MULTI-ARMED BANDITS

Saber Bahranifard

September 28, 2020

Abstract

In this report, the goal is to study highlights of the ”evaluative” nature of Reinforcement Learning (RL) which refers to the fact that each action is ”evaluated” based on the training information with a reward or penalty assigned to the action, accordingly. We’ are covering just a simplified version of the RL problem in which there is just one situation to respond to. An example of this so-called ”non-associative” setting is the Multi-armed Bandit problem in which the user makes one choice among K different Bandit Machines. Each choice is rewarded from a ”stationary reward distribution” and the objective is to maximize the total reward over time. The user repeats this process with possibly different action for a predefined number of times (steps per episode). After setting up the Bandit problem, the rest of this report focuses on different policies and evaluate them by the expected reward per step. In fact, the emphasis is on finding a good balance between ”exploring” and ”exploiting” [1]

1 Introduction

Policies map actions to states. That is, a policy tells us what actions to take next. The policies we experiment in this report try to maximize the expected reward, $Q_t(a)$ by compromising "exploration" and "exploitation". The algorithms we consider are:

Greedy action selection: explores at the beginning and as it finds a rewarding pattern, it exploits its knowledge and tries to maximize the "immediate" reward by repeating that pattern. For deterministic problems with fixed reward per action this approach works well. Performs poorly on non-stationary problems where action-value is changing over time (step).

ϵ-greedy action selection: at each step, with ϵ chance, the agent chooses to explore a new action while with (1-ϵ) chance it considers a greedy approach.

In the short run, greedy algorithms perform better as ϵ-greedy might face more penalties, but in the long run, it outperforms the greedy approach.

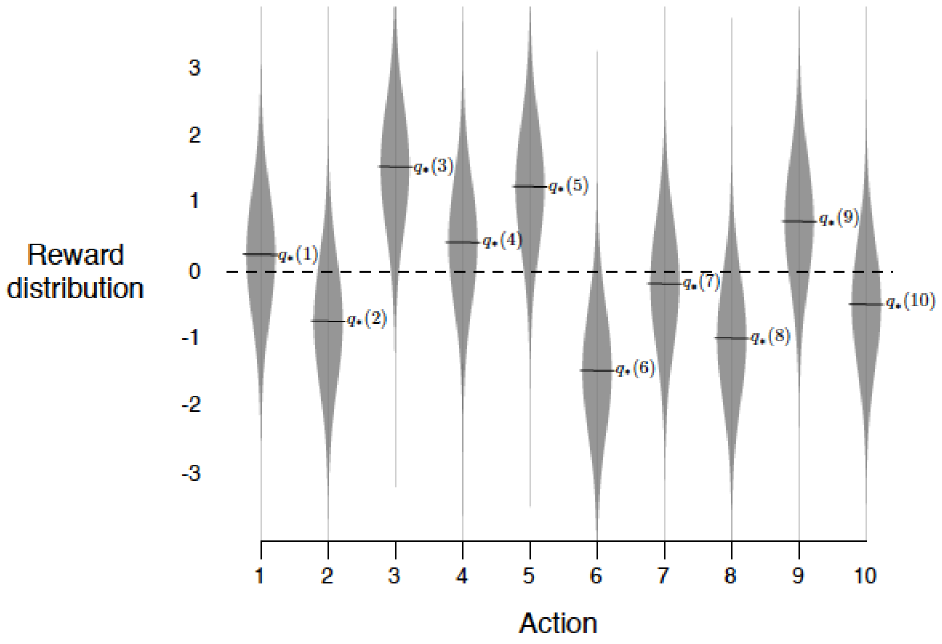
Similar to a Bayesian approach, setting initial values for action-value estimates force the learning agent to keep looking for better values. That is, by biasing the initial estimates, we can force the averaging approach to keep looking for values which might be out of its reach. This trick works well on stationary problems in which the conditions are not changing over time. "The beginning of time occurs only once and thus we shouldn't focus on it too much" [1].

