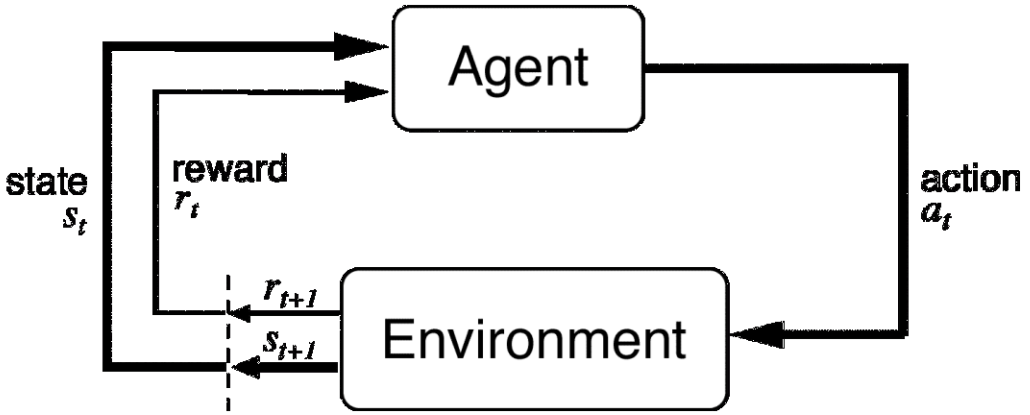
BANDIT PROBLEMS

By: Dimple Bohra

Formal Definition of RL



Agent and environment interact at discrete time steps : $t = 0, 1, 2, \ldots$

Agent observes state at step t: $s_t \in S$

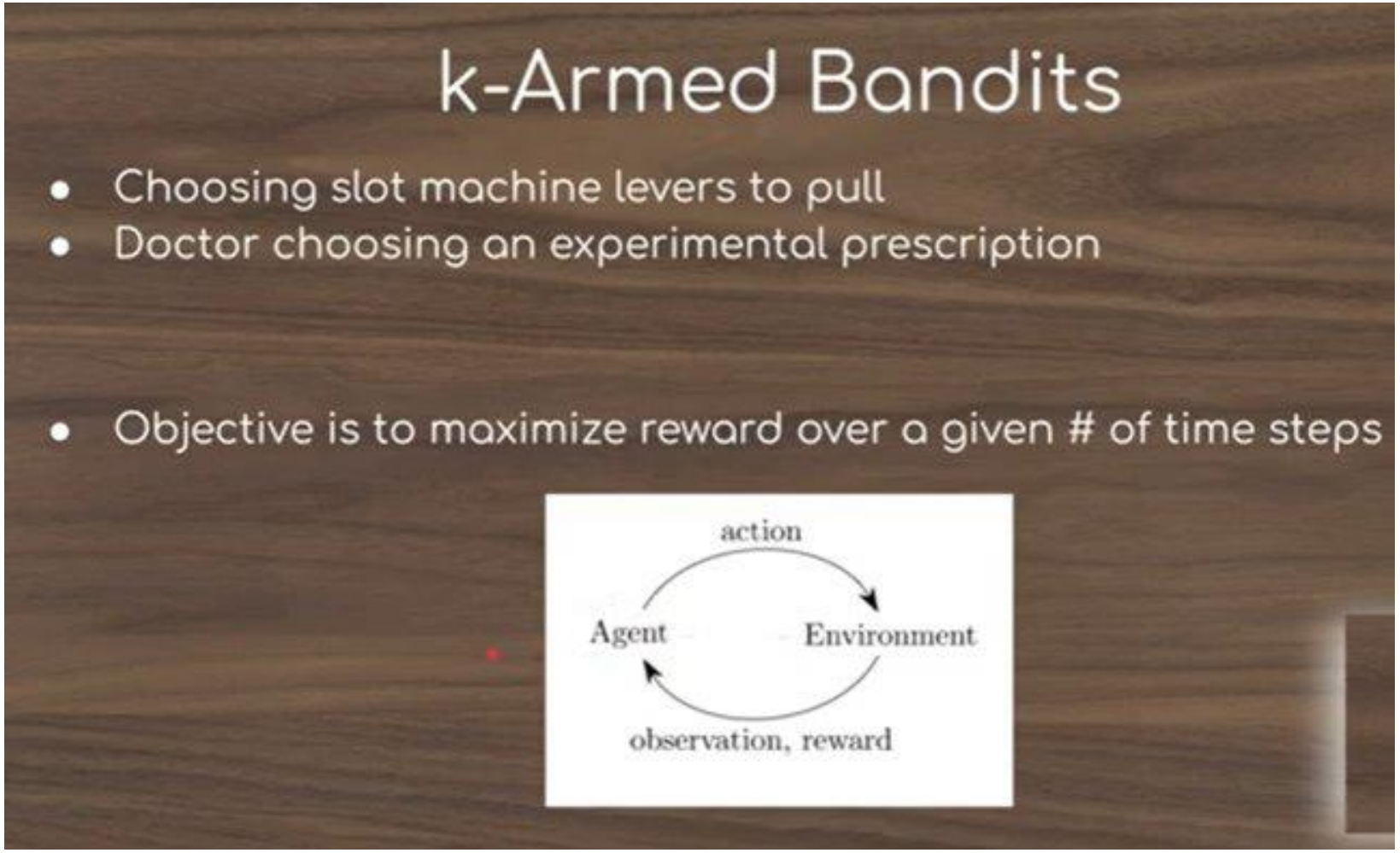
produces action at step t: $a_t \in A(s_t)$

