Reinforcement Learning (RL) is a type of machine learning where an agent learns how to make decisions by interacting with an environment to maximize a cumulative reward. It finds the best path by time taken on each path. It will come up with a unique solution by itself.

Agent is the model that is being trained via Reinforcement Learning by interacting with environment

Environment is the training situation in which the model must be optimized to

Policy is the strategy that tells how an agent will have to behave at anytime. It act as the mapping between state and present action

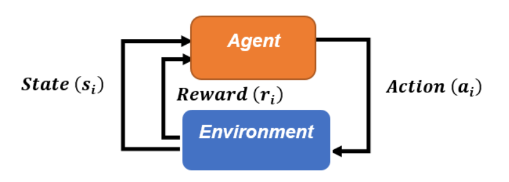
State S is the current position of an agent in an environment

Action A is the steps taken by the agent in a particular state based on the policy

Reward – Agent gets a positive or negative reward for every action it performs. It is given to appraise some action and helps the agent to move in right direction

Episodes - A sequence of actions from an initial state to a terminal state. When agent ends up in terminating state, it will not take any further action

Q value – It is a cumulative reward that determines how good an action A is taken in a particular state S



The key elements of Reinforcement Learning:

• Agent

• Environment

• State and Action

• Reward

Qa (average reward of arm a) =

The optimal arm a\* is the one that gives us the maximum average reward,that is,

a∗= argmax (Qa)

1. An agent is a software or a hardware program.
2. Initially an agent does not know about state, action, and reward.
3. An agent will explore different states.
4. An agent performs actions in different states using a policy defined to it.
5. An agent has to exploit the best action to gain rewards.
6. Thus, an agent learns more and more by interacting with the environment.

Exploration :

An agent primarily improves its knowledge about each action by gathering more information to make best overall action.

Exploitation:

An Agent aims at more(maximised) rewards by the estimated value (greedy).

Agent makes decision based on current information.

Once you found a restaurant you like, you might decide to visit that same restaurant for the rest of your life, exploiting only your positive experience. However, there is a certain appeal in exploring new venues as well. So that you might find a new favorite! or get some traumatic experiences as well. Finding the appropriate balancing between exploring and exploiting is critical.

1. Let’s say your friend and you digging in the hope that they will get diamond out of it. Your friend gets lucky and finds the diamond before you.
2. By seeing this, you might be a bit greedy and thinking that you might get luck. So, you start digging at the same spot as your friend.
3. Your action is called the greedy action and the policy is called the greedy policy.
4. However, in this situation the Greedy policy would fail because a bigger diamond is buried could be where you were digging in the beginning.
5. However, when your friend found the diamond, the only knowledge you got was the depth at which the diamond was buried.

You do not have the knowledge of what lies beyond that depth.

1. In reality the diamond may be where you were digging in the beginning or it may be where your friend was digging, or it may be completely at a different place.

• With such partial knowledge about future states and future rewards, our reinforcement learning agent will be in dilemma on whether to exploit the partial knowledge to receive some rewards or it should explore unknown actions which could result in much larger rewards.

• However, we cannot choose both explore and exploit simultaneously. This is critical.

• Hence, A strategy is required to be defined to overcome the Exploration-Exploitation Dilemma.

**Version 2**- Absolutely, let's break down the diamond digging analogy and the exploration-exploitation dilemma in simpler terms:

Imagine you and your friend are both digging in a hope to find diamonds. Your friend gets lucky and discovers a diamond before you do.

Seeing this, you feel a bit greedy and decide to dig in the same spot your friend found the diamond. This is what we call the "greedy" action, where you directly copy what seems to work for someone else. The plan seems good, right?

But here's the twist. It turns out, there's a bigger diamond buried where you initially started digging before your friend found one. So, the "greedy" policy of just copying what your friend did doesn't always lead to the best outcome.

The tricky part is, you only know how deep your friend found the diamond, but you have no idea what's below or beyond that depth. You don't know if the diamond is where you started digging or somewhere else entirely.

Now, your dilemma is whether to stick to the spot where you know there's something valuable (exploitation), or to try new spots to find potentially bigger rewards (exploration).

Here's the catch: You can't explore and exploit at the same time, it's one or the other. If you only exploit, you might miss out on bigger rewards hidden elsewhere. If you only explore, you might waste time without getting any rewards.

