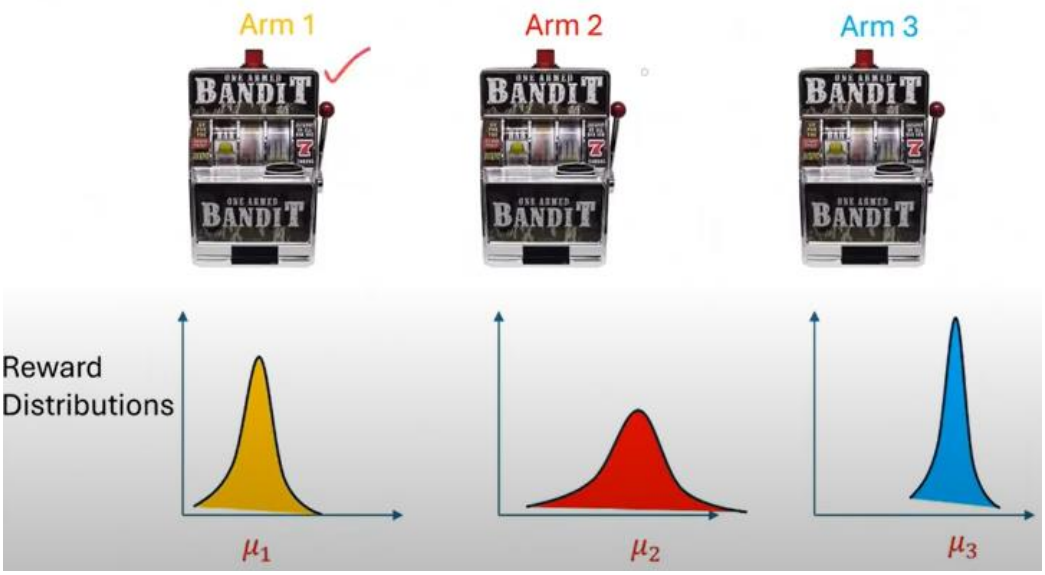
Unit - II

Multi-arm Bandits (MAB)

2.1. An n-Armed Bandit Problem

There are n slot machines (a.k.a. n-arm bandits) each with a stationary probability distribution of rewards. The probability distribution is not known to us. At each step t, we perform an action A_t. i.e. we choose to play with one of the n slot machines then we get a reward of R_t. The goal is to maximise the total reward received over some time period, say 1000 time steps.

This problem is called as n-armed bandit problem. It can be shown as given below.



Here, the mean value of third arm is more so it is better to play with third machine in order to get maximum total reward. But the mean values (i.e. probability distributions of rewards) of machines are not known to us. Then how would we know the best machine for playing??

Multi-arm bandit problem is the simplified version of RL problem because

- There is only one state
- But there are multiple actions
- A reward distribution is associated with each machine
- R_A denote the reward distribution for machine A.

To solve multi-arm bandit problem several approaches are there. One among them is greedy approach.

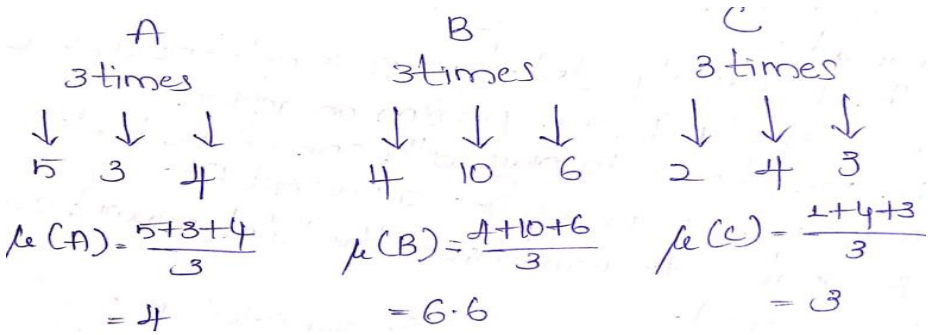
Greedy Approach

Assume that there are 3 machines A, B, C and you are asked to play with them 100 times. How to determine the best machine for playing?

The greedy approach works as follows.