gets resulting reward : $r_{t+1} \in \Re$

and resulting next state : s_{t+1}

$$\ldots \; s_t \xrightarrow[a_t]{r_{t+1}} s_{t+1} \xrightarrow[a_{t+1}]{r_{t+2}} s_{t+2} \xrightarrow[a_{t+2}]{r_{t+3}} s_{t+3} \xrightarrow{a_{t+3}} \ldots$$

k-armed Bandit problem



k-armed Bandit problem

The k-Armed Bandit Problem

- Here, 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

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

$$E\{r_t \mid a_t\} = Q^*(a_t)$$

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 k-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

$$a_t^* = \arg\max_a Q_t(a)$$

$$a_t = a_t^* \Rightarrow \text{exploitation}$$

$$a_t \neq a_t^* \Rightarrow \text{exploration}$$

- 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.

Action-Value Methods

- Methods that adapt action-value estimates and nothing else, e.g.: suppose by the *t*-th play, action a had been chosen k_a times, producing rewards then $r_1, r_2, \ldots, r_{k_a}$,

"sample average" $Q_t(a) = \dfrac{r_1 + r_2 + \cdots + r_{k_a}}{k_a}$ **OR** $Q_t(a) = \dfrac{R_1 + R_2 + \cdots + R_{N_t(a)}}{N_t(a)}.$

$$\lim_{k_a \to \infty} Q_t(a) = Q^*(a)$$

Sample Average Method

- If k_a = 0, then we define $Q_t(a)$ instead as some default value, such as $Q_1(a) = 0$.
- As $k_a \rightarrow \infty$, 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 a simple average of the sample of relevant rewards.

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

Greedy Action Selection

- The simplest action selection rule is to select the action (or one of the actions) with highest estimated action value, that is, to select at step t one of the greedy actions.
- This greedy action selection method can be written as

$$a_t = a_t^* = \arg\max_a Q_t(a)$$

- where $\arg\max_a$ denotes the value of a, at which the expression that follows is maximized

Greedy Policy

Greedy Policy

From the partially knowledge of actions in a state, associated rewards and resulting states, take the action which will maximize the total reward.

Greedy policy apparently fails here because we are assuming that bigger diamonds lie buried where you were digging.

