sutton-barto-reinforcement-exercises-ch-2

September 1, 2018

2.1:

When using an ϵ-greedy action selection criterion, in the case of two actions and $\epsilon = 0.5$, actions would be selected at random half of the time. Of the times when actions are selected at random, the greedy action will be select half of the time. Meaning the greedy action will be selected a total of 75% of the time.\

2.2:

Given that all of the reward estimates are initialized to 0, the first action will be selected arbitrarily, possible at random.

The following actions must have been selected at random as a result of the ϵ case: A_2, A_5.

This can be demonstrated by observing that $Q_1(1) = 1 > Q_1(2) = 0$ and $Q_4(2) = \frac{R_2+R_3+R_4}{3} = \frac{5}{3} > Q_4(3) = 0$.

Regarding the question of when random ϵ selection could have occured, it could have occured at any time step after the first.

2.3:

In the long run $\epsilon = 0.01$ will achieve the highest cumulative reward and probability of selecting the correct answer. The disadvantage of $\epsilon = 0.01$ relative to $\epsilon = 0.1$ is that it will cause our agent to take longer to find the optimal action, but it will select that action more consistently once it has been found.

The optimal action has a reward of 1.55. The expected reward per action once the agent has found the best possible action is equal to the sum of the greedy case and the ϵ random case: $1.55(1 - \epsilon) + \epsilon \sum_{i=1}^{10} \frac{q_*(i)}{10}$.

So the difference between the expected long term reward per action in the $\epsilon = .01$ and $\epsilon = .1$ cases is $1.55(.1 - .01) = 0.1395$.

2.4:

If α is not constant, but instead varies with with each update step n, then our incremental update rule utilizing α_n can be expressed as follows:

$$\begin{aligned}
Q_{n+1} &= Q_n + \alpha_n[R_n - Q_n] && (1)\\
&= \alpha_n R_n + (1-\alpha_n)Q_n && (2)\\
&= \alpha_n R_n + (1-\alpha_n)(\alpha_{n-1}R_{n-1} + (1-\alpha_{n-1})Q_{n-1}) && (3)\\
&= \alpha_n R_n + (1-\alpha_n)\alpha_{n-1}R_{n-1} + (1-\alpha_n)(1-\alpha_{n-1})(Q_{n-2} + \alpha_{n-2}[R_{n-2} - Q_{n-2}]) && (4)\\
&= \alpha_n R_n + (1-\alpha_n)\alpha_{n-1}R_{n-1} + (1-\alpha_n)(1-\alpha_{n-1})((1-\alpha_{n-2})Q_{n-2} + \alpha_{n-2}R_{n-2})) && (5)\\
&= \alpha_n R_n + (1-\alpha_n)\alpha_{n-1}R_{n-1} + (1-\alpha_n)(1-\alpha_{n-1})\alpha_{n-2}R_{n-2} + && (6)\\
&\dots + Q_1\prod_{i=1}^{n}(1-\alpha_i) + \alpha_1 R_1 \prod_{j=1}^{n-1}(1-\alpha_{n-j+1}) && (7)\\
&= Q_1\prod_{i=1}^{n}(1-\alpha_i) + \alpha_n R_n + \sum_{i=1}^{n-1}\alpha_i R_i \prod_{j=1}^{n-i}(1-\alpha_{n-j+1}) && (8)
\end{aligned}$$

2.5:

