cct | College Dublin
Computing • IT • Business

Machine Learning for Data Analysis

MSc in Data Analytics

CCT College Dublin

Reinforcement Learning

Week 12

Lecturer: Dr. Muhammad Iqbal*

Email: miqbal@cct.ie

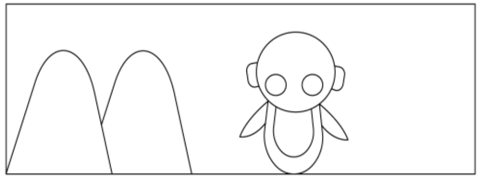
Agenda

- Introduction to Reinforcement Learning (RL)
- The First RL Algorithm
- Reinforcement Learning Examples & Terminology
- Types of Reinforcement Learning
- RL algorithm
- Multi-Armed Bandit Problem
- Exploration-Exploitation
- Action-Value Function
- Upper Confidence Bound
- Thompson Sampling
- Thompson Sampling Algorithm

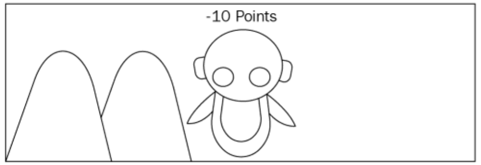
Introduction

Reinforcement Learning (RL)

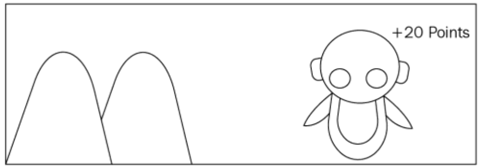
- Consider that you are teaching the dog to catch a ball, but you cannot teach the dog explicitly to catch a ball; instead, you will throw a ball, and every time the dog catches the ball, you will give it a cookie. If it fails to catch the ball, you will not give a cookie. The dog will figure out what actions made it receive a cookie and will repeat those actions.
- Similarly, in a RL environment, you will not teach the agent what to do or how to do instead, you will give a reward to the agent for each action it does.
- In the previous analogy, the dog represents the agent. Giving a cookie to the dog upon catching the ball is a positive reward, and not giving a cookie is a negative reward.
- Imagine you want to teach a robot to walk without getting stuck by hitting a mountain, but you will not explicitly teach the robot not to go in the direction of the mountain.



- Instead, if the robot hits and get stuck on the mountain, you will take away ten points so that robot will understand that hitting the mountain will result in a negative reward and it will not go in that direction again.

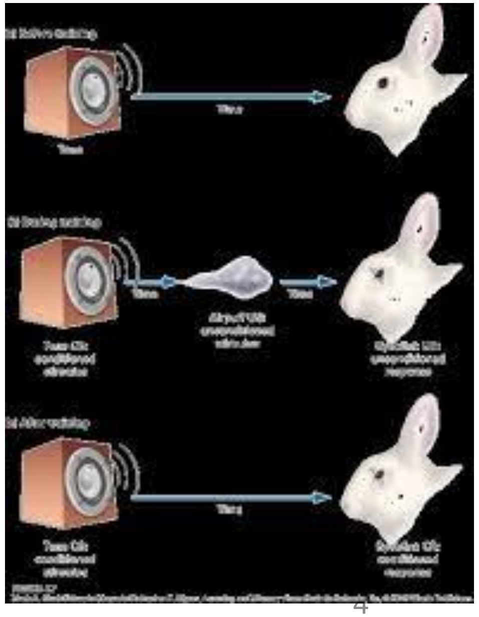


- You will give 20 points to the robot when it walks in the right direction without getting stuck. So the robot will understand which is the right path and will try to maximize the rewards by going in the right direction.



The First RL Algorithm

- In 1972 Robert Rescorla and Allan Wagner found an interesting phenomenon. They first blew a puff of air into a rabbit's eye, which caused it to blink. They then trained the rabbit to associate an external stimulus, a sound, with the puff of air.
- The rabbit blinked when it heard the sound, even if there was no puff of air. They then retrained the rabbit to blink when exposed to both **a sound** and **a light**.
- When the rabbit heard the sound and saw the light with no puff of air, it blinked. But next, when the researchers only flashed the light, the rabbit did not blink. The rabbit had developed a hierarchy of expectations; **sound and light equals blink.**
- When the rabbit did not observe the base expectation (the sound), this blocked all subsequent conditioning. You may have experienced this sensation yourself. Occasionally you learn something so incredible, so fundamental, you might feel that any derivative beliefs are also invalid.
- Your lower-level conditioning has been violated and your higher-order expectations have been blocked. The result of this work was the **Rescorla–Wagner model**.



Terminology

- Some important terms related to reinforcement learning are
- **Agent:** a hypothetical entity which performs actions in an environment to gain some reward.
- **Action (a):** All the possible moves that the agent can take.
- **Environment (e):** A scenario the agent has to face.
- **State (s):** Current situation returned by the environment.
- **Reward (R):** An immediate return sent back from the environment to evaluate the last action by the agent.
- **Policy (π):** The strategy that the agent employs to determine next action based on the current state.
- **Value (V):** The expected long-term return with discount, as opposed to the short-term reward **R**. $\mathbf{V_\pi(s)}$, is defined as the expected long-term return of the current state **s** under policy **π**.
- **Q-value or action-value (Q):** Q-value is similar to Value, except that it takes an extra parameter, the current action **a**. $\mathbf{Q_\pi(s, a)}$ refers to the long-term return of the current state **s**, taking action **a** under policy **π**.

Types of Reinforcement Learning

- **Passive Reinforcement Learning:** In case of passive RL, **the agent's policy is fixed** which means that it is told what to do. Agent executes a fixed policy and evaluates it.
  - **Direct Utility Estimation:** Suppose we have a 4 x 3 grid as the environment in which the agent can move either **Left**, **Right**, **Up** or **Down** (set of available actions). An example of a run and the total reward is 0.72.

$$(1,1)_{-0.04} \rightarrow (1,2)_{-0.04} \rightarrow (1,3)_{-0.04} \rightarrow (1,2)_{-0.04} \rightarrow (1,3)_{-0.04} \rightarrow (2,3)_{-0.04} \rightarrow (3,3)_{-0.04} \rightarrow (4,3)_{+1}$$

  - **Adaptive Dynamic Programming (ADP): ADP** is a smarter method than Direct Utility Estimation as it runs trials to learn the model of the environment by estimating the utility of a state as a sum of reward for being in that state and the expected discounted reward of being in the next state.

$$U^{\pi}(s) = R(s) + \gamma \sum_{s'} P(s'|s, \pi(s)) U^{\pi}(s')$$

  - **Temporal Difference Learning (TD):** TD learning does not require the agent to learn the transition model. The update occurs between successive states and agent only updates state that are directly affected.

$$U^{\pi}(s) \leftarrow U^{\pi}(s) + \alpha(R(s) + \gamma(U^{\pi}(s') - U^{\pi}(s))$$

Types of Reinforcement Learning

- **Active Reinforcement Learning:** In active RL, an agent *needs to decide what to do* as there's no fixed policy that it can act on. So, **the goal of an active RL agent is to act and learn an optimal policy**. Active Reinforcement Learning can be of the following categories
  - **Q-Learning:** Q-learning is a TD learning method which does not require the agent to learn the transitional model, instead learns Q-value functions *Q(s, a)*.

$$U(s) = \max_a Q(s, a)$$

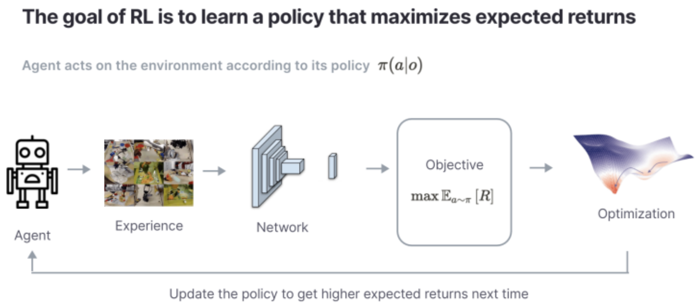
  - **ADP with exploration function:** As the goal of an active agent is to learn an optimal policy, the agent needs to learn the expected utility of each **state** and **update** its policy.
  - It can be done using a passive ADP agent and then using value or policy iteration it can learn optimal actions. But this approach results into a greedy agent.
  - Hence, we use an approach that gives higher weights to unexplored actions and lower weights to actions with lower utilities.

$$U_{i+1}(s) \leftarrow R(s) + \gamma \max_{a \in A} f\Big(\sum_{s'} P(s'|s, a) U_i(s'), N(s, a)\Big)$$

RL Algorithm

- The **steps** involved in typical **RL algorithm** are as follows

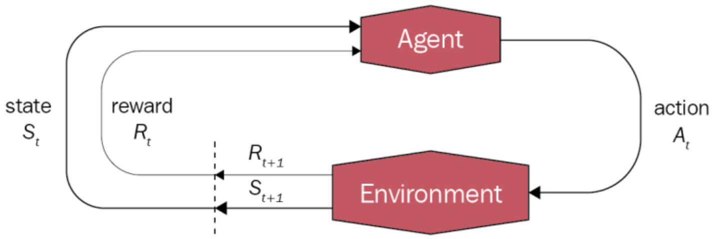
1. First, the **agent** interacts with the environment by performing an **action**.
2. The **agent** performs an **action** and moves from one **state** to another.
3. And then the **agent** will receive a **reward** based on the **action** it performed.
4. Based on the **reward**, the agent will understand whether the **action** was good or bad.
5. If the **action** was good, that is, if the **agent** received a positive **reward**, then the **agent** will prefer performing that **action** or else the **agent** will try performing another **action** which results in a positive **reward**. So it is basically a **trial and error** learning process.



