Домашнее задание 4

I. Taxi-v3

1. Введение

В данном эксперименте рассматриваются четыре алгоритма обучения с подкреплением: **Q-Learning, Cross-Entropy, Monte Carlo** и **SARSA**. Цель эксперимента — сравнить эффективность этих алгоритмов на задаче **Taxi-v3**. Мы будем сравнивать алгоритмы по средней награде, получаемой агентами, в зависимости от числа сгенерированных траекторий.

2. Описание эксперимента

2.1 Описание задачи (Taxi-v3)

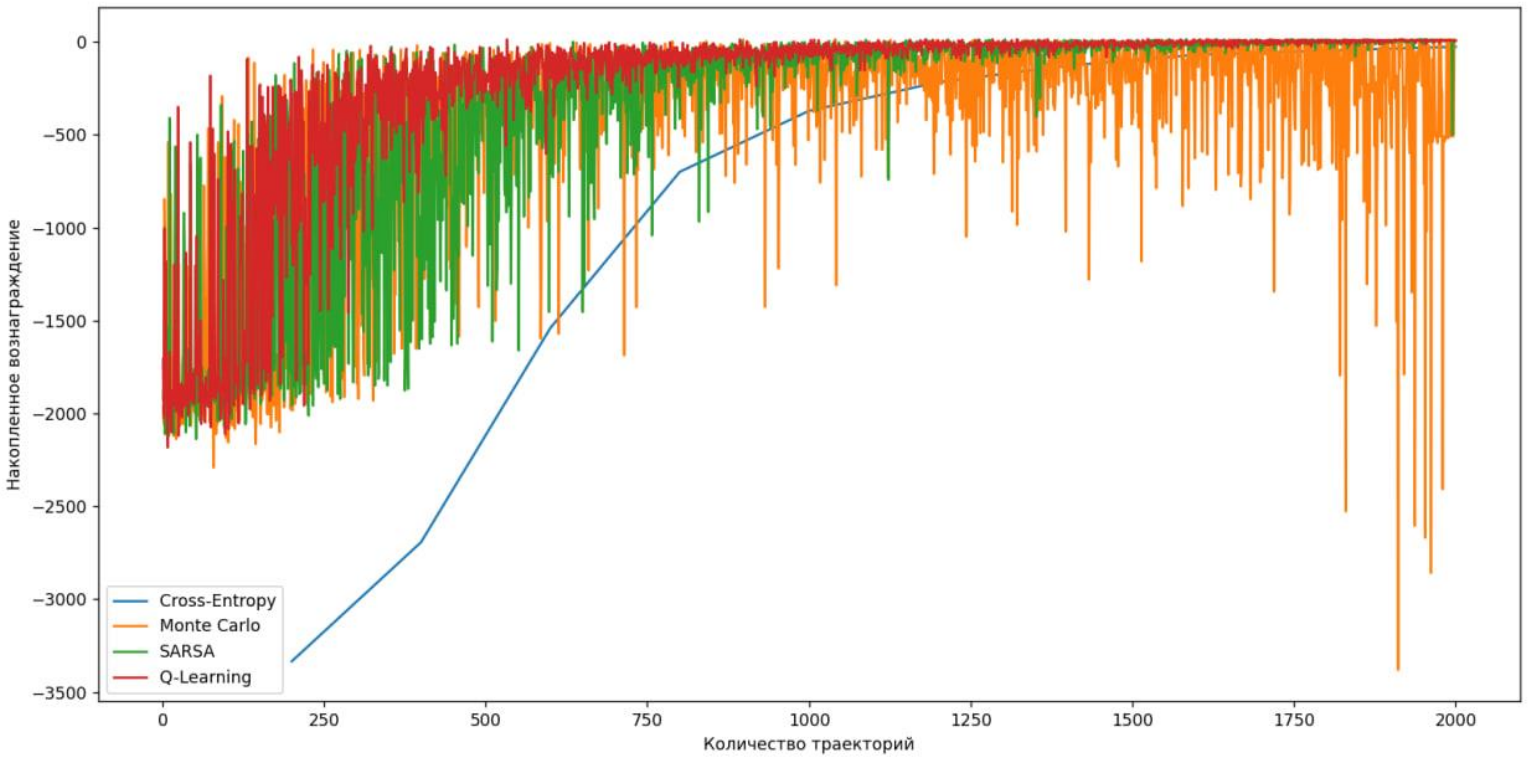
Задача Taxi-v3 является задачей, в которой агент (такси) должен перемещаться по сетке и забирать/высаживать пассажиров в разных точках. Задача состоит в оптимизации стратегии такси для максимизации награды. Награда зависит от правильности действий агента, включая забор и высаживание пассажиров в нужных местах, а также от использования минимального количества шагов.

