UNIT-II

Multi-arm Bandits: An n-Armed Bandit Problem, Action-Value Methods, Incremental Implementation, Tracking a Nonstationary Problem, Optimistic Initial Values, Upper-Confidence-Bound Action Selection, Gradient Bandits, Associative Search.

1. A k-armed Bandit Problem

- Consider the following learning 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, for example, over 1000 action selections, or time steps.
- This is the original form of the k-armed bandit problem, so named by analogy to a slot machine, or "one-armed bandit," 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.
- Through repeated action selections you are to maximize your winnings by concentrating your actions on the best levers.
- Another analogy is that of a doctor choosing between experimental treatments for a series of seriously ill patients.
- Each action is the selection of a treatment, and each reward is the survival or well-being of the patient.
- Today the term "bandit problem" is sometimes used for a generalization of the problem described above.
- In our k-armed bandit problem, 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.
- We denote the action selected on time step t as A_t, and the corresponding reward as R_t. The value then of an arbitrary action a, denoted $q_*(a)$, is the expected reward given that a is selected:

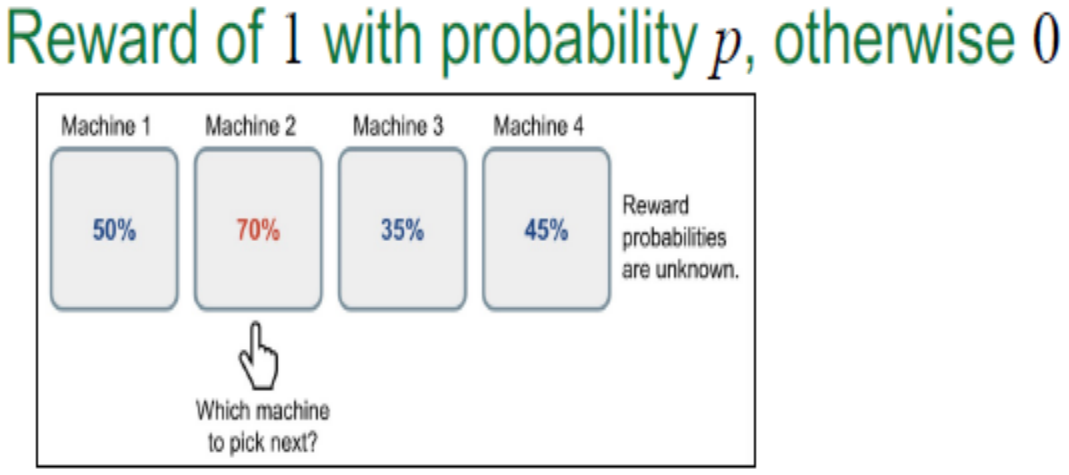
$$q_*(a) \doteq \mathbb{E}[R_t \mid A_t = a].$$

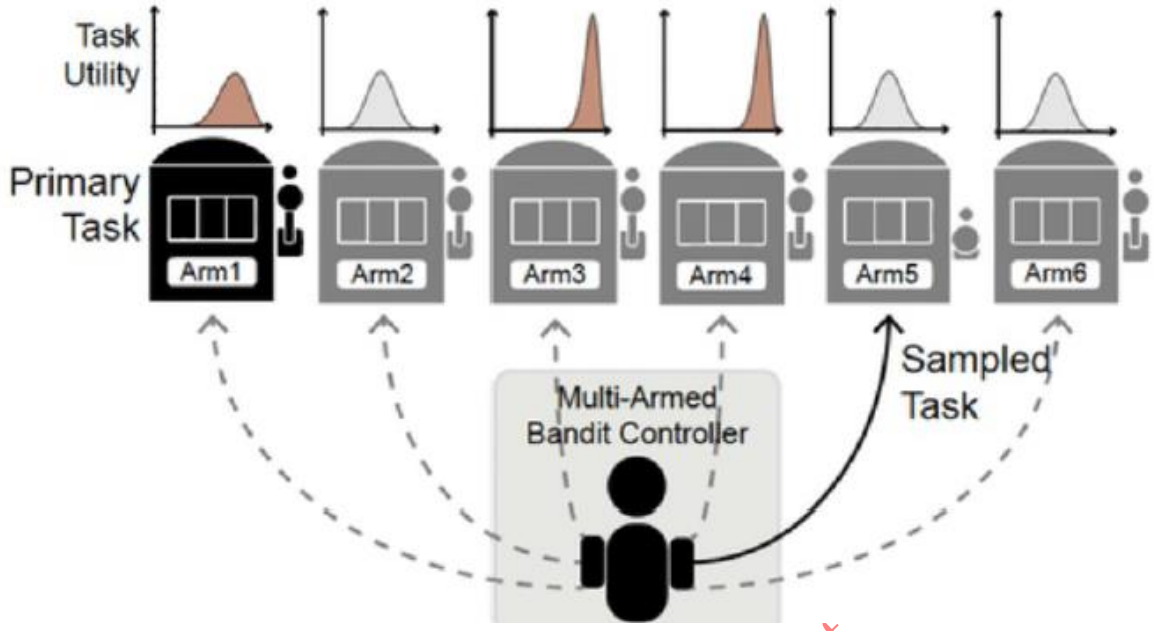
- If you knew the value of each action, then it would be trivial to solve the k-armed bandit problem: you would always select the action with highest value.
- We assume that you do not know the action values with certainty, although you may have estimates.
- We denote the estimated value of action a at time step t as $Q_t(a)$. We would like $Q_t(a)$ to be close to $q_*(a)$.
- If you maintain estimates of the action values, then at any time step there is at least one action whose estimated value is greatest. We call these the greedy actions.
- When you select one of these actions, we say that you are exploiting your current knowledge of the values of the actions.
- If instead you select one of the nongreedy actions, then we say you are exploring, because this enables you to improve your estimate of the nongreedy action's value.
- Exploitation is the right thing to do to maximize the expected reward on the one step, but exploration may produce the greater total reward in the long run.