The goal of RL is to learn a policy that maximizes expected returns

Agent acts on the environment according to its policy $\pi(a|o)$

Agent
Experience
Network
Objective
$\max \mathbb{E}_{a\sim\pi}[R]$
Optimization
Update the policy to get higher expected returns next time



Multi-Armed Bandit Problem

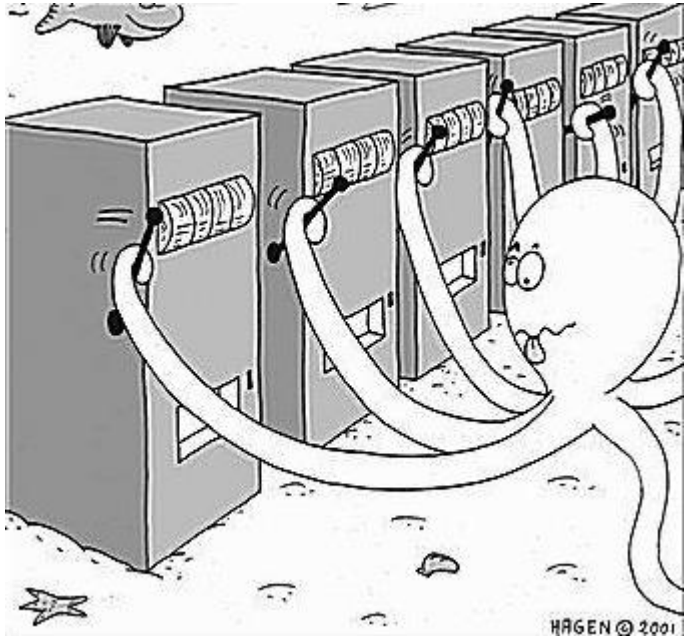
MABP

- Someone who steals your money is known as a **bandit**. A one-armed bandit is a simple slot machine in which you insert a coin, pull a lever, and immediately receive a payoff.

- Why is it referred to as a bandit? It turns out that all casinos set up these slot machines such that every gambler loses money!

- A multi-armed bandit is a sophisticated slot machine featuring numerous levers that a gambler can pull instead of just one, each with a varying return.

- The gambler has no idea what the probability distribution for the payoff associated with each lever is.



- The task is to identify which lever to pull in order to get maximum reward after a given set of trials. This problem statement is like a single step **Markov decision process**. Each arm chosen is equivalent to an action, which then leads to an immediate reward.

Exploration-Exploitation

in the context of Bernoulli MABP

- The table shows the sample results for a **5-armed Bernoulli bandit** with arms labelled as 1, 2, 3, 4 and 5. This is called **Bernoulli**, as the reward returned is either 1 or 0.

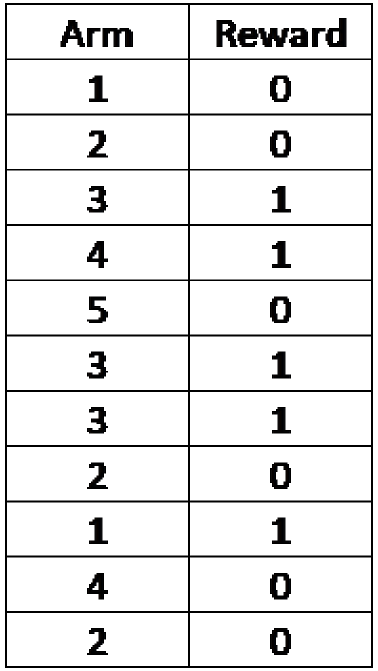
- In this example, it looks like the arm number 3 gives the maximum return and hence one idea is to keep playing this arm in order to obtain the maximum reward (pure exploitation).

- Based on our knowledge from the sample, 5 may appear to be a bad arm to play, but we must remember that we have only played this arm once and that we may need to play it a few more times (exploration) to get confidence.

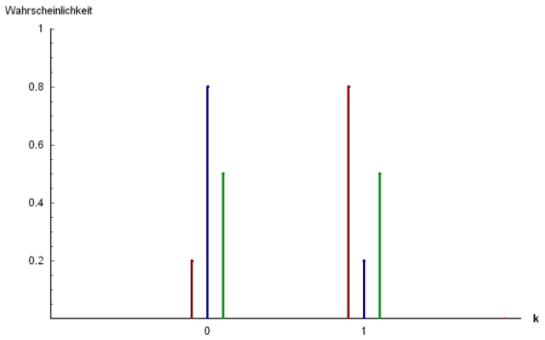
- Only then should we decide which arm to play (exploitation).

- **Use Cases:** Bandit algorithms are being used in a lot of research projects in the industry.

- **Clinical Trials:** The safety of patients participating in clinical trials is as crucial as the study's actual findings. Exploration refers to finding the optimal treatment, whereas exploitation refers to treating patients as effectively as feasible during the trial.



Arm	Reward
1	0
2	0
3	1
4	1
5	0
3	1
3	1
2	0
1	1
4	0
2	0



Bernoulli Distribution

Action-Value Function

- The expected payoff or expected reward can be called an **action-value function**. It is represented by **Q(a)** and defines the average reward for each action at a time **t**.

$$Q(a) = \mathbb{E}[r|a]$$

- Suppose the reward probabilities for a **K-armed bandit** are given by **{P1, P2, P3 …… Pk}**. If the ith arm is selected at time **t**, then **Qt(a) = Pi**.
- The question is, how do we decide whether a given strategy is better than the rest?
- One direct way is to compare the total or average reward which we get for each strategy after **n** trials. If we already know the best action for the given bandit problem, then an interesting way to look at this is the **concept of regret**.

Concept of Regret

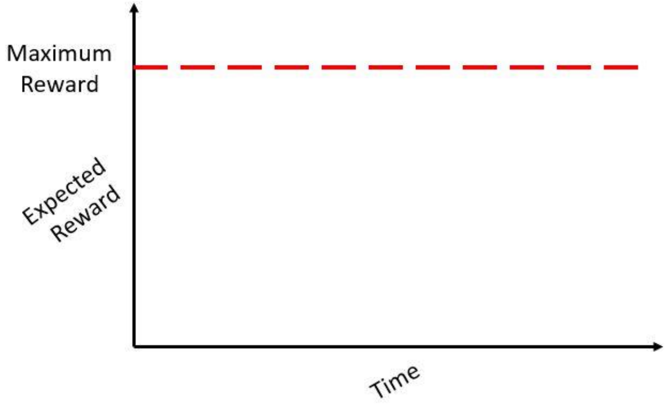
- Assume we already know which arm is the best for the given bandit situation. We can acquire a maximum expected benefit by continually tugging this arm, which can be depicted as a **horizontal line**.

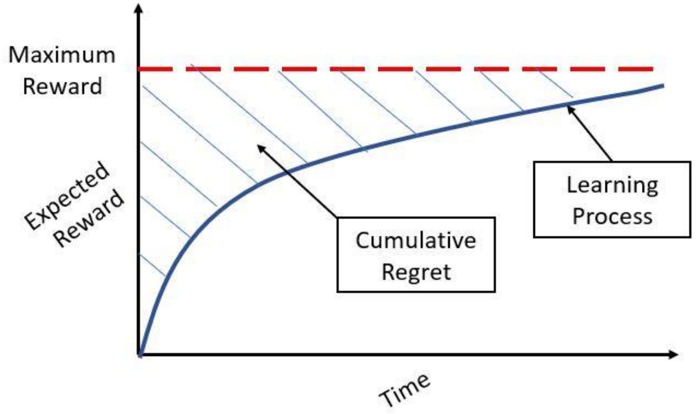
- In a genuine problem statement, we must conduct numerous trials by drawing various arms until we are reasonably certain of the arm to pull for highest average return at time **t**. **Regret** is the loss we suffer as a result of the time/rounds we spend learning.

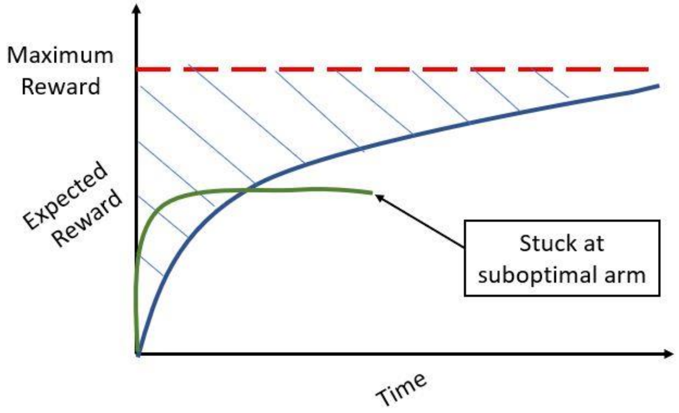
- In other words, we want to maximise the reward during the learning phase. One can wonder how the regret changes if we use a strategy that doesn't undertake enough exploration and relies on a suboptimal arm.

