Chapter 2 - Multi-armed Bandits

Key Feature: Evaluative Feedback

The defining feature that sets reinforcement learning (RL) apart from other types of learning is its use of **evaluative feedback** rather than **instructive feedback**. In RL, the agent receives feedback that assesses the actions it took, without necessarily telling it what the 'correct' or 'best' actions would have been. This contrasts with supervised learning methods, where the learner is explicitly told the correct action or label for each situation (instructive feedback).

Purely Evaluative vs. Purely Instructive Feedback

- **Purely Evaluative Feedback**: Tells the agent how good its action was but not if it was the best or worst possible action.
- **Purely Instructive Feedback**: Tells the agent exactly which action to take, regardless of what action the agent actually took.

Non-associative Setting: k-Armed Bandit Problem

The chapter focuses on a simplified case of RL known as the k-armed bandit problem. This problem involves a single situation or state, and thus is considered "non-associative." In this setting, the agent faces k different "arms" (options) and must decide which one to pull to maximize its total reward over time. The k-armed bandit problem serves as an introductory platform for studying evaluative feedback and basic learning methods that can later be extended to the full RL setting.

2.1 The k-Armed Bandit Problem

In the k-armed bandit problem, you're faced with *k* different options (or arms, like levers on a slot machine) to choose from at each time step. After choosing an arm, you receive a reward drawn from a stationary probability distribution specific to that arm. The goal is to maximize your expected total reward over a certain number of time steps (e.g., 1000).

Key Concepts

Value of an Action

The value $q*(a)$ of an action a is its expected reward: $q*(a)=E[Rt|At=a]$.

Estimation of Action Value

If you don't know the true value of each action, you can maintain an estimate $Qt(a)$ for each action a at time t.

Greedy and Non-Greedy Actions

- **Greedy Actions**: Actions for which the estimated value is currently the highest.
- **Non-Greedy Actions**: Actions that are not currently estimated to be the best.

Exploitation vs Exploration

- **Exploitation**: Choosing the action that has the highest estimated value based on current knowledge.

- **Exploration**: Choosing an action other than the one with the highest estimated value to improve your knowledge about other actions.

Conflict: Exploration vs Exploitation

The tension between exploration and exploitation is a key challenge in RL. Exploiting current knowledge maximizes short-term reward but can be suboptimal in the long run if better actions exist that have not been sufficiently explored. Conversely, exploration may result in lower immediate reward but can lead to higher long-term gains by discovering better actions.

Approaches to Balancing Exploration and Exploitation

While there are sophisticated ways to balance exploration and exploitation, often relying on strong assumptions about stationarity and prior knowledge, the book focuses on simpler methods that work well in practice. This is because the assumptions behind sophisticated methods often don't hold or can't be verified in real-world applications.

2.2 Action-value Methods

Estimating Action Values: Sample-Average Method

To estimate the value of an action *a*, one natural approach is to take the average of all the rewards received when *a* was selected. Mathematically, this is expressed as:

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

Here, 1predicate is an indicator function that returns 1 if the predicate is true and 0 otherwise. By the law of large numbers, as the number of samples approaches infinity, *Qt*(*a*) will converge to the true value *q*∗(*a*).

Action Selection: Greedy and ϵ-Greedy Methods

Greedy Method

The simplest way to select an action is to pick the one with the highest estimated value:

$$A_t \doteq \underset{a}{\arg\max}\, Q_t(a), \qquad (2.2)$$

This is known as greedy action selection. It exploits the current knowledge to maximize immediate rewards but doesn't explore other potentially better actions.

