Chapter 2: Evaluative Feedback

- Evaluating actions vs. instructing by giving correct actions
- Pure evaluative feedback depends totally on the action taken. 
  Pure instructive feedback depends not at all on the action taken.
- Supervised learning is instructive; optimization is evaluative
- Associative vs. Nonassociative:
  - Associative: inputs mapped to outputs; 
    learn the best output for each input
  - Nonassociative: “learn” (find) one best output
- $n$-armed bandit (at least how we treat it) is:
  - Nonassociative
  - Evaluative feedback
The $k$-Armed Bandit Problem

- Choose repeatedly from one of $k$ actions; each choice is called a play.
- After each play $A_t$, you get a reward $R_t$, where

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

These are unknown action values. Distribution of $R_t$ depends only on $a_t$.

- Objective is to maximize the reward in the long term, e.g., over 1000 plays.

To solve the $n$-armed bandit problem, you must explore a variety of actions and exploit the best of them.
The Exploration/Exploitation Dilemma

- Suppose you form estimates
  \[ Q_t(a) \approx q_*(a) \]
  action value estimates

- The greedy action at \( t \) is:
  \[ A_t = \arg\max_a Q_t(a) \]

- You can’t exploit all the time; you can’t explore all the time
- You can never stop exploring; but you should always reduce exploring. Maybe.
Action-Value Methods

- Methods that adapt action-value estimates and nothing else, e.g., we can use the sample average:

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

- Indicator function is one if argument is true and zero otherwise

- If we keep exploring all actions every now and then, then the action value will converge to the true value
ε-Greedy Action Selection

- Greedy action selection:
  \[ A_t = \text{arg \ max}_a Q_t(a) \]
  (we assume that ties are broken arbitrarily)

- ε-Greedy: choose a random action with probability \( \epsilon \) and choose the greedy action with probability \( 1 - \epsilon \)

... the simplest way to balance exploration and exploitation
10-Armed Testbed

- $n = 10$ possible actions
- Each $q_*(a)$ is chosen randomly from a standard normal distribution: $N(0,1)$
- Each $R_t$ is also normal and given by the $q_*(a)$ plus a Gaussian of zero mean and unit variance
- 1000 plays
- Repeat the whole thing 2000 times and average the results
Example Instantiation

where \( \arg\max a \) denotes the value of \( a \) at which the expression that follows is maximized (with ties broken arbitrarily). 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 \( \epsilon \), instead to select randomly from amongst all the actions with equal probability independently of the action-value estimates. We call methods using this near-greedy action selection rule \( \epsilon \)-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 \( \frac{1}{10} \), that is, to near certainty. These are just asymptotic guarantees, however, and say little about the practical effectiveness of the methods.

To roughly assess the relative effectiveness of the greedy and \( \epsilon \)-greedy methods, we compared them numerically on a suite of test problems. This was a set of 2000 randomly generated \( k \)-armed bandit problems with \( k = 10 \). For each bandit problem, such as that shown in Figure 2.1, the action values, \( q^*(a) \), \( a = 1, \ldots, 10 \), were selected according to a normal (Gaussian) distribution with mean 0 and variance 1. Then, when a learning method applied to that problem selected action \( A_t \) at time \( t \), the reward distribution was selected around \( q^*(a) \) with unit variance, as suggested by these gray distributions.

![Diagram of Reward Distribution](image.png)
$\varepsilon$-Greedy Methods on the 10-Armed Testbed

![Graph showing average reward and percentage of optimal action over plays for different values of $\varepsilon$.]
Let’s consider one specific action and let $R_i$ now be the reward that was obtained when this action has been chosen for the $i$-th time:

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

Can we do this incrementally (without storing all the rewards)? Yes (see blackboard 2.1)!

Let $Q_{n+1}$ be our estimate for the $n+1$ reward, which is the sample average of the first $n$ rewards. We could keep a running sum and count, or, equivalently:

$$Q_{n+1} = Q_n + \frac{1}{n} [R_n - Q_n]$$

This is a common form for update rules:

$$NewEstimate = OldEstimate + StepSize[Target - OldEstimate]$$
A simple bandit algorithm

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

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

The expression $\times$ Target $\times$ OldEstimate $\times$ 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 $n$th reward.
Tracking a Nonstationary Problem

Choosing $Q_n$ to be a sample average is appropriate in a stationary problem,

i.e., when none of the $Q^*(a)$ change over time,

But not in a nonstationary problem.

Better in the nonstationary case is:

$$Q_{n+1} = Q_n + \alpha [R_n - Q_n] \quad \alpha \in (0, 1]$$

$$= (1 - \alpha)^n Q_1 + \sum_{i=1}^{n} \alpha(1 - \alpha)^{n-i} R_i.$$ 

exponential, recency-weighted average
Tracking a Nonstationary Problem

To see this, just “unpack” $Q_k$ recursively:

$$Q_{n+1} = Q_n + \alpha \left[ R_n - Q_n \right]$$

$$= \alpha R_n + (1 - \alpha) Q_n$$

$$= \alpha R_n + (1 - \alpha) \left[ \alpha R_{n-1} + (1 - \alpha) Q_{n-1} \right]$$

$$= \alpha R_n + (1 - \alpha) \alpha R_{n-1} + (1 - \alpha)^2 Q_{n-1}$$

$$= \alpha R_n + (1 - \alpha) \alpha R_{n-1} + (1 - \alpha)^2 \alpha R_{n-2} +$$

$$\cdots + (1 - \alpha)^{n-1} \alpha R_1 + (1 - \alpha)^n Q_1$$

$$= (1 - \alpha)^n Q_1 + \sum_{i=1}^{n} \alpha (1 - \alpha)^{n-i} R_i.$$
All methods so far depend on $Q_1(a)$, i.e., they are biased.

