

Module 4

CSE3011 Reinforcement Learning

Credit Structure : 2-2-3



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Case Study – The MAB Problem

Topics : The MAB problem, The epsilon-greedy method, Softmax exploration, The upper confidence bound algorithm, The Thompson sampling algorithm, Applications of MAB, Finding the best advertisement banner using MAB, Contextual bandits.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Introduction...

- **Game of chance** : whose outcomes are depending on a randomising device (dice, spinning tops, random number generator, etc)
- **Gambler?**
 - A person who invests in a game of chance.
 - Take high risk to invest in a game of chance with a hope of increasing the money in hand.
- **Slot machine** : is a gambling machine that creates a game of chance for its customers
- A single slot machine is called a one-armed bandit and when there are multiple slot machines it is called a MAB or k-armed bandit, where k denotes the number of slot machines



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



- A MAB is a slot machine where we pull the arm (lever) and get a payout (reward) based on some probability distribution.



Figure 6.1: 3-armed bandit slot machines



PES
UNIVERSITY
Private University Estd. in Karnataka State by Act No. 41 of 2013

40
YEARS
OF ACADEMIC
WISDOM

Introduction

- Each arm has its own probability distribution indicating the probability of winning and losing the game when we pull this arm.
- For example, let's suppose we have two arms. 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.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Introduction

- Thus, we can say that pulling arm 1 is desirable as it makes us win the game 70% of the time.
- However, this probability distribution of the arm (slot machine) will not be given to us.
- We need to find out which arm helps us to win the game most of the time and gives us a good reward.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



MAB

- The MAB problem is one of the classic problems in reinforcement learning.
- The **Multi-Armed Bandit problem (MAB)** is a toy problem that models sequential decision tasks where the learner must simultaneously exploit their knowledge and explore unknown actions to gain knowledge for the future (exploration-exploitation tradeoff).



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Toy example: playing in a casino

- Imagine you have \$100 and you are facing a row of 10 slot machines
- How will you spend the \$100 across the machines to get the most rewards?
- Each of the machines has its own probability of return, and the more you play on one machine, the better you will know about the probability of its return.
- The problem here is: with the \$100, the more you spend across the machines trying (**exploration**), the better chance you have finding the highest returning one,
- But you also have less money left to invest in that winner; or you can bet on the known better performing ones (**exploitation**), and possibly miss the opportunity of finding the highest-returning arm.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Exploration strategies

To overcome exploration-exploitation dilemma in the MAB problem, we use the following exploration strategies:

- Epsilon-greedy
- Softmax exploration
- Upper confidence bound
- Thompson sampling



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Introduction

- Along with finding the best arm, our goal should be to minimize the cost of identifying the best arm, and this is usually referred to as **regret**.
- Thus, **we need to find the best arm while minimizing regret**.
- That is, we need to find the best arm, but we don't want to end up selecting the arms that make us lose the game in most of the rounds.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Introduction

- Let's denote the arm by a and define the average reward by pulling the arm a as:

$$Q(a) = \frac{\text{Sum of rewards obtained from the arm}}{\text{Number of times the arm was pulled}}$$

Where $Q(a)$ denotes the average reward of arm a .

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

$$a^* = \arg \max_a Q(a)$$

- The arm that gives the maximum average reward is the optimal arm.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Epsilon-greedy

- With epsilon-greedy, we select the best arm with a probability $1 - \epsilon$ and we select a random arm with a probability ϵ .
- 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.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Epsilon-greedy method

First, we initialize the count (the number of times the arm is pulled), `sum_rewards` (the sum of rewards obtained from pulling the arm), and `Q` (the average reward obtained by pulling the arm), as *Table 6.1* shows:

arm	count	sum_rewards	Q
arm 1	0	0	0
arm 2	0	0	0

Table 6.1: Initialize the variables with zero



Epsilon-greedy method

Round 1:

Say, in round 1 of the game, we select a random arm with a probability epsilon, and suppose we randomly pull arm 1 and observe the reward. Let the reward obtained by pulling arm 1 be 1. So, we update our table with count of arm 1 set to 1, and sum_rewards of arm 1 set to 1, and thus the average reward Q of arm 1 after round 1 is 1 as *Table 6.2* shows:

arm	count	sum_rewards	Q
arm 1	1	1	1
arm 2	0	0	0

Table 6.2: Results after round 1



Epsilon-greedy method

Round 2:

Say, in round 2, we select the best arm with a probability $1-\epsilon$. The best arm is the one that has the maximum average reward. So, we check our table to see which arm has the maximum average reward. Since arm 1 has the maximum average reward, we pull arm 1 and observe the reward and let the reward obtained from pulling arm 1 be 1.