ϵ-Greedy Method

In ϵ-greedy methods, you select the best action most of the time, but with probability ϵ, you select an action randomly. This allows for both exploitation and exploration. As the number of steps increases, each action will be explored an infinite number of times, ensuring *Qt*(*a*) converges to *q*∗(*a*)

Exercise 2.1

In ϵ-greedy action selection, for the case of two actions and ϵ=0.5, the probability of selecting the greedy action can be calculated as follows:

- With probability $1-\epsilon$=0.5, the greedy action will be selected (exploitation).
- With probability ϵ=0.5, a random action is selected. Since there are two actions, the probability of selecting the greedy action in this case is $\epsilon/2$=0.25.

So, the overall probability of selecting the greedy action is:

$(1-\epsilon)+(2\epsilon)=0.5+0.25=0.75$

Therefore, the probability of selecting the greedy action is 75%.

2.3 10-armed Testbed

The 10-armed testbed is a way to numerically compare different approaches to solving the k-armed bandit problem. In this setup, 2000 randomly generated k-armed bandit problems were used, each with k=10 arms. Each arm has an associated true value $q*(a)$ that was generated from a normal distribution with mean 0 and variance 1. When an action At is selected at time t, the reward Rt comes from a normal distribution centered around $q*(At)$ with variance 1.

This testbed showed that ϵ-greedy methods outperformed purely greedy methods in the long run. Greedy methods often got stuck exploiting suboptimal actions, while ϵ-greedy methods continued to explore and improve their performance over time.

Exercise 2.2

Given the sequence of actions $A1$=1, $A2$=2 ,$A3$=2, $A4$=2, $A5$=3 and rewards $R1$=−1, $R2$=1, $R3$=−2, $R4$=2, $R5$=0, we can make the following observations:

- For $A1$, it's not possible to tell whether exploration (ϵ-case) or exploitation (greedy case) happened since this is the first action.
- For $A2$, the reward for action 1 was negative, so action 2 could have been selected either due to ϵ or because it was the greedy choice.
- For $A3$ and $A4$, after receiving a reward of 1 for action 2, repeatedly choosing action 2 could be due to exploitation, but it could also be due to ϵ-greedy exploration.
- For $A5$, choosing action 3 could be due to ϵ-greedy exploration, especially if the average reward for action 2 was not good due to the -2 reward received on $A3$.

Therefore, exploration (ϵ-case) could have happened at any of these steps, but it definitely happened at $A5$ if the average reward for action 2 up to that point was better than for action 3.