In this report, we try to compare outcomes of the above mentioned approaches. Thus we've setup two different experiments:

- ϵ-Greedy policy with $\epsilon = [0.0, 0.1, 0.01]$ and $Q_0(a)$ initialized to 0
- ϵ-Greedy policy with $\epsilon = [0.0, 0.1, 0.01]$ and $Q_0(a)$ initialized to 5. To pick this initial value to initialize estimated action-value, the only concern was to make it bigger than average optimal value per step, which in this experiment was 1.54.

2 Expriments

The testbed created for these set of experiments consists of 1000 steps (iterations) per episode. Each episode starts with initializing expected action values, $Q_0(a)$, and then at each further step, we sample from a Normal distribution with mean equal to the true action-value, $q_*(a)$, and unit variance. The agent doesn't know the true action-value, $q_*(a)$, so the goal is to find an expected value of $q_*(a)$, and try to maximize this expectation. The true action-value, $q_*(a)$, is also a Normal distribution with zero mean and unit variance.



To do a computationally inexpensive averaging to calculate action-value estimates a running average algorithm is used. The running average method computes the estimated average value of a given action at current step based on the estimated average value of the same action at previous step plus a time-decaying weighted difference of the reward and averaged value of that action at the previous step, $Q_{n+1} = Q_n + \frac{1}{n}[R_n - Q_n]$.

A single run (episode) of the ϵ-greedy algorithm used to calculate expected action value and take an action accordingly is as below [1]:

A simple bandit algorithm

```
Initialize, for $a = 1$ to $k$:
    $Q(a) \leftarrow 0$
    $N(a) \leftarrow 0$

Repeat forever:
    $A \leftarrow \begin{cases} \arg\max_a Q(a) & \text{with probability } 1-\varepsilon \quad \text{(breaking ties randomly)} \\ \text{a random action} & \text{with probability } \varepsilon \end{cases}$
    $R \leftarrow bandit(A)$
    $N(A) \leftarrow N(A) + 1$
    $Q(A) \leftarrow Q(A) + \frac{1}{N(A)}\big[R - Q(A)\big]$
```

2000 number of episodes per experiment is considered for this testbed. Then, values of expected reward are averaged over all the episodes.

2.1 Experiment 1

For this experiment we employ a ϵ-Greedy policy with three different values for $\epsilon = [0.0, 0.1, 0.01]$. $Q_0(a)$ is initialized to 0 for all three policies.

- The greedy algorithm ($\epsilon = 0.0$) starts to optimize the results faster, as shown in *Figure 1-top*, but later its Average reward plot levels off. This is mainly because it doesn't explore for more rewards. As can be seen from *Figure 1-bottom*, the greedy policy chooses the optimal action for slightly more than 30% of the time.

- The ϵ-greedy algorithm ($\epsilon = 0.1$) explores more and finds optimal values more frequently, as shown in *Figure 1-top*. As can be seen from *Figure 1-bottom*, this policy chooses the optimal action more frequently than the greedy policy. However, if the code is running for longer steps, the diagram shows that ϵ-greedy algorithm with $\epsilon = 0.1$ can't select optimal action for more than 90% of the times.

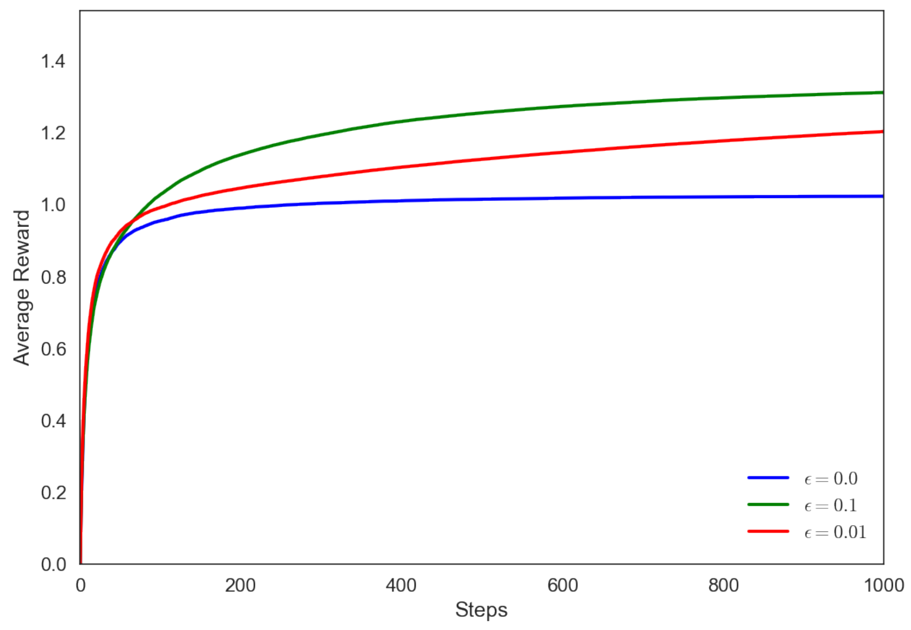
- The ϵ-greedy algorithm ($\epsilon = 0.01$) explores with some frequency between the two policies mentioned above. It takes some time for this policy to converge to optimal value, as shown in *Figure 1-top*, but it eventually passes the ϵ-greedy algorithm with $\epsilon = 0.1$. This is mainly because, as can be seen from *Figure 1-bottom*, this policy chooses the optimal action more frequently than other two policies. Running the code for longer steps, result in finding optimal action for more than 99% of the times.

