Chapter 2

Thursday, December 27, 2018 2:52 PM

Tabular solution methods:
Simplest forms: State and action spaces are small enough for the approximate value functions to be represented as arrays or tables. The methods can find exact solutions (exact optimal value function and the optimal policy)

Multi-armed Bandits:
RL uses training information that evaluates the actions taken rather than instructs by giving correct actions. It creates the need for active exploration, for an explicit search of good behavior. Pure evaluative feedback indicates how good the action taken was , not the best or the worst action. Pure instructive feedback indices the correct action to take, independently of the action actually taken.

2.1. A k-armed Bandit Problem:
It is an analogy to a slot machine or one-armed bandit problem, except that it has k-levers instead of one. Each action selection is like a play of one of the slot machine's levers and the rewards are the payoffs for hitting the jackpot.
In this game the agent has to repeatedly faced with a choice among 'k' different options and after each choice a numerical reward is chosen from a stationary probability distribution depending on the action selected. The objective is to maximize the expected total reward over some time period, for example over 1000 action selections or time steps.

Each of the k actions has an expected or mean reward given that that action is selected; let us call this the value of that action. The action selected on time step t as A_t, and the corresponding reward as R_t. The value of an arbitrary action a, denoted $q_*(a)$, is the expected reward given that a is selected:

$$q_*(a) \doteq \mathbb{E}[R_t \mid A_t = a] \,.$$

If the value of each action is known already with certainty it is easy to solve the k-armed bandit problem by only choosing the action which is having more value. But if the values are not known prior (most of the practical cases) then it is needed to be estimated with good certainty.

The estimated value of action 'a' at time step t as $Q_t(a)$. It is expected to have $Q_t(a)$ close to $q_*(a)$.
If the estimates of the action values are there, then at any time step there is at least one action whose estimated value is greatest. These are *greedy actions*. When one of those action is selected it is exploiting the knowledge. If one of the non-greedy actions is selected then it is called as exploring, it will improve the estimate of the non-greedy action's value. Exploitation can be used to maximize the expected reward on the one step, but the exploration may produce the greater total reward in the long run.
Suppose a greedy action's value is known with certainty, but other actions are estimated to be nearly as good but with substantial uncertainty. It denotes that at least one of the actions may be actually better than the greedy actions. If there are many time steps ahead, to make action selections, it may be better to explore the nongreedy actions and run, during exploration, but higher in the long run. Because after that we can use those good estimates and exploit them in the long run for many times. As it is not possible to both explore and exploit at any single action selection, it is called as the conflict between exploration and exploitation.
To exploit or explore depends in a complex way on the precise values of the estimates, uncertainties and the number of remaining steps.

2.2. Action Value Methods:
The methods for estimating the values of actions and for using the estimates to make action selection decisions are collectively called as *action-value* methods.
The true value of an action is the mean reward when that action is selected. One way to estimate is averaging the rewards actually received:

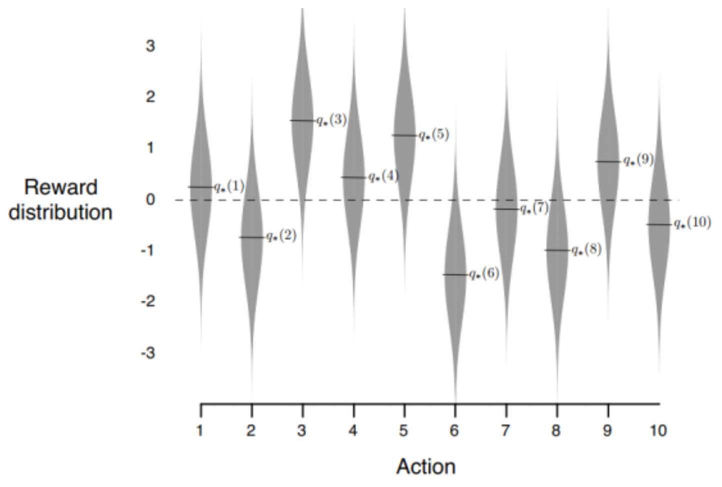
$$Q_t(a) \doteq \frac{\text{sum of rewards when } a \text{ taken prior to } t}{\text{number of times } a \text{ taken prior to } t} = \frac{\sum_{i=1}^{t-1} R_i \cdot \mathbb{1}_{A_i=a}}{\sum_{i=1}^{t-1} \mathbb{1}_{A_i=a}}$$

