[S&B Book] Chapter 2: Multi-armed Bandits

Tags

> The most important feature distinguishing reinforcement learning from other types of learning is that it uses training information that *evaluates* the action taken rather than *instructs* by giving correct actions

- **Evaluative feedback**

  Indicates how good the action taken was, but not whether it was the best or the worst action possible;
- **Instructive feedback**

  Indicates the correction to take, independently of the action actually taken.

In this section we study the evaluative aspect of reinforcement learning in a simplified setting, *k*-armed bandit problem.

▼ 2.1 A *k*-armed Bandit Problem

- *The k-armed bandit problem*

  > You are faced repeatedly with a choice among *k* different options, or actions. After each choice you receive a numerical reward chosen from a stationary probability distribution that depends on the action you selected. Your objective is to maximize the expected total reward over some time period.

- The value of an arbitrary action: $q_*(a) \doteq \mathbb{E}[R_t|A_t = a]$
- **greedy actions**:

  When maintaining estimates of the action values, then at any time step there is at least one action whose estimated value is the greatest.
- *Exploration* and *exploitation:*
  - Exploitation - selecting one of the greedy actions; exploiting the current knowledge of the values of the actions;
  - Exploration - selecting one of the non-greedy actions; exploring enables you to improve your estimate of the non-greedy action's value;

Reward is lower in the short run, during exploration, but higher in the long run because after you have discovered the better actions, you can exploit them many times.

▼ 2.2 Action-value Method

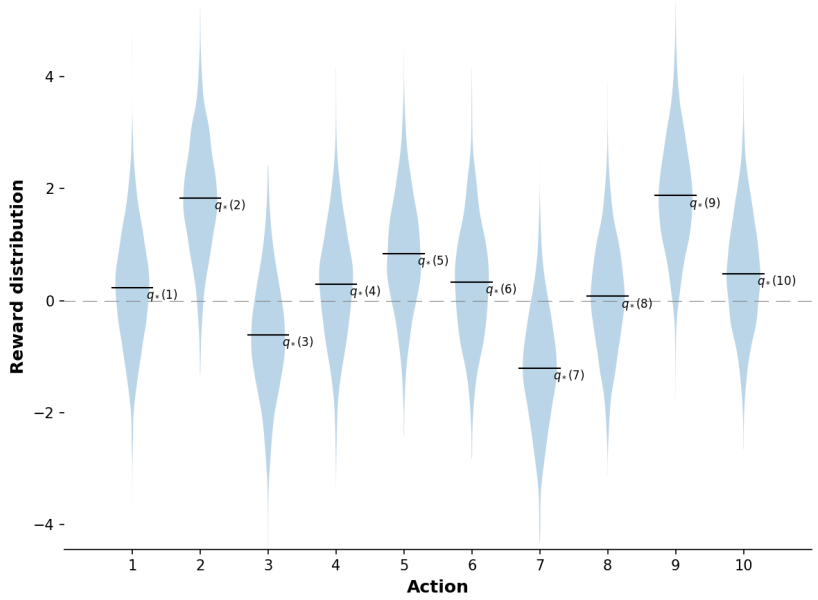
- The way of estimating action-value is by averaging the rewards actually received:

$$Q_t(a) \doteq \frac{\text{sum of rewards when } a \text{ taken prior to } t}{\text{number of times } a \text{ taken prior to } t} = \frac{\sum_{i=1}^{t-1} R_i \cdot 1_{A_i=a}}{\sum_{i=1}^{t-1} 1_{A_i=a}}$$

- *greedy* action selection method: $A_t \doteq \arg\max_a Q_t(a)$, with tie broken randomly; greedy action selection always exploits current knowledge to maximize immediate reward.
- $\varepsilon - greedy$ **method:** with a small probability ε, select randomly from among all the actions with equal probability; with probability $1 - \varepsilon$, select action greedily.

Example:

A set of 2000 randomly generated k-armed bandits problem with k = 10.



Implementation:

Reinforcement-Learning-2nd-Edition-Notes-Codes/example_2_1_distribution.py at main · terrence-ou/Reinforcement-Learning-2nd-Edition-Notes-Co

Notes and code implementations of examples and algorithms of the book Reinforcement Learning, 2nd Edition - Reinforcement-Learning-2nd-Edition-Notes-Codes/example_2_1_distribution.py at main · terr...

https://github.com/terrence-ou/Reinforcement-Learning-2nd-Edition-Notes-Codes/blob/main/chapter_02_k_armed_bandits/example_2_1_distribution.py

---

▼ 2.4 Incremental Implementation

Here I want to first take notes on the 2.4 then back to the e-greedy examples in 2.3 because 2.4 introduced the algorithms to implement the 10-armed bandit problem;

- **The action-value methods on a single action:**

  R_i denotes the reward received after the i-th selection of *this action*, and Q_n denotes the estimate of its action value after it has been selected $n-1$ times:

$$Q_n \doteq \frac{R_1 + R_2 + \cdots + R_{n-1}}{n-1}$$

  > This method maintain a record of all the rewards and then perform this computation whenever the estimated value was needed. But its disadvantage is that, if it is done, the memory and computational requirements would grow over time as more reward are seen. Each additional reward would require additional memory to store it and additional computation to compute the sum in the numerator.