- For example, suppose a greedy action's value is known with certainty, while several other actions are estimated to be nearly as good but with substantial uncertainty.
- The uncertainty is such that at least one of these other actions probably is actually better than the greedy action, but you don't know which one.
- If you have many time steps ahead on which to make action selections, then it may be better to explore the nongreedy actions and discover which of them are better than the greedy action.
- 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.
- Because it is not possible both to explore and to exploit with any single action selection, one often refers to the "conflict" between exploration and exploitation.

Analysis:

- The multi-armed bandit problem is used in reinforcement learning to formalize the notion of decision-making under uncertainty.
- In a multi-armed bandit problem, an agent(learner) chooses between *k* different actions and receives a reward based on the chosen action. The multi-armed bandits are also used to describe fundamental concepts in reinforcement learning, such as *rewards*, *timesteps*, and *values*.
- A bandit is defined as someone who steals your money. A one-armed bandit is a simple slot machine wherein you insert a coin into the machine, pull a lever, and get an immediate reward.

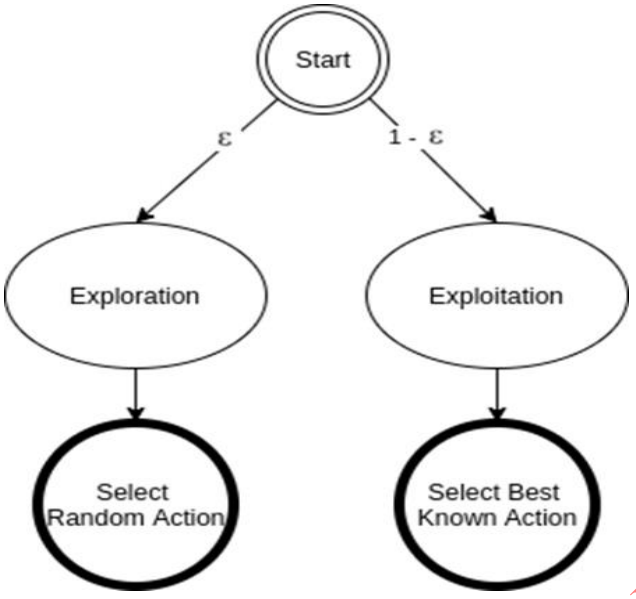




- A doctor choosing between experimental treatments for a series of seriously ill patients – Action is a treatment – Reward is survival or death of the patient.
- A classic problem named a multi-armed bandit problem is an example of a greedy epsilon algorithm.
- In this problem n arms or bandits are provided to the machine with the probability rate of success.
- Each arm is pulled out one by one, and based on the success rate, the reward is given. If a success, the reward is incremented by 1; otherwise, the reward is 0.
- Our objective is to pull out the arms sequentially to have the maximum reward at the end.

Exploration-Exploitation in Epsilon Greedy Algorithm

- Exploitation is when the agent knows all his options and chooses the best option based on the previous success rates.
- Whereas exploration is the concept where the agent is unaware of his opportunities and tries to explore other options to better predict and earn rewards.
- Going to the same pizza parlor and ordering the best pizza is an example of exploitation; here, the user knows all his options and selects the best option.
- On the other hand, going to the new Chinese restaurant and trying new things would come under exploration.
- The user is exploring new things; it could be better or worse, the user is unaware of the result.



2. Action-Value Methods

- We begin by looking more closely at methods for estimating the values of actions and for using the estimates to make action selection decisions, which we collectively call action-value methods.
- Recall that the true value of an action is the mean reward when that action is selected. One natural way to estimate this is by averaging the rewards actually received:

$$Q_t(a) \doteq \frac{\text{sum of rewards when } a \text{ taken prior to } t}{\text{number of times } a \text{ taken prior to } t} = \frac{\sum_{i=1}^{t-1} R_i \cdot \mathbb{1}_{A_i=a}}{\sum_{i=1}^{t-1} \mathbb{1}_{A_i=a}}, \tag{2.1}$$

where 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 $q_*(a)$.

- We call this the **sample-average** method for estimating action values because each estimate is an average of the sample of relevant rewards.
- The simplest action selection rule is to select one of the actions with the highest estimated value, that is, one of the greedy actions as defined in the previous section.
- If there is more than one greedy action, then a selection is made among them in some arbitrary way, perhaps randomly. We write this **greedy action selection method** as

$$A_t \doteq \underset{a}{\operatorname{argmax}}\, Q_t(a), \tag{2.2}$$

Where argmax_a denotes the action a for which the expression that follows is maximized.