where $1_{\text{predicate}}$ denotes the random variable that is 1 if predicate is true and 0 if it is not.
If the denominator is zero, then we instead define $Q_t(a)$ as some default value, such as 0. As the denominator goes to infinity, by the law of large numbers, $Q_t(a)$ converges to $q_*(a)$. This is called as the sample-average method because estimate is an average of the sample of relevant rewards. This is one of the ways for estimating the action values.
The simplest action selection rule is to select one of the actions with the highest estimated value. i.e. one of the greedy actions.

$$A_t \doteq \underset{a}{\arg\max}\, Q_t(a)$$

The above expression denotes that an action which gives maximum value in the function to the right of the argmax is selected and given to A_t.
Greedy action selection always exploits the current knowledge to maximize immediate reward. As an alternative the agent can behave greedily most of the times, but every once in a while, with small probability ε, the agent select randomly from among all actions with equal probability. This method of action selection is called as ε - greedy methods.
Advantage of this ε - greedy method is in the limit as the number of steps increases, every action will be sampled an infinite number of times and ensures that all the $Q_t(a)$ converge to $q_*(a)$. It implies that the optimal action converges to greater than 1 - ε.

2.3. The 10-armed testbed:
In order to test the effectiveness of the greedy and ε- greedy methods they were applied to a set of 2000 randomly generated k-armed bandit problems.
The action values, $q_*(a)$, a = 1, . . . , 10 were selected according to a normal distribution with mean 0 and variance 1. A learning method is applied to that problem, selected action A_t, at time step t, the actual reward R_t was selected from a normal distribution with mean $q_*(A_t)$ and variance 1. These distributions are shown in gray. The performance and behavior of any learning method is measured as it improves with experience over 1000 time steps when applied to one of the bandit problems. This 1000 time steps is one run. Then this run is repeated for 2000 times and then measures of the learning algorithm's average behavior is obtained.

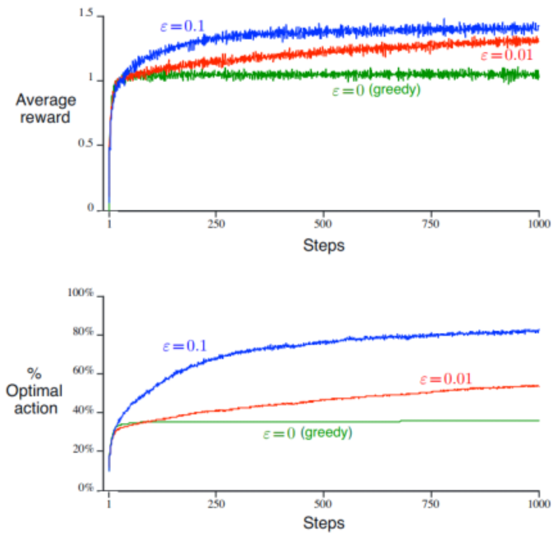


<u>*Figure Description:*</u>
An example bandit problem from the 10-armed testbed. The true value q(a) of each of the ten actions was selected according to a normal distribution with mean zero and unit variance, and then the actual rewards were selected according to a mean q*(a) unit variance normal distribution, as suggested by these gray distributions.*

The following figure compares the greedy method with two ε- greedy methods (ε = 0.01 and ε = 0.1). These methods estimate their action-values using the sample-average technique. The upper graph shows the increase in expected reward with experience. The greedy method improved slightly faster than the other methods at the very beginning but then leveled off at a lower level. It achieved reward-per-step of only about 1, compared to the best possible of 1.55 in the test bed. This shows the greedy method performed significantly worse in the long run because it got stuck in the suboptimal actions. The lower graph shows that the greedy method found optimal action only in one-third of the tasks. The ε - greedy methods *eventually* performed better because they continued to explore and to improve their chances of recognizing the optimal action.