- Although there may be no regret at first, the green curve in the next image shows that we are well below the maximum achievable payoff for the challenge.

- Based on how exploration is done, there are several ways to solve the MABP.





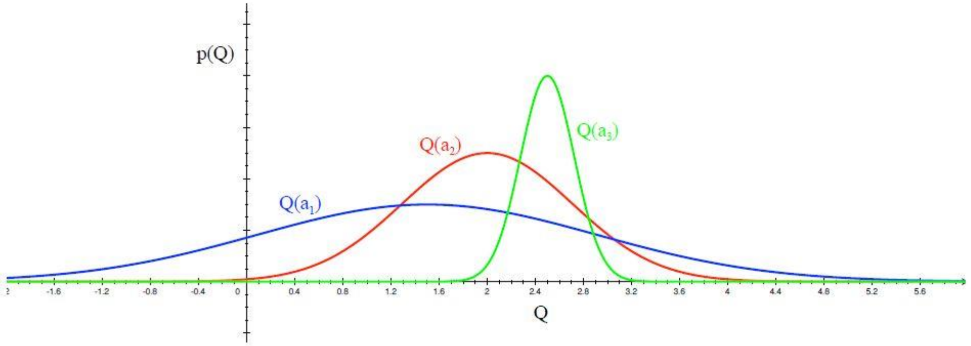


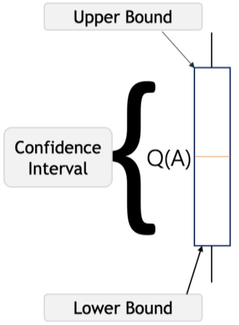
Upper Confidence Bound

UCB

- **Upper Confidence Bound (UCB)** is the most widely used solution method for multi-armed bandit problems. This algorithm is based on the **principle of optimism in the face of uncertainty**.

- In other words, the more uncertain we are about an arm, the more important it becomes to explore that arm.



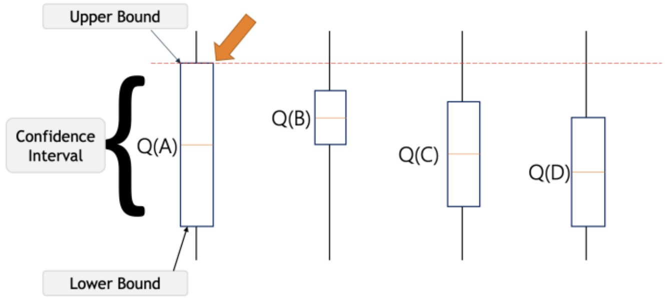


- Distribution of **action-value** functions for 3 different arms a_1, a_2 and a_3 after several trials is shown in the figure above. **This distribution shows that the action value for a_1 has the highest variance and hence maximum uncertainty.**

- **UCB** says that we should choose the arm **a_1** and receive a reward making us less uncertain about its action-value. For the next trial/timestep, if we still are very uncertain about **a_1,** we will choose it again until the uncertainty is reduced below a threshold.

Confidence Interval

UCB



- We can choose the optimal arm using a confidence interval using the UCB method.
- Assume that we each have two arms. After pulling these two arms, we discover that arm one pays us with 0.3 while arm two rewards us with 0.8. However, we cannot conclude that arm two will benefit us the most after just one round of arm pulling.
- It is necessary to pull the arms multiple times, calculate the average reward for each arm, and then choose the arm with the highest mean.
- How can we determine which of these arms' mean values is correct?
- he confidence interval specifies the interval within which the mean reward value of arms lies.
- If arm one's confidence interval is [0.2, 0.9], it suggests that arm one's mean value falls between 0.2 and 0.9. The lower confidence bound is 0.2, and the upper confidence bound is 0.9.
- A machine with a high UCB is chosen by the UCB algorithm.

Confidence Interval

UCB

- We can see that slot machine 3 has a high UCB. But we should not come to the conclusion that slot machine 3 will give us a good reward by just pulling ten times.

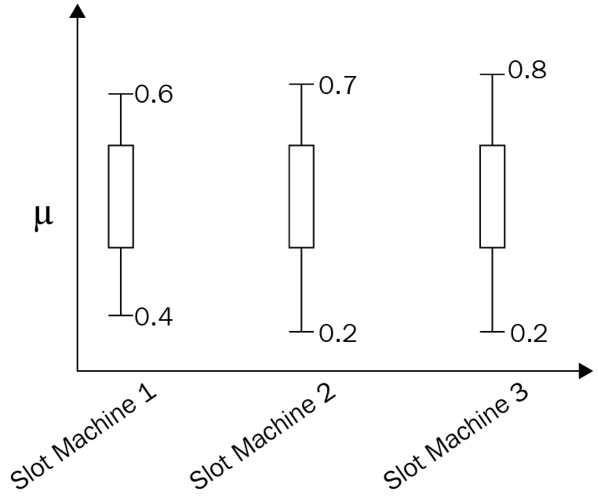
- Once we pull the arms several times, our confidence interval will be accurate.

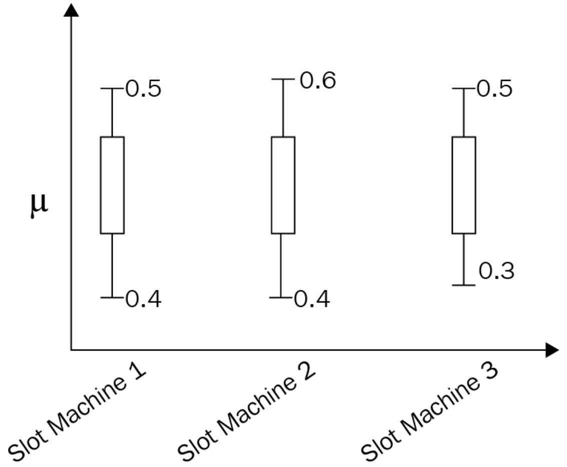
- So, over time, the confidence interval becomes narrow and shrinks to an actual value, as shown in the bottom diagram. So we can select slot machine 2, which has a high UCB. The idea behind UCB is very simple:

1. Select the action (arm) that has a high sum of average reward and upper confidence bound

2. Pull the arm and receive a reward

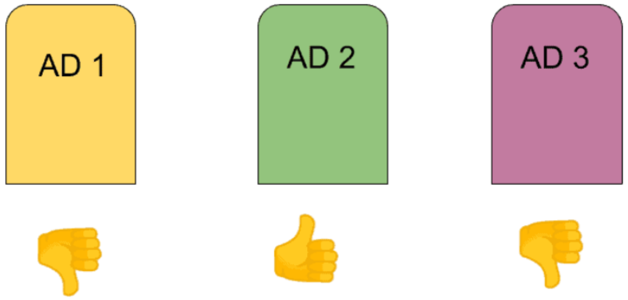
3. Update the arm's reward and confidence bound





$$Arm = argmax_a[Q(a) + \sqrt{\frac{2log(t)}{N(a)}}]$$

UCB Algorithm



- Suppose the advertiser must determine the **click rate for each ad** that promotes the same product. The advertiser's goal is to locate the most effective advertisement.

1. We have **m** ads. The advertiser displays these **ads** to there user when the user visits the web page.
2. Each time a user visits the web page, that makes **one round**.
3. At each round, the advertiser chooses one ad to display to the user.
4. At each round **n**, ad **j** gets reward $\boldsymbol{r_j(n) \in \{0,1\}}$ **:** $\boldsymbol{r_j(n) = 1}$ if user clicked on the **ad**, and **0** if user didn't click the ad.
5. The advertiser's goal is to maximize the total reward from all rounds.

UCB Algorithm

- We consider four advertisements for this computational experiment. And we need to select the best one or the one with the highest click rate.

- Suppose we consider the distributions behind every advertisement as shown in the Figure.