- Device incremental formulas for updating averages:

$$\begin{aligned}
Q_{n+1} &= \frac{1}{n}\sum_{i=1}^{n} R_i \\
&= \frac{1}{n}\left(R_n + \sum_{i=1}^{n-1} R_i\right) \\
&= \frac{1}{n}\left(R_n + (n-1)\frac{1}{n-1}\sum_{i=1}^{n-1} R_i\right) \\
&= \frac{1}{n}\left(R_n + (n-1)Q_n\right) \\
&= \frac{1}{n}(R_n + nQ_n - Q_n) \\
&= Q_n + \frac{1}{n}\left[R_n - Q_n\right]
\end{aligned}$$

  Thus, a general form of this method is:

  $NewEstimate \leftarrow OldEstimate + StepSize\left[Target - OldEstimate\right]$

- Put all together: Pseudocode for **A Simple Bandit Algorithm**

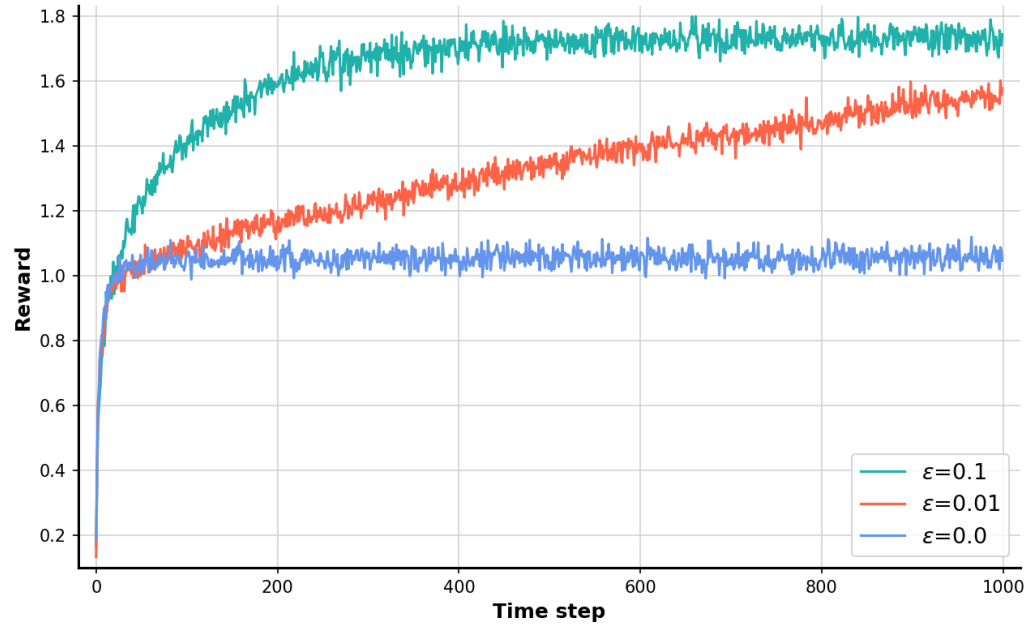
A simple bandit algorithm

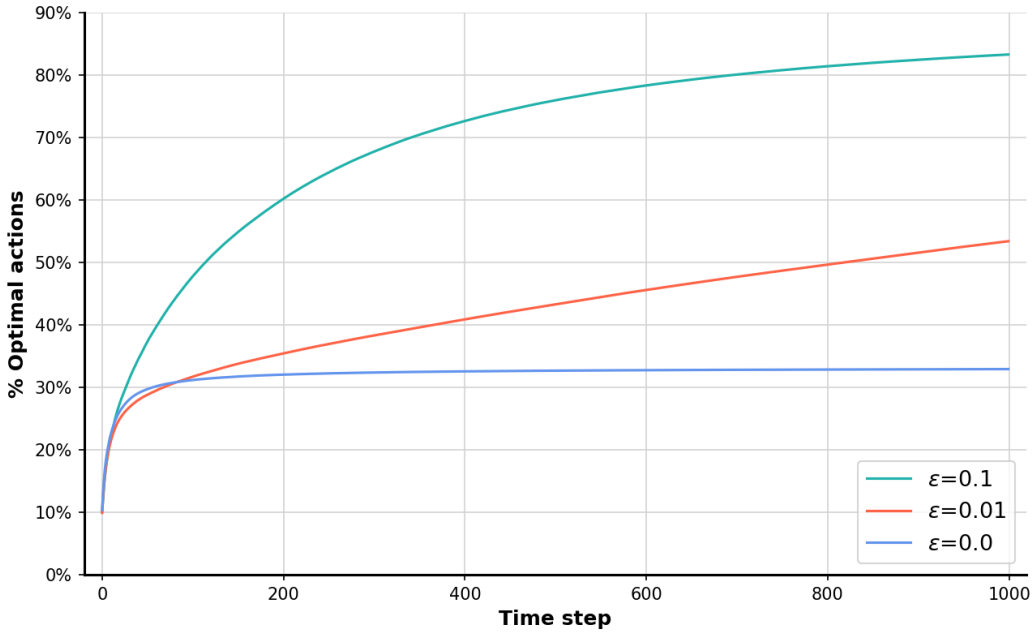
Initialize, for $a = 1$ to k:
  $Q(a) \leftarrow 0$
  $N(a) \leftarrow 0$

Loop forever:
  $A \leftarrow \begin{cases} \operatorname{argmax}_a Q(a) & \text{with probability } 1-\varepsilon \quad \text{(breaking ties randomly)} \\ \text{a random action} & \text{with probability } \varepsilon \end{cases}$
  $R \leftarrow bandit(A)$
  $N(A) \leftarrow N(A) + 1$
  $Q(A) \leftarrow Q(A) + \frac{1}{N(A)}\left[R - Q(A)\right]$

Example:

Run the 10-armed bandit algorithm on the testbed and compare the effect of using different ε value.