<u>Comparison between ε-greedy methods with different values of ε:</u>
The ε = 0.1 method explored more and found the optimal action earlier, but it never selected that action more than 91% of the time. The ε = 0.01 method improved more slowly but eventually would perform better than ε = 0.1 method on both the performance measures shown in the figure below. *So in order to get both the benefits of high and low values of ε it is recommended to reduce the value of ε over time.*

<u>Dependence of advantage of the ε - greedy methods over the greedy methods:</u>
The advantage depends on the task. If the reward variances are larger, then the ε - greedy methods are better than the greedy methods. But if the variances are zero, then the greedy method know the true value of each action after trying it once. So in this case greedy method perform well since it found out the optimal action earlier than the ε - greedy methods. Even in these deterministic cases if the tasks were not stationary, the true value of actions are changed over time, the ε-greedy methods are good, since the exploration is required to make sure that the non-greedy action values are not changed to become better than the greedy ones. Non-stationary is the case most commonly encountered in most of the practical RL problems.



<u>*Figure Description:*</u>
Average performance of " ε-greedy action-value methods on the 10-armed testbed. These data are averages over 2000 runs with different bandit problems. All methods used sample averages as their action-value estimates.

2.4. Incremental Implementation:

The action-value methods discussed till now estimate action values as sample averages of observed rewards. These computations have to efficient, i.e. with constant memory and constant per-time-stamp computation.

For simplification single action is concentrated.

R_i denote reward received after the *i*th selection of this action and Q_n denote the estimate of its action value after it has been selected for n-1 times which can be written as below

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

Obvious implementation would be to maintain a record of all the rewards and perform this when computation whenever the estimated value was needed. If this is followed then the memory and computational requirements grow over time as more rewards are seen. Each additional reward would require additional memory to store it and additional computation to compute the sum in the numerator.

It is easy to devise incremental formulas for updating the averages with small, constant computation required for each new reward.

Given Q_n and the *n*th reward , R_n, the new average of all *n* rewards can be computed using the following steps:

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

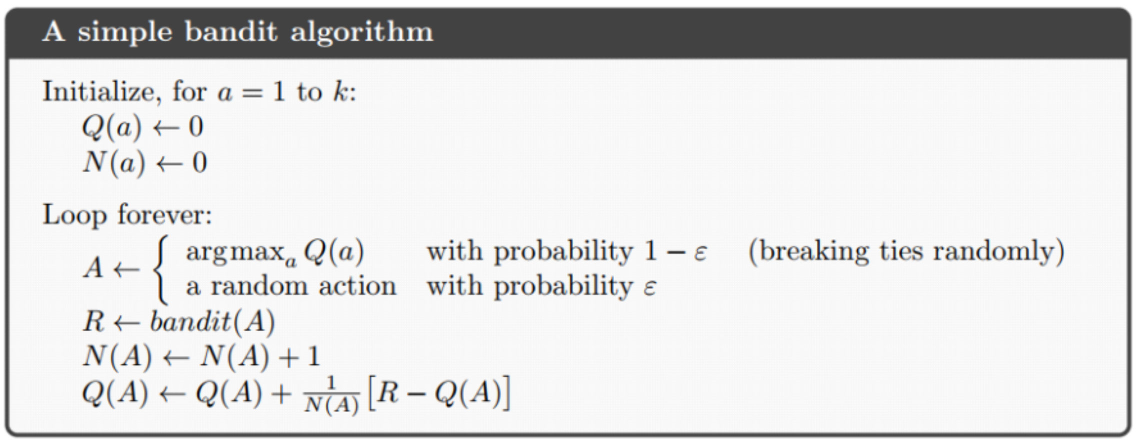
The above formula even holds for n=1, obtaining $Q_2 = R_1$ for arbitrary Q_1. This implementation requires memory only for Q_n and n, and only the small computation for each new reward. The general form of the approach mentioned above is

New Estimate <-- Old Estimate + Step Size [Target - Old Estimate]

The term *[Target - Old Estimate]* is the error in the estimate and it is reduced by taking a step toward the *"Target".* Here the target is the nth reward.