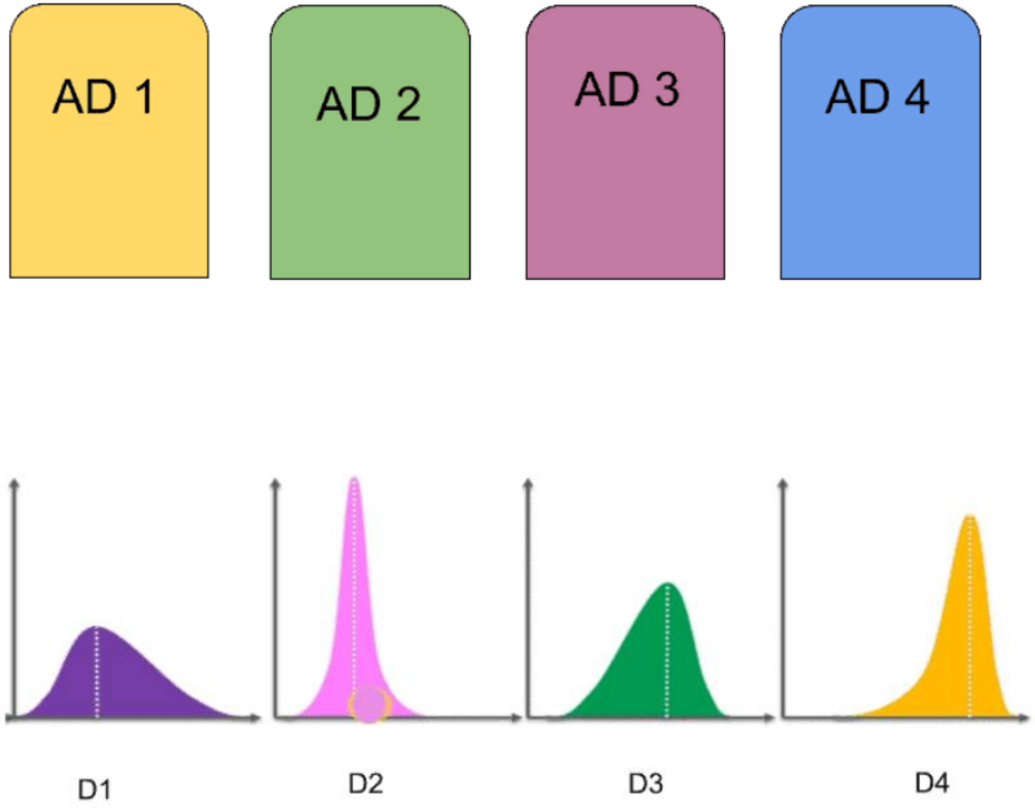
- By observing at these distributions, we can find which one is the best ad.

- The D4 distribution is the left-skewed and we can say that Ad D4 is the best.

- We do not know it before. This is not defined in our problem.

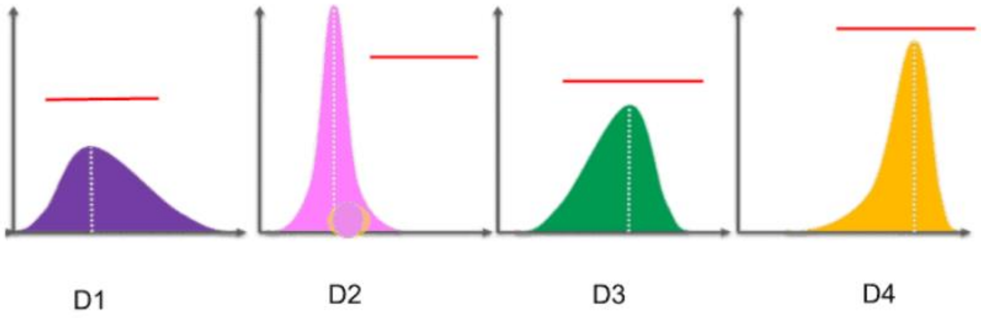
- **Upper Confident Bound Algorithm will find out.**

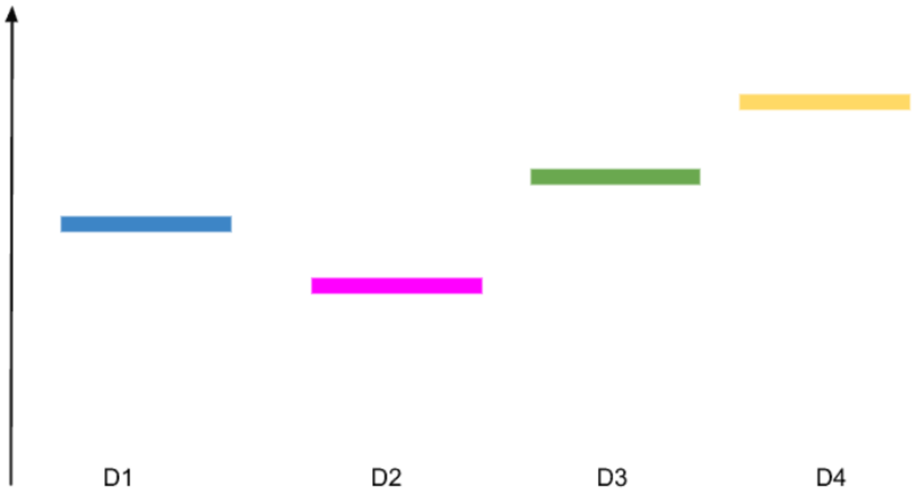
- With the help of Upper Confidence Bound, we find out step by step using iterative approach.



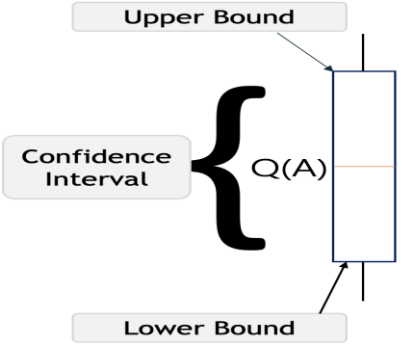
UCB Algorithm

- **Step 1:**
- To understand how Upper confidence bound works.
- Transform the vertical lines of distribution into the horizontal lines.
- After transforming into horizontal lines, it will look something as shown in the bottom figure.
- These are the **expected values or return** for each of these distributions for each ad.

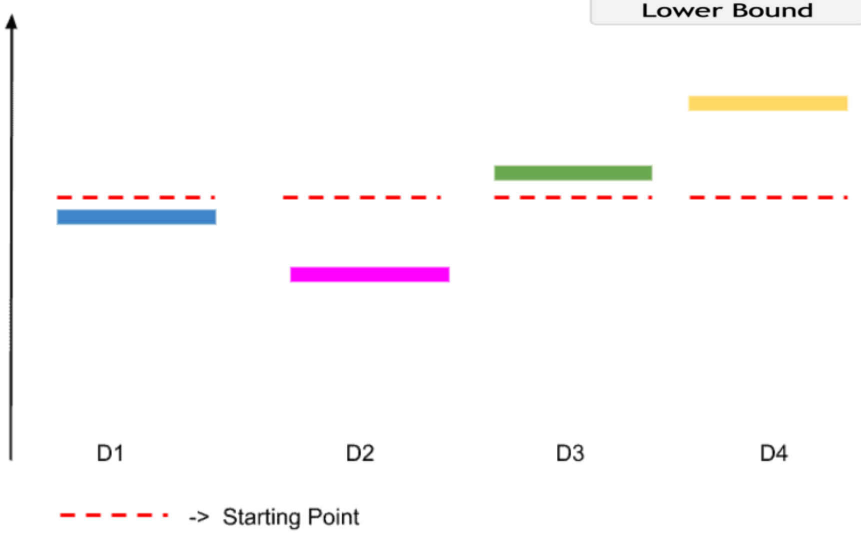


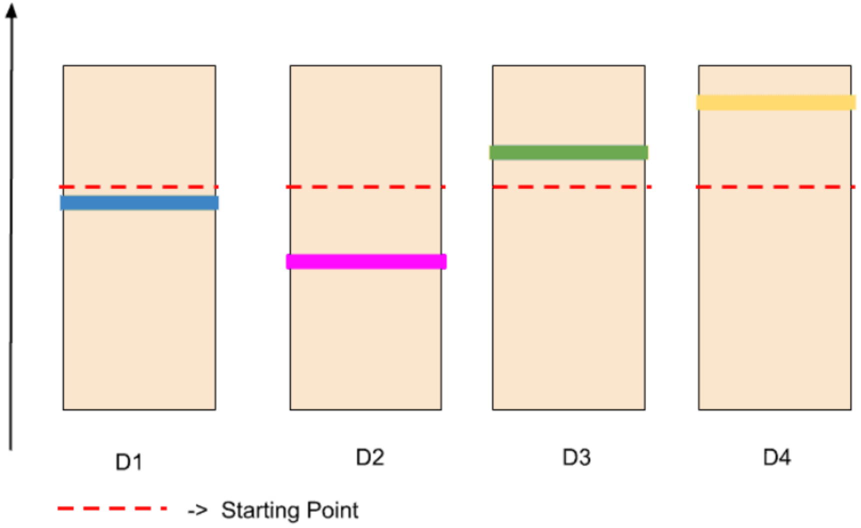


UCB Algorithm

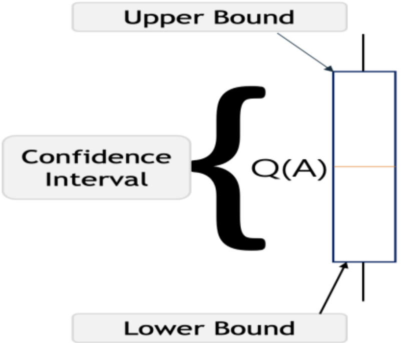


- **Step 2:**
- For each distribution, the upper confidence bound presupposes a starting point.
- We do not know which ad is best, so all ads have the same starting point as shown with dotted red line.
- Assume each ad has this starting point as shown in the figure at the top.
- **Step 3:**
- The formula behind this upper confidence bound algorithm creates confidence bound.
- These bounds are designed in such a way that we have a very **high level of certainty** that confidence bound will include the actual expected return as shown in the bottom image.