So, you need a smart strategy to tackle this Exploration-Exploitation Dilemma. This strategy should balance trying new things to uncover potentially better outcomes (exploration) while also making use of what you already know works (exploitation). It's like making decisions that give you a good mix of immediate gains and the possibility of even greater gains in the future. That's the challenge in many real-life situations, and it's what reinforcement learning aims to address.

**MULTI ARM BANDITS**

1. Slot machines are one of the most popular games in the casino, where we pull the arm and get a reward.
2. If we get 0 reward then we lose the game, and if we get +1 reward then we win the game.
3. There can be several slot machines, and each slot machine is referred to as an arm.
4. For instance, slot machine 1 is referred to as arm 1, slot machine 2 is referred to as arm 2, and so on. Thus, whenever we say arm n, it actually means that we are referring to slot machine n.

Each arm has its own probability distribution indicating the probability of winning and losing the game.

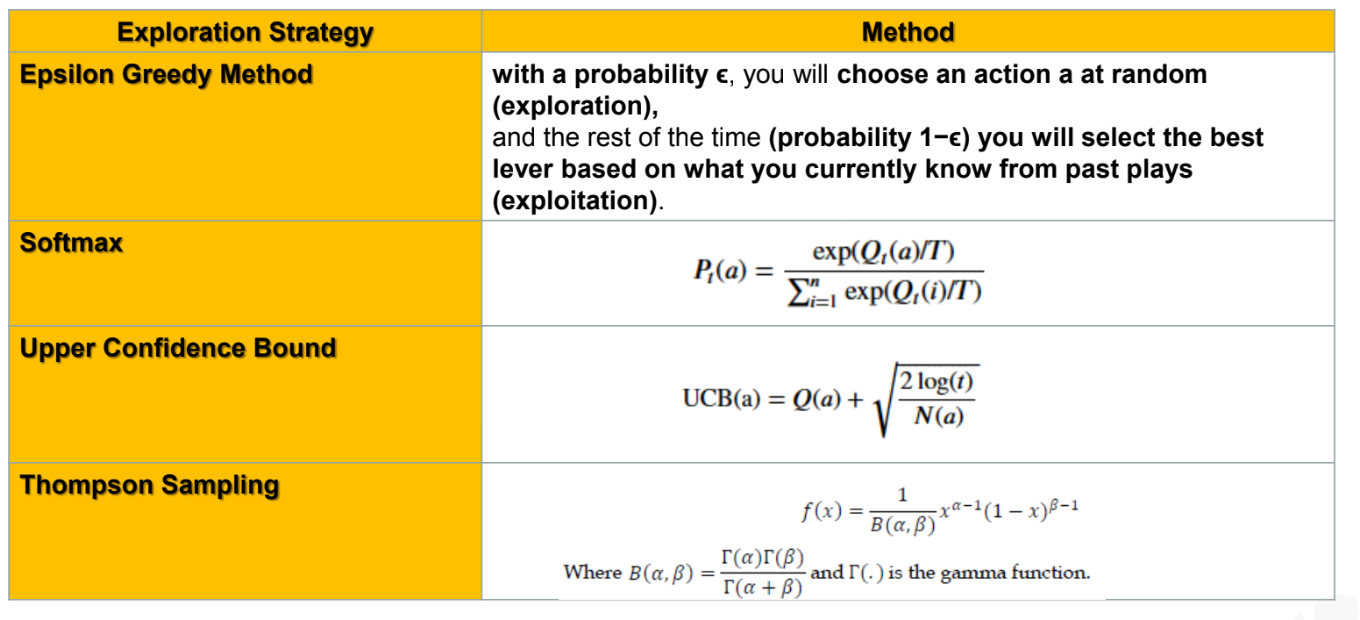
Let the probability of winning if we pull arm 1 (slot machine 1) be 0.7 and the probability of winning if we pull arm 2 (slot machine 2) be 0.5.

**Version 2** : Imagine you're in a casino, and one of the most popular games there involves pulling a lever on a machine. When you pull the lever, you either win or lose. If you win, you get a reward of +1, and if you lose, you get nothing (reward of 0).

Now, picture this: There are multiple of these machines in the casino, and each machine is like a separate game. We call each of these machines an "arm." So, when we talk about arm 1, we mean the first machine, arm 2 is the second machine, and so on. Each arm (machine) has its own way of giving out rewards.