Implementation:

Reinforcement-Learning-2nd-Edition-Notes-Codes/example_2_2_bandits_algo.py at main · terrence-ou/Reinforcement-Learning-2nd-Edition-Notes-C

Notes and code implementations of examples and algorithms of the book Reinforcement Learning, 2nd Edition - Reinforcement-Learning-2nd-Edition-Notes-Codes/example_2_2_bandits_algo.py at main · terr...

https://github.com/terrence-ou/Reinforcement-Learning-2nd-Edition-Notes-Codes/blob/main/chapter_02_k_armed_bandits/example_2_2_bandits_algo.py

▼ 2.5 Tracking a Nonstationary Problem

- All of the discussions so far are based on the assumption that the bandit problem is stationary. But in most of the time, the reinforcement problem we're facing is nonstationary.
- In the nonstationary case, **it makes sense to give more weights to recent reward than to long-past rewards.** One of the most popular ways of doing this is to use a constant step-size parameter:

$$Q_{n+1} \doteq Q_n + \alpha \left[R_n - Q_n\right], \ \alpha \in (0, 1]$$

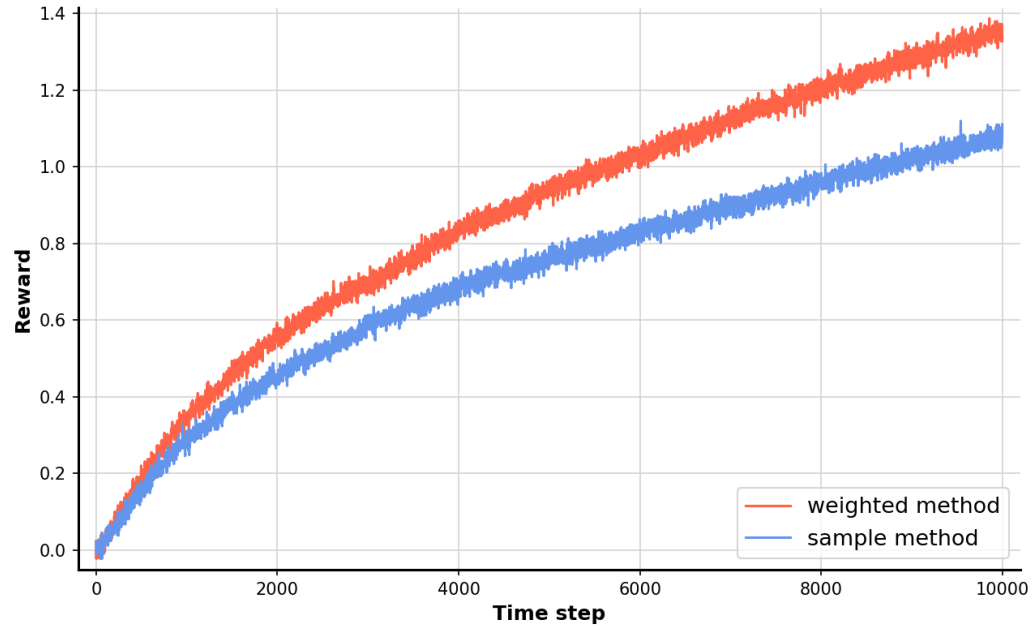
Proof: Q_{n+1} is a weighted average of past rewards and the initial estimate Q_1:

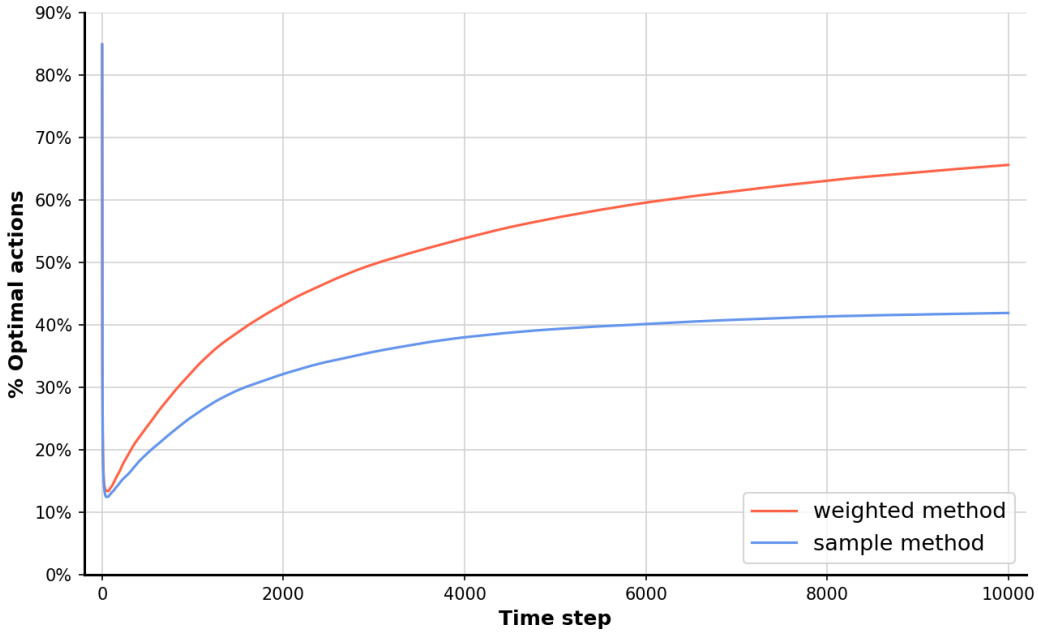
$$\begin{aligned}
Q_{n+1} &= Q_n + \alpha \left[R_n - Q_n\right] \\
&= \alpha R_n + (1-\alpha) Q_n \\
&= \alpha R_n + (1-\alpha) \left[\alpha R_{n-1} + (1-\alpha) Q_{n-1}\right] \\
&= \alpha R_n + (1-\alpha)\alpha R_{n-1} + (1-\alpha)^2 Q_{n-1} \\
&= \alpha R_n + (1-\alpha)\alpha R_{n-1} + (1-\alpha)^2 \alpha R_{n-2} + \cdots + (1-\alpha)^{n-1} \alpha R_1 + (1-\alpha)^n Q_1 \\
&= (1-\alpha)^n Q_1 + \sum_{i=1}^{n} \alpha (1-\alpha)^{n-i} R_i
\end{aligned}$$