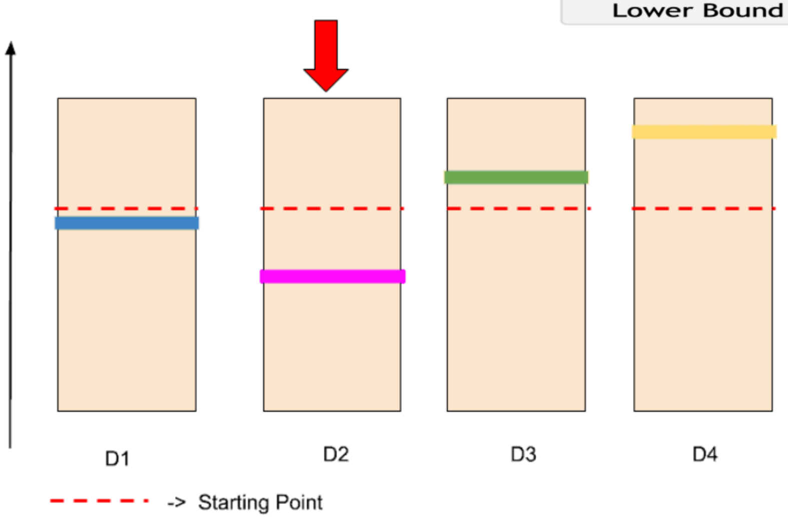


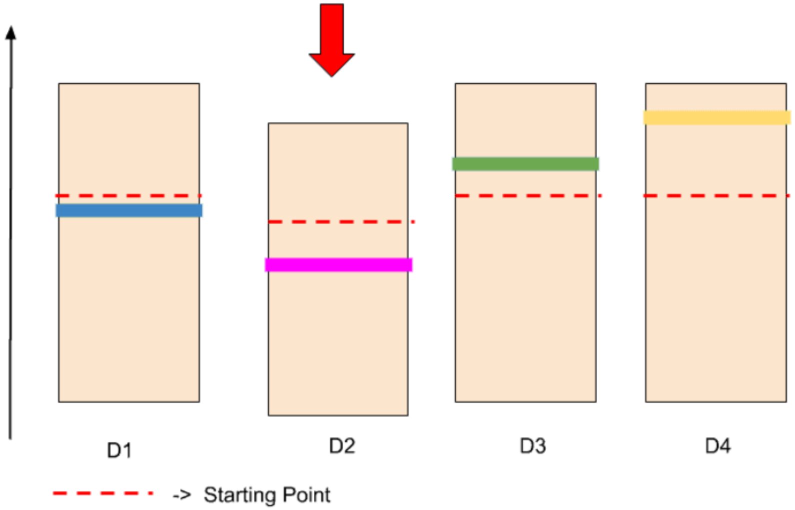


UCB Algorithm

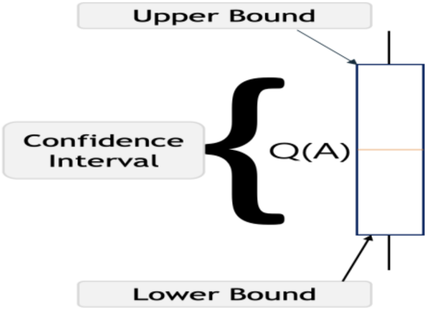


- **Step 4:**
- As Upper confidence bound works on upper bound, we consider those distributions which are closer to the upper bound.
- Initially all have on the same level. **We consider randomly one ad D2.**
- **Step 5:**
- Test this by displaying this ad to the user.
- Assume that the user did not click on this ad.
- Because the user did not click on this ad, so the confidence bound becomes lower. And the red dotted line (the starting point) also goes down. Consequently, the confidence bound shrink also, which means it becomes smaller as shown in the bottom figure.
- Every time, the confidence bound may go up or low, depending upon the ad is clicked or not. During this process, the size of confidence bound also becoming smaller.

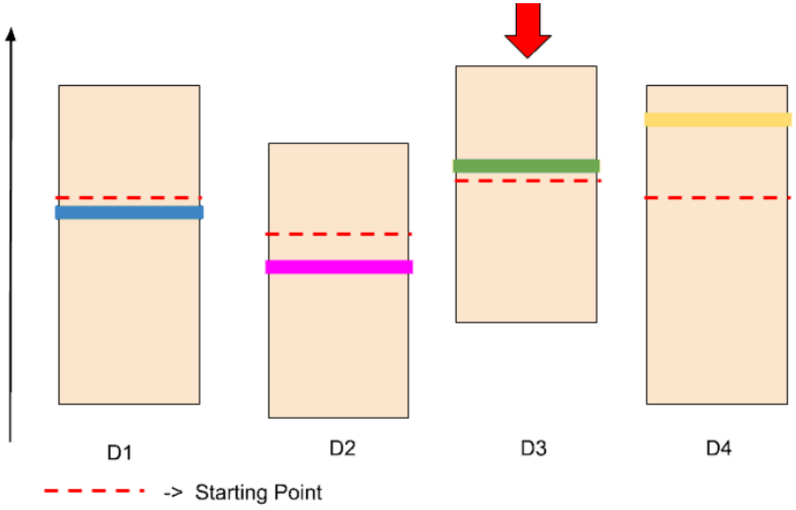


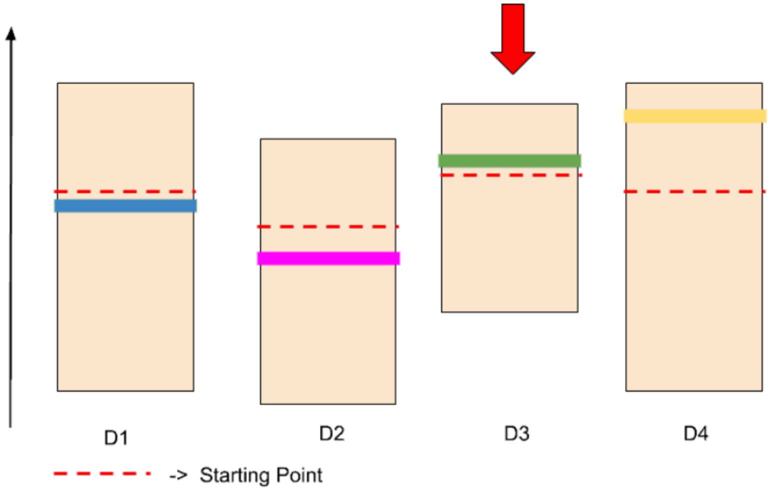


UCB Algorithm

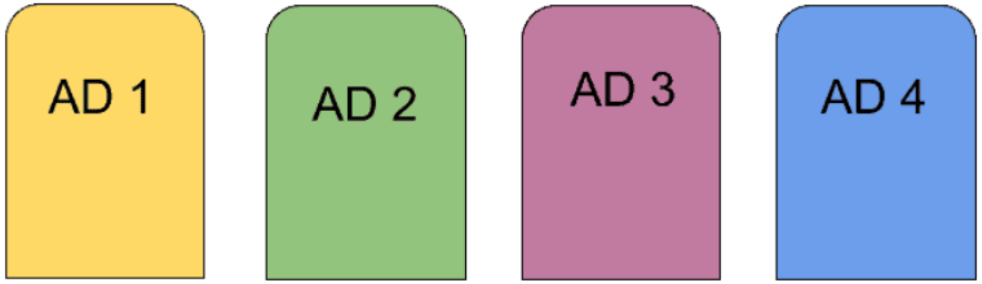


- **Step 6:**
- Now we find the next ad with the high confidence bound. It may be D1, D3, and D4. Because of they all on the same level.
- Assume, we have chosen randomly this ad D3. We perform the same task as we did with D2.
- This will be displayed to the user. **The user click on the ad.** So, this ad or distribution goes a little bit up as shown in the top figure.
- D3 goes up from other distribution. So, this D3 has reached the upper confidence level.
- In the upper confidence bound algorithm, the size of confidence bound also becoming smaller as the user clicks on the ads. Therefore, the confidence bound of D3 shrinks.
- After shrinking, it looks something like that as show in the bottom figure. The red dotted line goes up, but the size of confidence bound becomes small.
- Now, D3 is not on the top of other ads.