- Play with all machines equal number of times say 3 times each (called exploration)
- Compute the mean reward value of each machine.
- Then play the remaining steps with the machine that has highest mean value (called exploitation)

Example: Assume that there are 3 machines A, B, C and you are asked to play with them 100 times. How to determine the best machine for playing?



Since, machine B has highest mean value, so, play remaining 91 times with machine B to get maximum total reward.

The problems with greedy approach are

- we don't know how much exploration is done before finding the best machine
- though we do exploration to find the best machine but no guarantee that it is the best machine.
- If we do too much exploration then we may left with less trails remaining.
- If we do too low exploration then we may miss the best machine for playing

We may overcome some of these problems using Ɛ-greedy method.

Ɛ-greedy Method

The epsilon-greedy algorithm is one of the simplest strategies for solving the MAB problem. It works as follows:

- With probability epsilon, explore a random arm.
- With probability 1 – epsilon, exploit the arm with the highest estimated reward.

Algorithm

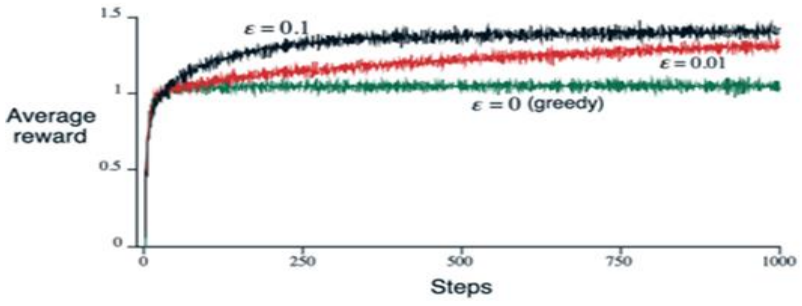
1. Initialize the estimated values of all arms to zero or a small positive number.
2. For each trial:
   - Generate a random number between 0 and 1.
   - If the number is less than epsilon, select a random arm (exploration).
   - Otherwise, select the arm with the highest estimated reward (exploitation).
   - Update the estimated reward of the selected arm based on the observed reward.

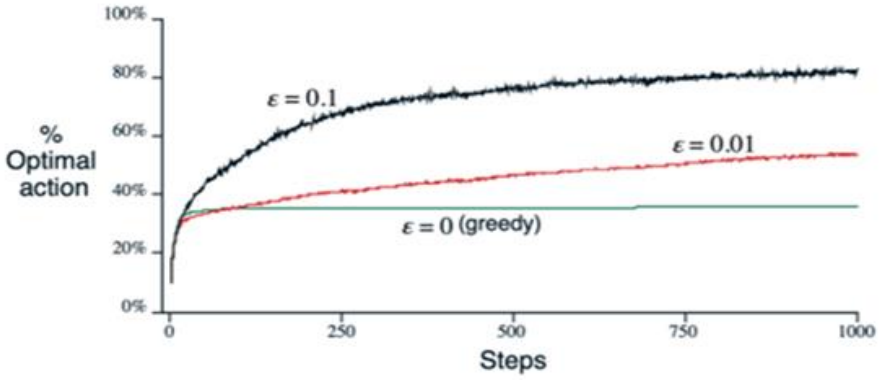
More precisely, Ɛ-greedy method takes the best action or chooses a random action from the set of actions. It always produces maximum total reward than the greedy method.

The following graph shows the comparison between greedy and Ɛ-greedy methods. The first graph shows the following.

- At the beginning the greedy method improves slightly after that it is levelled off.
- The greedy method performs worst in the long run because it gets stuck in a sub-optimal action.
- In long run Ɛ-greedy performs well than greedy method.

The second graph shows that the greedy method finds optimal action (or machine) only 1/3 of tasks whereas the Ɛ-greedy method finds optimal action (or machine) in 2/3 of the tasks.