Exercise 2.3

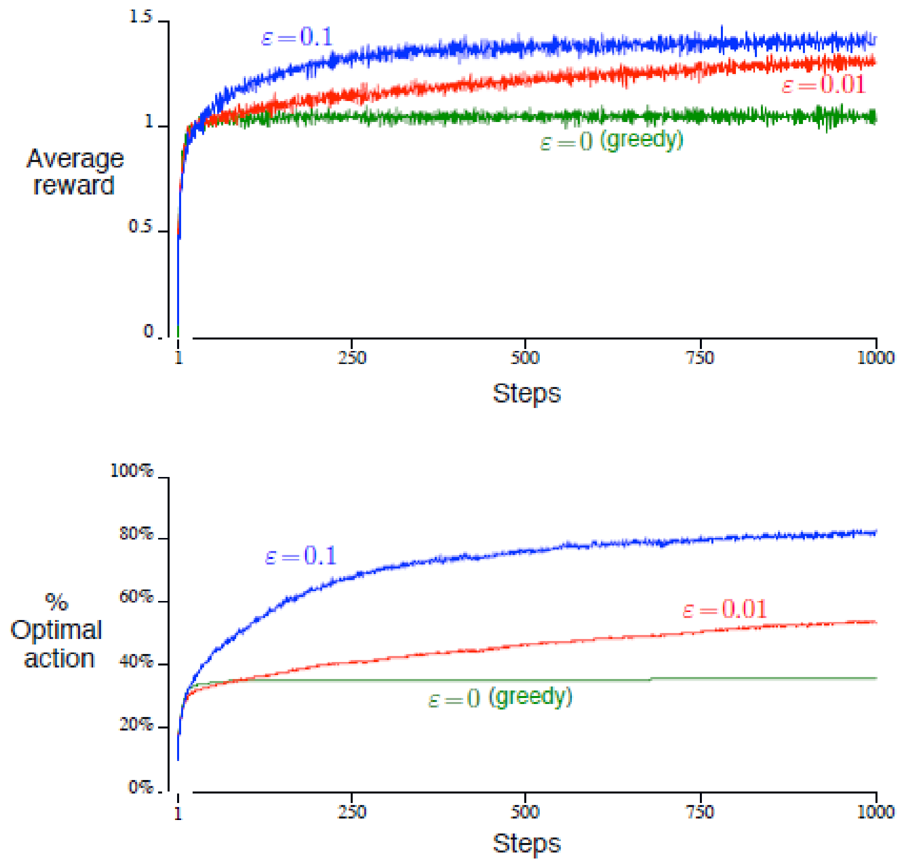

Figure 2.2: Average performance of "-greedy action-value methods on the 10-armed testbed.

These data are averages over 2000 runs with different bandit problems. All methods used sample

averages as their action-value estimates.

Based on the description of Figure 2.2, the ϵ=0.01 method seems to perform best in the long run in terms of both cumulative reward and the probability of selecting the best action. The ϵ=0.01 method improves more slowly but is likely to be more accurate in the long term compared to the ϵ=0.1 method.

Quantitatively, it's hard to specify "how much better" the ϵ=0.01 method will be without specific numerical results. However, according to the text, the greedy method achieved a reward-per-step of only about 1, compared to the best possible of about 1.55. ϵ-greedy methods would be expected to approach this upper limit more closely, particularly for ϵ=0.01, given enough time.

2.4 Incremental Implementation of Action-Value Methods

The incremental implementation of action-value methods is aimed at computing action values with constant memory and constant computational effort per time step. The key idea is to update the average reward for each action incrementally, as new rewards are observed, rather than storing all past rewards and computing the average from scratch each time.

Incremental Update Formula

The incremental formula to update the estimated action value *Qn* after observing a new reward *Rn* is:

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

Here, *n* is the number of times the action has been taken, and *Rn* is the reward observed on the *n*-th selection of this action.

This formula updates *Qn*+1 based on the old estimate *Qn* and the new reward *Rn*, with a step-size parameter *n*1. This ensures that the method uses constant memory and computational time per step, which is a significant advantage.

General Update Rule

The general form of this kind of update is:

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

Here, [Target−Old Estimate] is the error in the estimate, and the step-size parameter determines how much of this error is used to update the estimate. α is used to denote the step size (1/k) in the rest of this book.

Pseudocode for a Simple Bandit Algorithm

The pseudocode for a simple bandit algorithm uses ϵ-greedy action selection and incrementally computed sample averages.

A simple bandit algorithm

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

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

In this algorithm, *Q*(*a*) is the estimated value of action *a*, *N*(*a*) is the number of times action *a* has been selected, ϵ is the probability of selecting a random action (exploration), and **bandit(A)** is a function that returns the reward for taking action *A*.

This approach provides a computationally efficient way to solve the k-armed bandit problem, balancing the trade-off between exploration and exploitation.

2.6 Tracking a Nonstationary Problem

When dealing with nonstationary problems, where the underlying reward distribution can change over time, it's beneficial to give more weight to recent rewards than to older rewards. This can be accomplished by using a constant step-size parameter *α* in the incremental update formula:

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

where the step-size parameter α ∈ (0, 1] is constant. This results in Qk+1 being a weighted average of the past rewards and the initial estimate Q1:

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

This is known as an exponential recency-weighted average, as the weight given to each past reward decays exponentially over time.

Because the weight given to each reward depends on how many rewards ago it was observed, we can see that more recent rewards are given more weight. Note the weights α sum to 1 here, ensuring it is indeed a weighted average where more weight is allocated to recent rewards.