- Greedy action selection always exploits current knowledge to maximize immediate reward; it spends no time at all sampling apparently inferior actions to see if they might really be better.

- A simple alternative is to behave greedily most of the time, but every once in a while, say with small probability ε, instead select randomly from among all the actions with equal probability, independently of the action-value estimates.
- We call methods using this near-greedy action selection rule " ε-greedy methods. An advantage of these methods is that, in the limit as the number of steps increases, every action will be sampled an infinite number of times, thus ensuring that all the $Q_t(a)$ converge to $q_*(a)$.
- This of course implies that the probability of selecting the optimal action converges to greater than $1-\varepsilon$, that is, to near certainty.
- To roughly assess the relative effectiveness of the greedy and " ε-greedy methods, we compared them numerically on a suite of test problems.
- This was a set of 2000 randomly generated n-armed bandit tasks with n = 10. For each bandit, the action values, q(a), a = 1; : : : ; 10, were selected according to a normal (Gaussian) distribution with mean 0 and variance 1.
- On tth time step with a given bandit, the actual reward Rt was the q(At) for the bandit (where
- At was the action selected) plus a normally distributed noise term that was mean 0 and variance 1.
- Averaging over bandits, we can plot the performance and behavior of various methods as they improve with experience over 1000 steps, as in Figure 2.1. We call this suite of test tasks the 10-armed testbed.

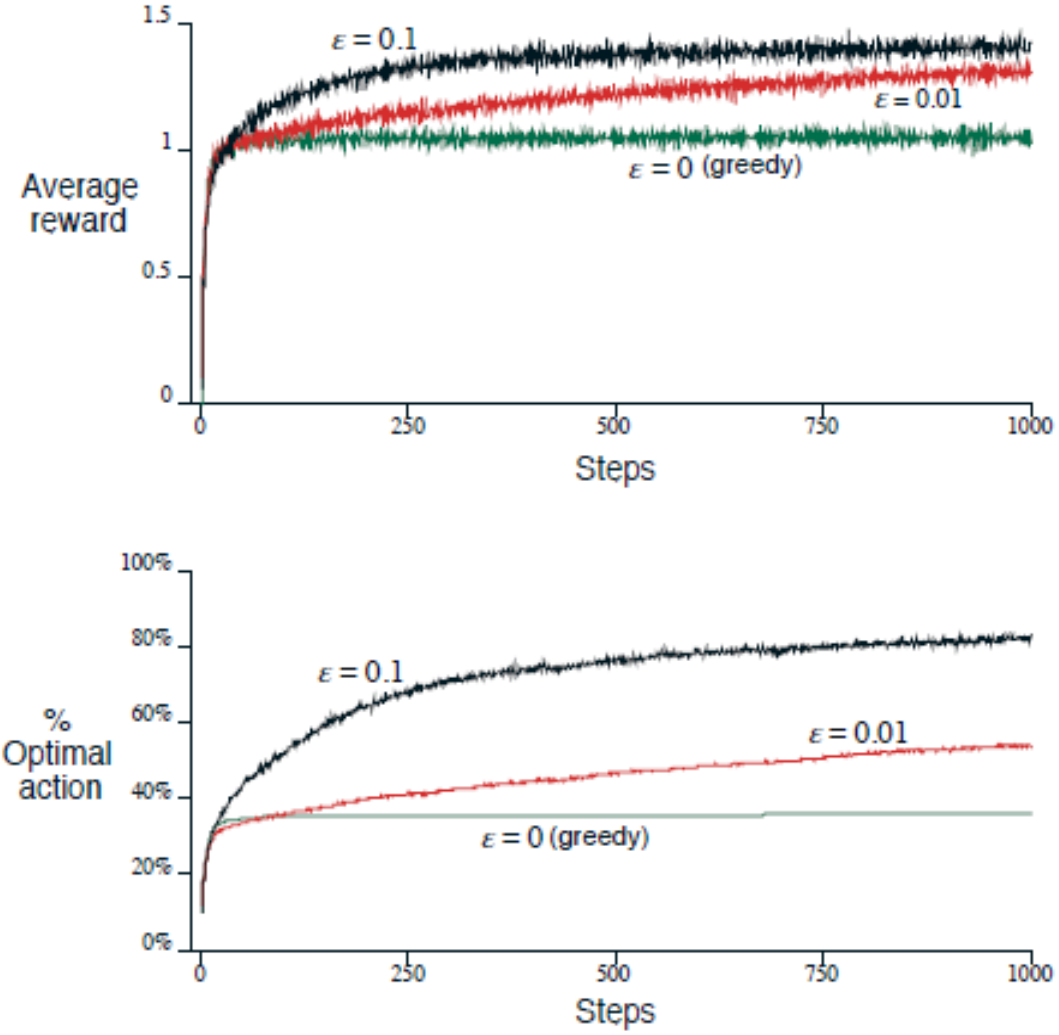


Figure 2.1: Average performance of ε-greedy action-value methods on the 10-armed testbed. These data are averages over 2000 tasks. All methods used sample averages as their action-value estimates. The detailed structure at the beginning of these curves depends on how actions are selected when multiple actions have the same maximal action value. Here such ties were broken randomly. An alternative that has a similar effect is to add a very small amount of randomness to each of the initial action values, so that ties effectively never happen.