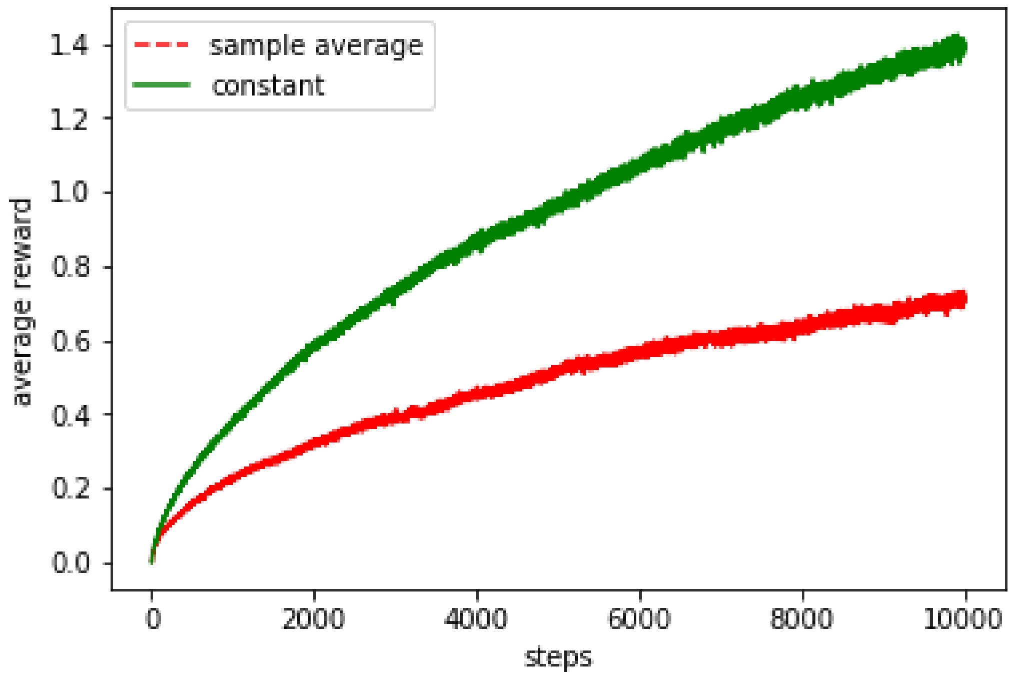
```
In [146]: import numpy as np
          import random
          import numpy.random
          import matplotlib.pyplot as plt

          # EXERCISE 2.5

          steps = 10000
          runs = 2000

          """
          returns data in the format

          {
              "rewards": np.array([1.5, .2, ...]),
              "optimal": np.array([1, 0, ...])
          }
          """
          def ten_armed_bandit(sample_average = False):
              q = np.zeros(10)
              Q = np.zeros(10)

              rewards = []
              optimal = []

              for i in range(steps):
                  arg_max_reward = None
                  arg_max_actions = []

                  for j in range(len(Q)):
```

```
                estimated_reward = Q[j]

                if arg_max_reward == None or estimated_reward > arg_max_reward:
                    arg_max_reward = estimated_reward
                    arg_max_actions = [j]

                elif estimated_reward == arg_max_reward:
                    arg_max_actions.append(j)

            if random.uniform(0, 1) < .1:
                selected_action = random.choice(range(10))
            else:
                selected_action = random.choice(arg_max_actions)

            actual_reward = q[selected_action]
            alpha = 1/(i + 1) if sample_average else .1
            Q[selected_action] = Q[selected_action] + alpha * (actual_reward -
    Q[selected_action])

            was_action_optimal = 1 if actual_reward == max(q) else 0

            rewards.append(actual_reward)
            optimal.append(was_action_optimal)

            random_walk_values = np.random.normal(0, .01, len(q))

            for j in range(len(q)):
                q[j] = q[j] + random_walk_values[j]

        return {
            "rewards": np.array(rewards),
            "optimal": np.array(optimal)
        }


    def average_bandit_results(sample_average = False):
        rewards = np.zeros(steps)
        optimal = np.zeros(steps)

        for i in range(runs):
            results = ten_armed_bandit(sample_average)
            rewards = rewards + (results["rewards"] / runs)
            optimal = optimal + (results["optimal"] / runs)

        return {
            "rewards": rewards,
            "optimal": optimal
        }

    def plot_average_rewards(with_sample_average, without_sample_average):
        plt.plot(with_sample_average, "r--", label = "sample average")
        plt.plot(without_sample_average, "g", label = "constant")
        plt.xlabel("steps")
        plt.ylabel("average reward")
        plt.legend()
        plt.show()

    def plot_average_optimality(with_sample_average, without_sample_average):
        plt.plot(with_sample_average, "r--", label = "sample average")
        plt.plot(without_sample_average, "g", label = "constant")
        plt.xlabel("steps")
        plt.ylabel("percent optimal")
        plt.legend()
        plt.show()
```