In fact, the weight given to each reward decays exponentially into the past. This sometimes called an exponential or recency-weighted average.

Convergence Conditions

For the estimates to converge, two conditions must be met:

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

The first condition ensures that the updates are large enough to overcome initial conditions or random fluctuations. The second condition ensures that the updates eventually become small enough for convergence.

In the case of a constant α, only the first condition is met, meaning the estimates will continue to vary but won't converge. This is often desirable in nonstationary environments, which are common in reinforcement learning.

2.7Optimistic Initial Values

Initial action values can have a strong influence on the behavior of action-value methods. This is particularly true for methods with a constant step-size parameter α, where the bias introduced by initial estimates decreases over time but never completely vanishes. One simple yet effective strategy to encourage exploration is to initialize action values optimistically. That is, you set initial values higher than the expected rewards from any of the actions.

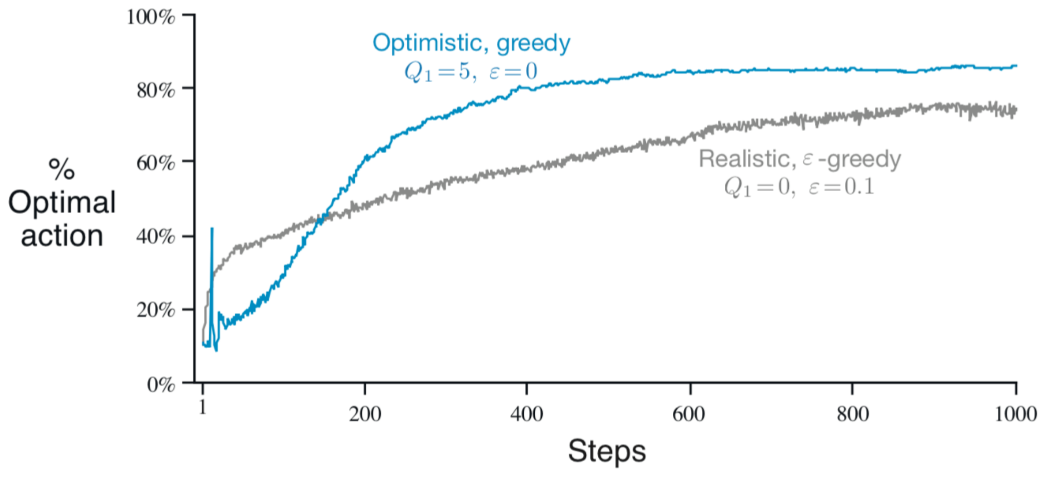

Figure 2.3: The effect of optimistic initial action-value estimates on the 10-armed testbed.

Both methods used a constant step-size parameter, ↲ = 0.1.

How It Works

When action values are initialized optimistically, the agent starts by being "disappointed" with the real rewards it receives, as they don't live up to the optimistic expectations. This encourages the agent to try other actions, leading to a significant amount of exploration in the early stages. This is especially effective when a greedy policy is used, as the agent quickly learns about the environment before settling into a more exploitative strategy.

For example, in the 10-armed bandit problem, if $q*(a)$ values are drawn from a normal distribution with mean 0 and variance 1, initializing $Q(a)$ to +5 would be extremely optimistic. A greedy policy with these initial values would explore more at the beginning, before settling down to exploiting the action with the highest estimated value.

Advantages and Limitations

Advantages: Encourages exploration in the early stages, which can be beneficial in stationary environments. Simple to implement.

Limitations: Not well-suited for non-stationary problems, as the optimistic initialization only has an impact in the beginning. Introduces another set of parameters that may require tuning.

Applicability

Optimistic initial values can be a useful approach for encouraging exploration in stationary problems, but they are less useful in non-stationary settings where the need for exploration can re-emerge over time. In such cases, other exploration strategies like ϵ-greedy or Upper Confidence Bound (UCB) methods may be more appropriate.