In the incremental methods, the step size changes from time step to time step. In processing the *n*th reward for action *a*, the step size is 1/n. The step size parameter can be generally denoted using $\alpha_t(a)$.

Pseudocode for a complete bandit algorithm using incrementally computed sample averages and ε - greedy action selection is given below.



A simple bandit algorithm

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

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

2.5. Tracking a Nonstationary problem:

The averaging methods used are suitable for stationary problems. *Stationary problems are the problems in which the reward probabilities do not change over time.* But most of the reinforcement learning problems are nonstationary. In those cases more weightage has to be given to the recent rewards than to long-past rewards. One of the most popular ways is to *use constant step-size parameter*.

The incremental update rule for updating an average Q_n of the *n-1* past rewards can be modified to the following form.

$$Q_{n+1} \doteq Q_n + \alpha\Big[R_n - Q_n\Big],$$

the step size parameter α is constant and α ∈ (0,1]. So this results in Q_{n+1} being a weighted average of past rewards and the initial estimate Q_1.

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

It is called as weighted-average because the sum of the weights is equal to 1. The weight given to each reward depends on how many rewards ago it was observed. The quantity (1 - α) is less than 1 and thus the weight given to R_i decreases as the number of intervening rewards increases. This weight decays exponentially because of the exponent on (1 - α).

For instance if (1 - α) = 0, then all the weight goes to the very last rewards, R_n, because of the convention $0^0 = 1$. This is called an *exponential recency-weighted average method*.

It is also convenient to vary the step-size parameter from step to step. Let $\alpha_n(a)$ denote the step-size parameter used to process the reward received after the *n*th selection of action a. In the *simple-average method* this step size parameter $\alpha_n(a) = 1/n$, which is guaranteed to converge to the true value because of the law of large-numbers.

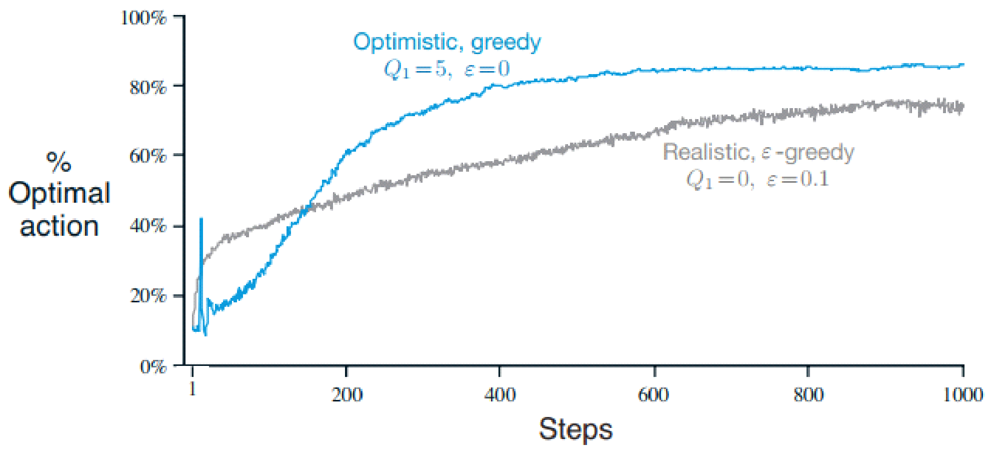
In stochastic approximation theory, the conditions required to assure convergence with probability 1:

$$\sum_{n=1}^{\infty} \alpha_n(a) = \infty \qquad \text{and} \qquad \sum_{n=1}^{\infty} \alpha_n^2(a) < \infty.$$

First expression denotes that the steps are large enough to eventually overcome any initial conditions or random fluctuations.
Second expression states that eventually the steps become small enough to assure convergence.
Both conditions are met in the case of $\alpha_n(a) = 1/n$ but not in the case of constant step-size parameter, where the second condition is not satisfied, indicating that the estimates never completely converge but continue to vary in response to the most recently received rewards. This is desired in nonstationary conditions and in turn most of the RL problems.