- Figure 2.1 compares a greedy method with two ε-greedy methods (ε = 0.01 and ε = 0.1), as described above, on the 10-armed testbed. Both methods formed their action-value estimates 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 levelled off at a lower level.
- It achieved a reward per step of only about 1, compared with the best possible of about 1.55 on this testbed.

- The greedy method performs significantly worse in the long run because it often gets stuck performing suboptimal actions.
- The lower graph shows that the greedy method found the optimal action in only approximately one-third of the tasks. In the other two-thirds, its initial samples of the optimal action were disappointing, and it never returned to it.
- The ε-greedy methods eventually perform better because they continue to explore, and to improve their chances of recognizing the optimal action.
- The ε = 0.1 method explores more, and usually finds the optimal action earlier, but never selects it more than 91% of the time.
- The ε = 0.01 method improves more slowly, but eventually performs better than the ε = 0.1 method on both performance measures. It is also possible to reduce ε over time to try to get the best of both high and low values.
- The advantage of ε-greedy over greedy methods depends on the task. For example, suppose the reward variance had been larger, say 10 instead of 1. With noisier rewards it takes more exploration to find the optimal action, and ε-greedy methods should fare even better relative to the greedy method.

3. Incremental Implementation

- The action-value methods we have discussed so far all estimate action values as sample averages of observed rewards.
- We now turn to the question of how these averages can be computed in a computationally efficient manner in particular, with constant memory and constant per-time-step computation.
- To simplify notation, we concentrate on a single action. Let R_i now denote the reward received after the i^{th} selection of this action, and let Q_n denote the estimate of its action value after it has been selected n-1 times which we can now write simply as

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

- The obvious implementation would be to maintain a record of all the rewards and then perform this computation whenever the estimated value was needed.
- However, if this is done, then the memory and computational requirements would 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.
- As you might suspect, this is not really necessary.
- It is easy to devise incremental formulas for updating averages with small, constant computation required to process each new reward.
- Given Q_n and the nth reward, R_n, the new average of all n rewards can be computed by

$$
\begin{aligned}
Q_{n+1} &= \frac{1}{n}\sum_{i=1}^{n} R_i \\
&= \frac{1}{n}\left(R_n + \sum_{i=1}^{n-1} R_i\right) \\
&= \frac{1}{n}\left(R_n + (n-1)\frac{1}{n-1}\sum_{i=1}^{n-1} R_i\right) \\
&= \frac{1}{n}\Big(R_n + (n-1)Q_n\Big) \\
&= \frac{1}{n}\Big(R_n + nQ_n - Q_n\Big) \\
&= Q_n + \frac{1}{n}\Big[R_n - Q_n\Big],
\end{aligned}
\qquad 2.3
$$

- Which holds even 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 (2.3) for each new reward.
- This update rule (2.3) is of a form that occurs frequently throughout,
- The general form is

$$NewEstimate \leftarrow OldEstimate + StepSize\Big[Target - OldEstimate\Big]. \qquad (2.4)$$

- The expression [Target-Old Estimate] is an error in the estimate.
- It is reduced by taking a step toward the "Target." The target is presumed to indicate a desirable direction in which to move, though it may be noisy.
- In the case above, for example, the target is the nth reward.
- Note that the step-size parameter (Step Size) used in the incremental method (2.3) changes from time step to time step.
- In processing the nth reward for action a, the method uses the step-size parameter 1 /n we denote the step-size parameter by α or, more generally, by $\alpha_t(a)$.
- Pseudo code for a complete bandit algorithm using incrementally computed sample averages and "-greedy action selection is shown in the box below. The function bandit(a) is assumed to take an action and return a corresponding reward

A simple bandit algorithm

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

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

4.Tracking a Nonstationary Problem

- The averaging methods discussed so far are appropriate for stationary bandit problems, that is, for bandit problems in which the reward probabilities do not change over time.
- As noted earlier, we often encounter reinforcement learning problems that are effectively nonstationary.
- In such cases it makes sense to give more weight to recent rewards than to long-past rewards. One of the most popular ways of doing this is to use a constant step-size parameter.
- For example, the incremental update rule (2.3) for updating an average Q_n of the n-1 past rewards is modified to be

$$Q_{n+1} \doteq Q_n + \alpha\Big[R_n - Q_n\Big], \tag{2.5}$$

where the step-size parameter $\alpha \in (0, 1]$ is constant. This results in Q_{n+1} being a weighted average of past rewards and the initial estimate Q_1:

$$\begin{aligned} Q_{n+1} &= Q_n + \alpha\Big[R_n - Q_n\Big] \\ &= \alpha R_n + (1-\alpha)Q_n \\ &= \alpha R_n + (1-\alpha)\left[\alpha R_{n-1} + (1-\alpha)Q_{n-1}\right] \\ &= \alpha R_n + (1-\alpha)\alpha R_{n-1} + (1-\alpha)^2 Q_{n-1} \\ &= \alpha R_n + (1-\alpha)\alpha R_{n-1} + (1-\alpha)^2\alpha R_{n-2} + \\ &\qquad \cdots + (1-\alpha)^{n-1}\alpha R_1 + (1-\alpha)^n Q_1 \\ &= (1-\alpha)^n Q_1 + \sum_{i=1}^{n} \alpha(1-\alpha)^{n-i} R_i. \end{aligned} \tag{2.6}$$