2.8 Upper-Confidence-Bound (UCB) Action Selection

The Upper-Confidence-Bound (UCB) action selection method is another approach to solving the exploration-exploitation dilemma in reinforcement learning, particularly in the k-armed bandit problem. It seeks to balance not only the estimated value of each action but also the uncertainty in each estimate.

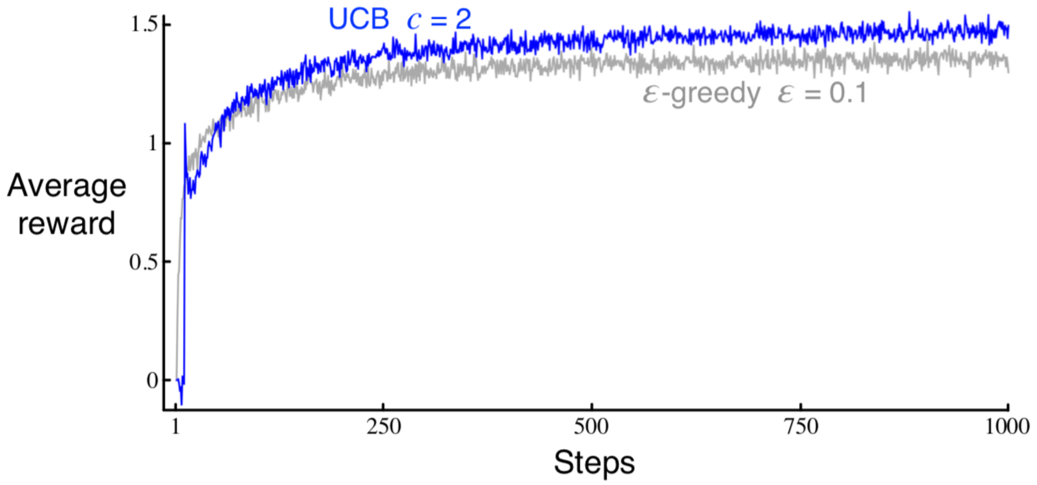

Figure 2.4: Average performance of UCB action selection on the 10-armed testbed. As shown,

UCB generally performs better than "-greedy action selection, except in the first k steps, when it selects randomly among the as-yet-untried actions.

How It Works

The UCB method selects actions based on the upper confidence bound of their estimated value. Specifically, at each time step *t*, the action *At* is chosen according to:

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

Here, *Qt*(*a*) is the estimated value of action *a* at time *t*, *Nt*(*a*) is the number of times action *a* has been taken up to time *t*, and *c* is a constant that controls the degree of exploration.

The term $\sqrt{\frac{\ln t}{N_t(a)}}$ represents the uncertainty or confidence interval around *Qt*(*a*), and ln*t* is the natural logarithm of *t*.

Advantages and Limitations

Advantages:

- Encourages exploration of actions that could potentially be optimal.
- Takes into account both the estimated value and the uncertainty around the value of each action.
- Generally effective in stationary bandit problems.

Limitations:

- More computationally expensive than ϵ-greedy, as it requires additional calculations involving logarithms and square roots.
- Not straightforward to extend to non-stationary problems or larger state-action spaces (common in more general reinforcement learning scenarios).

Applicability

The UCB method often works well in stationary, finite-armed bandit problems, where it balances exploration and exploitation in a more nuanced way compared to �ϵ-greedy action selection. However, its applicability is limited in non-stationary environments and in problems with large or infinite state-action spaces.

Comparisons with ϵ-Greedy

In many situations, UCB outperforms ϵ-greedy action selection, especially as the number of time steps increases. ϵ-greedy methods explore indiscriminately, while UCB targets its exploration towards actions that are uncertain or have the potential to be optimal. However, the advantage of UCB comes at the cost of increased computational complexity.

