Summary: Reinforcement Learning - An Introduction

Chapter 2

Multi-armed Bandits

The most important feature distinguishing R.L. from other types of learning is that it uses training information that *evaluates* the actions taken rather than *instructs* by giving correct actions. In their pure forms, two kinds of feedback are quite distinct: evaluative feedback depends entirely on the action taken, whereas instructive feedback is independent of the action taken.

The evaluative aspect of R.L. in a simplified setting, one that does not involve learning to act in more than one situation. (*nonassociativity*) The particular nonassociative, evaluative feedback problem that we explore is a simple version of the k-armed bandit problem.

2.1 A k-armed Bandit Problem

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 we are to maximize our winnings by concentrating our actions on the best levers.

In classical 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. 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 we knew the value of each action, then it would be trivial to solve the k-armed bandit problem: we would always select the action with highest value. We assume that we do not know the action values with certainty, although we 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 we 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 we select one of these actions, we say that we are *exploiting* our current knowledge of the

values of the actions. If instead we select one of the nongreedy actions, then we say we are *exploring*, because this enables us to improve our estimate of the nongreedy action's value. The uncertainty is such that at least one of these other actions probably is actually better than the greedy action, but we don't know which one. Reward is lower in the short run, during exploration, but higher in the long run because after we have discovered the better actions, we can exploit them many times. In any specific case, whether it is better to explore or exploit depends in a complex way on the precise values of the estimates, uncertainties, and the number of remaining steps.

2.2 Action-value Methods

Action-value methods are collectively called for using the estimates to make action selection decisions and for estimating the values of actions. **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:

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

where $\mathbb{1}_{predicate}$ 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 $q_*(a)$. (*sample-average*) If there is more than one greedy action, then a selection is made among actions 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. 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. Using this near-greedy action selection rule is called as *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 numer of times, thus ensuring that all the $Q_t(a)$ converge to $q_*(a)$. The probability of selecting the optimal action converges to greater than $1-\varepsilon$, that is, to near certainty. These are just asymptotic guarantees, however, and say little about the practical effectiveness of the methods.

2.3 The 10-armed Testbed

Read the pages between 28 and 30.

For any learning method, we can measure its performance and behavior as it improves with experience over 1000 time steps when applied to one of the bandit problems. This makes up one *run*. Repeating this for 2000 independent runs, each with a different bandit problem, we obtained measures of the learning algorithm's average behavior.

The greedy method improved slightly faster than the other methods at the very beginning, but then leveled off at a lower level. The greedy method performed significantly worse in the long run because it often got stuck performing suboptimal actions. The $\varepsilon - greedy$ methods eventually performed better because they continued to explore and to improve their chances of recognizing the optimal action. It is also possible to reduce ε over time to try to get the best of both high and low values.

The advantage of $\varepsilon - greedy$ 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 $\varepsilon - greedy$ methods should fare even better relative to the greedy method. On the other hand, if the reward variances were zero, then the greedy method would know the true value of each action after trying it once. In this case the greedy method might actually perform best because it would soon find the optimal action and then never explore. But even in the deterministic case there is a large advantage to exploring if we weaken some of the other assumptions. For example, suppose the bandit task were nonstationary, that is, the true values of the actions changed over time. In this case exploration is needed even in the deterministic case to make sure one of the nongreedy actions has not changed to become better than the greedy one. **Nonstationarity is the case most commonly encountered in reinforcement learning. R.L. requires a balance between exploration and exploitation.**

2.4 Incremental Implementation

The question of how sample averages can be computed in a computationally efficient manner, in particular, with constant memory and constant per-time-step computation is now turned. Let R_i now denote the reward received after the ith 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

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

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. 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

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

$$
\begin{aligned}
&= \frac{1}{n}\Big(R_n + (n-1)Q_n\Big) \\
&= \frac{1}{n}\Big(R_n + nQ_n - Q_n\Big) \\
&= Q_n + \frac{1}{n}\Big[R_n - Q_n\Big], \qquad (2.3)
\end{aligned}
$$

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 can be written in the general forms as follows

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

The expression $\big[Target - OldEstimate\big]$ is an *error* in the estimate. It is reduced by taking a step forward to "Target". The target presumed to indicate a desirable direction in which to move, though it may be noisy. Note that the step-size parameter *StepSize* used in the incremental method 2.3 changes from time step to time step.