ε-Greedy Action Selection

- A simple alternative is to behave greedily most of the time, but every once in a while, say with small probability **ε,** 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 **ε-greedy methods**

ε-Greedy Action Selection

- Greedy action selection:

$$a_t = a_t^* = \arg\max_a Q_t(a)$$

- ε-Greedy:

$$a_t = \begin{cases} a_t^* & \text{with probability} \quad 1-\varepsilon \\ \text{random action with probability} & \varepsilon \end{cases}$$

. . . the simplest way to balance exploration and exploitation

ε-Greedy Action Selection

ε- Greedy Policy (Exploration with Exploitation)

Greedy action with probability = 1−ε (Exploitation)

Exploration with probability = ε

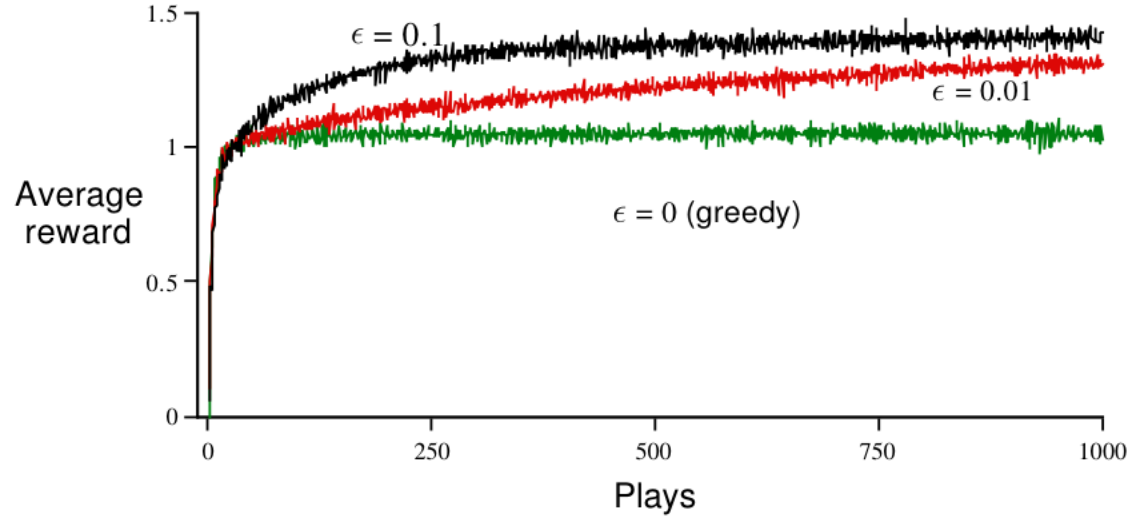
$0 < \varepsilon < 1$

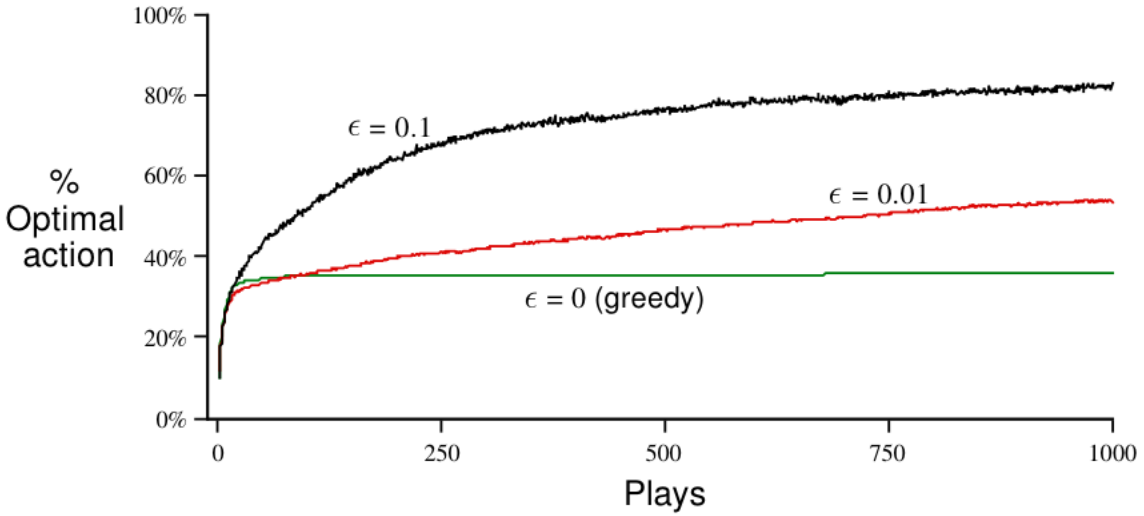
With partial or no knowledge about future rewards, ε–greedy policy gives best results as it balances between exploitation of current knowledge and exploration of unknown territory (actions). We can start with ε closer to 1 initially (more exploration) and bring it closer to 0 in steps as we learn more and more about the environment.