2.6. Optimistic Initial Values

Methods discussed so far relies to some extent on the initial action values, $Q_1(a)$. These methods are biased by their initial estimates. The bias disappears in the sample-average methods once all the actions have been selected at least once, but for the methods with constant step size parameter (α), the bias is permanent, but decrease over time. In practice, this kind of bias is usually not a problem, and can be sometimes very helpful. The initial estimate become, in effect, a set of parameters that has to be chosen by the user, if only set all of them to zero. They also provide an easy way to provide prior knowledge about the level of rewards can be expected.
Initial values to make exploration:
Initial action values can also be used to encourage exploration. In the 10-bed test problem discussed before we can choose the initial action values to +5 instead of 0. The $q_*(a)$ for this problem is selected from a distribution with mean 0 and variance 1. So the initial estimate of +5 is wildly optimistic and this optimism encourages the action-value methods to explore. Whatever actions are initially selected, the reward is less than the starting estimates, being disappointed with this the method tries other actions. So this results in the selection of all actions several times before the value estimates converge. *The system does a fair amount of exploration even if greedy actions are selected all the time.*
The following figure shows the performance of the approach on the 10-armed bandit testbed with the greedy approach having the value of $Q_1(a) = +5$ for all a and it is compared with the ε-greedy method with $Q_1(a) = 0$. These optimistic methods may perform worst initially because it explores more, but eventually it performs better because its exploration decreases with time. This technique for encouraging the explorations is called *optimistic initial values*.

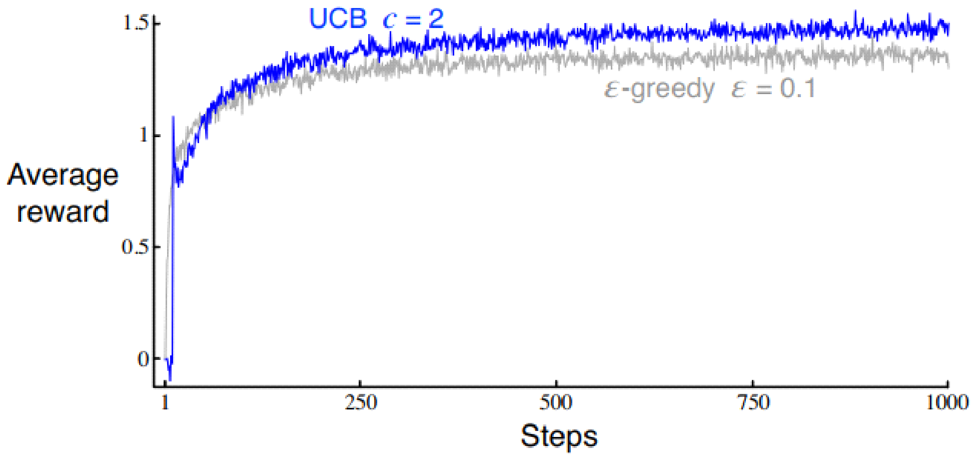


It is a simple trick which is effective in the stationary problems. But it is not a general approach to encourage exploration. It is not well suited for the non-stationary problems because the optimistic initial values approach's drive for exploration is inherently temporary. Suppose the task changes which produces the need for exploration this method won't hold. So, any method that focuses on the initial values won't perform good in the non-stationary cases. It also suits for the sample-average method which averages all subsequent rewards with equal weights.

2.7. Upper-Confidence-Bound Action Selection:
Exploration is essential since there is always uncertainty about the action-value estimates. ε-greedy methods forces the non-greedy actions to be tried indiscriminately, with no preference for those that are greedy or particularly certain. Selecting the non-greedy actions according to their potential for actually being optimal, taking into account both how close their estimates are to being maximal and the uncertainties in those estimates. One effective way to do this is to select actions based on the following expression.