Pseudocode for a complete bandit algorithm using incrementally computed sample averages and $\varepsilon-greedy$ action selection is shown in the box below. The function $bandit(a)$ is assumed to take an action and return a corresponding reward.

Algorithm 1 A simple bandit algorithm

Initialize, for $a = 1$ to k:
$Q(a) \leftarrow 0$
$N(a) \leftarrow 0$
Loop forever:
$$A \leftarrow \begin{cases} \text{argmax}_a\, Q_t(a) & \text{with probability } 1-\varepsilon \quad \text{(breaking ties randomly)} \\ \text{a random action with probability } \varepsilon \end{cases}$$
$R \leftarrow bandit(A)$
$N(A) \leftarrow N(A) + 1$
$Q(A) \leftarrow Q(A) + \frac{1}{N(A)}\big[R - Q(A)\big]$

2.5 Tracking a Nonstationary Problem

R.L. 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. The incremental update rule 2.3 for updating an average Q_n of the $n-1$ past rewards is modified to be

$$Q_{n+1} \doteq Q_n + \alpha\big[R_n - Q_n\big], \qquad (2.5)$$

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:

$$
\begin{aligned}
Q_{n+1} &= Q_n + \alpha\big[R_n - Q_n\big] \\
&= \alpha R_n + (1-\alpha)Q_n
\end{aligned}
$$

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

We call this weighted average because the sum that of the the weights is $(1-\alpha)^n + \sum_{i=1}^{n} \alpha(1-\alpha)^{n-i} = 1$. The quantity $1-\alpha$ is less than 1, and thus the weight given to R_i decreases as the number of intervening rewards increases. In fact, the weight decays exponentially according to the exponent on $1-\alpha$. (If $1-\alpha = 0$, then all the weight goes on the very last reward, R_n, because of the convention that $0^0 = 1$.) (*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. 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:

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

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, $\alpha_n(a) = a$. 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. 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.

Exercise 2.4 Suppose the step sizes α_n are not constant. According to the updating rule 2.5, the general case on each prior reward can be written as

$$
\begin{aligned}
Q_{n+1} &= Q_n + \alpha_n\big[R_n - Q_n\big] \\
&= \alpha_n R_n + (1-\alpha_n)Q_n \\
&= \alpha_n R_n + (1-\alpha_n)\big[\alpha_{n-1}R_{n-1} + (1-\alpha_{n-1})Q_{n-1}\big] \\
&= \alpha_n R_n + (1-\alpha_n)\alpha_{n-1}R_{n-1} + (1-\alpha_n)(1-\alpha_{n-1})Q_{n-1} \\
&= \alpha_n R_n + (1-\alpha_n)\alpha_{n-1}R_{n-1} + (1-\alpha_n)(1-\alpha_{n-1})\alpha_{n-2}R_{n-2} + \\
&\quad \cdots + (1-\alpha_n)(1-\alpha_{n-1})\ldots\alpha_1 R_1 + (1-\alpha_n)(1-\alpha_{n-1})\ldots(1-\alpha_1)Q_1 \\
&= \left(\prod_{i=1}^{n}(1-\alpha_i)\right)Q_1 + \left(\sum_{i=1}^{n-1}\left(\prod_{j=i+1}^{n}(1-\alpha_j)\right)\alpha_i R_i\right) + \alpha_n R_n.
\end{aligned}
$$

2.6 Optimistic Initial Values

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. 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. Employing an 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. Initially, the optimistic method performs worse because it explores more, but eventually it performs better because its exploration decreases with time.(*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. Keep in mind that, **any method that focuses on the initial conditions in any special way is unlikely to help with the general nonstationary case.**

Exercise 2.7 (Unbiased Constant-Step-Size Trick) Sample averages do not produce the initial bias that constant step sizes do. However, sample averages are not a completely satisfactory solution because they may perform poorly on nonstationary problems. It is possible to avoid the bias of constant step sizes while retaining their advantages on nonstationary problems? One way is to use a step size of

$$\beta_n \doteq \frac{\alpha}{\bar{o}_n}, \tag{2.8}$$

to process the nth reward for a particular action, where $\alpha > 0$ is a conventional constant step size, and $\bar{o}_n$ is a trace of one that starts at 0:

$$\bar{o}_n \doteq \bar{o}_{n-1} + \alpha(1 - \bar{o}_{n-1}), \quad \text{for } n \geq 0, \quad \text{with} \bar{o}_0 \doteq 0. \tag{2.9}$$

