функцию. Было предложено учитывать как награду, так и номер эпизода.

Epsilon = min(0.1, 1 - max(1 - episode / episode_n, 1 - 2 * last_reward / 500)

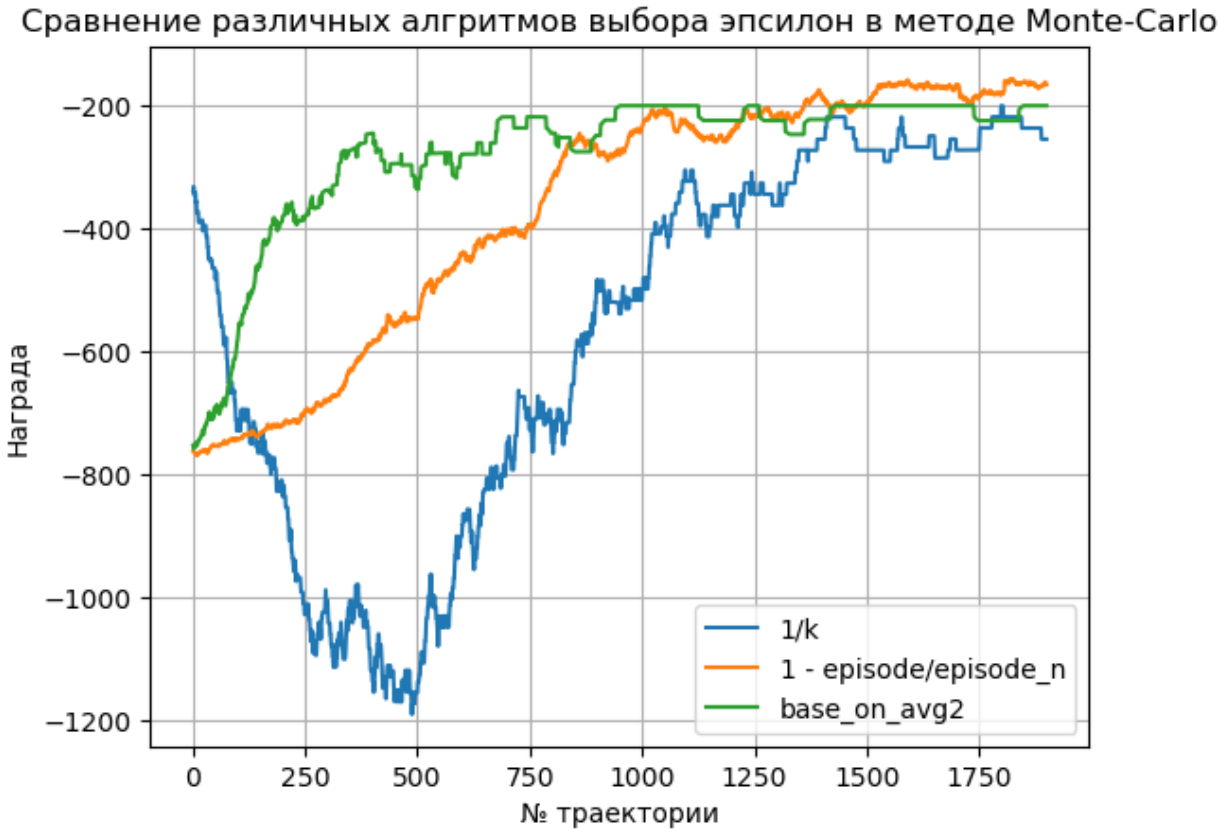
Вывод всё тот же))

Далее я вспомнил, что сам указывал на очень шумные результаты метода. Так что решил чуть-чуть улучшить свой вариант с выбором эпсилон по награде – брать среднее по последним 10 наградам.



Идея была многообещающей. Может по 100, как и с окном для графика…

Последняя идея, связанная со сравнением среднего последних 10 наград со средним всех наград. Если последние награды больше, то уменьшаем текущий эпсилон в 0.9, иначе увеличиваем в 1.1 раз.



Неудача… Далее, к сожалению, я не успел попробовать что-то ещё. Возможно нужно было попробовать как-то связать эпсилон непосредственно с семантикой задачи…