Applications of N-armed Bandit problems

- Design of experiments (Clinical Trials)
- Online ad placement
- Web page personalization
- Games
- Network(packet routing)

2.2. Action-Value Methods

- Let **q(a**) denotes the true value of action **a**.
- **Q_t(a)** denote the estimated value of action a till time step **t.**
- One method to estimate **Q_t(a)** is given by averaging the reward received.
- Assume that action a is played **N_t(a)** times till **t** and the rewards received are **R_1,R_2,…..$R_{Nt(a)}$.**
- Then the estimated action value is

  $$Q_t(a) = R_1 + R_2 + \cdots + R_{N(a)}/N_{t(a)}$$

  As $N_{t(a)} \rightarrow \infty$, **$Q_t$(a)** converges to **q(a)**(by law of convergence)

- Using this method the action selected is the one that has maximum estimated value. It is called as selecting the greedy action at t and A_t^* for which

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

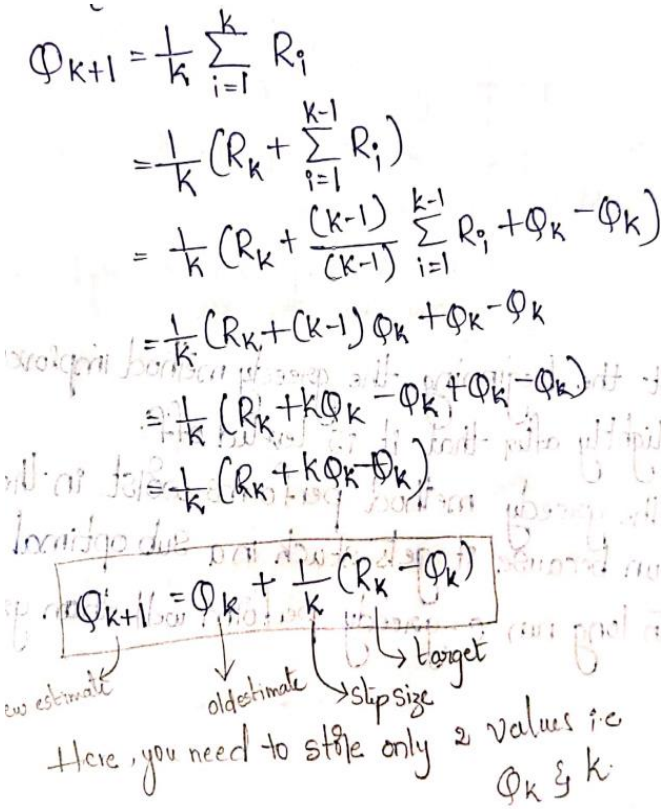
- This can be written as

  $$A_t = \operatorname{argmax}_a Q_t(a) \quad (2)$$

- Simplest action selection rule is to select the action with the highest estimated value.
- argmax_a means the value of **a** for which Q_t is maximised.
- This method is also called as naive method.
- The problem with this method is that it requires storing of huge number of rewards and many computations.
- One solution to this is incremental computation of Q values.

2.3. Incremental Implementation

Let **Q_t(a)** denote the estimated value of **a** till time step **t.** Then q value of an action at time step t+1 can be computed as



- Here you need to store only two values Q_k and k.
- We are updating our estimate of Q_{k+1} as

  NewEstimate←OldEstimate+StepSize [Target−Old Estimate]
- It works even for k=1

2.4. Tracking a Nonstationary Problem

- In a stationary problem the reward probabilities do not change over time .In other words they are fixed.
- But in a non-stationary problem, the probabilities changes over time.

$$\boldsymbol{Q_{k+1} = Q_k + \frac{1}{k}(R_k - Q_k)}$$

$$\boldsymbol{Q_{k+1} = Q_t + \propto (R_k - Q_k)}$$

Where the step-size parameter $\propto \in$ [0,1] is contant

$$\boldsymbol{Q_{k+1} = Q_k + \propto (R_k - Q_k)}$$

$$\boldsymbol{Q_{k+1} = \propto R_k + (1 - \propto) Q_k} \textbf{ (1)}$$

To get Q_k replace k by k-1 in equation (1)

$$\boldsymbol{Q_k = \propto R_{k-1} + (1 - \propto) Q_{k-1}} \qquad (2)$$

Substitute (2) in (1)

$$\boldsymbol{Q_{k+1} = \propto R_k + (1 - \propto)[\propto R_{k-1} + (1 - \propto) \propto Q_{k-1}]}$$

$$\boldsymbol{Q_{k+1} = \propto R_k + (1 - \propto) \propto R_{k-1} + (1 - \propto)^2 Q_{k-1}} \qquad (3)$$