For example, let's say there are two machines (arms) in front of you. The first one, arm 1, has a 70% chance of giving you a reward (winning), and the second one, arm 2, has a 50% chance of giving you a reward. In other words, if you pull arm 1, there's a higher chance of winning compared to pulling arm 2.

This idea of having multiple arms, each with its own way of giving rewards, is what we call a "multi-armed bandit" problem. The challenge here is to figure out which arm to pull to get the most rewards over time. It's like deciding which machine to play to increase your chances of winning. Just like in a casino, you want to find the best strategy to maximize your overall rewards by pulling the right arms.



**Probability Distribution**

1. The uniform distribution, which assigns probability 1/n to each arm, is a stationary distribution.
2. Uniform distribution is used to model events which has the same probability of occurring, such as coin toss, roll of a die, etc.
3. Bernoulli distribution describes events which have a binary outcome, i.e. gives success or failure with a probability.
4. Bernoulli bandits are a RL model used to improve decisions with binary outcomes.
5. They have various applications ranging from headline news selection to clinical trials.
6. Bernoulli Distribution is a type of discrete probability distribution where every experiment conducted asks a question that can be answered only in yes or no.
7. In other words, the random variable can be 1 with a probability p or it can be 0 with a probability (1 - p).
8. Such an experiment is called a Bernoulli trial.
9. A pass or fail exam can be modelled by a Bernoulli Distribution.

**Version 2 :**

**Uniform Distribution:** Imagine you're playing a game where you have several options, and each option has an equal chance of happening. For example, when you toss a fair coin, the chances of getting heads or tails are the same. This kind of situation, where all options are equally likely, is described by a uniform distribution. It's like every possibility has a "fair share" of the chance.

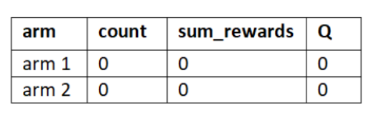
**Bernoulli Distribution:** Think about a situation where there are only two possible outcomes: success or failure. Like flipping a coin and getting heads (success) or tails (failure). Bernoulli distribution is used to describe events like this, where the result can only be a "yes" (success) or "no" (failure). For instance, if you're taking a test and you either pass (success) or fail (failure), that can be modeled using Bernoulli distribution. It's all about describing binary choices.

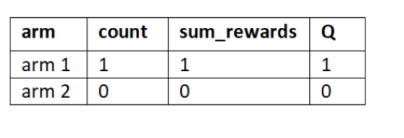
In reinforcement learning, we use Bernoulli bandits when we're making decisions that have binary outcomes (like clicking or not clicking on an ad). These distributions help us understand how likely these outcomes are and guide our choices.

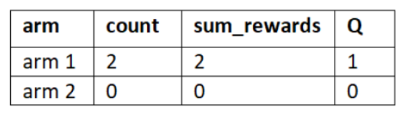
Say we have two arms—arm 1 and arm 2. Suppose with arm 1 we win the game 80% of the time and with arm 2 we win the game 20% of the time.

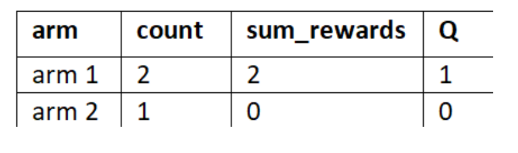
So, we can say that arm 1 is the best arm as it makes us win the game 80% of the time. Now, let's learn how to find this with the epsilon-greedy method.

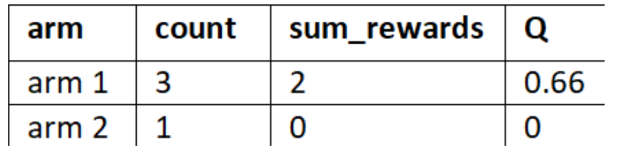
Initialize the variables to 0

R=1,

R=1

R=0

R=0



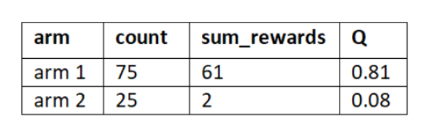
Repeat this process for several rounds (iterations) by pulling the best arm with a probability 1-epsilon

and a random arm with probability epsilon.

The best arm is the one that has the maximum average reward from the table.

