# Monte-Carlo, SARSA, QLearning. Отчет

## Содержание

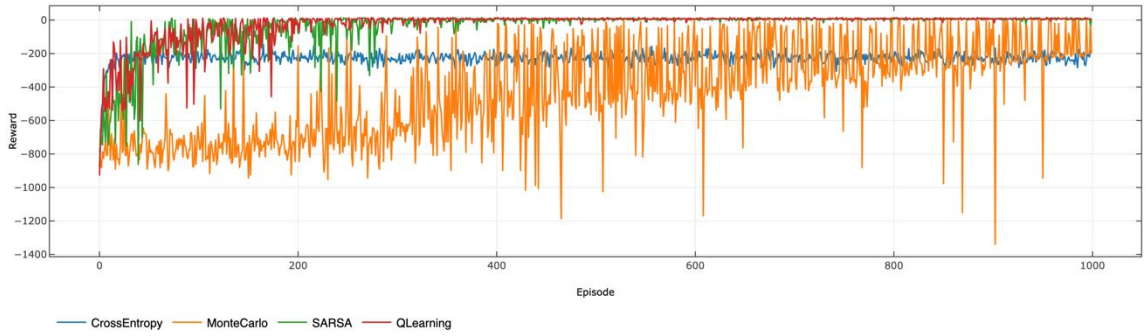
1. Реализовать Q-Learning и сравнить его результаты с реализованными ранее алгоритмами: Cross-Entropy, Monte Carlo, SARSA в задаче Taxi-v3. Для сравнения как минимум нужно использовать графики обучения. Причем графики лучше делать относительно количества сгенерированных траекторий.

Результаты можно посмотреть в [ClearML]

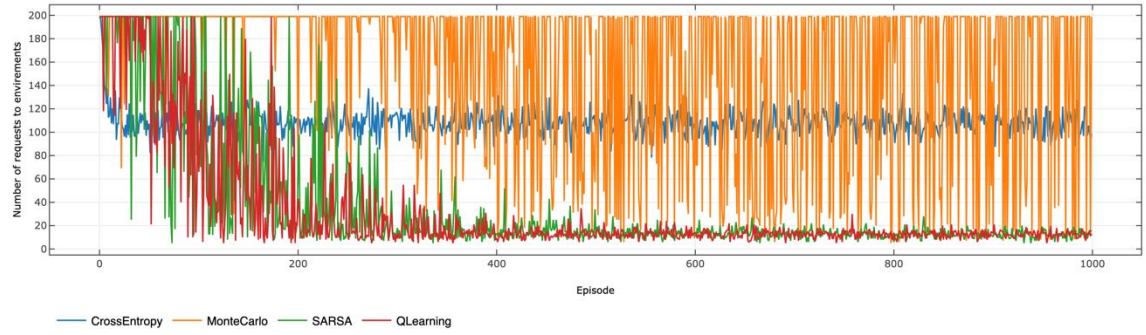
**Иследуемые алгоритмы:**

```
ALGORITHMS = {
    'CrossEntropy': dict(env=ENV, episode_n=1000, q_params=0.6, trajectory_len=100),
    'MonteCarlo': dict(env=ENV, episode_n=1000, trajectory_len=1000, gamma=0.99),
    'SARSA': dict(env=ENV, episode_n=1000, trajectory_len=1000, gamma=0.999, alpha=0.5),
    'QLearning': dict(env=ENV, episode_n=1000, trajectory_len=1000, gamma=0.999, alpha=0.5)
}
```

*Reward на итерациях обучения:*




*Кол-во обращений к среде на итерациях обучения:*




Вывод:

Лучшие результаты продемонстрировали алгоритмы SARSA и QLearning, как по reward, так и по кол-ву обращений к среде на итерациях обучения.

2. Дискретизировать (можно использовать numpy.round()) пространство состояний и обучить Агента решать CartPole-v1, Acrobot-v1, MountainCar-v0, или LunarLander-v2 (одну на выбор) методами Monte Carlo, SARSA и Q-Learning. Сравнить результаты этих алгоритмов и реализованного ранее алгоритма Deep Cross-Entropy на графиках.

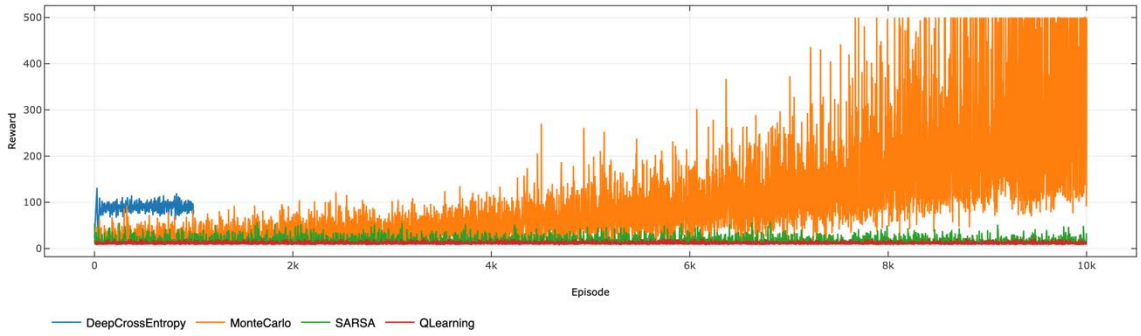
Была выбрана среда **CartPole-v1**.