because $1 - \alpha$ is less than 1, thus weight given to R_i decreases as the number of intervening rewards increases.

Exercise:

Design and conduct an experiment to demonstrate the difficulties that sample-average methods have for non-stationary problems. Use a modified version of the 10-armed testbed in which all the $q_*(a)$ start out equal and then take independent random walks (say by adding a normally distributed increment with mean zero and standard deviation 0.01 to all the $q_*(a)$ on each step). Prepare plots like Figure 2.2 for an action-value method using sample averages, incrementally computed, and another action-value method using a constant step-size parameter, $\alpha = 0.1$ Use $\varepsilon = 0.1$ and longer runs, say of 10,000 steps.





Implementation:

Reinforcement-Learning-2nd-Edition-Notes-Codes/exercise_2_5_non_stationary.py at main · terrence-ou/Reinforcement-Learning-2nd-Edition-Notes-

Notes and code implementations of examples and algorithms of the book Reinforcement Learning, 2nd Edition - Reinforcement-Learning-2nd-Edition-Notes-Codes/exercise_2_5_non_stationary.py at main · t...

https://github.com/terrence-ou/Reinforcement-Learning-2nd-Edition-Notes-Codes/blob/main/chapter_02_k_armed_bandits/exercise_2_5_non_stationary.py

▼ 2.6 Optimistic Initial Values

- **Bias:**

  The initial action-value estimates $Q_1(a)$, are **bias** in terms of statics.

  For the sample-average method, the bias disappears once all actions have been selected at lease once, but for the weighted-average method, the bias is permanent, though decreasing over time.

- **Optimistic initial values:**

  The assumption of this method: initial action values can also be used as a simple way to encourage exploration.

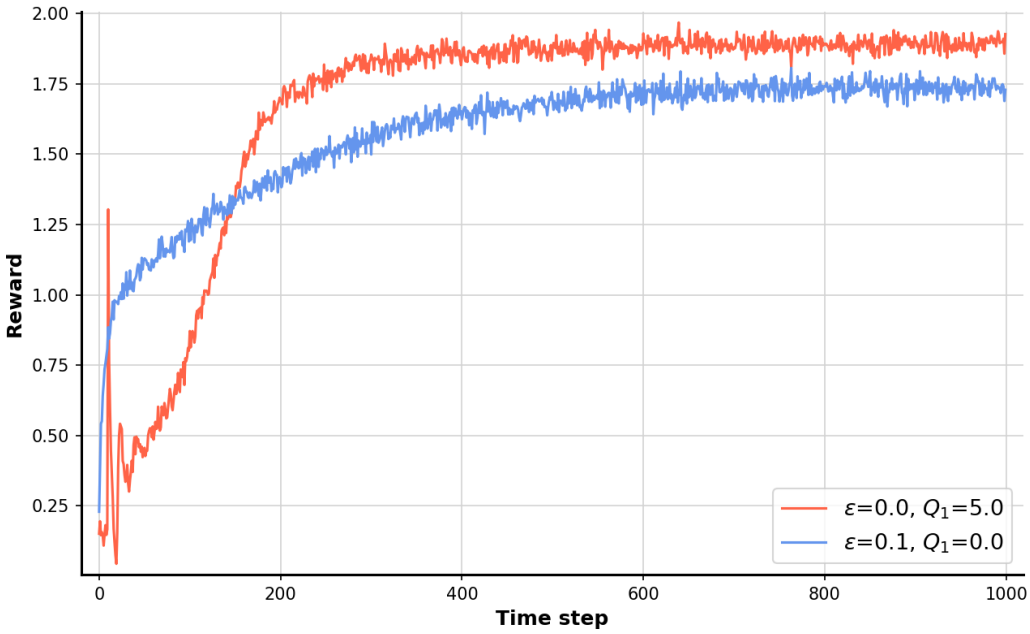
  In the k-armed bandit testbed, $q_*(a)$ are sampled from a normal distribution with mean 0 and variance 1. An initial estimate of +5 is thus wildly optimistic; Whichever actions are initially selected, the reward is less than the starting estimates; the learner switches to other actions, being “disappointed” with the rewards it is receiving.