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

where ln *t* denotes the natural log of t, $N_t(a)$ denotes the number of times that action '*a*' has been selected prior to time t and the number *c > 0* controls the degree of exploration. If $N_t(a) = 0$, then a is considered to be a maximizing action.
The square root term is a measure of the uncertainty or variance in the estimate of a's value. So the quantity being max'ed over is the upper bound on the possible true action value of action a, with c determining the confidence level.
So each time action *'a'* is selected then the uncertainty decreases, the occurrence of action *'a'* increments $N_t(a)$ and since it is in the denominator, the uncertainty term decreases. On the other hand each time an action other than the action *'a'* is selected then t increases but $N_t(a)$ won't increase, since t appears in the numerator this will increase the uncertainty term.
Natural log is used such that the increase gets small over time, but are unbounded, so all actions will be eventually selected, but actions with lower value estimates or that have been selected frequently, will be selected with decreasing frequency over time.
The following plot shows the improvement using the UCB approach.



UCB often performs well, but they are very hard to be extended beyond the bandit problem. Another difficulty is dealing with the bandit problems. Another difficulty is dealing with the problems having large state spaces (when using function approximations). In these more advanced settings the idea of UCB action selection is not practical.

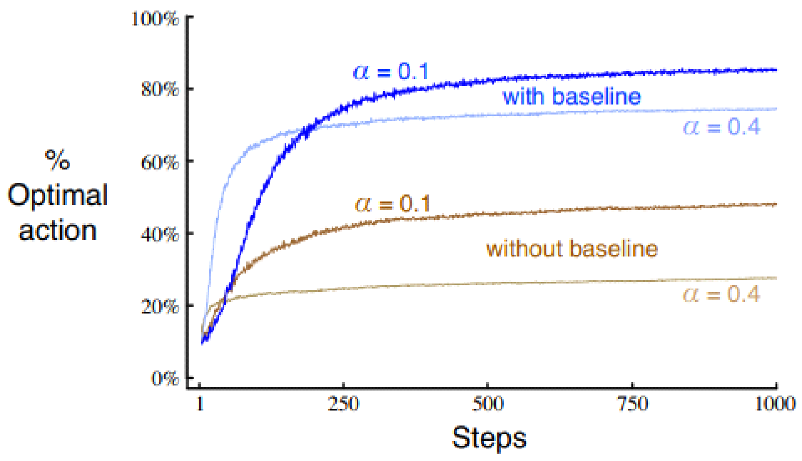
2.8 Gradient Bandit Algorithms:
Till now methods that estimate action values and use those estimates to select actions are considered. It is not the only method that is possible. In Gradient Based Algorithm method, the learning of a numerical preference for each action *a*, which is denoted as $H_t(a)$. The larger the preference, the more often that action is taken, but the preference has no interpretation in terms of reward.
The relative preference of one action over another is important. Even if 1000 is added to all the action preferences there is no effect on action probabilities since preferences are determined through a soft-max distribution. The formula for the same is shown below

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

$\Pi_t(a)$ is the probability of taking an action *a*, at time t. Initial preference for all the actions are the same ($H_1(a) = 0$, for all actions) and have equal probability to get selected.
On each step after selecting each action, the action preferences are updated by the formula below:

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

This is the natural learning algorithm based on the stochastic gradient ascent.
$\alpha > 0$ is the step-size parameter, $\bar{R}_t \in$ R is the average of all rewards up through and including time *t*, which can be computed incrementally using the incremental methods.
The term $\bar{R}_t$ is considered as the baseline with which the reward is compared and if the reward is higher than the baseline then the probability of taking A_t in the future is increased and if the reward is low then probability is decreased. The non-selected actions move in the opposite direction.
The following figure the result with the gradient bandit algorithm on a variant of 10 - armed testbed in which the true selected rewards are selected according to a normal distribution with mean +4 (zero was chosen in the previous approaches) and with unit variance. The shift of all the rewards from 0 to +4 has no effect on the gradient bandit algorithms, because of the reward baseline term, which can adapt to the new level. If the baseline were omitted then the performance degrades.



<u>Figure Description:</u> Average performance of the gradient bandit algorithm with and without a reward baseline on the 10-armed testbed when the $q_*(a)$ are chosen to be near +4 rather than near zero.