Randomly pull arm 2/Pull the best arm and observe the reward.

Let the reward obtained by pulling arm 2 be 0.



**Version 2** :

**Epsilon-greedy strategy and its steps:**

\*\*1. Initialization:\*\*

- We start with two arms: arm 1 and arm 2.

- We set up three variables: count, sum\_rewards, and Q for each arm. These help us keep track of how many times we pulled an arm, the total rewards obtained from that arm, and the average reward for that arm.

\*\*2. Round 1:\*\*

- We randomly select an arm with a chance determined by epsilon (exploration rate).

- Let's say we chose arm 1 and got a reward of 1.

- We update the count, sum\_rewards, and calculate the average reward Q for arm 1.

\*\*3. Round 2:\*\*

- Now we want to make a smarter choice by exploiting the information we have. We choose the arm with the highest Q (average reward).

- As arm 1 has the highest Q from the first round, we pull arm 1 again and get a reward of 1.

- We update the count, sum\_rewards, and Q for arm 1.

\*\*4. Round 3:\*\*

- We decide to explore again, so we randomly select an arm with a probability determined by epsilon.

- Let's say we chose arm 2 and got a reward of 0.

- We update the count, sum\_rewards, and Q for arm 2.

\*\*5. Round 4:\*\*

- Back to exploiting: we choose arm 1 because it has the highest Q.

- This time we get a reward of 0.

- We update the count, sum\_rewards, and Q for arm 1.

\*\*6. Repeating Rounds:\*\*

- We continue this process for more rounds, sometimes exploring with epsilon probability and sometimes exploiting by choosing the arm with the highest Q.

- We keep updating the counts, sum\_rewards, and Q values for each arm.

\*\*7. Final Decision:\*\*

- After many rounds, we see that arm 1 has the highest Q value, which means it's the best arm to choose.

- The epsilon-greedy method helps us balance between exploring unknown arms and exploiting what we know to maximize our rewards.

\*\*8. Exploration vs. Exploitation:\*\*

- Epsilon-greedy strategy explores (chooses arms randomly) with a certain chance (epsilon) and exploits (chooses the best-known arm) the rest of the time.

- It's like trying new things sometimes (exploration) and sticking to what works best most of the time (exploitation).

Remember, epsilon-greedy helps us learn the best action over time, but it might not always distribute exploration equally among all options. It's a trade-off between trying new things and going with what seems to be the best.

**MAB : SOFTMAX EXPLORATION**

1. For example, say we have 4 arms where arm 1 is the best arm.
2. Say arm 3 is never a good arm and it always gives a reward of 0. Instead of exploring arm 3 again, we can spend more time exploring arm 2 and arm 4.
3. But the problem with the epsilon-greedy method is that we explore all the non-best arms [arm 2, arm 3, arm 4] equally.

**How to avoid this?**

To avoid this, we can give priority to arm 2 and arm 4 over arm 3 for exploring.

**Okay, but how can we give priority to the arms?**

1. We can give priority to the arms by assigning a probability to all the arms based on the average reward Q.
2. The arm that has the maximum average reward will have high probability, and all the non-best arms have a probability proportional to their average reward.

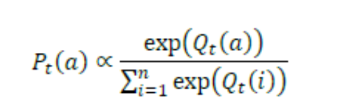
As average reward of arm 3 is 0.

So, we give more priority to arms 2 and 4 than arm 3 as the probability of arm 2 and 4 will be high compared to arm 3.

In Softmax exploration, select the arms based on a probability. The probability of each arm is directly proportional to its average reward:

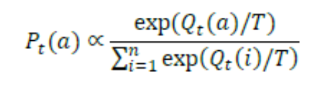


The average reward (Q value) will not sum to 1. So, we convert them into probabilities with the softmax function



In the initial rounds, correct average reward of each arm is not known, so selecting the arm based on the probability of average reward will be inaccurate in the initial rounds.

To avoid this, we introduce a new parameter called T the temperature parameter.



**Okay, how will this T help us?**

When T is high, all the arms have an equal probability of being selected and when T is low, the best arm that has the maximum average reward will have a high probability.

Set T to a high number in the initial rounds to explore all the arms equally, and after a series of rounds we reduce the value of T to select the best arm that has a high probability.