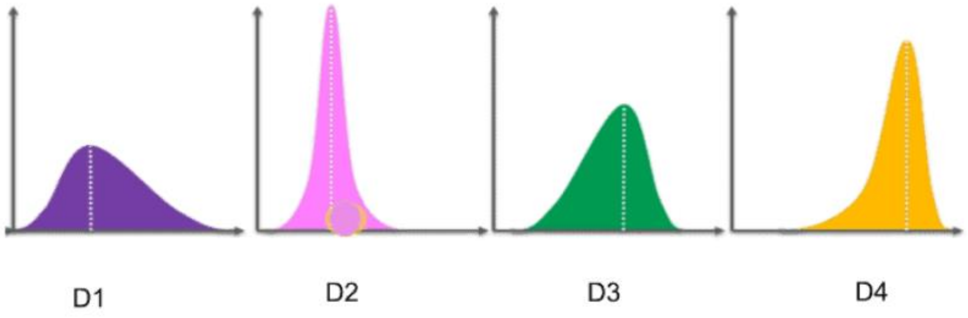


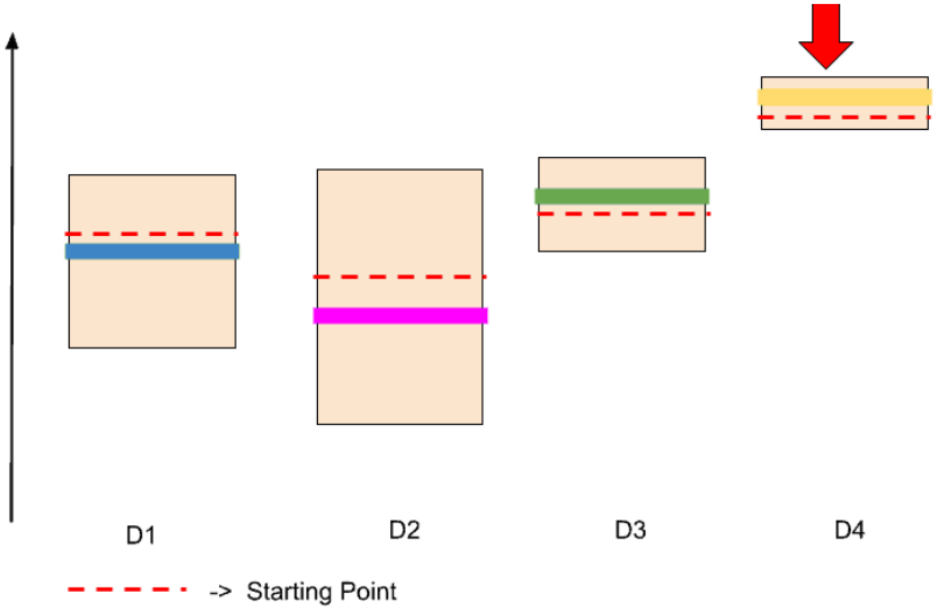


UCB Algorithm



- **Step 7:**
- This process continues until we find the best ad. There may be hundreds or thousands of iterations.
- **The best ad is defined who has highest confidence bound.**
- After several iterations, we found that D4 has the highest confidence bound. Therefore, Ad 4 is the best ad that has more click rates than others.
- The upper confidence Bound Algorithms works perfectly for Advertisement or for any Campaigns.





Upper Confidence Bound

UCB

- **Optimism** is justified and we get a **positive reward** which is the objective ultimately.

- The **optimism** was not justified. In this case, we play an arm that we believed that might give a large reward when in fact it does not. If this happens sufficiently, then we will learn what is the true payoff of this action and not choose it in the future.

- **Steps involved in UCB:**

- Play each of the **K** actions once, giving initial values for mean rewards corresponding to each action for each round **t = K**

- ***$Q_t(a)$*** represents the current estimate for action ***a*** at time ***t***.

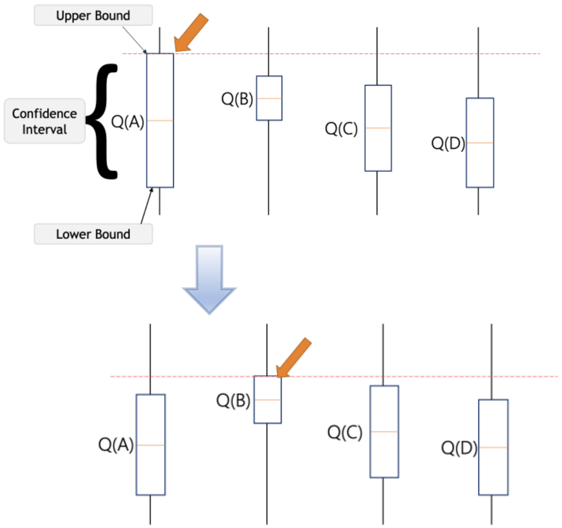
- ***$N_t(a)$*** represent the number of times action **a** was played or taken

- **‘c’** is a confidence value that controls the level of exploration.

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

Explore

Exploit



- Observe the **reward** and update the **mean reward** or **expected payoff** for the chosen action

Upper Confidence Bound

Exploitation vs. Exploration

Exploitation

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

Exploration

- **Exploitation:** ***Q*$_t$*(a)*** represents the exploitation part of the equation. UCB is based on the principle of “optimism in the fact of uncertainty”, which basically means if you don’t know which action is best then choose the one that currently looks to be the best. Taking this half of the equation by itself will do exactly that: the action that currently has the highest estimated reward will be the chosen action.

- **Exploration:** The second half of the equation adds exploration, with the degree of exploration being controlled by the **hyper-parameter ‘c’**. This part of the equation provides a measure of the uncertainty for the action’s reward estimate.

- If an action hasn’t been tried very often, or not at all, then ***N*$_t$*(a)*** will be small. Consequently, the uncertainty term will be large, making this action more likely to be selected. Every time an action is taken we become more confident about its estimate. In this case **N$_t$(a) increments, and so the uncertainty term decreases**, making it less likely that this action will be selected as a result of exploration.

- When an action is not being selected, the uncertainty term will grow slowly, due to **the log function in the numerator**. Whereas, every time that the action is selected, the uncertainty will shrink rapidly due to the increase in ***N*$_t$*(a)*** being linear. So the exploration term will be larger for actions that have been selected infrequently, due to the uncertainty in the estimates of their rewards.

- As time progresses, the exploration term gradually decreases (since as ’n’ goes to infinity log n/n goes to zero), until eventually actions are selected based only on the exploitation term.

Thompson Sampling

- Thompson Sampling is a method that uses **exploration** and **exploitation** to maximise the total rewards gained from completing a task. **Posterior Sampling or Probability** Matching are other names for Thompson Sampling.

- **Exploration** is when an action is repeated several times, while **exploitation** is when additional actions are conducted with the intention of maximising the return depending on the results of the previous actions, either rewards or penalties.

- Some of the areas where it has been used are **revenue management, marketing, web site optimisation, A/B testing, advertisement, recommendation system, gaming and more**.

- How would you determine maximum success if you were given five slot machines and a restricted amount of lever turns? Thompson Sampling is a method that effectively addresses the issue.

- To forecast the success rates of each Slot machine, Thompson Sampling uses Probability Distribution and Bayes Rule.

Thompson Sampling has been around for quite some time. William R. Thompson proposed it for the first time in 1933

single arm/lever slot machine

Intuition Behind

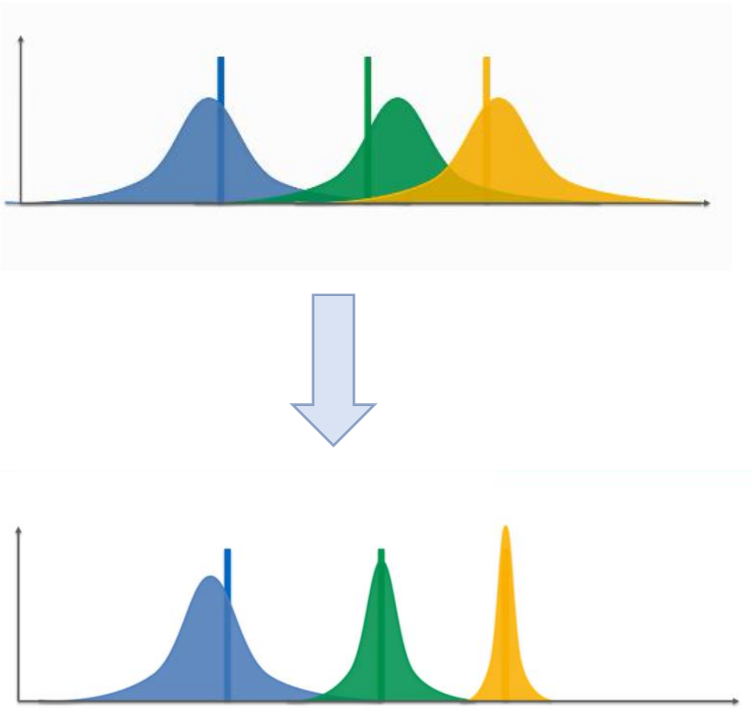
Thompson Sampling

1. To begin with, all machines are assumed to have a **uniform distribution** of the probability of success, in this case getting a reward.

2. For each observation obtained from a Slot machine, based on the reward, a new distribution is generated with probabilities of success for each slot machine.

3. Further observations are made based on these **prior probabilities** obtained on each round or observation which then updates the success distributions.

4. After sufficient observations, each slot machine will have a **success distribution** associated with it which can help the player in choosing the machines wisely to get the maximum rewards.



Thompson Sampling Algorithm

Conceptual Understanding

- Consider the scale, the **horizontal axis is the Return** that we get from the Bandit machine. We are considering **three Bandits** to understand the concept. Each of these machines has a distribution behind it and we do not know.

1. At the start, all the machines are identical.

2. In Figure 2, we have some trial runs. Based on the **trial runs**, the Algorithm will construct a distribution. For example, in Figure 2, for the blue machine we are pulling the values, and we get some values say four based on these values we are constructing a distribution.

3. It does not represent the distributions behind the machines. **We are constructing the distributions of where we think the actual expected value might lie.**

4. This demonstrates that it is a Probabilistic Algorithm (consider Figure 3, the mean values (μ*) can be anywhere within the distribution curve).