Greedy Vs ε-Greedy Methods

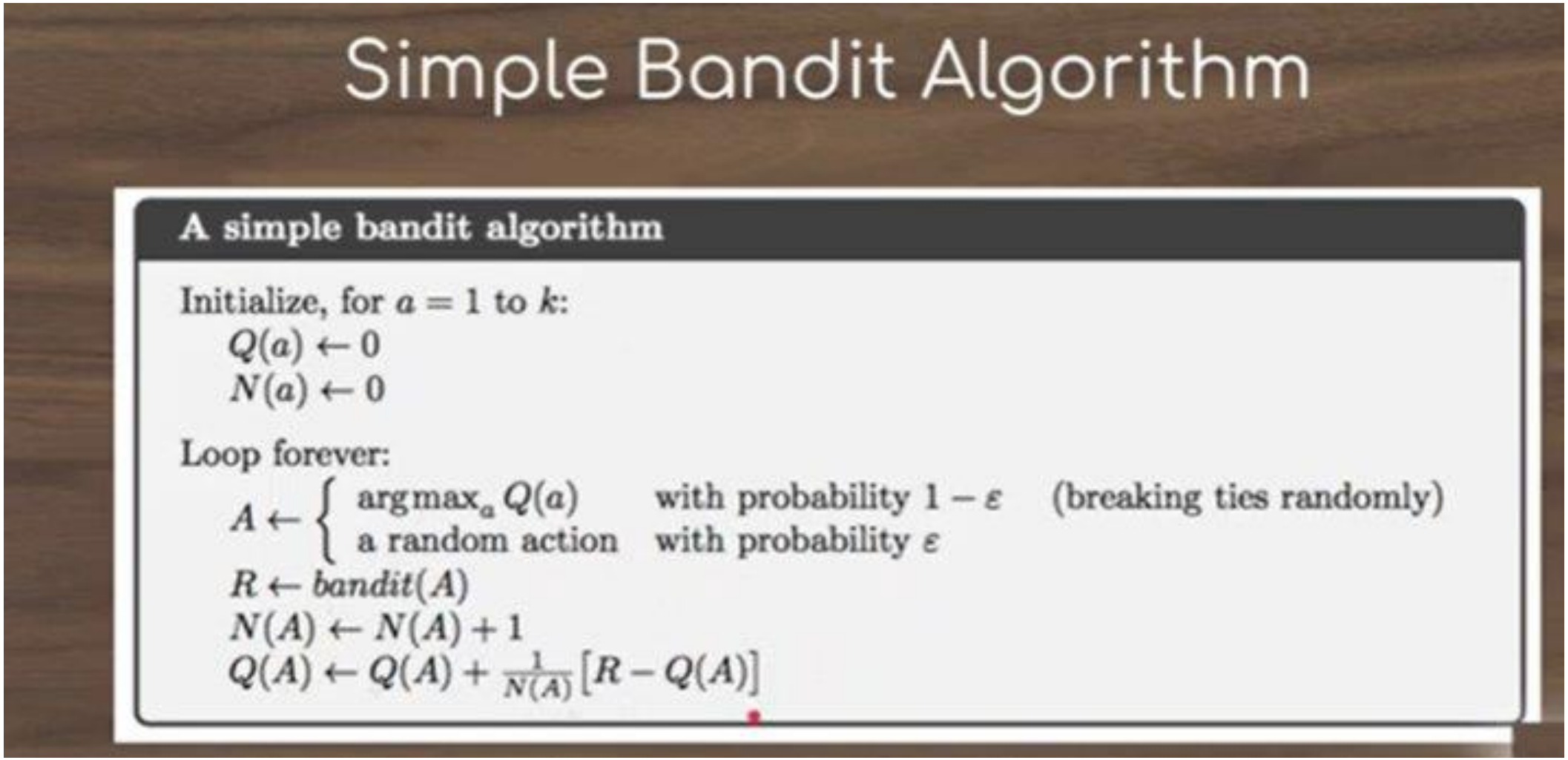
- The advantage of ε-greedy over greedy methods depends on the task. For example, suppose the reward variance had been larger, say 10. With noisier rewards, it takes more exploration to find the optimal action, and ε-greedy methods should fare even better relative to the greedy method.
- On the other 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.