**Version 2:**

Certainly, let's break down how to address this issue and use the Softmax exploration method:

\*\*1. Giving Priority to Arms:\*\*

- To avoid exploring all non-best arms equally, we want to give more priority to better-performing arms.

- This can be achieved by assigning probabilities to each arm based on their average reward Q.

- The best arm gets the highest probability, and other arms get probabilities proportional to their rewards.

\*\*2. Softmax Exploration:\*\*

- We will select arms based on probabilities. Each arm's probability is directly tied to its average reward.

- However, in the beginning, when we don't know the accurate average rewards, this approach might be less accurate.

\*\*3. Introducing the Temperature Parameter (T):\*\*

- The temperature parameter T helps in the transition from exploration to exploitation.

- High T values make all arms equally likely to be chosen, while low T values prioritize the arm with the highest reward.

- Initially, we set T to a high value to explore arms equally and then decrease T over time to favor the best arm.

\*\*4. Initial Exploration and Later Exploitation:\*\*

- With high T, we start by exploring arms with similar probabilities to find better options.

- As we collect more data and reduce T, we shift towards exploiting the best arm with a higher probability.

\*\*5. Benefits of Softmax with T:\*\*

- Softmax exploration with T provides a balanced approach: it allows us to explore widely at first and then focus on the best option.

- It adjusts the exploration-exploitation balance dynamically as we gather more information about arm performance.

\*\*6. Remembering the Concept:\*\*

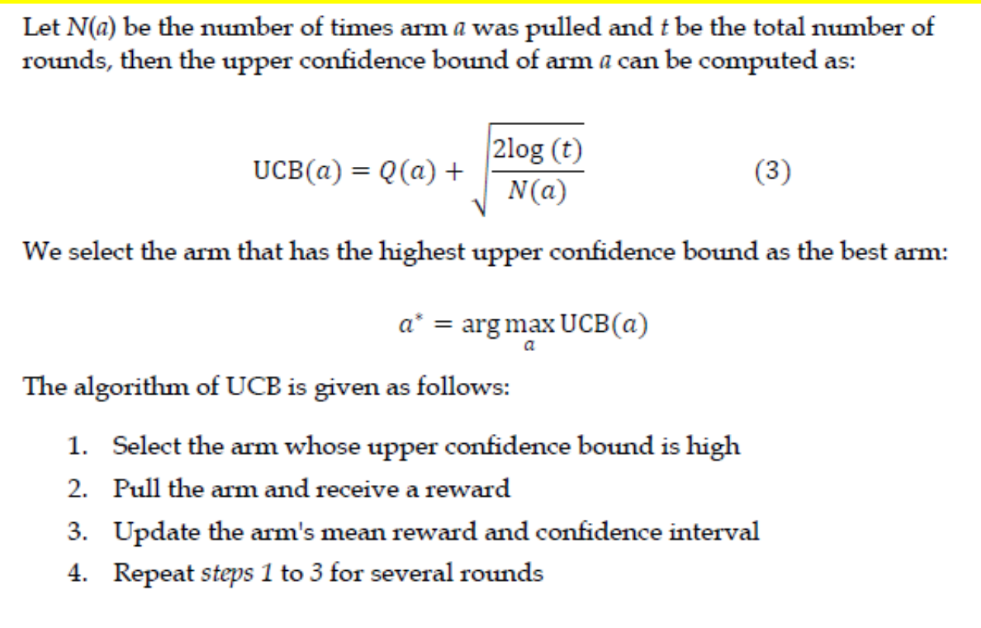
- Think of T as a dial that controls how much we prioritize exploration versus exploitation.

- High T cranks up exploration, allowing us to consider various options.

- Low T turns up exploitation, leading us to favor the known best-performing option.

So, Softmax exploration with the temperature parameter helps us manage the trade-off between exploring new options and exploiting the best-known option, adapting over time to make more informed decisions.