Overall, the choice between ϵ-greedy and UCB will depend on the specific requirements of the problem at hand, such as whether the environment is stationary, the computational resources available, and the need for targeted exploration.

2.8 Gradient Bandit Algorithms

Gradient bandit algorithms take a different approach from the value-based methods like ϵ-greedy or UCB. Instead of maintaining an estimate of the value for each action, they maintain a preference for each action. These preferences are used to compute the probabilities of selecting each action, but they do not have any direct interpretation in terms of expected rewards.

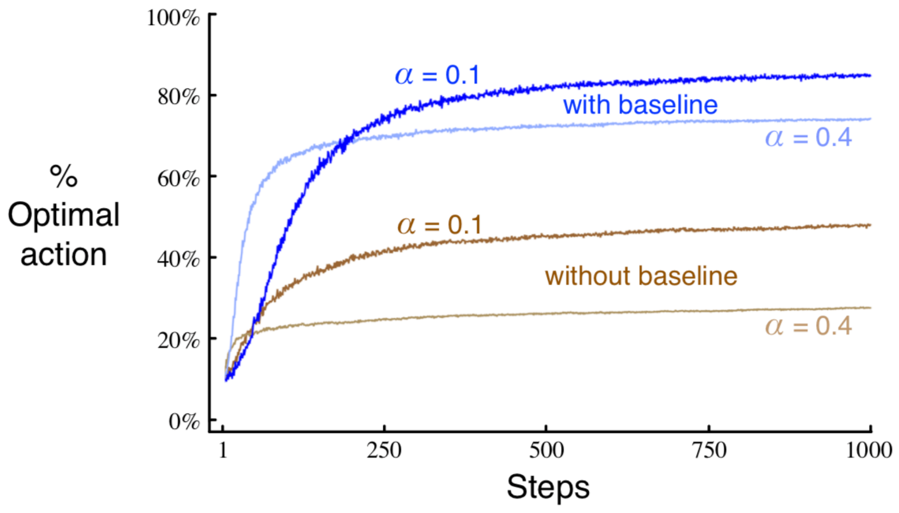

Figure 2.5: Average performance of the gradient bandit algorithm with and without a reward

baseline on the 10-armed testbed when the q⇜(a) are chosen to be near +4 rather than near zero.

Soft-max Distribution for Action Selection

The probability of taking action *a* at time *t* is determined by a soft-max distribution:

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

Here, *Ht*(*a*) represents the preference for action *a* at time *t*, and *πt*(*a*) is the probability of taking action *a* at time *t*.

Learning Algorithm

The learning algorithm for updating the preferences is based on stochastic gradient ascent. After each action *At* and corresponding reward *Rt*, the preferences are updated as:

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

Here, *α*>0 is a step-size parameter, and *R*¯*t* is the average of all rewards received up to time *t*. This average serves as a baseline that helps in adjusting the preferences.

The Role of Baseline *R*¯*t*

The baseline *R*¯*t* is crucial for the performance of the gradient bandit algorithm. If the reward *Rt* is greater than the baseline, then the probability of taking action *At* in the future is increased. Conversely, if *Rt* is below the baseline, the probability is decreased. This makes the algorithm more robust to shifts in the reward distribution.

Advantages and Limitations

Advantages:

- More nuanced exploration strategy compared to ϵ-greedy.
- Adapts quickly to different reward levels due to the use of a baseline.

Limitations:

- Computationally more expensive due to the need to calculate exponentials for the soft-max probabilities.
- No direct interpretation of action preferences in terms of expected rewards.

2.9 Associative Search (Contextual Bandits)

Thus far we have been discussing the stationary k-armed bandit problem, where the value of each arm is unknown but nonetheless remains stationary. Now, we consider a problem where the task could change from step to step, but the value distributions of the arms in each task remain the same. This is called contextual bandits, and in the toy example we are usually given a hint that the task has changed e.g. the slot machine changes colour for each task. Now we want to learn the correct action to take in a particular setting, given the task colour observed. This is an intermediary between the stationary problem and the full reinforcement learning problem.