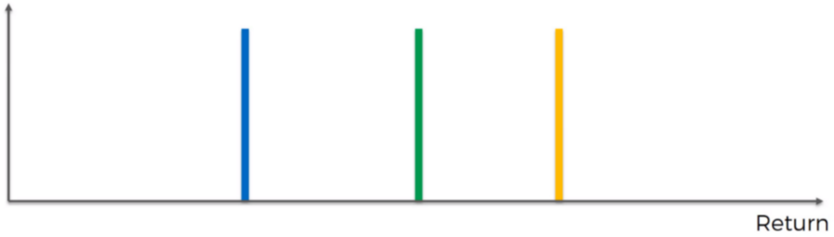


Figure 1

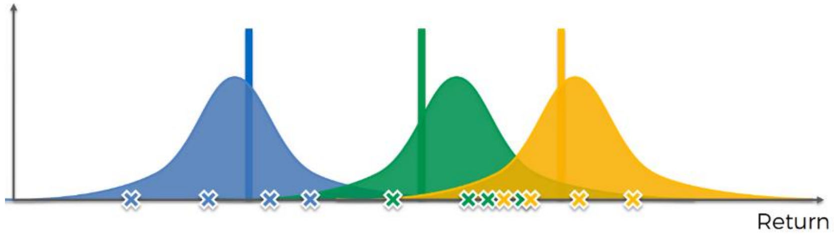
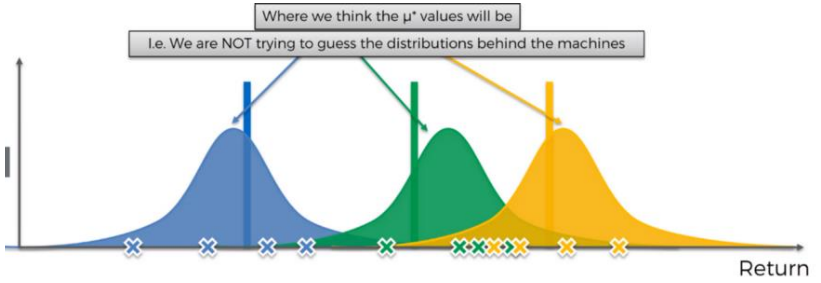


Figure 2


Where we think the μ* values will be

I.e. We are NOT trying to guess the distributions behind the machines

Figure 3

Thompson Sampling Algorithm

Conceptual Understanding

5. In Figure 4, the values are pulled from each distribution (blue, green, yellow), which is highlighted as “**X**”.

6. To solve the problem, we pick the green machine because it has the highest expected return.

7. We have to translate these results which we got from our imaginary set into the actual world.

8. In the hypothetical world, we have selected green machine, so in the actual world the algorithm also selects the green machine and what that does is it pulls the lever for this machine. This value will be based on the distribution behind this green machine (this is the actual expected value of the distribution as shown in Figure 5).

9. We obtained new information (i.e, the new value) which we need to add and adjust the distribution based on this new value. In Figure 6, we can see the shift in the distribution, it has also become narrower. Every time we add new information our distribution becomes more refined.

We have generated our own Bandit Configuration.

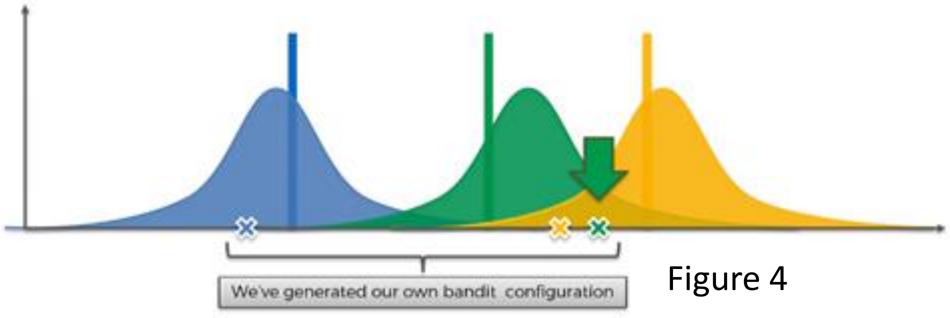


Figure 4

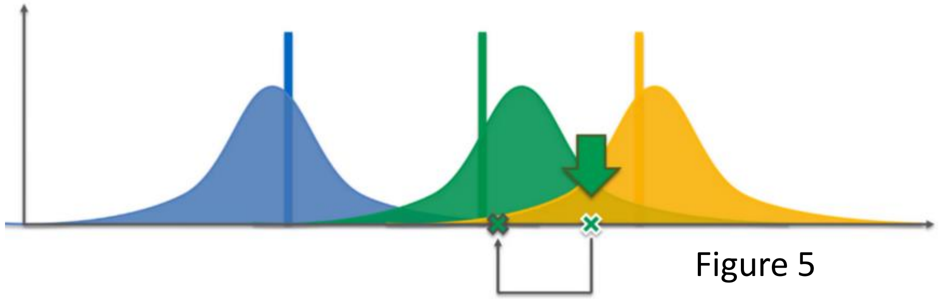


Figure 5

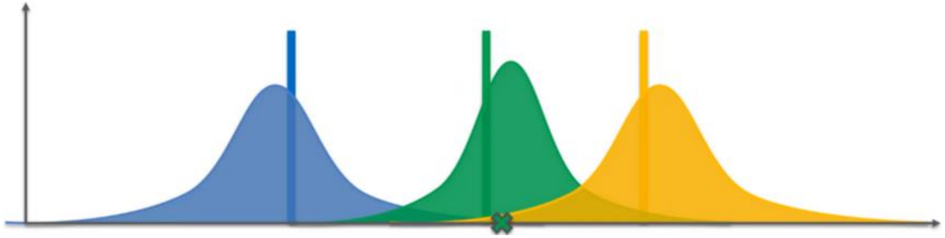


Figure 6

Thompson Sampling Algorithm

Conceptual Understanding

9. What happens next is, **NEW ROUND**.

10. We pull out some new values and generate our own bandit configuration in our hypothetical/ imaginary world. And pick the best bandit, pull the lever of that bandit which gives some value, this is the actual value that we received from the real world. Then we need to in-cooperate this value into our real world and make adjustments accordingly and continue!

11. We need to continue doing this until we get into the point where we have refined the distributions substantially as shown in Figure 8.