**UPPER CONFIDENCE BOUND EXPLORATION**



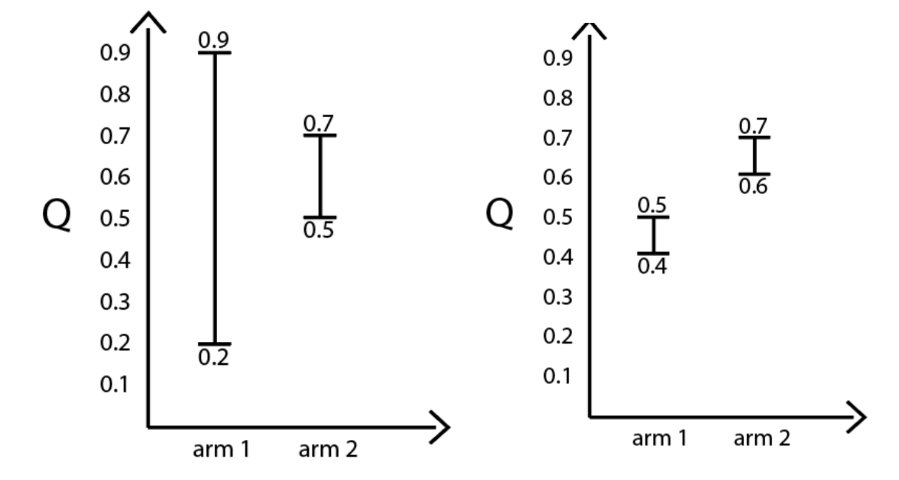
Suppose we have two arms – arm 1 and arm 2.

• Let's say we played the game for 20 rounds by pulling arm 1 and arm 2 randomly and found that the mean reward of arm 1 is 0.6 and the mean reward of arm 2 is 0.5.

• Is it accurate? Does it represents the true mean?

• We use confidence interval denotes the **interval within which the true mean value lies.**

1. Let’s say we played the game for 20 rounds by pulling arm 1 and arm 2 randomly
2. Say arm 2 has been pulled 15 times and arm 1 has been pulled only 5 times.
3. Since arm 2 has been pulled many times, the confidence interval of arm 2 is small and it denotes a certain mean reward.
4. Since arm 1 has been pulled fewer times, the confidence interval of the arm is large and it denotes an uncertain mean reward.
5. Thus, it indicates that arm 2 has been explored a lot more than arm 1.
6. When confidence interval is large, we will not be sure about the mean reward.
7. For example, select arm 1 since it has a high upper confidence bound of 0.9; however, since the confidence interval of arm 1 is large, our mean reward could be anywhere from 0.2 to 0.9, and so get a low reward.
8. Still select arm 1 as it promotes exploration.
9. When the arm is explored well, then the confidence interval gets smaller.
10. As the game is played for several rounds by selecting the arm that has a high UCB, confidence interval of both arms will get narrower and denote a more accurate mean value.



**Version 2:**

Certainly, let's break down the concept of Upper Confidence Bound (UCB) exploration:

\*\*1. Confidence Intervals Reflect Exploration:\*\*

- Confidence intervals show how confident we are in the estimated mean reward of each arm.

- Larger intervals indicate more uncertainty about the true mean reward.

- Smaller intervals indicate more confidence in the estimated mean reward.

\*\*2. Initial Confidence Intervals:\*\*

- Initially, when we have limited data from playing the game, the confidence intervals are wide.

- For example, arm 1 has been pulled only a few times, so its confidence interval is large.

- On the other hand, arm 2 has been pulled more, leading to a smaller confidence interval.

\*\*3. Role of Confidence Intervals in Selection:\*\*

- When deciding which arm to pull, we look at the upper confidence bound (UCB) of each arm.

- UCB considers both the estimated mean reward and the uncertainty (width of the confidence interval).

- A higher UCB means that arm has a better chance of having a higher true mean reward.

\*\*4. Early Exploration:\*\*

- In the beginning, selecting an arm with a high UCB might lead to low actual rewards due to the uncertainty.

- We're willing to explore arms with larger UCBs to gather more information about their rewards.

\*\*5. Reducing Uncertainty Through Exploration:\*\*

- As we play more rounds, the arm that's been pulled more times (like arm 2) will have a smaller confidence interval.

- This is because more data reduces uncertainty and gives us a clearer picture of the true mean reward.

\*\*6. Smaller Confidence Intervals with Time:\*\*

- Over time, both arms' confidence intervals become smaller and closer to the actual mean rewards.

- This means we become more certain about the rewards associated with each arm.

\*\*7. Visual Representation:\*\*

- Imagine the initial graph where arm 2's interval is smaller than arm 1's, indicating more exploration for arm 2.

- After more rounds, the intervals become smaller for both arms, showing that we've learned more about their rewards.