2.2 Experiment 2

Similar to the former experiment, in this experiment we employ a ϵ-Greedy policy with three different values for $\epsilon = [0.0, 0.1, 0.01]$, but this time $Q_0(a)$ is initialized to 5.0 for all three policies. We are to measure the performance of these policy under the initial state conditions. As is evident from *Figure 2*, this initial conditions motivates the greedy algorithm to catch up with others initially, but later in the process, it looses its momentum as the effect of early rewards kicks out.

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction.* MIT press, 2018.



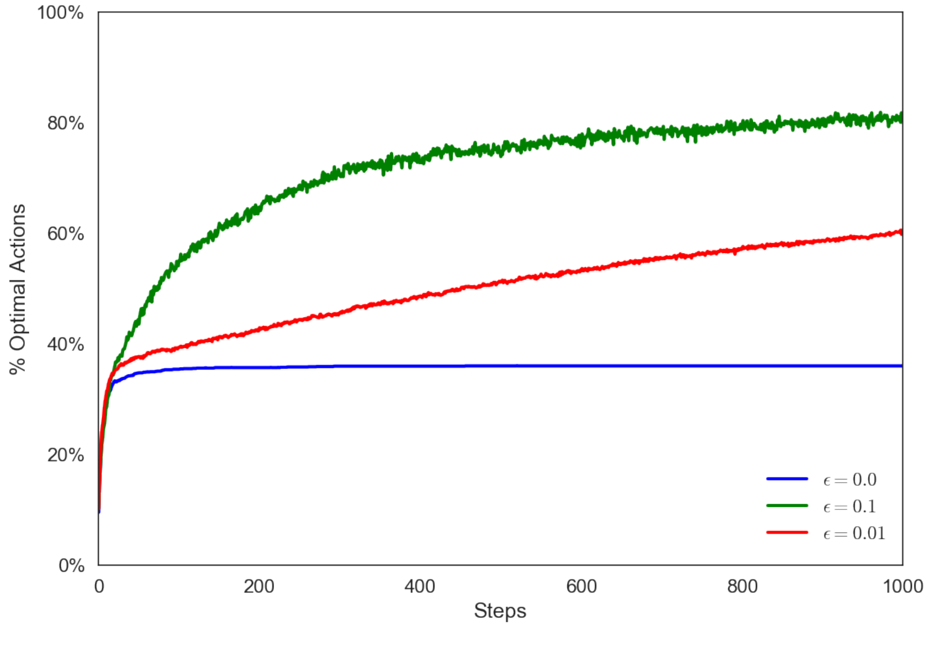
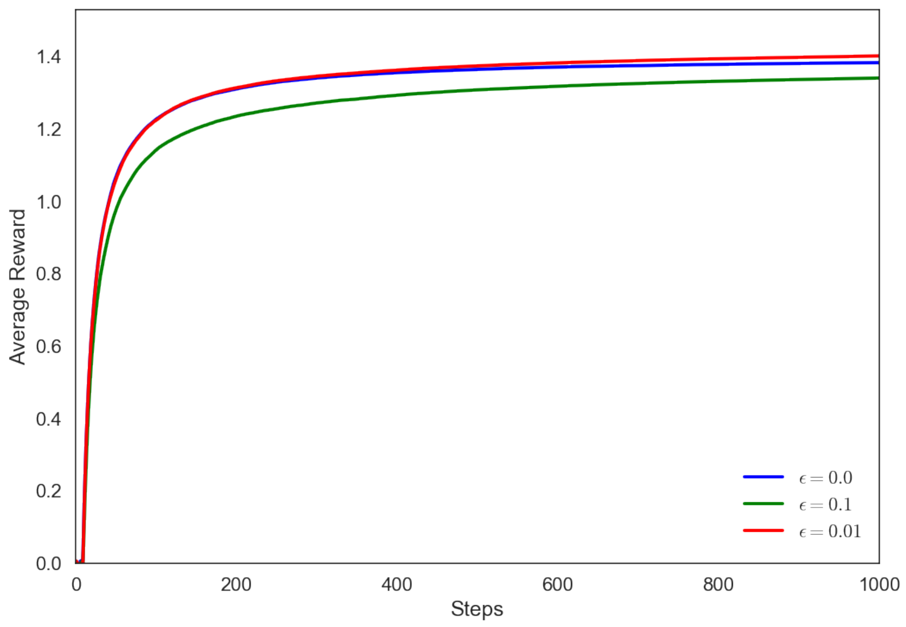