Greedy Vs ε-Greedy Methods





Simple Bandit Algorithm



Incremental Implementation

The average of the first k rewards is $$Q_t(a) = \frac{r_1 + r_2 + \cdots + r_{k_a}}{k_a}$$

Can we do this incrementally (without storing all the rewards)?

$$\begin{aligned} Q_{k+1} &= \frac{1}{k}\sum_{i=1}^{k} R_i \\ &= \frac{1}{k}\left(R_k + \sum_{i=1}^{k-1} R_i\right) \\ &= \frac{1}{k}\Big(R_k + (k-1)Q_k + Q_k - Q_k\Big) \\ &= \frac{1}{k}\Big(R_k + kQ_k - Q_k\Big) \\ &= Q_k + \frac{1}{k}\Big[R_k - Q_k\Big], \end{aligned}$$

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

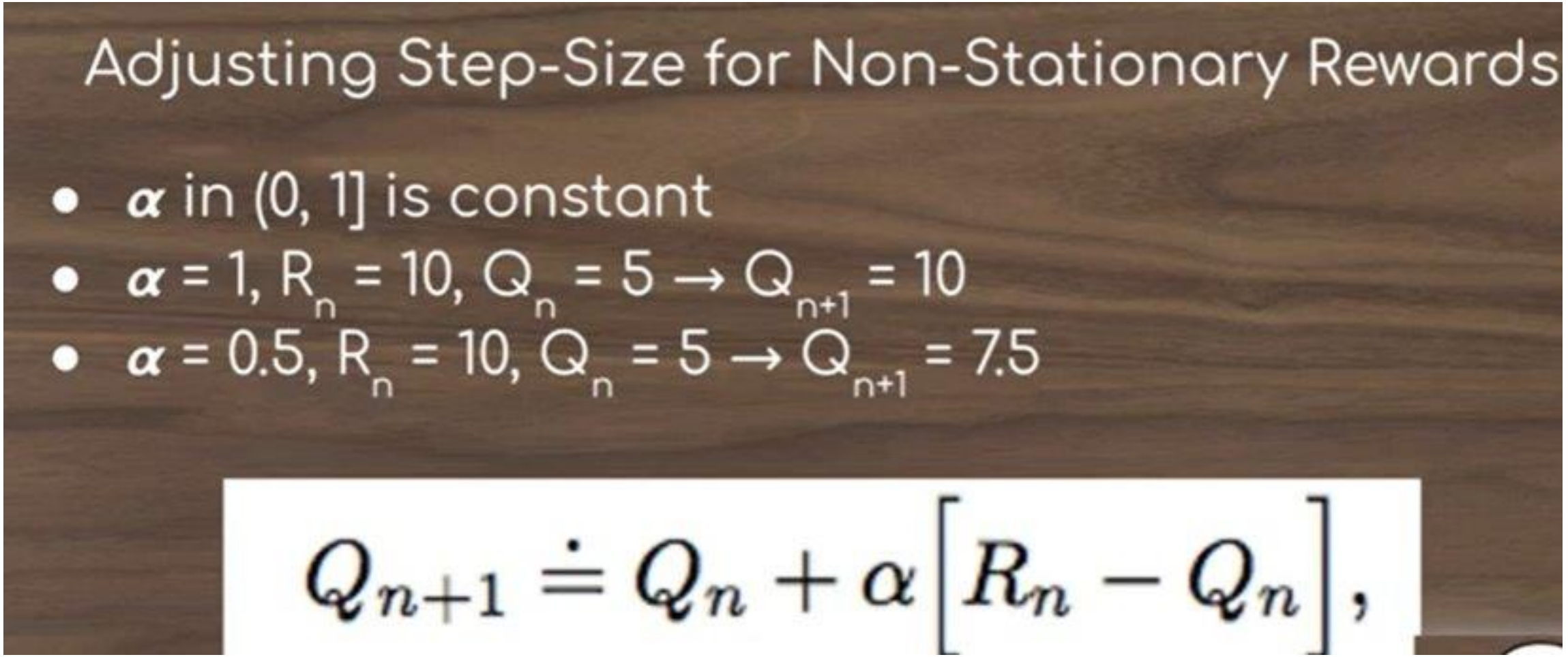
This is a common form for update rules:

$$NewEstimate = OldEstimate + StepSize[Target - OldEstimate]$$

Incremental Implementation

- The expression [Target − OldEstimate] 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 kth reward. Note that the step-size parameter (StepSize) used in the incremental method described above changes from time step to time step.
- In processing the kth reward for an action a, that method uses a step-size parameter of 1/k .
- Step size parameter is denoted by α or $\alpha_t(a)$.

Tracking a Nonstationary Problem



Tracking a Nonstationary Problem

- In case of Nonstationary, it makes sense to weight recent rewards more heavily than long-past ones.
- One of the most popular ways of doing this is to use a constant step-size parameter.
- For example, the incremental update rule for updating an average Q_k of the $k-1$ past rewards is modified to be :

$$Q_{k+1} = Q_k + \alpha \left[R_k - Q_k \right]$$

where the step-size parameter $\alpha \in (0, 1]$ is constant. This results in Q_{k+1} being a weighted average of past rewards and the initial estimate Q1

Tracking a Nonstationary Problem

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

- sometimes called an exponential, recency-weighted average

Tracking a Nonstationary Problem

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

E.g.

α=0.8,

so,

$Q2 = (1-0.8)^2 + (0.8\ (1-0.8)^{2-1} + 0.8\ (1-0.8)^{2-2})$

$= (0.2)^2 + (0.16 + 0.8)$

$= 0.04 + 0.96$

$= 1$

Tracking a Nonstationary Problem

- Sometimes it is convenient to vary the step-size parameter from step to step. Let $\alpha_k(a)$ denote the step-size parameter used to process the reward received after the kth selection of action a.
- The choice $\alpha_k(a) = 1/k$ 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 $\{\alpha_k(a)\}$.

Tracking a Nonstationary Problem

- A well known result in stochastic approximation theory gives us the conditions required to assure convergence with probability 1:

$$\sum_{k=1}^{\infty} \alpha_k(a) = \infty \qquad \text{and} \qquad \sum_{k=1}^{\infty} \alpha_k^2(a) < \infty$$

- 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.

Optimistic Initial Values

- All the methods we have discussed so far are dependent to some extent on the initial action-value estimates, Q1(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 (Qk+1)as given by equation is previous slide.
- In practice, this kind of bias is usually not a problem, and can sometimes be very helpful.