To get Q_{k-1} replace k by k-2 in (1)

$$\boldsymbol{Q_{k+1} = \propto R_k + (1 - \propto) \propto R_{k-1} + (1 - \propto)^2 Q_{k-2} +}$$
$$\boldsymbol{\ldots\ldots\ldots + (1 - \propto)^{k-1} \propto R_1 + (1 - \propto)^k Q_1}$$

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

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

$$(1-\propto)^k Q_1 + \sum_{i=1}^{k} \propto (1-\propto)^{k-i} = 1$$

- In fact, the weight given to each reward decays exponentially into the past. this sometimes called an exponential or recency-weighted average
- Two conditions should be met :
  - Step size must be large enough to overcome the initial conditions.
  - Step size must be small enough to ensure the convergence.

2.5. Optimistic Initial Values

In most of the problems, the estimated value is initialized to 0. i.e.

Q1(a) = 0 Q1(b)=0 Q1(c)=0 ..... q1(k) =0

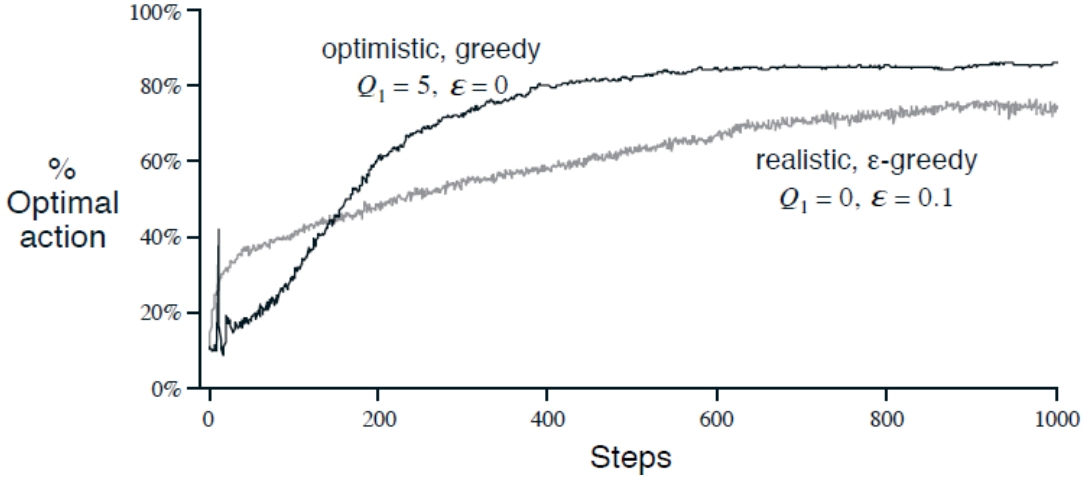
The initial value acts as a bias. Assume that we have initialized the Q values as given below.

	Q1(a) = 0	Q1(b)=0	Q1(c)=0
1st trail reward received	1	10	1.5
2nd trail reward received	1.5	5	2.5

From this we think that machine b is good and the rest of the trails will be played with b in the hope of receiving maximum total reward. In fact, C may be good. Here, the inotialization is not allowing us to play more number of times with the machines to find the best one.

Whereas optimistic intialization will allow us to play more number of times with the machines to find the best one. In optimistic inialization, the q values are initialized to a large value say 10. This allows us to play more number of times with the machines until the average q value of all amchines comes closer to 10. Hence, it leads to finding the best machine correctly.

The performance of this method is shown below.



- At the beginning the performance of optmistic initialization is poor, but in long run it outperforms than Ɛ-greedy method.
- In the above problem, by setting initial values to +5 rather than 0 we encourage

exploration, even in the greedy case. More exploration leads to finding the best machine with a high probability.

- The above method of exploration is called Optimistic Initial Values.

2.6. Upper-confidence-bound Action Selection

The UCB algorithm is based on the principle of optimism in the face of uncertainty. It selects the arm with the highest upper confidence bound, balancing the estimated reward and the uncertainty of the estimate. It works as follows.