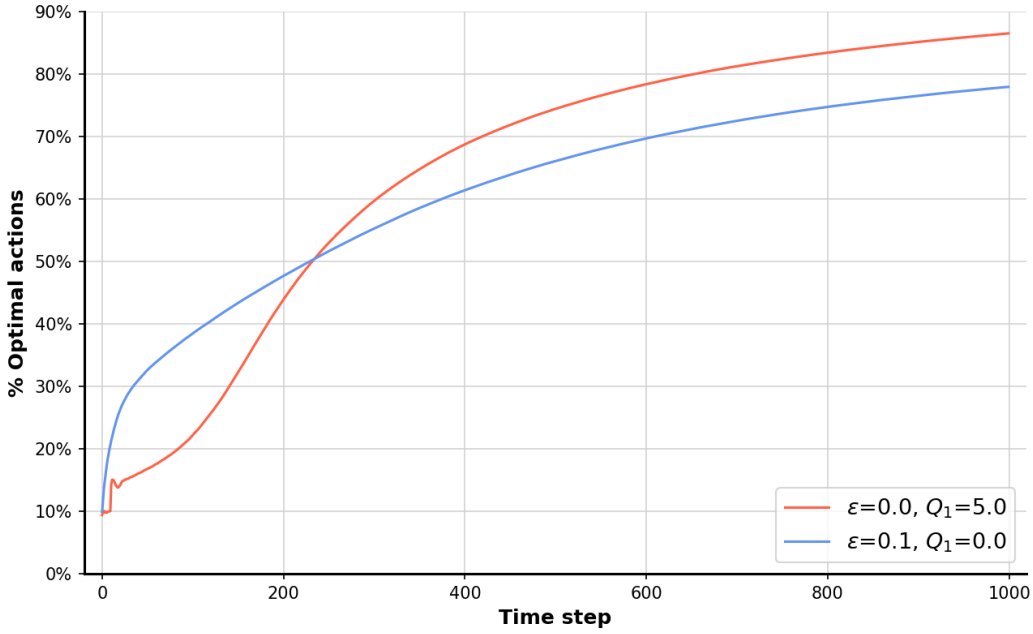
- **Limitation:**

  This method is not well suited to nonstationary problems because its drive for exploration is inherently temporary. If the task changes, creating a renewed need for exploration, this method cannot help.

Example:

The effect of optimistic initial action-value estimates on the 10-armed testbed. Both methods used a constant step-size parameter, $\alpha = 0.1$





Implementation:

Reinforcement-Learning-2nd-Edition-Notes-Codes/example_2_3_OIV.py at main · terrence-ou/Reinforcement-Learning-2nd-Edition-Notes-Codes

Notes and code implementations of examples and algorithms of the book Reinforcement Learning, 2nd Edition - Reinforcement-Learning-2nd-Edition-Notes-Codes/example_2_3_OIV.py at main · terrence-ou/R...

https://github.com/terrence-ou/Reinforcement-Learning-2nd-Edition-Notes-Codes/blob/main/chapter_02_k_armed_bandits/example_2_3_OIV.py

▼ 2.7 Upper-Confidence-Bound Action Selection

- The limitation of ε-greedy method:

  The greedy actions look best at present, but some of other actions may be better;

  ε-greedy action selection forces the non-greedy actions to be tried, but indiscriminately, with no preference for those that are nearly greedy or particularly uncertain.

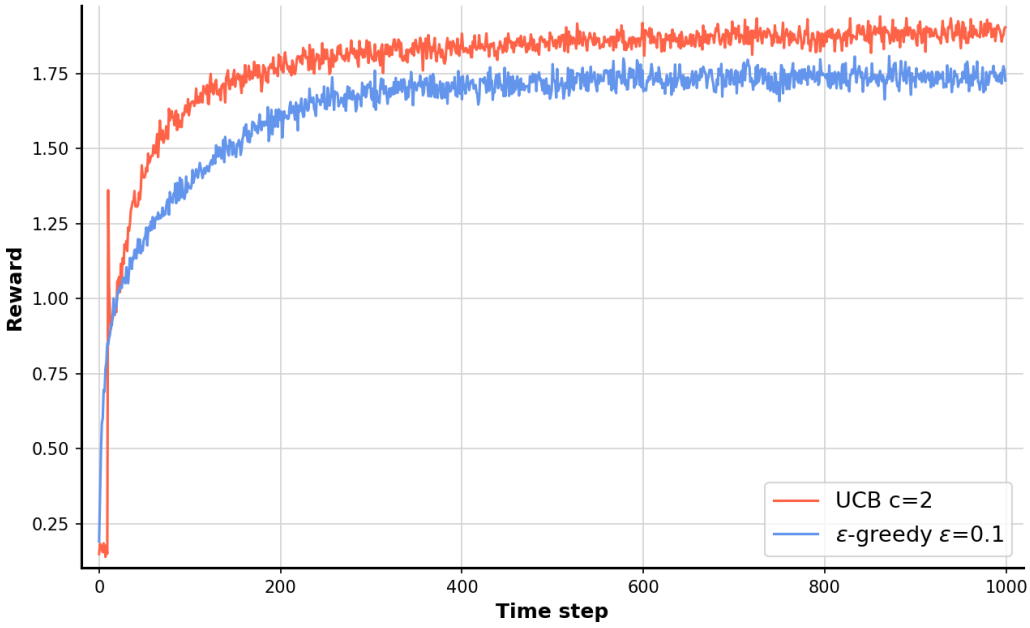
- The idea of UCB helps select among the non-greedy actions according to their *potential for actually being optimal*.