2.10 Summary

The chapter covers various algorithms and techniques for balancing the exploration and exploitation dilemma in k-armed bandit problems:

- ε-greedy methods: These methods are simple and perform random exploration a fraction ϵ of the time.
- UCB methods: These methods are deterministic and favor actions that are under-explored, taking into account both estimated value and uncertainty.

- Gradient Bandit Algorithms: These methods don't estimate action-values but rather learn a preference for each action. Actions are then selected probabilistically according to these preferences.
- Optimistic Initial Values: This method starts with high initial action values encouraging exploration before convergence.

The chapter also discusses how different methods perform over a variety of parameter settings in the 10-armed bandit testbed. The methods are compared based on the average value over 1000 steps, showing UCB generally performs best in these tests.

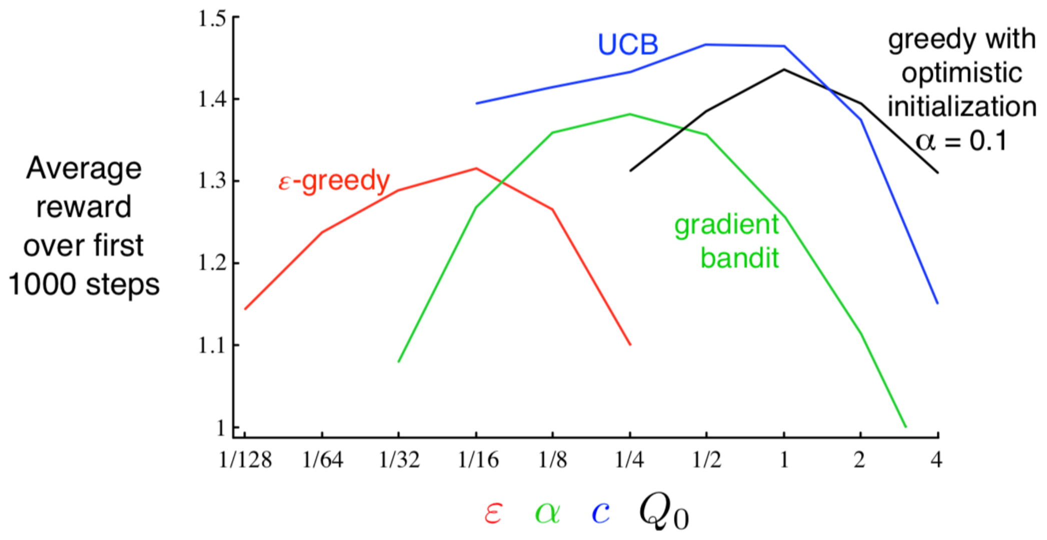

Figure 2.6: A parameter study of the various bandit algorithms presented in this chapter.

Each point is the average reward obtained over 1000 steps with a particular algorithm at a particular setting of its parameter.

Key Takeaways

- The value of an action can be summarised by Qt(a), the sample average return from an action.
- When selecting an action, it is preferable to maintain exploration, rather than only selecting the action we believe to be most valuable at any given timestep, to ensure we continue to improve our best estimate of the optimal action. We do so use ϵ-greedy policies.
- If our problem is non-stationary, rather than taking a standard average of every return received after an action, we can take a weighted average that gives higher value to more recently acquired rewards. We call this an exponential or recency-weighted average.
- Optimistic initial values encourage lots of early exploration as our returns always decrease our estimate of Qt meaning the greedy actions remain exploratory. Only useful for stationary problems.
- ϵ-greedy policies can be adapted to give more value to actions that have been selected less often, i.e. actions where we have higher uncertainty in their value, using upper-confidence bound action selection.

- Lastly, each of these techniques have varied performance on the n-armed bandit test dependent on their parametrisation. Their performance is plotted in Figure 2.6