Algorithm

1. Initialize the counts and values of all arms.
2. For each trial:
   - Calculate the upper confidence bound for each arm using

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

Wher t is the time step

$Q_t(a)$ is the estimated reward of action 'a' at time step t,

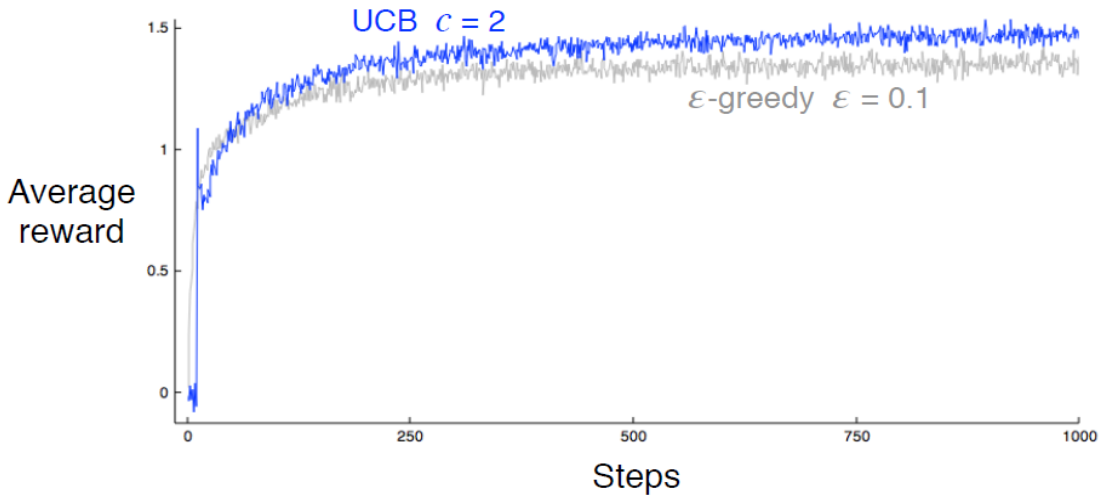
C is a parameter used to control exploration

ln t is the natural logarithm of t

$N_t(a)$ denote number of times the action 'a' was selected prior to t.

   - Select the arm with the highest UCB value.
   - Update the estimated reward of the selected arm based on the observed reward.

As the time step increases, the denominator dominates the numerator.
Each time we select an action our uncertainty decreases because N is the denominator of this equation.
UCB will often perform better than e-greedy methods.
The performance of UCB is shown below.



2.7. Gradient Bandits

The methods discussed so far such as Ɛ-greedy, UCB etc. are based on action value estimation.
But the gradient bandit algorithm uses a different approach. It learns a numerical preference value $H_t(a)$ for each action 'a'.

- Larger the preference, the most often the action is taken.
- Smaller the preference, the less often the action is taken.

It assumes that the probability of taking an action A_t is based on softmax distribution (a.k.a. Gibb's or Boltzman distribution) and is given as below.

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

$\Pi_t(a)$ denote the probability of taking an action 'a' at time t.
Initially $H_t(a) = 0$ for all a.
On each subsequent step an action is played and its preference is updated as

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

Where α is a step parameter, α>0
And $\bar{R}_t$ is the average of all rewards received up to t.

Derivation for the updation equation

The gradient bandit algorithm uses stochastic approximation.

$$H_{t+1}(a) \doteq H_t(a) + \alpha \frac{\partial \mathbb{E}(R_t)}{\partial H_t(a)}$$

where $\mathbb{E}(R_t) = \sum_x \pi_t(x) q(x)$

$$\frac{\partial \mathbb{E}(R_t)}{\partial H_t(a)} = \frac{\partial}{H_t(a)} \sum_x \pi_t(x) q(x)$$

$$= \sum_x q(x) \frac{\partial \pi_t(x)}{\partial H_t(a)}$$

$$= \sum_x (q(x) - B_t) \frac{\partial \pi_t(x)}{\partial H_t(a)}$$

$$= \sum_x \pi_t(x) (q(x) - B_t) \frac{\partial \pi_t(x)}{\partial H_t(a)} \Big/ \pi_t(x)$$