  $A_t \doteq \arg\max_a \left[Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}} \right]$

  - The square-root term measures the uncertainty or variance in the estimate of a's value;
  - c determines the confidence level;
  - Each time a is selected the uncertainty is presumably reduced; each time an action other than a is selected, t increases but N_t(a) does not, the uncertainty estimate increases.

  The actions with lower value estimates, or that have already been selected frequently, will be selected with decreasing frequency over time.

Example:



Implementation:

Reinforcement-Learning-2nd-Edition-Notes-Codes/example_2_4_UCB.py at main · terrence-ou/Reinforcement-Learning-2nd-Edition-Notes-Codes

Notes and code implementations of examples and algorithms of the book Reinforcement Learning, 2nd Edition - Reinforcement-Learning-2nd-Edition-Notes-Codes/example_2_4_UCB.py at main · terrence-ou/R...

https://github.com/terrence-ou/Reinforcement-Learning-2nd-Edition-Notes-Codes/blob/main/chapter_02_k_armed_bandits/example_2_4_UCB.py

- **The limitation of UCB**

  1. It is difficult in dealing with nonstationary problems;
  2. Another difficulty is dealing with large state space, particularly when using function approximation .

  In these settings, the idea of UCB action selection is usually not practical;

▼ 2.8 Gradient Bandit Algorithms

- In this section, we consider learning a numerical *preference* for each action a, which we denote $H_t(a) \in \mathbb{R}$; the larger the preference, the more often that action is taken.
- $\pi_t(a)$: a notation for the probability of taking action a at time t;

$$\Pr\{A_t = a\} \doteq \frac{e^{H_t(a)}}{\sum_{b=1}^{k} e^{H_t(b)}} \doteq \pi_t(a)$$

  Initially, all action preferences are the same so that all actions have an equal probability of being selected.

- **Gradient ascent:**

  On each step, after selecting action A_t and receiving the reward R_t, the action preference are updated by:

$$H_{t+1}(A_t) \doteq H_t(A_t) + \alpha(R_t - \bar{R}_t)(1 - \pi_t(A_t)), \quad \text{and}$$
$$H_{t+1}(a) \doteq H_t(a) - \alpha(R_t - \bar{R}_t)\pi_t(a), \quad \text{for all } a \neq A_t$$

- $\alpha > 0$ is a step size parameter;
- $\bar{R}_t \in \mathbb{R}$ is the average of the rewards up to but not including time t (with $\bar{R}_1 \doteq R_1$). This term serves as a baseline with which reward is compared; if the reward is higher than baseline, then the probability of taking A_t in the future is increased, and if the reward is below baseline, then the probability is decreased.

---

proof of the gradient ascent equation:

$$H_{t+1}(a) \doteq H_t(a) + \alpha \frac{\partial \mathbb{E}[\mathbb{R}_t]}{\partial H_t(a)}, \text{where } \mathbb{E}[R_t] = \sum_x \pi_t(x) q_*(x)$$

First we take a closer look at the exact performance gradient:

$$\begin{aligned}
\frac{\partial \mathbb{E}[R_t]}{\partial H_t(a)} &= \frac{\partial}{\partial H_t(a)} \left[\sum_x \pi_t(x) q_*(x) \right] \\
&= \sum_x q_*(x) \frac{\partial \pi_t(x)}{\partial H_t(a)} \\
&= \sum_x \left(q_*(x) - B_t \right) \frac{\partial \pi_t(x)}{\partial H_t(a)}
\end{aligned}$$

Then we multiply each term for the sum by $\pi_t(x)/\pi_x(x)$:

$$\begin{aligned}
\frac{\partial \mathbb{E}[R_t]}{\partial H_t(a)} &= \sum_x \pi_t(x) \left(q_*(x) - B_t \right) \frac{\partial \pi_t(x)}{\partial H_t(a)} / \pi_t(x) \\
&= \mathbb{E} \left[(q_*(A_t) - B_t) \frac{\partial \pi_t(A_t)}{\partial H_t(a)} / \pi_t(A_t) \right] \\
&= \mathbb{E} \left[(R_t - \bar{R}_t) \frac{\partial \pi_t(A_t)}{\partial H_t(a)} / \pi_t(A_t) \right]
\end{aligned}$$

Get partial derivative $\frac{\partial \pi_t(x)}{\partial H_t(a)}$:

$$\text{recall that: } \frac{\partial}{\partial x} \left[\frac{f(x)}{g(x)} \right] = \frac{\frac{\partial f(x)}{\partial x} g(x) - f(x) \frac{\partial g(x)}{\partial x}}{g(x)^2}$$