Результаты можно посмотреть в [ClearML]

**Иследуемые алгоритмы:**

```
ALGORITHMS = {
    'DeepCrossEntropy': dict(env=ENV, lr=1e-2, episode_n=1000, q_params=0.8, trajectory_len=100),
    'MonteCarlo': dict(env=ENV, episode_n=10000, trajectory_len=1000, gamma=0.99),
    'SARSA': dict(env=ENV, episode_n=10000, trajectory_len=1000, gamma=0.999, alpha=0.5),
    'QLearning': dict(env=ENV, episode_n=10000, trajectory_len=1000, gamma=0.999, alpha=0.5)
}
```

*График reward на итерациях обучения:*




Так как в данной среде ревард – это кол-во действий, то график обращений к среде его полностью повторяет.

Вывод:
Лучшие результаты были продемонстрированы алгоритмом Monte-Carlo. Алгоритмы SARSA и QLearning так и не вышли на оптимальный уровень работы.

3. Придумать стратегию для выбора epsilon позволяющую агенту наилучшим образом решать Taxi-v3 алгоритмом Monte Carlo

Входе эксперимента был исследован алгоритм Monte-Carlo с различными стратегиями epsilon.

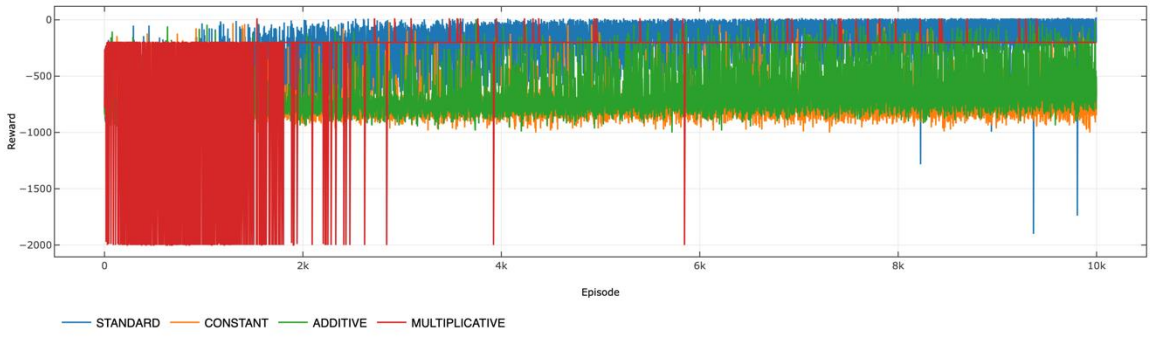
Результаты можно посмотреть в ClearML

**Стратегии epsilon:**

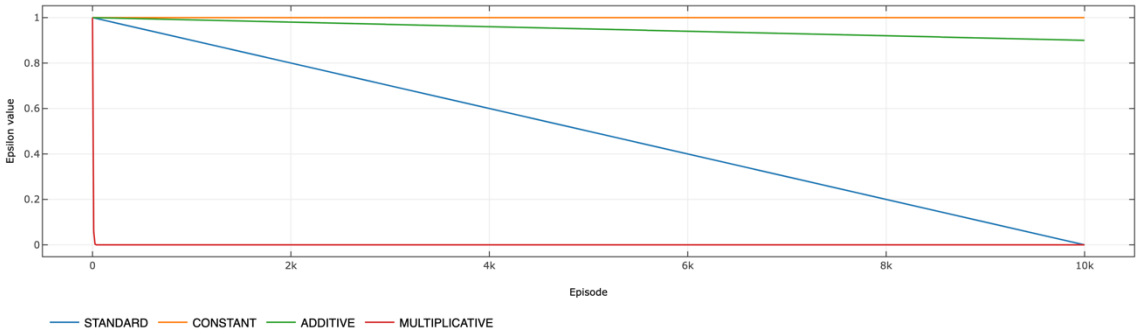
```
EPSILON_STRATEGIES = {
   'STANDARD': dict(env=ENV, episode_n=10000, trajectory_len=1000, gamma=0.99,
epsilone_strategy="STANDARD", constant=None),
   'CONSTANT': dict(env=ENV, episode_n=10000, trajectory_len=1000, gamma=0.99,
epsilone_strategy="CONSTANT", constant=1),
   'ADDITIVE': dict(env=ENV, episode_n=10000, trajectory_len=1000, gamma=0.99, epsilone_strategy="ADDITIVE",
constant=0.1),
   'MULTIPLICATIVE': dict(env=ENV, episode_n=10000, trajectory_len=1000, gamma=0.99,
epsilone_strategy="MULTIPLICATIVE", constant=0.8),
}
```

```
{
            "STANDARD": epsilon = 1 - episode / episode_n
            "CONSTANT": epsilon = constant
            "ADDITIVE": (
            if epsilon - constant/episode_n > 0:
              epsilon -= constant/episode_n
            )
            "MULTIPLICATIVE": epsilon *= constant
```

*График reward на итерациях обучения:*



*Значения epsilon на итерациях обучения:*




Вывод:

Из рассмотренных стратегий epsilon самой оптимальной является стандартная (фактически она линейная), которая была предложена на лекциях, так как при такой стратегии награда на итерациях обучения достигает лучших результатов, однако на итерациях обучения у такого вида стратегии есть достаточно большая дисперсия в отличии от мультипликативной стратегии