2.2 Алгоритмы и методы

В рамках эксперимента были использованы следующие алгоритмы:

- **Q-Learning: off-policy** алгоритм, который обновляет свою политику независимо от текущей стратегии, что позволяет ему быстро улучшать свою производительность. Алгоритм использует функцию Q, которая оценивает полезность действия в каждом состоянии, и обновляется на основе максимальной возможной награды.
- **Cross-Entropy Method**: использует вероятностное распределение для поиска оптимальной политики, может привести к более гибкому поиску решения. Метод оптимизации, который ищет вероятность для каждой действия на основе текущей политики, обучая агента с помощью выборки лучших действий.
- **Monte Carlo: on-policy** метод, который обновляет свои оценки только после завершения эпизода, что может привести к медленному, но точному обучению.
- **SARSA: on-policy** алгоритм, схожий с Q-Learning, но обновляющий свои оценки в процессе выполнения действия согласно текущей политике.

3. Результаты



• **Q-Learning** демонстрирует быстрый рост награды, стабилизируясь на высоком уровне с увеличением числа траекторий, что подтверждает эффективность **off-policy** подхода в данной задаче. Это типично для алгоритма Q-Learning, так как со временем происходит накопление полезной информации о состоянии среды и эффективных действиях. Алгоритм быстро улучшает свои результаты, начиная с отрицательных значений награды и уверенно приближаясь к нейтральным или положительным.

• **Cross-Entropy** начинает с низкой награды, но постепенно улучшается. Алгоритм показывает стабильное улучшение среднего вознаграждения с увеличением количества траекторий. Использует подход с вероятностным выбором действий, что может привести к менее эффективному поведению в начале, но способствует более гибкому поиску оптимальной стратегии, хотя для этого требуется большое количество траекторий. Скорость обучения меньше, чем у Q-Learning.

• **Monte Carlo** демонстрирует тренд улучшения среднего вознаграждения по мере увеличения числа траекторий, хотя улучшения происходят неравномерно. В начале видно, что значения среднего вознаграждения крайне низкие и варьируются в широком диапазоне. Особенность: **On-policy** метод - стратегия обновляется на основе текущих действий, тем самым затрудняя быстрое исследование пространства решений. Требует больше времени для стабилизации и достижения высокой награды, что может быть ограничением для задач, где требуется быстрое обучение, как у **Q-Learning**.

• **SARSA:** Показывает результаты, аналогичные Q-Learning, но с некоторыми отклонениями; его более осторожный подход делает его менее эффективным в ранних стадиях обучения. В отличие от Q-Learning, он обновляет свою стратегию на основе текущих действий, что делает его более осторожным в исследовании.

II. LunarLander-v2

1. Введение

В данном эксперименте мы сравнили эффективность четырех алгоритмов обучения с подкреплением — **Monte Carlo, SARSA, Q-Learning и Deep CEM** — в решении задачи **LunarLander-v2**. Целью было определить, какой из алгоритмов лучше справляется с задачей, и оценить различия в скорости их обучения и стабильности награды. Сравнение проводилось **по количеству траекторий**, необходимому для достижения высокой средней награды.