We can write:

$$
\begin{aligned}
\frac{\partial \pi_t(x)}{\partial H_t(a)} &= \frac{\partial}{\partial H_t(a)} \left[\frac{e^{H_t(x)}}{\sum_{y=1}^{k} e^{H_t(y)}} \right] \\
&= \frac{\frac{\partial e^{H_t(x)}}{\partial H_t(a)} \sum_{y=1}^{k} e^{H_t(y)} - e^{H_t(x)} \frac{\partial \sum_{y=1}^{k} e^{H_t(y)}}{\partial H_t(a)}}{\left(\sum_{y=1}^{k} e^{H_t(y)} \right)^2} \\
&= \frac{1_{a=x} e^{H_t(x)} \sum_{y=1}^{k} e^{H_t(y)} - e^{H_t(x)} e^{H_t(a)}}{\left(\sum_{y=1}^{k} e^{H_t(y)} \right)^2} \\
&= \frac{1_{a=x} e^{H_t(x)}}{\sum_{y=1}^{k} e^{H_t(y)}} - \frac{e^{H_t(x)} e^{H_t(a)}}{\left(\sum_{y=1}^{k} e^{H_t(y)} \right)^2} \\
&= 1_{a=x} \pi_t(x) - \pi_t(x) \pi_t(a) \\
&= \pi_t(x) (1_{a=x} - \pi_t(a))
\end{aligned}
$$

Now we put this result back into the equation of $\frac{\partial \mathbb{E}[R_t]}{\partial H_t(a)}$:

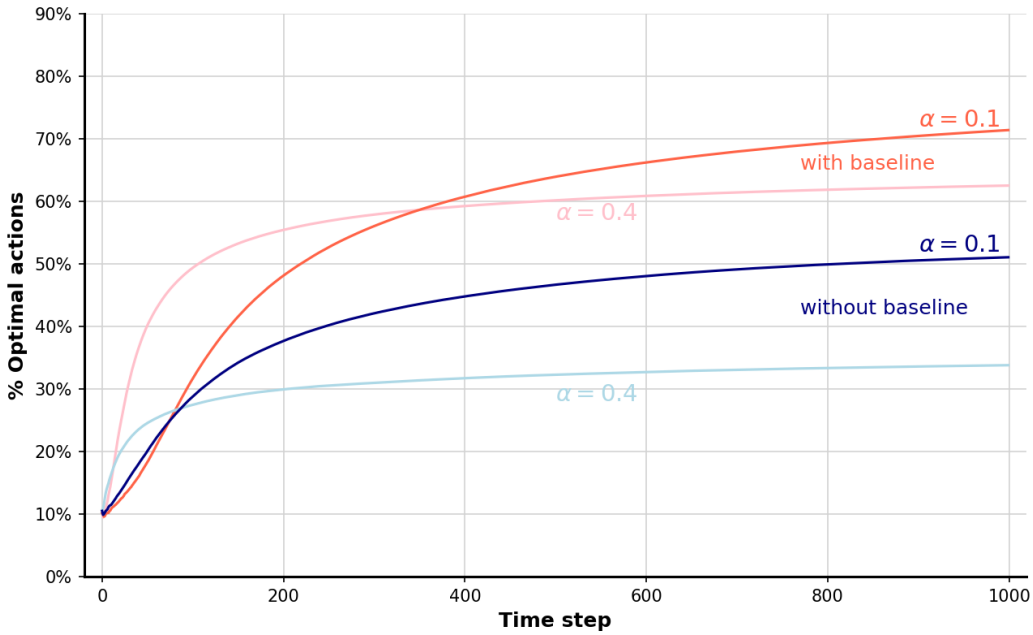
$$
\begin{aligned}
\frac{\partial \mathbb{E}[R_t]}{\partial H_t(a)} &= \mathbb{E} \left[(R_t - \bar{R}_t) \pi_t(A_t) (1_{a=A_t} - \pi_t(a)) / \pi_t(A_t) \right] \\
&= \mathbb{E} \left[(R_t - \bar{R}_t) (1_{a=A_t} - \pi_t(a)) \right]
\end{aligned}
$$

Thus, we get the gradient ascent equation:

$$
H_{t+1}(a) = H_t(a) + \alpha (R_t - \bar{R}_t)(1_{a=A_t} - \pi_t(a)), \quad \text{for all } \alpha
$$

Example:

Average performance of the gradient bandit algorithm with and without reward baseline. The reward are chosen from a normal distribution with mean=+4 and unit variance:



Implementation:

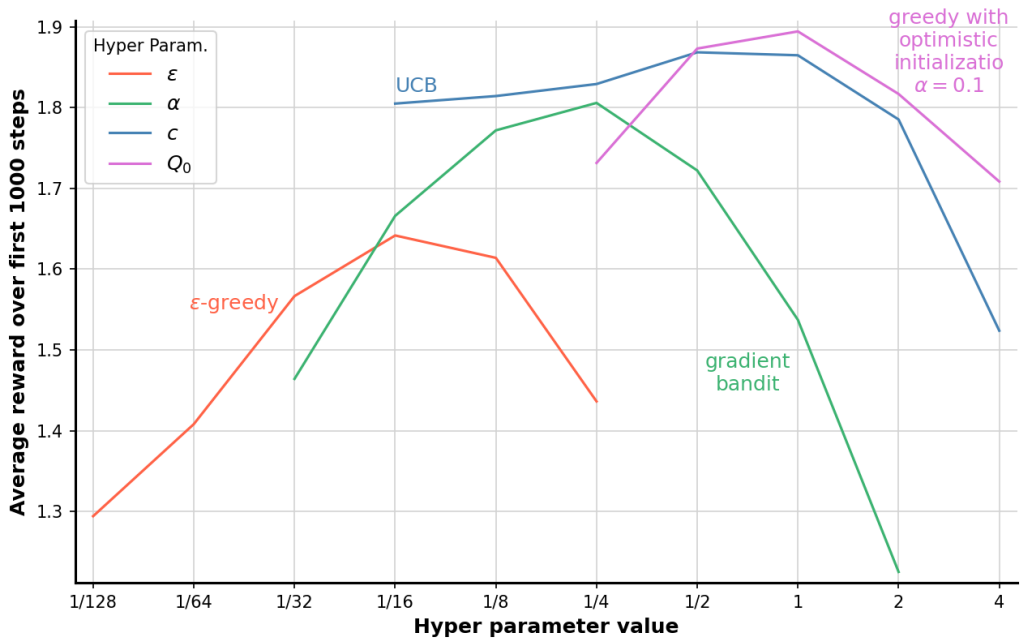
Reinforcement-Learning-2nd-Edition-Notes-Codes/example_2_5_gradient.py at main · terrence-ou/Reinforcement-Learning-2nd-Edition-Notes-Codes
Notes and code implementations of examples and algorithms of the book Reinforcement Learning, 2nd Edition - Reinforcement-Learning-2nd-Edition-Notes-Codes/example_2_5_gradient.py at main · terrence...
https://github.com/terrence-ou/Reinforcement-Learning-2nd-Edition-Notes-Codes/blob/main/chapter_02_k_armed_bandits/example_2_5_gradient.py