\*\*8. Choosing Exploitation After Exploration:\*\*

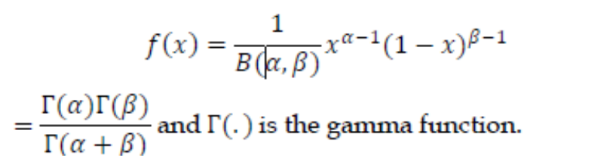
- As confidence intervals narrow, we gain confidence in the estimated rewards.

- We transition from exploration (choosing arms with high UCBs) to exploitation (choosing the arm with the highest estimated reward) as we gather more information.

So, UCB exploration takes into account the balance between exploration and exploitation. It initially explores options with high UCBs to gather data, which helps narrow down the uncertainty and allows for more accurate decisions in the long run.

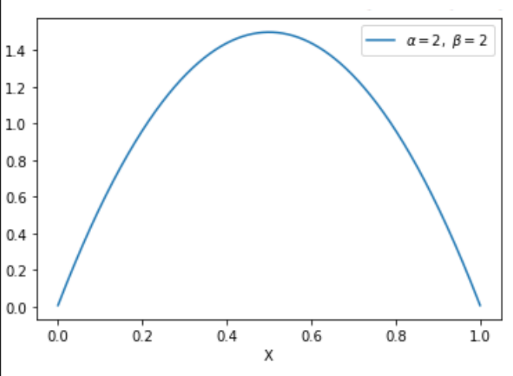
**Thompson sampling (TS)** is based on a beta distribution.

The beta distribution is a probability distribution function and it is expressed as:

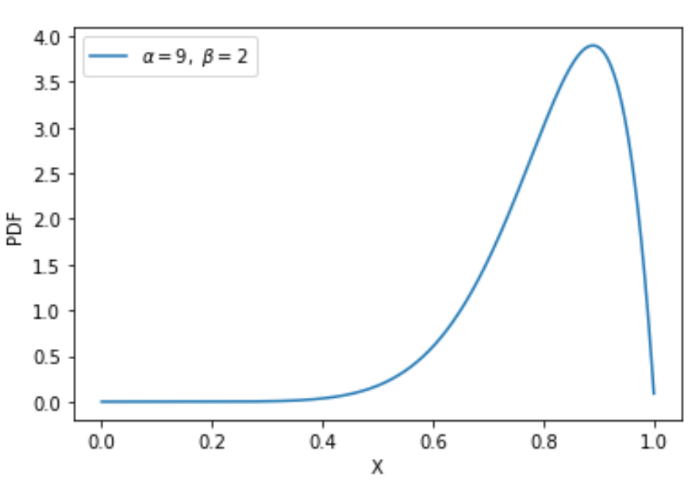


The shape of the distribution is controlled by the two parameters alpha α and beta β.

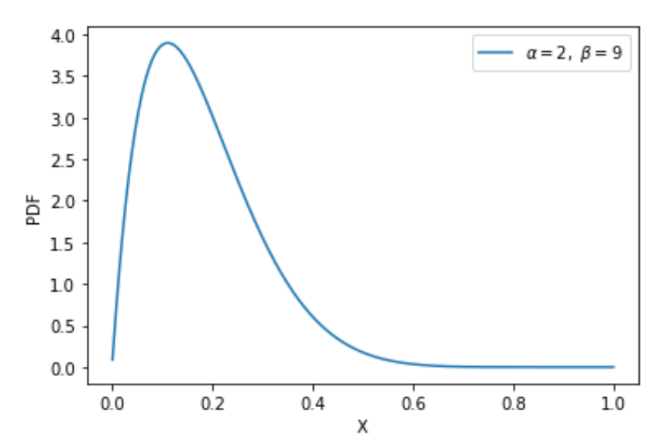
1. When the values of alpha α and beta β are the same, then we will have a symmetric distribution.