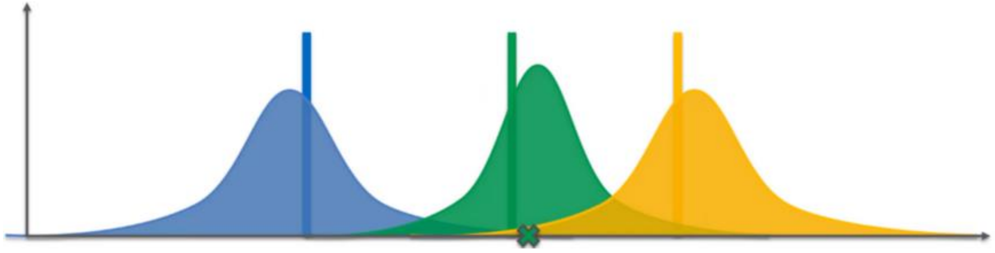


Figure 6

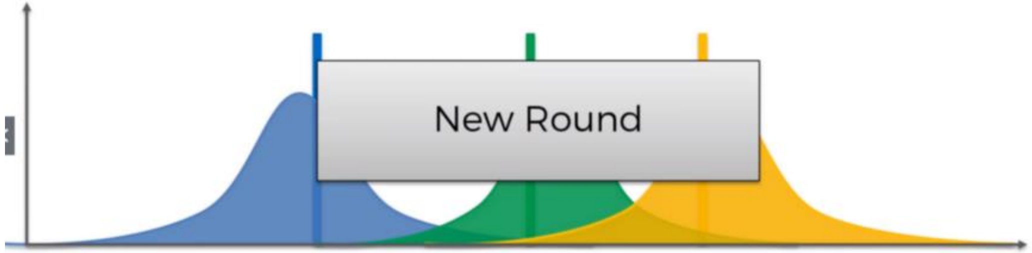


Figure 7

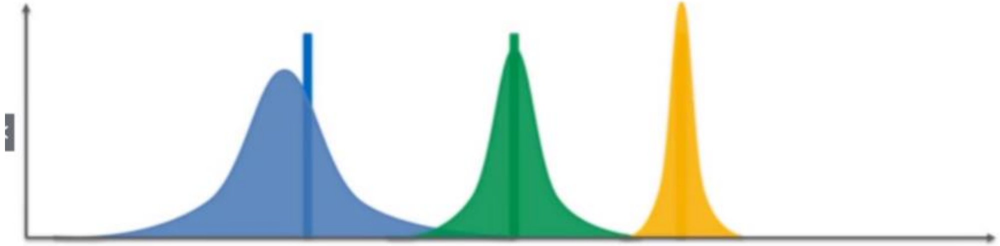
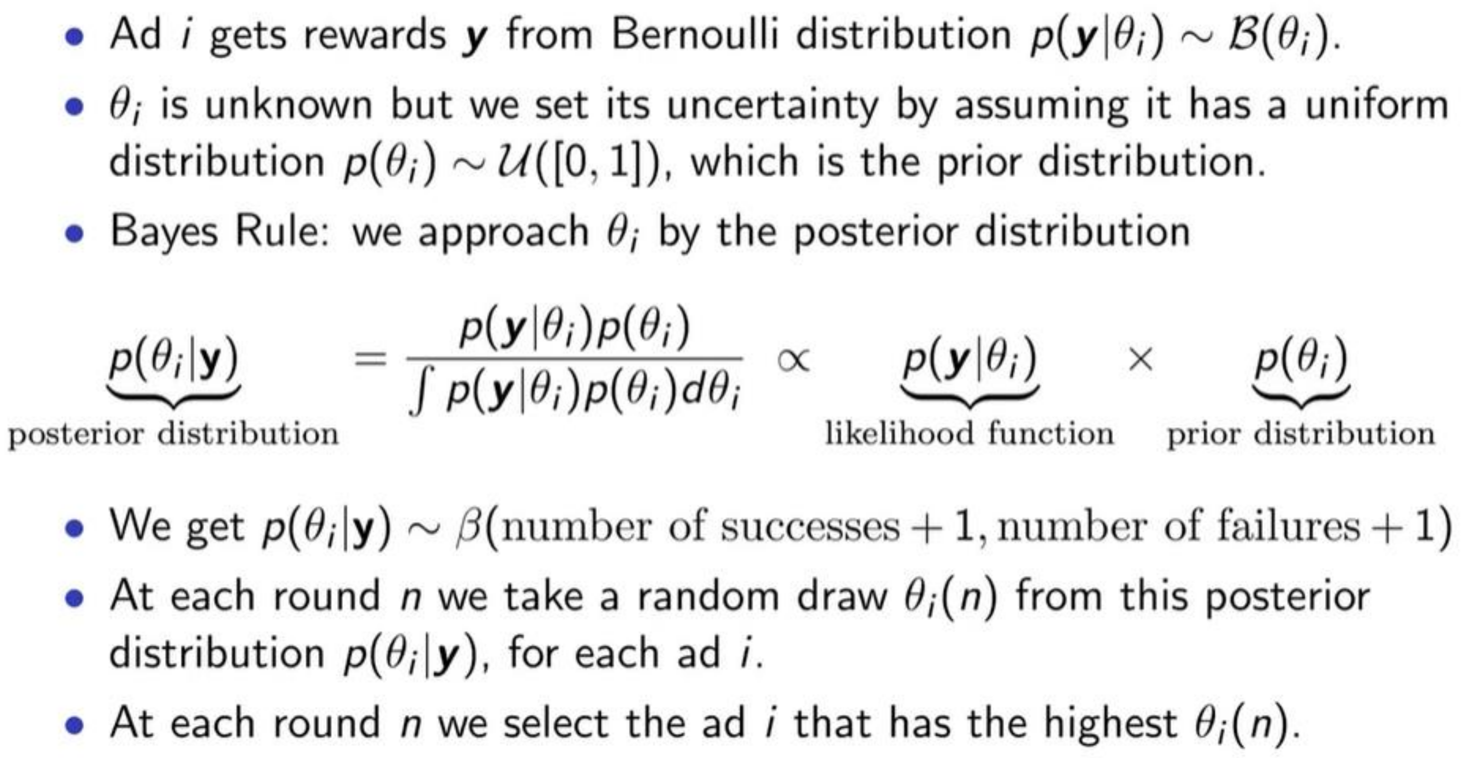


Figure 8

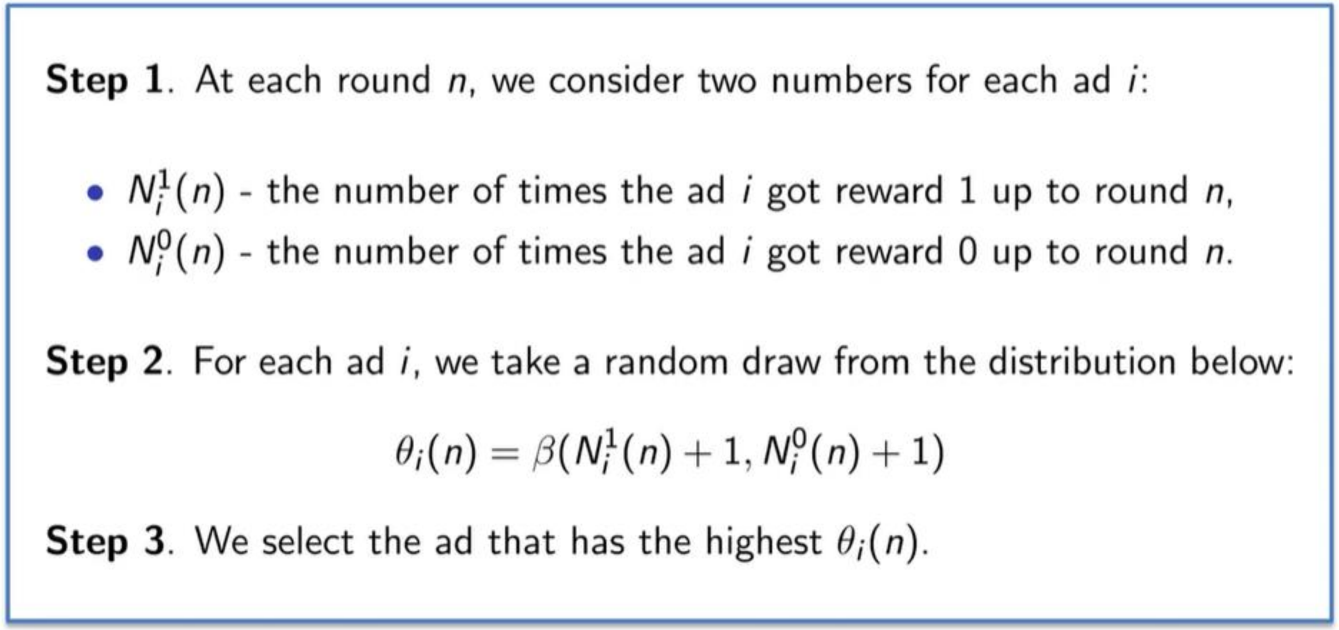
Thompson Sampling Algorithm



- Ad i gets rewards $\mathbf{y}$ from Bernoulli distribution $p(\mathbf{y}|\theta_i) \sim \mathcal{B}(\theta_i)$.
- θ_i is unknown but we set its uncertainty by assuming it has a uniform distribution $p(\theta_i) \sim \mathcal{U}([0,1])$, which is the prior distribution.
- Bayes Rule: we approach θ_i by the posterior distribution

$$\underbrace{p(\theta_i|\mathbf{y})}_{\text{posterior distribution}} = \frac{p(\mathbf{y}|\theta_i)p(\theta_i)}{\int p(\mathbf{y}|\theta_i)p(\theta_i)d\theta_i} \propto \underbrace{p(\mathbf{y}|\theta_i)}_{\text{likelihood function}} \times \underbrace{p(\theta_i)}_{\text{prior distribution}}$$

- We get $p(\theta_i|\mathbf{y}) \sim \beta(\text{number of successes} + 1, \text{number of failures} + 1)$
- At each round n we take a random draw $\theta_i(n)$ from this posterior distribution $p(\theta_i|\mathbf{y})$, for each ad i.
- At each round n we select the ad i that has the highest $\theta_i(n)$.



Step 1. At each round n, we consider two numbers for each ad i:

- $N_i^1(n)$ - the number of times the ad i got reward 1 up to round n,
- $N_i^0(n)$ - the number of times the ad i got reward 0 up to round n.

Step 2. For each ad i, we take a random draw from the distribution below:

$$\theta_i(n) = \beta(N_i^1(n) + 1, N_i^0(n) + 1)$$

Step 3. We select the ad that has the highest $\theta_i(n)$.

Reference and Resources

- Reinforcement Learning, Phil Winder, PhD., O'Reilly Media, Inc., 2020.
- The Reinforcement Learning Workshop, Alessandro Palmas, Emanuele Ghelfi, Dr. Alexandra Galina Petre, Mayur Kulkarni, Anand N.S., Quan Nguyen, Aritra Sen, Anthony So, Saikat Basak, Packt Publishing, August 2020.
- Deep Reinforcement Learning with Python - Second Edition by Sudharsan Ravichandiran Published by Packt Publishing, 2020
- https://www.analyticsvidhya.com/blog/2018/09/reinforcement-multi-armed-bandit-scratch-python/
- https://analyticsindiamag.com/thompson-sampling-explained-with-python-code/
- https://medium.com/analytics-vidhya/reinforcement-learning-thompson-sampling-to-solve-the-multi-armed-bandit-problem-63fbf3f9e37d
- https://learning.oreilly.com/library/view/deep-reinforcement-learning/9781839210686/#publisher_resources
- Introduction to Reinforcement Learning – DataCamp
- Some images are used from Google search repository to enhance the level of learning for educational purpose.
- Thanks to Marina Soledad for providing some slides for preparation of this lecture