▼ 2.9 Summary

It is natural to ask which of the method in this chapter is the best; it is a difficult question to answer, but we can run them all on the 10-armed bandit testbed and compare their performance

Example:

A *parameter study* of the various bandit algorithm in this chapter, each as a function of its own parameter shown on a single scale on the x-axis.



Implementation:

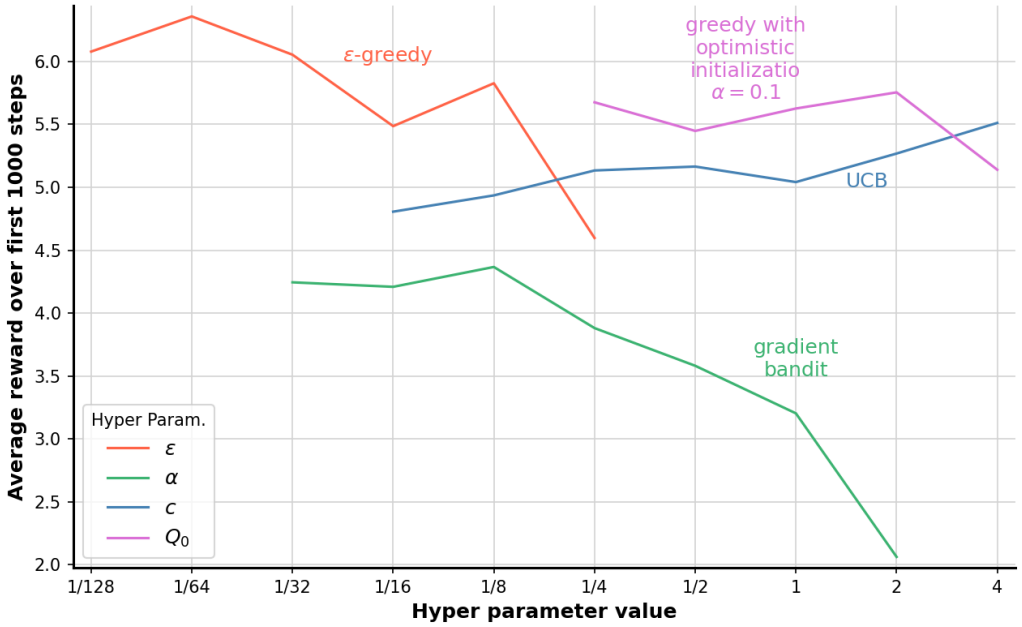
Reinforcement-Learning-2nd-Edition-Notes-Codes/example_2_6_summary.py at main · terrence-ou/Reinforcement-Learning-2nd-Edition-Notes-Code
Notes and code implementations of examples and algorithms of the book Reinforcement Learning, 2nd Edition - Reinforcement-Learning-2nd-Edition-Notes-Codes/example_2_6_summary.py at main · terrence-...
https://github.com/terrence-ou/Reinforcement-Learning-2nd-Edition-Notes-Codes/blob/main/chapter_02_k_armed_bandits/example_2_6_summary.py

> Despite the simplicity of these algorithms, the methods presented in this chapter can fairly be considered the state of art. There are more sophisticated methods, but their complexity and

> assumptions make them impractical for the full reinforcement learning problem that is our real focus.

Exercise:

Make a figure analogous to the previous figure for the non-stationary case in Exercise 2.5. Include the constant-step-size ε-greedy algorithm with α=0.1. Use runs of 200,000 steps and, as a performance measure for each algorithm and parameter setting, use the average reward over the last 100,000 steps.



Implementation:

Reinforcement-Learning-2nd-Edition-Notes-Codes/exercise_2_6_summary_non_stationary.py at main · terrence-ou/Reinforcement-Learning-2nd-Edi

Notes and code implementations of examples and algorithms of the book Reinforcement Learning, 2nd Edition - Reinforcement-Learning-2nd-Edition-Notes-Codes/exercise_2_6_summary_non_stationary.py at ...

https://github.com/terrence-ou/Reinforcement-Learning-2nd-Edition-Notes-Codes/blob/main/chapter_02_k_armed_bandits/exercise_2_6_summary_non_stationary.py