- We call this a weighted average because the sum of the weights is

$$(1-\alpha)^n + \textstyle\sum_{i=1}^{n} \alpha(1- \ \alpha)^{n-i} = 1,$$

- Note that the weight, $\alpha(1-\alpha)^{n-i}$, given to the reward R_i depends on how many rewards ago, n-I it was observed. The quantity $1-\alpha$ is less than 1.
- And thus the weight given to Ri decreases as the number of intervening rewards increases.
- In fact, the weight decays exponentially according to the exponent on $1-\alpha$.
- Accordingly, this is sometimes called an exponential recency-weighted average.
- Sometimes it is 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 nth selection of action a. As we have noted, the choice $\alpha_n(a)$ = 1/n results in the sample-average method, which is guaranteed to converge to the true action values by the law of large numbers.
- But of course convergence is not guaranteed for all choices of the sequence $\{\alpha_n(a)\}$.
- A well-known result in stochastic approximation theory gives us the conditions required to assure convergence with probability 1:

$$\sum_{n=1}^{\infty} \alpha_n(a) = \infty \qquad \text{and} \qquad \sum_{n=1}^{\infty} \alpha_n^2(a) < \infty. \tag{2.7}$$

- The first condition is required to guarantee that the steps are large enough to eventually overcome any initial conditions or random fluctuations.
- The second condition guarantees that eventually the steps become small enough to assure convergence.
- Note that both convergence conditions are met for the sample-average case, $\alpha_n(a) = \frac{1}{n}$, , but not for the case of constant step-size parameter, $\alpha_n(a) = \alpha$. .
- In the latter case, the second condition is not met, indicating that the estimates never completely converge but continue to vary in response to the most recently received rewards.
- As we mentioned above, this is actually desirable in a nonstationary environment, and problems that are effectively nonstationary are the most common in reinforcement learning.
- In addition, sequences of step-size parameters that meet the conditions (2.7) often converge very slowly or need considerable tuning in order to obtain a satisfactory convergence rate.
- Although sequences of step-size parameters that meet these convergence conditions are often used in theoretical work, they are seldom used in applications and empirical research.

5. Optimistic Initial Values

- All the methods we have discussed so far are dependent to some extent on the initial action-value estimates, $Q_1(a)$. In the language of statistics, these methods are biased by their initial estimates.
- For the sample-average methods, the bias disappears once all actions have been selected at least once, but for methods with constant α, the bias is permanent, though decreasing over time as given by (2.6).
- In practice, this kind of bias is usually not a problem and can sometimes be very helpful. T
- The downside is that the initial estimates become, in effect, a set of parameters that must be picked by the user, if only to set them all to zero.
- The upside is that they provide an easy way to supply some prior knowledge about what level of rewards can be expected.
- Initial action values can also be used as a simple way to encourage exploration.
- Suppose that instead of setting the initial action values to zero, as we did in the 10-armed testbed, we set them all to +5.
- Recall that the $q_*(a)$ in this problem are selected from a normal distribution with mean 0 and variance 1.
- An initial estimate of +5 is thus wildly optimistic.
- But this optimism encourages action-value methods to explore.
- 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.
- The result is that all actions are tried several times before the value estimates converge. The system does a fair amount of exploration even if greedy actions are selected all the time.
- Figure 2.3 shows the performance on the 10-armed bandit testbed of a greedy method using $Q_1(a) = +5$, for all a.
- For comparison, also shown is an "-greedy method with Q1(a) = 0.
- Initially, the optimistic method performs worse because it explores more, but eventually it performs better because its exploration decreases with time.
- We call this technique for encouraging exploration optimistic initial values.
- We regard it as a simple trick that can be quite effective on stationary problems, but it is far from being a generally useful approach to encouraging exploration.
- For example, it is not well suited to nonstationary problems because its drive for exploration is inherently temporary.

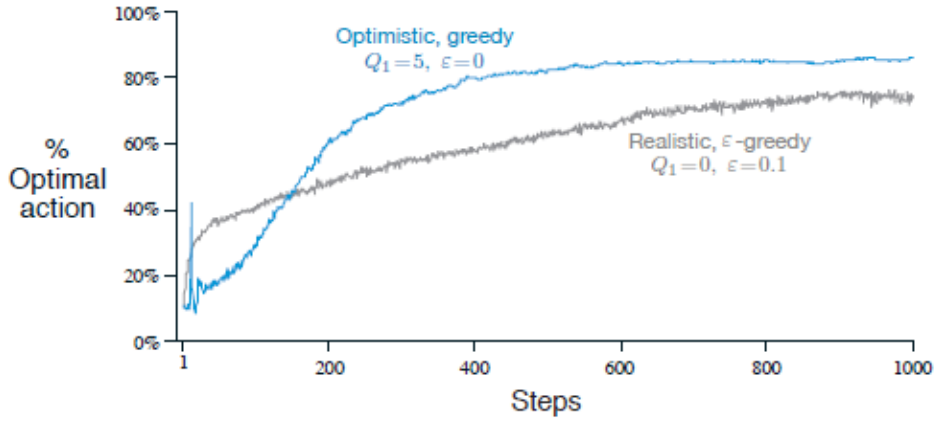


Figure 2.3: The effect of optimistic initial action-value estimates on the 10-armed testbed. Both methods used a constant step-size parameter, $\alpha = 0.1$.