2. Описание эксперимента

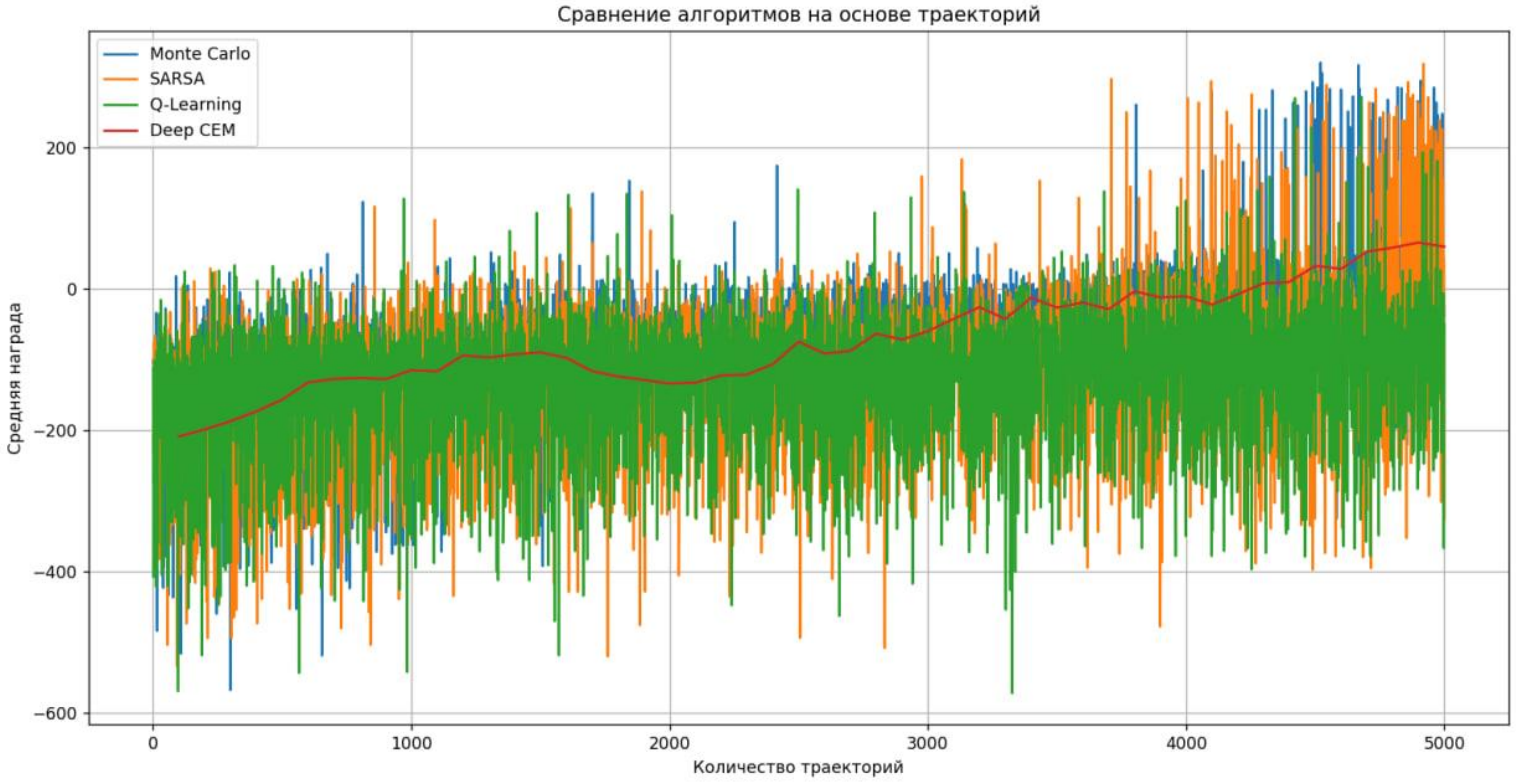
Дискретизация пространства состояний

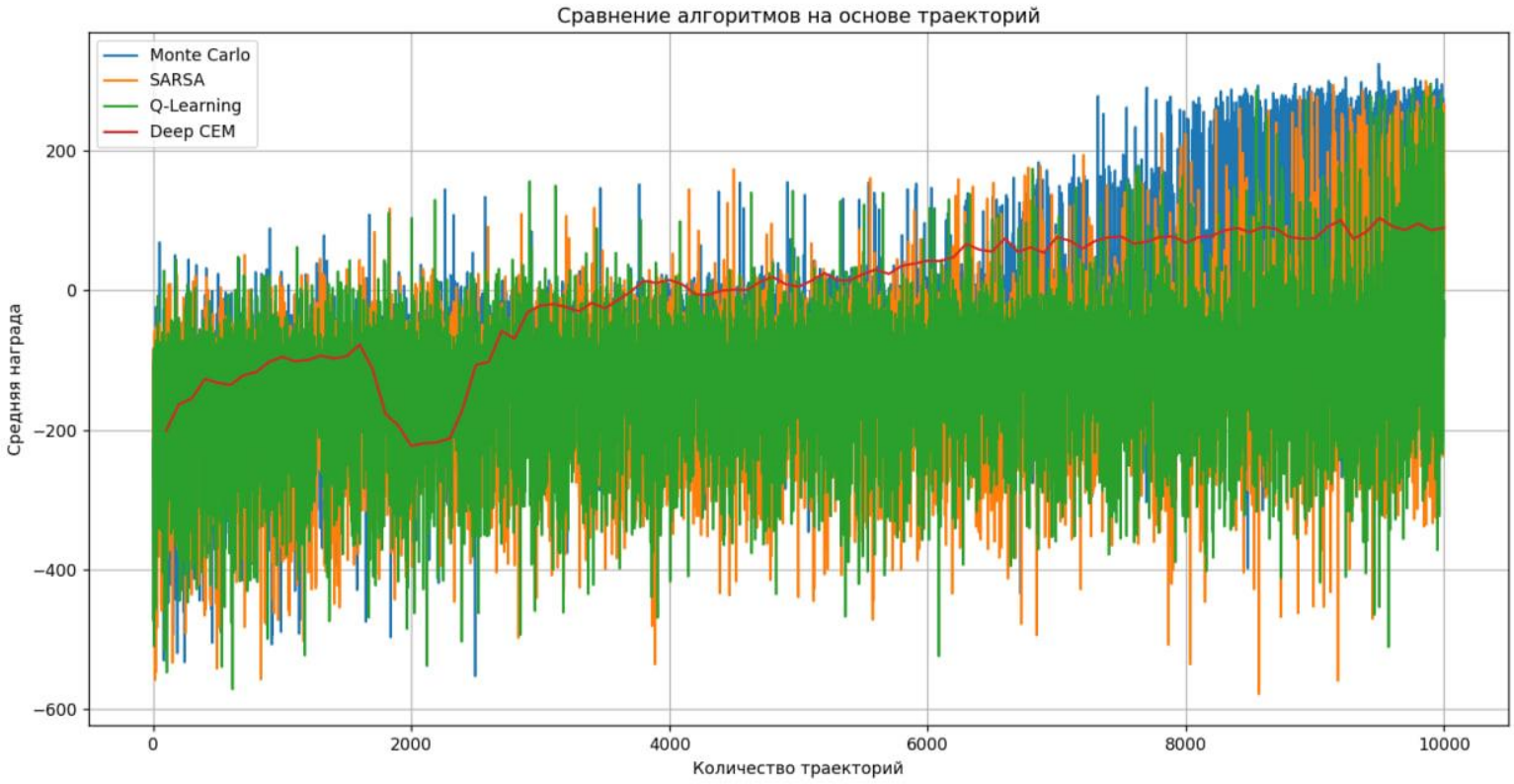
Задача LunarLander-v2 имеет непрерывное пространство состояний размерностью 8, включающее:

- положение по горизонтали и вертикали,
- горизонтальную и вертикальную скорости,
- угол наклона и угловую скорость,
- наличие контакта с землей для каждой из ног.

Для упрощения обучения и адаптации алгоритмов Monte Carlo, SARSA и Q-Learning, которые плохо работают в непрерывном пространстве состояний, мы дискретизировали это пространство. Мы разбили каждую из 8 составляющих состояния на ограниченное количество **"корзин" (buckets)**. Например, горизонтальное и вертикальное положение были дискретизированы в 5 "корзин" каждая, угловая скорость — в 2 "корзины". Это позволяет преобразовать каждое состояние в дискретное представление, что упрощает вычисления и ускоряет обучение.