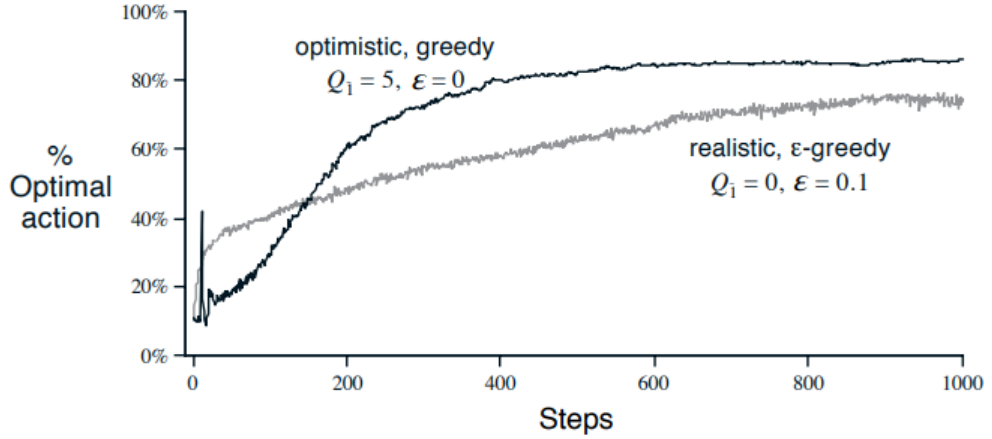
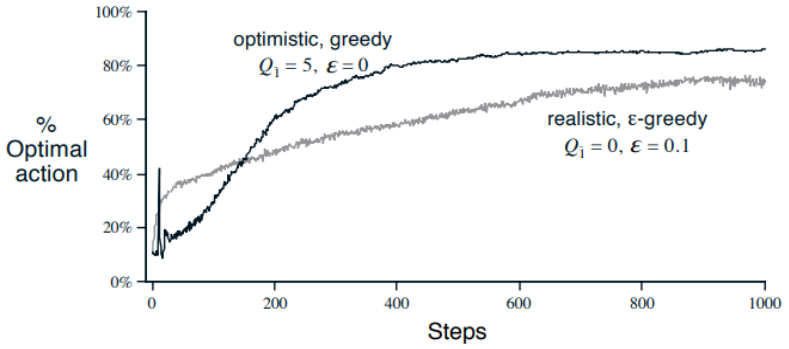


Figure : The effect of optimistic initial action-value estimates on the 10- armed testbed. Both methods used a constant step-size parameter, α = 0.1.

Optimistic Initial Values

- 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 of encouraging exploration

Optimistic Initial Values



- Figure shows the performance on the 10-armed bandit testbed of a greedy method using Q1(a) = +5, for all a.
- For comparison, also shown is an ε-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

Upper-Confidence-Bound Action Selection

- Considering the working of Greedy and ε-Greedy algorithm, 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 as

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

- 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), 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 Action Selection

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

- 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 the c parameter determining the confidence level.
- Each time a is selected the uncertainty is presumably reduced; $N_t(a)$ is incremented and, as it appears in the denominator of the uncertainty term, the term is decreased.

UCB Action Selection

- As shown, UCB generally performs better than ε-greedy action selection, except in the first n 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.

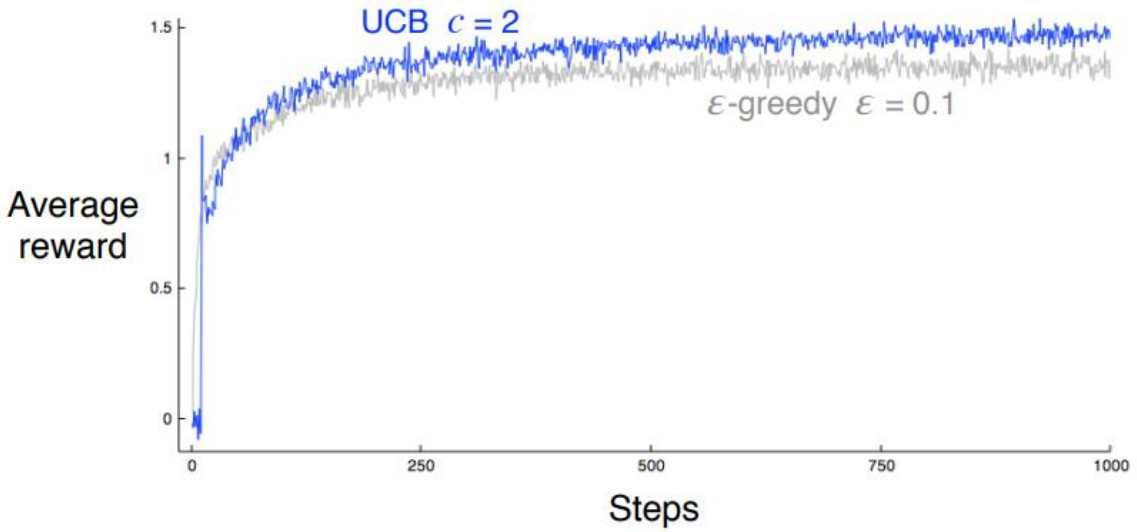


Figure: Average performance of UCB action selection on the 10-armed testbed.