2.9 Associative Search

Till now non-associative tasks are considered. It means that there is no necessity to associate different actions with different situations. In the tasks discussed above the learner tries to find a single best action when the task is stationary or tries to track the best action as it changes over time when the task is nonstationary. In general RL tasks there is more than one situation possible. In those scenarios, the goal is to learn a policy : a mapping from situations to the actions that are best in those situations.
The following assumption can be made for these situations. Suppose there are several different k-armed bandit tasks and on each step, the learner confronts one of these chosen at random. So, the bandit task changes randomly from one step to another. This can be similar to a single non-stationary k-armed bandit task whose true action values change randomly from step to step. One of the methods described above, that is good in nonstationary scenarios, can be used to solve this problem. But these methods only work when true action values change very slowly.
When a specific bandit task is selected and some distinctive clue (no the action values) about the task is provided to the learner, the learner can perform well with these clues than without them.

Example for the above assumption: Slot machine that changes the color of its display when it changes its action values. In this case the learner has to learn a policy that can associate each task (from the color of the display) to the best action to take when facing the task.
This is an example of an associative search task, because it involves both trial-and-error learning to search for the best actions and association of these actions with the situations in which they are best. These tasks are also referred as "Contextual Bandits". These tasks are intermediate between the k-armed bandit problem and the full RL problem.
Like full RL they learn a policy but like the k-armed bandit problem mentioned above each action affects only the intermediate reward. If the actions are allowed to affect the next situation as well as reward, it leads to the full RL problem.

2.10 Summary
The chapter has provided simple ways to balance exploration and exploitation. *The ε - greedy methods makes action selections randomly a small fraction of time. UCB methods choose deterministically but achieve exploration by subtly favoring at each step the actions that have so far received fewer samples. Gradient Bandit Algorithms estimate action preferences, instead of action values and favor the most preferred actions in a graded, probabilistic manner using soft-max distribution*. *Initializing the estimates optimistically causes even the greedy methods to do significant exploration.*

So which of the methods is best?
All of these methods are evaluated on the 10-armed testbed that was used in this chapter and their performances are compared. But these methods have their own parameters and the graphs so far have shown the performance of these methods, by showing for each method learning over time with different parameter settings. It will be difficult to understand if the same is done for all of these methods and summarized in a single graph. So, the average value over the 1000 steps in considered and this is proportional to the area under the learning curve.
The following plot shows this measure for various bandit algorithms, each as a function of its own parameter shown on a single scale on the x-axis. This graph is called as parameter study. The parameter values are varied by factors of two and represented in a log scale. The inverted U-shapes are observed for each algorithm's performance since all the algorithms perform best at an intermediate value of their parameter, neither too large not too small.

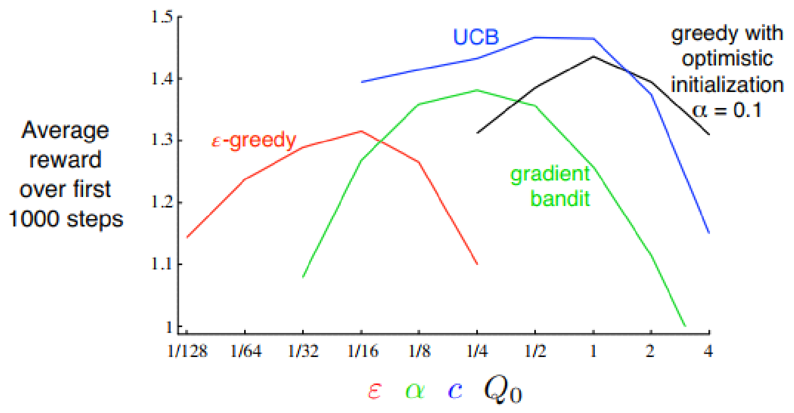


Figure Description: *A parameter study of the various bandit algorithms presented in this chapter. Each point is the average reward obtained over 1000 steps with a particular algorithm at a particular setting of its parameter.*

The method should not be evaluated based on just how well it is performing in its best parameter setting, but also to how sensitive it is to its parameter value. All of the algorithms evaluated are insensitive over a range of a parameter values. Overall, UCB performs well.