3. Результаты





Monte Carlo, SARSA и Q-Learning: Эти алгоритмы достигли значений средней награды выше 100 уже на ранних этапах, несмотря на сильные колебания. Колебания обусловлены тем, что каждый алгоритм обучается на

основе оценки награды, полученной за каждый эпизод, что приводит к высокому уровню случайности. Однако, несмотря на это, все три алгоритма демонстрируют уверенный рост средней награды, что свидетельствует о высоком качестве обучения.

Deep CEM: Этот метод показал более плавное и стабильное улучшение награды, но потребовал большее количество траекторий для достижения удовлетворительных результатов. Он не превзошел отметку в 100 баллов, что делает его менее эффективным для данной задачи по сравнению с другими методами. Плавность обучения Deep CEM обусловлена тем, что он использует элитные траектории для обновления параметров, что снижает случайность, но увеличивает количество необходимых данных для обучения.

Все рассмотренные алгоритмы достаточно хорошо справляются с решением задачи и способны достичь высокой награды. Выбор подходящего алгоритма зависит от предпочтений между скоростью обучения и его стабильностью.

III. Стратегии для выбора epsilon

Для оптимизации работы алгоритма Monte Carlo в задаче Taxi-v3 важно правильно подобрать стратегию для выбора значения параметра ε (эпсилон), который определяет степень исследования (exploration) и эксплуатации (exploitation). Эпсилон-гриди стратегия помогает агенту изучать окружающую среду (выбирая случайные действия) и, по мере обучения, сосредотачиваться на лучших действиях (следуя политикам на основе Q-значений).

Для задачи Taxi-v3 можно предложить несколько стратегий изменения ε в процессе обучения, чтобы достичь наилучших результатов:

1. Экспоненциальное уменьшение ε (ε decay): Один из подходов — использовать экспоненциальное уменьшение значения ε, что позволяет на начальных этапах лучше исследовать среду, а затем постепенно уменьшать ε, чтобы агент использовал накопленные знания для улучшения награды, т.е. быстро уменьшается в начале и замедляет снижение в дальнейшем, что способствует интенсивному исследованию на начальных этапах и фокусируется на эксплуатации позже.

```
epsilon = epsilon_min + (epsilon_max - epsilon_min) * np.exp(-decay_rate * episode)
```

- epsilon_min — минимальное значение ε (например, 0.01),
- epsilon_max — начальное значение ε (например, 1.0),
- decay_rate — скорость уменьшения (например, 0.005).

2. Линейное уменьшение ε: Линейное уменьшение подходит для простых задач и позволяет плавно переходить от исследования к эксплуатации.