Carrying out an analysis like that in 2.6

$$\begin{aligned}
\bar{o}_n &= \bar{o}_{n-1} + \alpha(1 - \bar{o}_{n-1}) \\
&= \alpha + (1-\alpha)\bar{o}_{n-1} \\
&= \alpha + (1-\alpha)(\bar{o}_{n-2} + \alpha(1 - \bar{o}_{n-2})) \\
&= \alpha + (1-\alpha)(\alpha + (1-\alpha)\bar{o}_{n-2}) \\
&= \alpha + (1-\alpha)\alpha + (1-\alpha)^2\bar{o}_{n-2} \\
&= \alpha + (1-\alpha)\alpha + (1-\alpha)^2\alpha + (1-\alpha)^3\bar{o}_{n-3} \\
&= \left(\sum_{i=1}^{n}(1-\alpha)^{n-i}\alpha\right) + \underbrace{(1-\alpha)^n\bar{o}_0}_{=0,\ \text{since}\ \bar{o}_0 = 0}
\end{aligned}$$

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

β_n, the step size for non-stationary problems becomes,

$$\beta_n \doteq \frac{\alpha}{\sum_{i=1}^{n}(1-\alpha)^{n-i}\alpha} = \frac{1}{\sum_{i=1}^{n}(1-\alpha)^{n-i}}.$$

2.7 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. 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

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

where $\ln t$ denotes the natural logarithm 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.

Upper confidence bound (UCB) action selection with its 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. **In some more advanced settings (e.g. nonstationary problems, large state spaces) the idea of UCB action selection is usually not practical.**

2.8 Gradient Bandit Algorithms

A numerical *preference* for each action a is typically denoted as H_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:

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

where $\pi_t(a)$, 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.

Exercise 2.9 Show that in the case of two actions, the soft-max distribution is the same as that given by the logistic, or sigmoid, function often used in statistics and artificial neural networks.

In fact the *sigmoid function* is a special case of the *soft-max function* for a classifier with only two input classes. A sigmoid function is the logistic function defined by the formula

$$S(x) \doteq \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1}.$$

Using above function definition we would have:

$$\Pr\{A_t = a\} = \frac{e^{H_t(a)}}{e^{H_t(a)}+1},$$

$$\Pr\{A_t = b\} = 1 - \Pr\{A_t = a\} = \frac{1}{e^{H_t(a)}+1}.$$

The soft-max distribution for the case of two ($k = 2$) is formulated as follows:

$$\Pr\{A_t = a\} = \frac{e^{H_t(a)}}{e^{H_t(a)}+e^{H_t(b)}},$$

$$\Pr\{A_t = b\} = \frac{e^{H_t(b)}}{e^{H_t(a)}+e^{H_t(a)}},$$

Addition or summation from all action preferences does not affect the probabilities for the action selection. Therefore we can rewrite the preference values in the following form:

$$\begin{aligned} H_t(b)\{=0\} &\leftarrow H_t(b) - H_t(b), \\ H_t(a) &\leftarrow H_t(a) - H_t(b), \end{aligned}$$

and we get

$$\begin{aligned} \Pr\{A_t = a\} &= \frac{e^{H_t(a)}}{e^{H_t(a)}+e^0} = \frac{e^{H_t(a)}}{e^{H_t(a)}+1}, \\ \Pr\{A_t = b\} &= \frac{e^0}{e^{H_t(a)}+e^0} = \frac{1}{e^{H_t(a)}+1}. \end{aligned}$$

There is a natural learning algorithm for the setting 2.11 based on the idea of stochastic gradient ascent. On each step, after selecting action A_t and receiving the reward R_t , the action preferences are updated by:

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

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. 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. **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.** (*Look Figure 2.5*)

2.8.1 The Bandit Gradient Algorithm as Stochastic 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. 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 exact performance gradient can be written as:

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

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, $\sum_x \frac{\partial \pi_t(x)}{\partial H_t(a)}$ - as H_a is changed, some action's 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 $\frac{\pi_t(x)}{\pi_t(x)}$:

$$\frac{\partial \mathbb{E}[R_t]}{\partial H_t(a)} = \sum_x \pi_t(x) \big(q_*(x) - B_t\big) \frac{\partial \pi_t(x)}{\partial H_t(a)} \frac{1}{\pi_t(x)}.$$

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:

$$= \mathbb{E}\left[\big(q_*(A_t) - B_t\big) \frac{\partial \pi_t(A_t)}{\partial H_t(a)} \frac{1}{\pi_t(A_t)} \right],$$

$$= \mathbb{E}\Big[\big(R_t - \bar{R}_t\big)\frac{\partial \pi_t(A_t)}{\partial H_t(a)}\frac{1}{\pi_t(A_t)}\Big],$$

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, $\dfrac{\partial \pi_t(x)}{\partial H_t(a)} = \pi_t(x)\big(\mathbb{1}_{a=x} - \pi_t(a)\big)$ can be established, where $\mathbb{1}_{a=x}$ is defined to be 1 if $a = x$, else 0. Keeping in mind that, we now have

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

Substituting a sample of the expectation above for the performance gradient in 2.13 yields:

$$H_{t+1}(a) \doteq H_t(a) + \alpha(R_t - \bar{R}_t)\big(\mathbb{1}_{a=x} - \pi_t(a)\big), \text{ for all } a. \text{ (Look also 2.12)}$$

The derivative term, $\dfrac{\partial \pi_t(x)}{\partial H_t(a)}$ can be proved in term of $\pi_t(x)\big(\mathbb{1}_{a=x} - \pi_t(a)\big)$ by recalling the standard quotient rule for derivatives:

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

Using this, we can write

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

It can be 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. **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**. Choosing it as the average of the rewards may not be the very best, but it is simple and works well in practice.

2.9 Associative Search (Contextual Bandits)

So far it have been considered only *nonassociative tasks*, that is, tasks in which there is no need to associate different actions with different situations. 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.** With the right policy we can usually do much better than we could in the absence of any information distinguishing one bandit task from another.

Associative search task 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. If actions are allowed to affect the *next situation* as well as the reward, then we have the full R.L. problem.

Exercise 2.10 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 we 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?

For the first situation, we cannot hold individual estimates for the case A and B. Therefore, the best approach is to select the action that has best value estimate in combination. In this case, the estimates of both actions the same. Therefore the best expectation of success is 0.5 and it can be achieved by selecting an action randomly at each step.

$$A_1 = \mathbb{E}\big[R_t | A_t = A_1\big] = 0.5 \times 0.1 + 0.5 * 0.9 = 0.5$$
$$A_2 = \mathbb{E}\big[R_t | A_t = A_2\big] = 0.5 \times 0.2 + 0.5 * 0.8 = 0.5$$

For the second statement, we can hold independent estimates for the case A and B, thus we can learn the best action for each one treating them as independent bandit problems. The best expectation of success is 0.55 obtained from selecting A_2 in case A and A_1 in case B.

$$q_* = 0.5 \times 0.2 + 0.5 \times 0.9 = 0.55$$

2.10 Summary

One well-studied approach to balancing exploration and exploitation in k-armed bandit problems is to compute a special kind of action value called a *Gittins index*. In certain important special cases, this computation is tractable and leads directly to optimal solutions, although it does require complete knowledge of the prior distribution of possible

problems, which we generally assume is not available.

The Gittins-index approach is an instance of *Bayesian methods*, which assume a known initial distribution over the action values and then update the distribution exactly after each step (assuming that the true action values are stationary). **In general, the update computations can be very complex, but for certain special distributions** (called *conjugate priors*) **they are easy**. One possibility is to then select actions at each step according to their posterior probability of being the best action. This method, sometimes called *posterior sampling* or *Thompson sampling*, often performs similarly to the best of the distribution-free methods.

In the Bayesian setting it is even conceivable to compute the *optimal* balance between exploration and exploitation. One can compute for any possible action the probability of each possible immediate reward and the resultant posterior distributions over action values. This evolving distribution becomes the *information state* of the problem.

Chapter 3

Finite Markov Decision Processes