$$= \mathbb{E}\left[(q_x - B_t) \frac{\partial \pi_t(A_t)}{\partial H_t(a)} \Big/ \pi_t(A_t)\right]$$

$$\Rightarrow \frac{\partial \mathbb{E}(R_t)}{\partial H_t(a)} = \mathbb{E}\left[(R_t - \bar{R}_t) \frac{\partial \pi_t(A_t)}{\partial H_t(a)} \Big/ \pi_t(A_t)\right]$$

$$\frac{\partial \pi_t(A_t)}{\partial H_t(a)} = \frac{\partial}{\partial H_t(a)} \pi_t(x)$$

$$= \frac{\partial}{\partial H_t(a)} \frac{e^{H_t(x)}}{\sum_{y=1}^{k} e^{H_t(y)}} \qquad \left[\frac{\partial u}{\partial v} = \frac{v du - u dv}{v^2}\right]$$

$$= \frac{\sum_{y=1}^{k} e^{H_t(y)} \frac{\partial e^{H_t(x)}}{\partial H_t(a)} - e^{H_t(x)} \frac{\partial \sum_{y=1}^{k} e^{H_t(y)}}{\partial H_t(a)}}{\left(\sum_{y=1}^{k} e^{H_t(y)}\right)^2}$$

$$= \frac{\mathbb{1}_{a=x}\, e^{H_t(x)} \sum_{y=1}^{k} e^{H_t(y)} - e^{H_t(x)} \cdot e^{H_t(a)}}{\left(\sum_{y=1}^{k} e^{H_t(y)}\right)^2}$$

$$= \frac{\mathbb{1}_{a=x}\, e^{H_t(x)}}{\sum_{y=1}^{k} e^{H_t(y)}} - \frac{e^{H_t(x)} \cdot e^{H_t(a)}}{\left(\sum_{y=1}^{k} e^{H_t(y)}\right)^2}$$

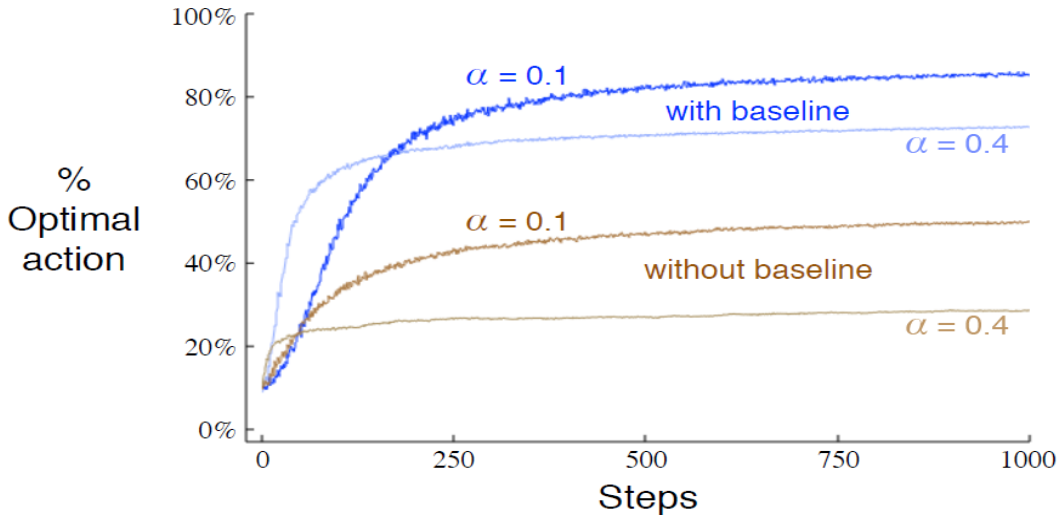
$$= \mathbb{1}_{a=x}\, \pi_t(x) - \pi_t(x) \cdot \pi_t(a)$$

$$\Rightarrow \frac{\partial H_t(x)}{\partial H_t(a)} = \pi_t(x)\left[\mathbb{1}_{a=x} - \pi_t(a)\right]$$

$$\Rightarrow \frac{\partial \mathbb{E}(R_t)}{\partial H_t(a)} = \mathbb{E}\left[(R_t - \overline{R_t})\left(\mathbb{1}_{a=x} - \pi_t(a)\right)\right]$$

$$\therefore H_{t+1}(a) = H_t(a) + \alpha (R_t - \overline{R_t})\left(\mathbb{1}_{a=x} - \pi_t(a)\right)$$

The performance of gradient bandit algorithm is shown below.