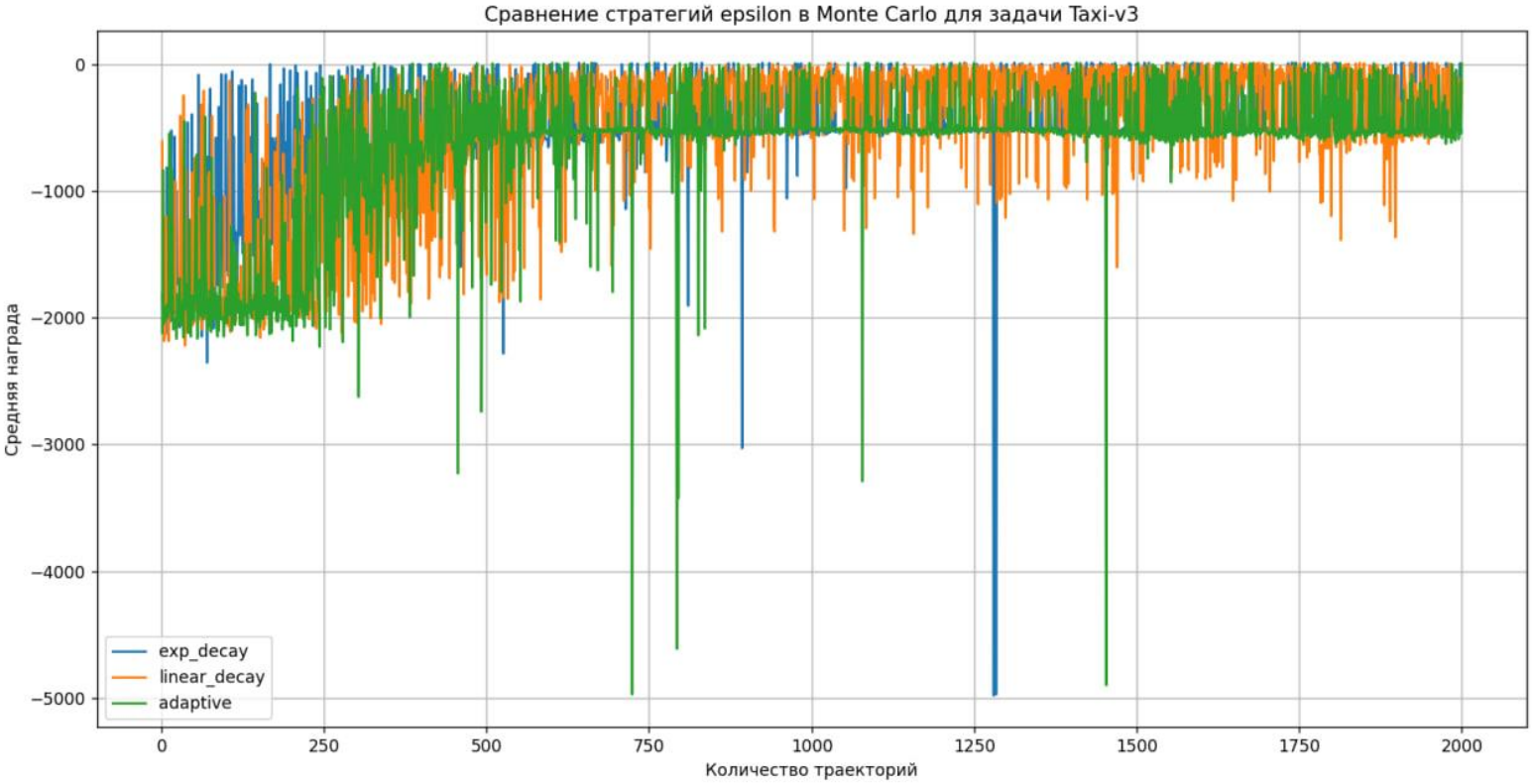
epsilon = max(epsilon_min, epsilon_start - episode * epsilon_decay)

- epsilon_start — начальное значение ε (например, 1.0),
- epsilon_decay — скорость линейного уменьшения (например, epsilon_start / episode_n),
- epsilon_min — минимальное значение ε (например, 0.01).

3. Адаптивное уменьшение ε на основе успехов агента: В этой стратегии ε уменьшается быстрее, если агент успешно выполняет задания. Если на протяжении определенного количества эпизодов (например, 100) средняя награда растет, ε уменьшается. В случае снижения награды — ε может снова немного увеличиться, что стимулирует исследование.

```
if episode > 100:
    recent_rewards = total_rewards[-100:]
    if np.mean(recent_rewards) > np.mean(total_rewards[-200:-100]):
        epsilon *= 0.99  # Уменьшаем `ε` на 1% при успехе
    else:
        epsilon = min(epsilon_max, epsilon * 1.01)  # Увеличиваем на 1% при снижении
```

Результаты:



- **Exp Decay (Экспоненциальное уменьшение)** – ε постепенно уменьшается с экспоненциальной скоростью. Результаты выглядят стабильно, но на некоторых отрезках все же наблюдаются резкие падения средней награды, что указывает на недостаточную устойчивость в отдельных траекториях.
- **Linear Decay (Линейное уменьшение)** – ε линейно уменьшается, и эта стратегия выглядит наиболее стабильной. Колебания награды меньше, и агент быстро достигает высокой стабильной награды.
- **Adaptive (Адаптивное уменьшение)** – ε меняется адаптивно, в зависимости от успехов агента. На начальных этапах обучения наблюдается больше колебаний, так как агент динамически подстраивает ε. К концу обучения, однако, результат стабилизируется, и награда оказывается сопоставимой с остальными стратегиями.