UCB Action Selection

- 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.
- One difficulty is in dealing with nonstationary problems; something more complex than the methods discussed so far would be needed.
- Another difficulty is dealing with large state spaces

Thompson Sampling Algorithm

- All the methods we've seen for tackling the Bandit Problem have selected their actions based on the current averages of the rewards received from those actions.
- Thompson Sampling (also sometimes referred to as the ***Bayesian Bandits*** **algorithm**) takes a slightly different approach; rather than just refining an estimate of the mean reward it extends this, to instead build up a probability model from the obtained rewards, and then samples from this to choose an action.

Thompson Sampling

- Thompson Sampling generates a model of the reward probabilities. When, as in this case, the available rewards are binary (win or lose, yes or no, charge or no charge) then the Beta distribution is ideal to model this type of probability.
- The Beta distribution takes two parameters, 'α' (alpha) and 'β' (beta). In the simplest terms these parameters can be thought of as respectively the count of successes and failures.
- Additionally, a Beta distribution has a mean value given by:

$$\frac{\alpha}{\alpha+\beta} = \frac{\text{number of successes}}{\text{total number of trials}}$$

Thompson Sampling

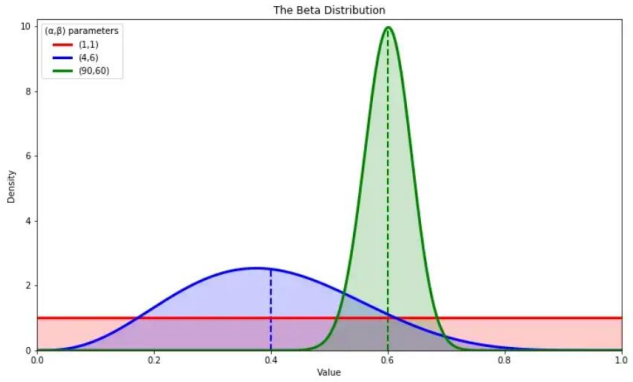


Figure: The Beta distribution for various values of alpha and beta.

Algorithm 1: Thompson Sampling using Beta priors

For each arm $i = 1, \ldots, N$ set $S_i = 0, F_i = 0$.

foreach $t = 1, 2, \ldots,$ **do**

  For each arm $i = 1, \ldots, N$, sample $\theta_i(t)$ from the Beta$(S_i + 1, F_i + 1)$ distribution.

  Play arm $i(t) := \arg\max_i \theta_i(t)$ and observe reward r_t.

  If $r_t = 1$, then $S_{i(t)} = S_{i(t)} + 1$, else $F_{i(t)} = F_{i(t)} + 1$.

end

Prior: Beta(α,β)

Posterior: Head: Beta(α,β+1)

Tail: Beta(α+1,β)

Policy Gradient Approaches

- The goal of any Reinforcement Learning(RL) algorithm is to determine the optimal policy that has a maximum reward.
- Policy gradient algorithm is a policy iteration approach where policy is directly manipulated to reach the optimal policy that maximizes the expected return.
- This type of algorithms is model-free reinforcement learning(RL).

Policy Gradient Approaches

- The environment dynamics or transition probability is indicated as below: $p(s_{t+1}|s,a)$
- It can be read the probability of reaching the next state st+1 by taking the action from the current states.
- Sometimes transition probability is confused with policy.
- Policy π is a distribution over actions given states. In other words, the policy defines the behavior of the agent.

$$\pi(a|s)$$

Return and Reward

- We can define our return as the sum of rewards from the current state to the goal state i.e. the sum of rewards in a trajectory

$$\mathrm{R}(\tau) = \sum_{t=0}^{T-1} R(s_t, a_t)$$

- Where τ = (*s*0 ,*a*0 ,…,*sT*−1 ,*aT*−1).