The diagram shows that gradient bandit algorithm performs well with baseline.

2.8. Associative Search (or Contextual Bandits)

The bandit problems discussed so far shall not associate context to the actions or situations.

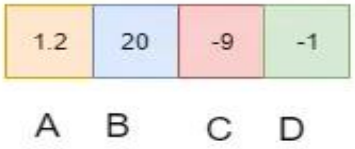
But in some applications, associating context with the situations makes sense.

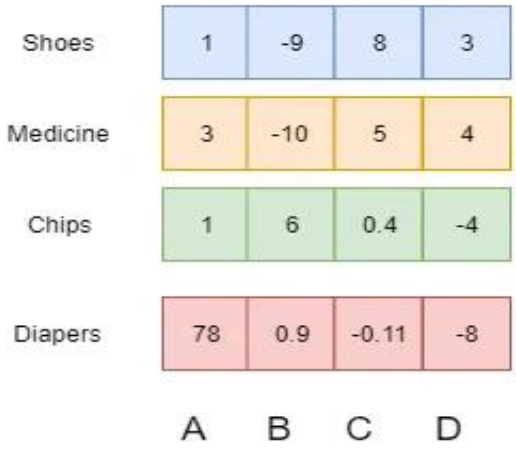
For example, in Add placements. Instead of showing the same add to all the users it is better to show different adds to different users. For example

- Show add related shoes for all users who are interested in shoes
- Show add related medicines for all users who are searching for medicines
- Show add related diapers for all users who are interested in diapers

- And so on

This is shown below.





The above figure shows that first one is a normal bandit. In this whenever we want to play then machine B is given as it is best.

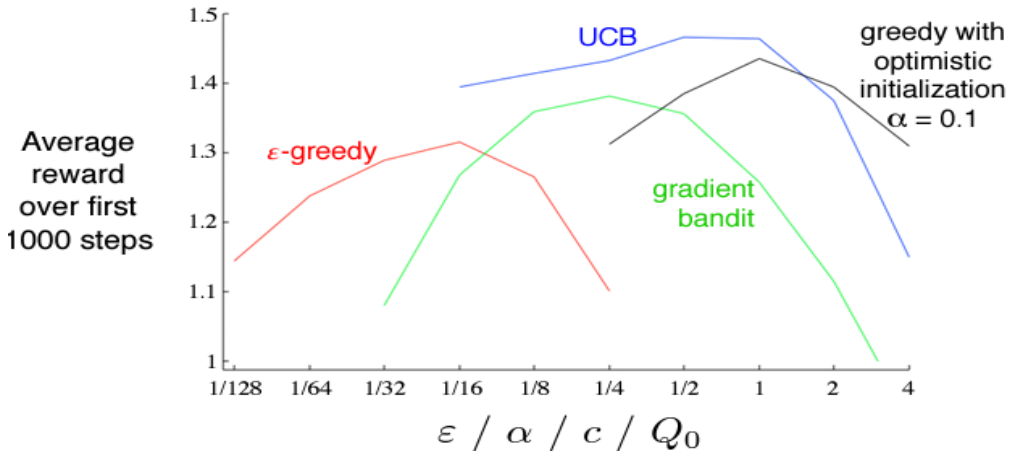
The second one shows contextual bandits where depending on the context, the appropriate machine is given. For example

- if the context is shoes then machine C is given
- if the context is medicines then machine C is given
- if the context is chips then machine B is given
- if the context is diapers then machine A is given

Such bandits are called as contextual bandits.

2. 9. Summary

The following diagram shows the summary of all bandit algorithms.



The figure shows that UCB performs better

Then Greedy algorithm with optimistic initial values

Then gradient bandit

And last is Ɛ-greedy