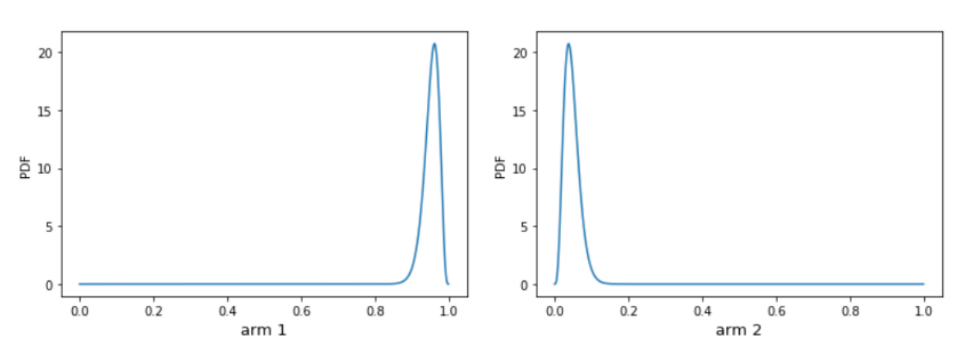


1. When the value of alpha α is higher than beta β then we will have a higher probability closer to 1 than 0.

 α > β

1. When the value of beta β is higher than alpha α then we will have a high probability closer to 0 than 1.

 β > α



Arm 1 is better than arm 2 because it has a high probability close to 1, but arm 2 has a high probability close to 0.

So, if we pull arm 1, we get a reward of 1 and win the game, but if we pull arm 2 we get a reward of 0 and lose the game.

Thus, once we know the true distribution of the arms then we can find the best arm.

First, we take n samples from arm 1 and arm 2 and compute their distribution.

In the initial iterations, the computed distributions of arm 1 and arm 2 will not be the same as the true distribution, and this is called as the prior distribution.

The steps involved in the Thomson sampling method are given here:

1. Initialize the beta distribution with alpha and beta set to **equal values for all k arms**
2. Sample a value from the beta distribution of all k arms
3. Pull the arm whose sampled value is high
4. If we win the game, then update the alpha value of the distribution to

α = α + 1

1. If we lose the game, then update the beta value of the distribution to

β = β + 1

1. Repeat steps 2 to 5 for many rounds

**Version 2 :**

**concept of Thompson Sampling using a simple explanation:**

\*\*Beta Distribution and Arm Selection:\*\*

- The beta distribution is a way to model probabilities, and it's defined by two parameters: alpha (α) and beta (β).

- The shape of the distribution is influenced by these parameters.

- If α and β are the same, it results in a symmetric distribution, as shown in the graph.

- When α is higher than β, the distribution leans towards 1, and when β is higher, it leans towards 0.

\*\*Thompson Sampling for Arm Selection:\*\*

- Thompson Sampling is a method to find the best arm to pull in a Multi-Armed Bandit problem.

- Initially, we don't know the true distribution of arms, so we start with a prior distribution (an educated guess).

- We collect samples from arms to estimate their distributions, which starts as our prior belief.

- As we gather more data, our estimates become more accurate and converge towards the true distributions.

\*\*Learning the True Distribution:\*\*

1. Start with initialized prior distributions for all arms.

2. Sample a value from each arm's distribution and choose the arm with the highest sample.

3. If you win, update the distribution's alpha (α) value by adding 1.

4. If you lose, update the distribution's beta (β) value by adding 1.

5. Repeat steps 2 to 4 for many rounds.

\*\*Evolution of Distributions:\*\*

- Initially, prior distributions are guesses, shown by wide intervals.

- As you play more rounds, distributions shift based on wins and losses, getting closer to the true distributions.

- After many rounds, the distributions become more accurate, resembling the actual win probabilities.

\*\*Thompson Sampling: Balancing Exploration and Exploitation:\*\*

- Thompson Sampling balances exploration (trying different arms) and exploitation (choosing arms with high probabilities).

- By sampling from the beta distribution, it incorporates uncertainty and encourages exploration.

- Over time, as distributions become more accurate, it favors arms with higher probabilities.

\*\*Steps in Thompson Sampling:\*\*

1. Initialize arms with prior distributions (equal α and β values).

2. Sample values from each arm's distribution and pick the arm with the highest sample.

3. Play the selected arm and receive a reward (win or lose).

4. Update the distribution's parameters based on the outcome (win: increase α, lose: increase β).

5. Repeat steps 2 to 4 for multiple rounds to refine distribution estimates.

In simple terms, Thompson Sampling uses prior beliefs to estimate arm probabilities, samples from these estimates, and adjusts beliefs based on outcomes. It cleverly balances exploring new arms with exploiting the ones that seem promising, making it a valuable strategy in the Multi-Armed Bandit problem.