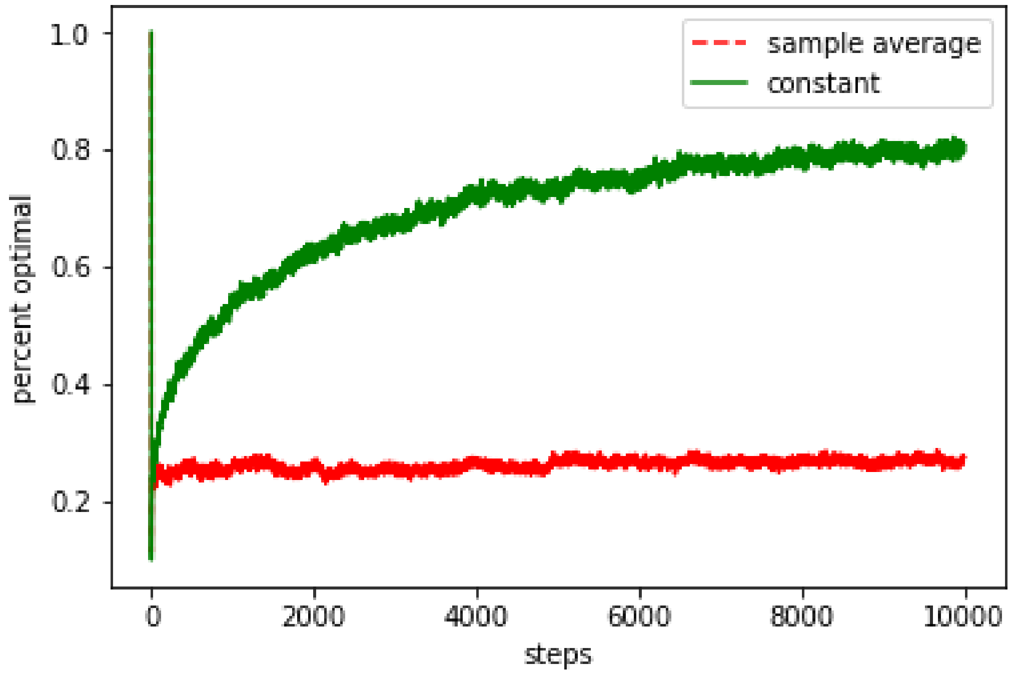
```
In [78]: %matplotlib inline
```

```
In [147]: with_sample_average = average_bandit_results(True)
In [148]: without_sample_average = average_bandit_results()
```

```
In [149]: plot_average_rewards(with_sample_average["rewards"], without_sample_average["rewards"])
```



```
In [150]: plot_average_optimality(with_sample_average["optimal"],
          without_sample_average["optimal"])
```



2.6:

The optimistic greedy agent's initial spike in rewards can be explained as follows.

$q_*(a)$ is normally distributed with a mean of 0 and a variance of 1. This means $Q_1 > max\{q_*(a)\}$ in the vast majority of cases.

For the reward distributions where the above is true, agents with the optimistic greedy action-value method will attempt every action at least once at the onset.

The spike we observe is a product of agents selecting the optimal action once during this preliminary period.

2.7:

The softmax distribution with two actions consist of,

$$\pi_t(a) = \frac{e^{H_t(a)}}{e^{H_t(1)} + e^{H_t(2)}}.$$

Logistic classifiers are binary classifiers. In this case let $L(a)$ represent the probability assigned by a logistic classifier of selecting a, where $a \in \{1, 2\}$. $L(2)$ is formulated as,

$$L(2) = \frac{1}{1 + e^{-f(2)}}$$

where $f(a)$ is a function which accepts an action and outputs a preference score. In the case of the softmax distribution, the probability of selection action $a = 2$ is,

$$\tag{9}$$

$$\pi_t(2) = \frac{e^{H_t(2)}}{e^{H_t(1)} + e^{H_t(2)}} \tag{10}$$

$$= \frac{1}{1 + e^{H_t(1) - H_t(2)}}. \tag{11}$$

$$\tag{12}$$

$$\tag{13}$$

Now let $f(a)$ be a function which outputs a preference score such that $f(2) = H_t(2) - H_t(1)$, so $\pi_t(2)$ can be re-expressed as,

$$\pi_t(2) = \frac{1}{1 + e^{-f_t(2)}} = L(2).$$

Both $L(a)$ and $\pi(a)$ are normalized across a, so if $L(2) = \pi_t(2)$ it must also be the case that $L(1) = \pi_t(1)$, ensuring that the two distributions are the same.