- If the task changes, creating a renewed need for exploration, this method cannot help.
- Indeed, any method that focuses on the initial conditions in any special way is unlikely to help with the general nonstationary case.
- The beginning of time occurs only once, and thus we should not focus on it too much.
- This criticism applies as well to the sample-average methods, which also treat the beginning of time as a special event, averaging all subsequent rewards with equal weights.

6. Upper-Confidence-Bound Action Selection

- Exploration is needed because there is always uncertainty about the accuracy of the action-value estimates.
- The greedy actions are those that look best at present, but some of the other actions may actually be better.
- The 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.
- It would be better to select among 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 of doing this is to select actions according to

$$A_t \doteq \underset{a}{\operatorname{argmax}} \left[Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}} \right], \qquad (2.10)$$

where ln t denotes the natural logarithm of t (the number that e = 2.71828 would have to be raised to in order to equal t), $N_t(a)$ denotes the number of times that action a has

been selected prior to time t (the denominator in (2.1)), 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 idea of this upper confidence bound (UCB) action selection is that the square-root term is a measure of the uncertainty or variance in the estimate of a's value.
- The quantity being max'ed over is thus a sort of upper bound on the possible true value of action a, with c determining the confidence level.
- Each time a is selected the uncertainty is presumably reduced: $N_t(a)$ increments, and, as it appears in the denominator, the uncertainty term decreases.
- On the other hand, each time an action other than a is selected, t increases but $N_t(a)$ does not; because t appears in the numerator, the uncertainty estimate increases.
- The use of the natural logarithm means that the increases get smaller over time, but are unbounded; all actions will eventually be selected, but actions with lower value estimates, or that have already been selected frequently, will be selected with decreasing frequency over time.
- Results with UCB on the 10-armed testbed are shown in Figure 2.4. UCB often performs well, as shown here, but is more difficult than ε-greedy to extend beyond bandits to the more general reinforcement learning settings.
- One difficulty is s in dealing with nonstationary problems; methods more complex than those presented in Section 2.5 would be needed. Another difficulty is dealing with large state spaces, particularly when using function approximation as developed in Part II of this book. In these more advanced settings the idea of UCB action selection is usually not practical.

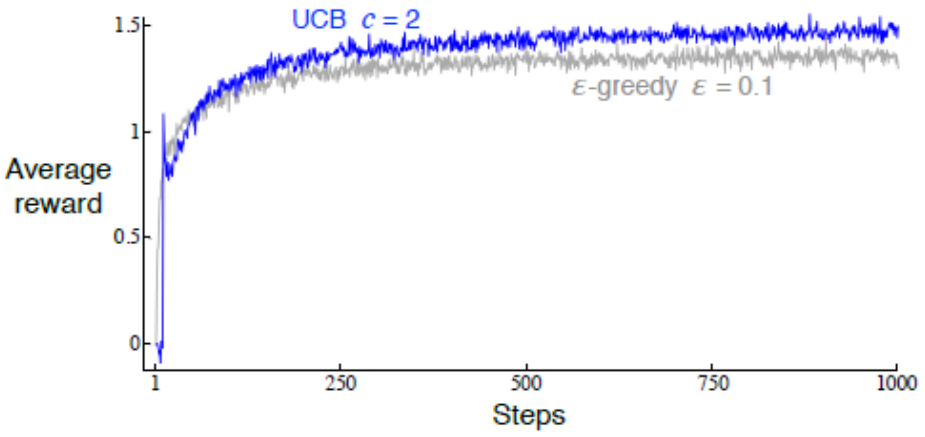


Figure 2.4: Average performance of UCB action selection on the 10-armed testbed. As shown, UCB generally performs better than ε-greedy action selection, except in the first k steps, when it selects randomly among the as-yet-untried actions.

7. Gradient Bandit Algorithms

- In this section we consider learning a numerical preference for each action a, which we denote $H_t(a)$.
- The larger the preference, the more often that action is taken, but the preference has no interpretation in terms of reward.
- Only the relative preference of one action over another is important; if we add 1000 to all the action preferences there is no effect on the action probabilities, which are determined according to a soft-max distribution (i.e., Gibbs or Boltzmann distribution) as follows:

$$\Pr\{A_t = a\} \doteq \frac{e^{H_t(a)}}{\sum_{b=1}^{k} e^{H_t(b)}} \doteq \pi_t(a), \qquad (2.11)$$

  where here we have also introduced a useful new notation, $\pi_t(a)$, for the probability of taking action a at time t. Initially all action preferences are the same (e.g., $H_1(a) = 0$, for all a) so that all actions have an equal probability of being selected.
- There is a natural learning algorithm for this setting based on the idea of stochastic gradient ascent. On each step, after selecting action At and receiving the reward R_t, the action preferences are updated by:

$$\begin{aligned} H_{t+1}(A_t) &\doteq H_t(A_t) + \alpha(R_t - \bar{R}_t)(1 - \pi_t(A_t)), && \text{and} \\ H_{t+1}(a) &\doteq H_t(a) - \alpha(R_t - \bar{R}_t)\pi_t(a), && \text{for all } a \neq A_t, \end{aligned} \qquad (2.12)$$