Figure 1: Performance of ϵ-greedy action-value methods tested on a 10-armed bandit problem. The $Q_0(a)$ is initialized to 0 for each policy and ϵ values of $0.0, 0.1$, and 0.01 are selected. The optimal action per step is 1.54. *top*) Average reward per step *bottom*) Average optimal actions per step



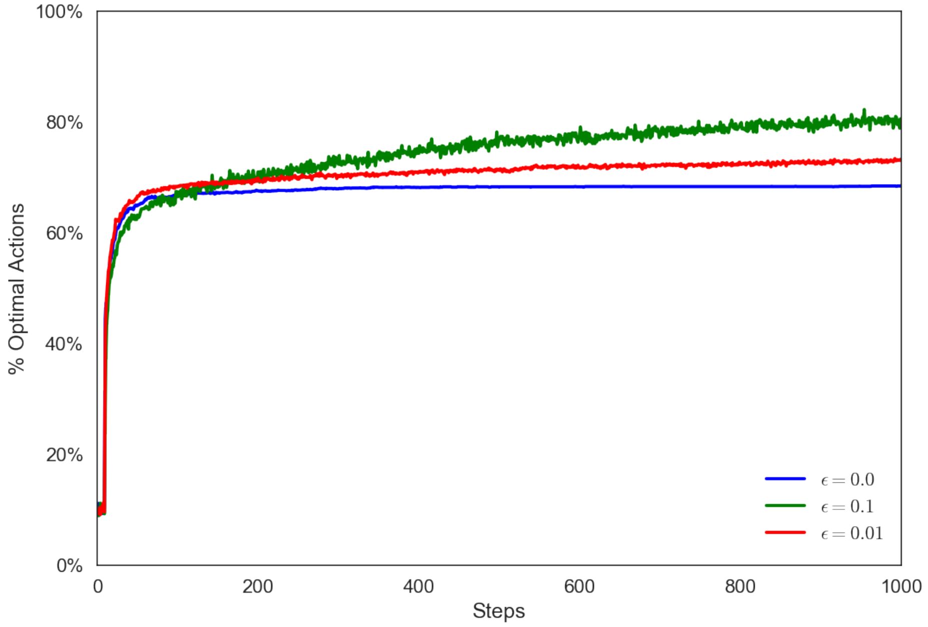


Figure 2: Performance of ϵ-greedy action-value methods tested on a 10-armed bandit problem. The $Q_0(a)$ is initialized to 5.0 for each policy and ϵ values of $0.0, 0.1$, and 0.01 are selected. The optimal action per step is 1.54. *top*) Average reward per step *bottom*) Average optimal actions per step