Suppose instead we initialize the action values optimistically, i.e., on the 10-armed testbed, use $Q_1(a) = 5$ for all $a$. 

Figure 2.2 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 $\varepsilon$-greedy method with $Q_1(a) = 0$. 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.
Upper-Confidence-Bound Action Selection

- **Idea:** prefer those non-greedy actions that have a good chance of turning out to be better than the current greedy action.

- One specific method (UCB1, Auer et al. 2002) uses:

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

- \( c \) controls the degree of exploration

- **Note:** not easy to apply to non-stationary problems and high-dimensional state spaces requiring function approximation
Upper-Confidence-Bound Action Selection

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

Intuition: when estimating the mean from averaging $N$ measurements, the so-called standard error of the mean scales with $1/\sqrt{N}$ (see blackboard 2.2)
CHAPTER 2. MULTI-ARM BANDITS

Upper-Confidence-Bound Action Selection

Figure 2.3: 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 few plays, when it selects randomly among the as-yet-unplayed actions. UCB with $c=1$ would perform even better but would not show the prominent spike in performance on the 11th play. Can you think of an explanation of this spike?

As time goes by it will be a longer wait, and thus a lower selection frequency, for actions with a lower value estimate or that have already been selected more times.

Results with UCB on the 10-armed testbed are shown in Figure 2.3. UCB will often perform well, as shown here, but is more difficult than ε-greedy to extend beyond bandits to the more general reinforcement learning settings considered in the rest of this book. One difficulty is in dealing with nonstationary problems; something more complex than the methods presented in Section 2.4 would be needed. Another difficulty is dealing with large state spaces, particularly function approximation as developed in Part III of this book. In these more advanced settings there is currently no known practical way of utilizing the idea of UCB action selection.

2.7 Gradient Bandits

So far in this chapter we have considered methods that estimate action values and use those estimates to select actions. This is often a good approach, but it is not the only one possible. In this section we consider learning a numerical preference $H_t(a)$ for each action $a$. The larger the preference, the better.

Question: what causes the early spike?
Gradient Bandits

- Use Softmax action selection based on action preferences $H_t$ (not value estimates)

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

- Directly performing stochastic gradient ascent on the expected reward (see blackboard 2.3) gives:

\[
H_{t+1}(A_t) \overset{\doteq}{=} H_t(A_t) + \alpha(R_t - \bar{R}_t)(1 - \pi_t(A_t)), \quad \text{and} \\
H_{t+1}(a) \overset{\doteq}{=} H_t(a) - \alpha(R_t - \bar{R}_t)\pi_t(a), \quad \forall a \neq A_t
\]

- Where we chose the average reward obtained so far as a free parameter: $\bar{R}_t \in \mathbb{R}$

Gradient Bandits

% Optimal action

$\mathbb{E}[q(a)] = 4$

$\alpha = 0.1$ with baseline

$\alpha = 0.1$ without baseline

$\alpha = 0.4$ with baseline

$\alpha = 0.4$ without baseline

Steps
2.9. SUMMARY

It is natural to ask which of these methods is best. Although this is a difficult question to answer in general, we can certainly run them all on the 10-armed testbed that we have used throughout this chapter and compare their performances. A complication is that they all have a parameter; to get a meaningful comparison we will have to consider their performance as a function of their parameter. Our graphs so far have shown the course of learning over time for each algorithm and parameter setting, but it would be too visually confusing to show such a learning curve for each algorithm and parameter value. Instead we summarize a complete learning curve by its average value over the 1000 steps; this value is proportional to the area under the learning curves we have shown up to now. Figure 2.6 shows this measure for the various bandit algorithms from this chapter, each as a function of its own parameter shown on a single scale on the x-axis. Note that the parameter values are varied by factors of two and presented on a log scale. Note also the characteristic inverted-U shapes of each algorithm's performance; all the algorithms perform best at an intermediate value of their parameter, neither too large nor too small. In assessing a method, we should attend not just to how well it does at its best parameter setting, but also to how sensitive it is to its parameter value. All of these algorithms are fairly insensitive, performing well over a range of parameter values varying by about an order of magnitude. Overall, on this problem, UCB seems to perform best. Despite their simplicity, in our opinion the methods presented in this chapter can fairly be considered the state of the art. There are more sophisticated methods, but their complexity and assumptions make them impractical for the full reinforcement learning problem that is our real focus. Starting in Chapter 5 we present learning methods for solving the full reinforcement learning problem that use in part the /c/Q/0

Figure 2.6: 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.
Associative Search (Contextual Bandits)

- One step towards the full reinforcement learning problem

- Now consider that you can be in different situations (facing different bandits) and you can identify which bandit you are currently facing. You must map the different situations to the action that’s best for this situation. What should you do?

- This setting is intermediate between the n-armed bandit problem and the full reinforcement learning problem

- Note: reward is still immediate

- Note: your action does not yet change what the next situation will be
Conclusions

- These are all very simple methods
  - but they are complicated enough—we will build on them
  - we should understand them thoroughly

- Ideas for improvements:
  - “action elimination” methods
  - approximating Bayes optimal solutions
  - Gittens indices