  where $\alpha > 0$ is a step-size parameter, and $\bar{R}_t \in \mathbb{R}$ is the average of all the rewards up through and including time t, which can be computed incrementally as described.
- The $\bar{R}_t$ term serves as a baseline with which the reward is compared.
- 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 below baseline, then probability is decreased.
- The non-selected actions move in the opposite direction.
- Figure 2.5 shows results with the gradient bandit algorithm on a variant of the 10-armed testbed in which the true expected rewards were selected according to a normal distribution with a mean of +4 instead of zero (and with unit variance as before).
- This shifting up of all the rewards has absolutely no effect on the gradient bandit algorithm because of the reward baseline term, which instantaneously adapts to the new level.
- But if the baseline were omitted (that is, if $\bar{R}_t$ was taken to be constant zero in (2.12)), then performance would be significantly degraded, as shown in the figure.

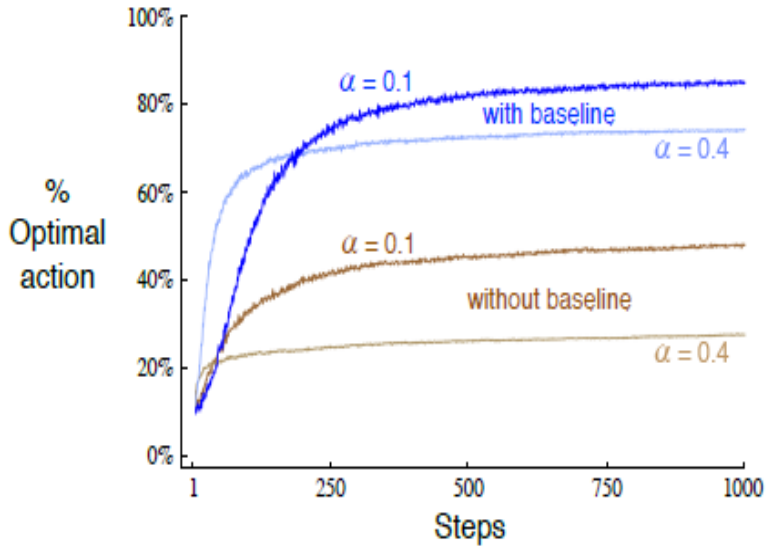


Figure 2.5: Average performance of the gradient bandit algorithm with and without a reward baseline on the 10-armed testbed when the $q_*(a)$ are chosen to be near $+4$ rather than near zero.

The Bandit Gradient Algorithm as Stochastic Gradient Ascent

One can gain a deeper insight into the gradient bandit algorithm by understanding it as a stochastic approximation to gradient ascent. In exact *gradient ascent*, each action preference $H_t(a)$ would be incremented proportional to the increment's effect on performance:

$$H_{t+1}(a) \doteq H_t(a) + \alpha \frac{\partial \mathbb{E}[R_t]}{\partial H_t(a)}, \tag{2.13}$$

where the measure of performance here is the expected reward:

$$\mathbb{E}[R_t] = \sum_x \pi_t(x) q_*(x),$$

and the measure of the increment's effect is the *partial derivative* of this performance measure with respect to the action preference. Of course, it is not possible to implement gradient ascent exactly in our case because by assumption we do not know the $q_*(x)$, but in fact the updates of our algorithm (2.12) are equal to (2.13) in expected value, making the algorithm an instance of *stochastic gradient ascent*. The calculations showing this require only beginning calculus, but take several

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

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

where B_t, called the *baseline* , can be any scalar that does not depend on x. We can include a baseline here without changing the equality because the gradient sums to zero over all the actions, $\sum_x \frac{\partial \pi_t(x)}{\partial H_t(a)} = 0$—as $H_t(a)$ is changed, some actions' probabilities go up and some go down, but the sum of the changes must be zero because the sum of the probabilities is always one.

Next we multiply each term of the sum by $\pi_t(x)/\pi_t(x)$:

$$\frac{\partial \mathbb{E}[R_t]}{\partial H_t(a)} = \sum_x \pi_t(x)\big(q_*(x) - B_t\big)\frac{\partial \pi_t(x)}{\partial H_t(a)}/\pi_t(x).$$

The equation is now in the form of an expectation, summing over all possible values x of the random variable A_t, then multiplying by the probability of taking those values. Thus:

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

where here we have chosen the baseline $B_t = \bar{R}_t$ and substituted R_t for $q_*(A_t)$, which is permitted because $\mathbb{E}[R_t|A_t] = q_*(A_t)$. Shortly we will establish that $\frac{\partial \pi_t(x)}{\partial H_t(a)} = \pi_t(x)\big(\mathbb{1}_{a=x} - \pi_t(a)\big)$, where $\mathbb{1}_{a=x}$ is defined to be 1 if $a = x$, else 0. Assuming that for now, we have

$$\begin{aligned}&= \mathbb{E}\big[\big(R_t - \bar{R}_t\big)\pi_t(A_t)\big(\mathbb{1}_{a=A_t} - \pi_t(a)\big)/\pi_t(A_t)\big] \\ &= \mathbb{E}\big[\big(R_t - \bar{R}_t\big)\big(\mathbb{1}_{a=A_t} - \pi_t(a)\big)\big].\end{aligned}$$

Recall that our plan has been to write the performance gradient as an expectation of something that we can sample on each step, as we have just done, and then update on each step proportional to the sample. Substituting a sample of the expectation above for the performance gradient in (2.13) yields:

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

which you may recognize as being equivalent to our original algorithm (2.12).

Thus it remains only to show that $\frac{\partial \pi_t(x)}{\partial H_t(a)} = \pi_t(x)\big(\mathbb{1}_{a=x} - \pi_t(a)\big)$, as we assumed. Recall the standard quotient rule for derivatives:

$$\frac{\partial}{\partial x}\left[\frac{f(x)}{g(x)}\right] = \frac{\frac{\partial f(x)}{\partial x}g(x) - f(x)\frac{\partial g(x)}{\partial x}}{g(x)^2}.$$

Using this, we can write

$$\begin{aligned}
\frac{\partial \pi_t(x)}{\partial H_t(a)} &= \frac{\partial}{\partial H_t(a)}\pi_t(x) \\
&= \frac{\partial}{\partial H_t(a)}\left[\frac{e^{H_t(x)}}{\sum_{y=1}^{k} e^{H_t(y)}}\right] \\
&= \frac{\frac{\partial e^{H_t(x)}}{\partial H_t(a)}\sum_{y=1}^{k} e^{H_t(y)} - e^{H_t(x)}\frac{\partial \sum_{y=1}^{k} e^{H_t(y)}}{\partial H_t(a)}}{\left(\sum_{y=1}^{k} e^{H_t(y)}\right)^2} && \text{(by the quotient rule)} \\
&= \frac{\mathbb{1}_{a=x}e^{H_t(x)}\sum_{y=1}^{k} e^{H_t(y)} - e^{H_t(x)}e^{H_t(a)}}{\left(\sum_{y=1}^{k} e^{H_t(y)}\right)^2} && \text{(because } \tfrac{\partial e^x}{\partial x} = e^x\text{)} \\
&= \frac{\mathbb{1}_{a=x}e^{H_t(x)}}{\sum_{y=1}^{k} e^{H_t(y)}} - \frac{e^{H_t(x)}e^{H_t(a)}}{\left(\sum_{y=1}^{k} e^{H_t(y)}\right)^2} \\
&= \mathbb{1}_{a=x}\pi_t(x) - \pi_t(x)\pi_t(a) \\
&= \pi_t(x)\big(\mathbb{1}_{a=x} - \pi_t(a)\big). && \text{Q.E.D.}
\end{aligned}$$

We have just shown that the expected update of the gradient bandit algorithm is equal to the gradient of expected reward, and thus that the algorithm is an instance of stochastic gradient ascent. This assures us that the algorithm has robust convergence properties.

Note that we did not require any properties of the reward baseline other than that it does not depend on the selected action. For example, we could have set it to zero, or to 1000, and the algorithm would still be an instance of stochastic gradient ascent. The choice of the baseline does not affect the expected update of the algorithm, but it does affect the variance of the update and thus the rate of convergence (as shown, e.g., in Figure 2.5). Choosing it as the average of the rewards may not be the very best, but it is simple and works well in practice.

8. Associative Search (Contextual Bandits)

- So far in this chapter we have considered only nonassociative tasks, that is, tasks in which there is no need to associate different actions with different situations.
- In these tasks the learner either 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.
- However, in a general reinforcement learning task there is more than one situation, and the goal is to learn a policy: a mapping from situations to the actions that are best in those situations.
- To set the stage for the full problem, we briefly discuss the simplest way in which nonassociative tasks extend to the associative setting.
- As an example, suppose there are several different k-armed bandit tasks, and that on each step you confront one of these chosen at random. Thus, the bandit task changes randomly from step to step.
- This would appear to you as a single, nonstationary k-armed bandit task whose true action values change randomly from step to step.
- Now suppose, however, that when a bandit task is selected for you, you are given some distinctive clue about its identity (but not its action values).
- Maybe you are facing an actual slot machine that changes the colour of its display as it changes its action values.
- Now you can learn a policy associating each task, signalled by the colour you see, with the best action to take when facing that task—for instance, if red, select arm 1; if green, select arm 2.
- With the right policy you can usually do much better than you could in the absence of any information distinguishing one bandit task from another.
- This is an example of an associative search task, so called 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.
- Associative search tasks are often now called contextual bandits in the literature.
- Associative search tasks are intermediate between the k-armed bandit problem and the full reinforcement learning problem.
- They are like the full reinforcement learning problem in that they involve learning a policy, but like our version of the k-armed bandit problem in that each action affects only the immediate reward.
- If actions are allowed to affect the next situation as well as the reward, then we have the full reinforcement learning problem.
- Suppose you face a 2-armed bandit task whose true action values change randomly from time step to time step. Specifically, suppose that, for any time step, the true values of actions 1 and 2 are respectively 0.1 and 0.2 with probability 0.5 (case A), and 0.9 and 0.8 with probability 0.5 (case B).
- If you are not able to tell which case you face at any step, what is the best expectation of success you can achieve and how should you behave to achieve it? Now suppose

that on each step you are told whether you are facing case A or case B (although you still don't know the true action values).

- This is an associative search task. What is the best expectation of success you can achieve in this task, and how should you behave to achieve it.