So, we update our table with count of arm 1 to 2 and sum_rewards of arm 1 to 2, and thus the average reward Q of arm 1 after round 2 is 1 as *Table 6.3* shows:

arm	count	sum_rewards	Q
arm 1	2	2	1
arm 2	0	0	0

Table 6.3: Results after round 2



Epsilon-greedy method

Round 3:

Say, in round 3, we select a random arm with a probability epsilon. Suppose we randomly pull arm 2 and observe the reward. Let the reward obtained by pulling arm 2 be 0. So, we update our table with count of arm 2 set to 1 and sum_rewards of arm 2 set to 0, and thus the average reward Q of arm 2 after round 3 is 0 as

Table 6.4 shows:

arm	count	sum_rewards	Q
arm 1	2	2	1
arm 2	1	0	0

Table 6.4: Results after round 3



Epsilon-greedy method

Round 4:

Say, in round 4, we select the best arm with a probability $1 - \epsilon$. So, we pull arm 1 since it has the maximum average reward. Let the reward obtained by pulling arm 1 be 0 this time. Now, we update our table with count of arm 1 to 3 and sum_rewards of arm 2 to 2, and thus the average reward Q of arm 1 after round 4 will be 0.66 as *Table 6.5* shows:

arm	count	sum_rewards	Q
arm 1	3	2	0.66
arm 2	1	0	0

Table 6.5: Results after round 4



Epsilon-greedy method

- We repeat this process for several rounds; that is, for several rounds of the game, we pull the best arm with a probability $1 - \epsilon$ and we pull a random arm with probability ϵ .

Table 6.6 shows the updated table after 100 rounds of the game:

arm	count	sum_rewards	Q
arm 1	75	61	0.81
arm 2	25	2	0.08

Table 6.6: Results after 100 rounds

From *Table 6.6*, we can conclude that arm 1 is the best arm since it has the maximum average reward.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Instructions to install gym-bandits

First, let's clone the Gym bandits repository:

```
git clone https://github.com/JKCooper2/gym-bandits
```

Next, we can install it using pip:

```
cd gym-bandits  
pip install -e .
```

After installation, we import `gym_bandits` and also the `gym` library:

```
import gym_bandits  
import gym
```

`gym_bandits` provides several versions of the bandit environment. We can examine the different bandit versions at <https://github.com/JKCooper2/gym-bandits>.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Implementation

#import necessary libraries

```
import gym
```

```
import gym_bandits
```

```
import numpy as np
```

#create the envt and print the no. of actions, prob distbn of the arms

```
env = gym.make("BanditTwoArmedHighLowFixed-v0")
```

```
print(env.action_space.n)
```

```
print(env.p_dist)
```

```
env.reset()
```

#initialize count, sum-of-rewards and average reward of all arms to zero

```
count = np.zeros(2)
```

```
sum_rewards = np.zeros(2)
```

```
Q = np.zeros(2)
```

#define a function for epsilon-greedy policy

```
def epsilon_greedy(epsilon):
```

```
    if np.random.uniform(0,1) < epsilon:
```

```
        return env.action_space.sample()
```

```
    else:
```

```
        return np.argmax(Q)
```



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



#initialize the parameters

```
num_rounds = 100
```

#generate several rounds

```
for i in range(num_rounds):
```

```
    arm = epsilon_greedy(epsilon=0.5)
```

```
    next_state, reward, done, info = env.step(arm)
```

```
    count[arm] += 1
```

```
    sum_rewards[arm] += reward
```

```
    Q[arm] = sum_rewards[arm]/count[arm]
```



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



#print the average reward of both arms

```
print(Q)
```

#print the optimal arm

```
print('The optimal arm is arm {}'.format(np.argmax(Q)+1))
```



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Softmax exploration

- Softmax exploration, also known as Boltzmann exploration, is another useful exploration strategy for finding the optimal arm.
- In the epsilon-greedy policy, all the non-best arms are explored equally.
- That is, all the non-best arms have a uniform probability of being selected.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Softmax exploration

For instance, as *Table 6.7* shows, arm 1 is the best arm as it has a high average reward Q . So, we assign a high probability to arm 1. Arms 2, 3, and 4 are the non-best arms, and we need to explore them. As we can observe, arm 3 has an average reward of 0. So, instead of selecting all the non-best arms uniformly, we give more priority to arms 2 and 4 than arm 3. So, the probability of arm 2 and 4 will be high compared to arm 3:

arm	Q
arm 1	0.84
arm 2	0.24
arm 3	0.0
arm 4	0.13

Table 6.7: Average reward for a 4-armed bandit



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Softmax exploration

Thus, in softmax exploration, we select the arms based on a probability. The probability of each arm is directly proportional to its average reward:

$$p_t(a) \propto Q_t(a)$$

But wait, the probabilities should sum to 1, right? The average reward (Q value) will not sum to 1. So, we convert them into probabilities with the softmax function, as shown here:

$$P_t(a) \propto \frac{\exp(Q_t(a))}{\sum_{i=1}^n \exp(Q_t(i))} \quad (1)$$



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Softmax exploration

- So, now the arm will be selected based on the probability. However, in the initial rounds we will not know the correct average reward of each arm, 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 . T is called the temperature parameter.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Softmax exploration

We can rewrite the preceding equation with the temperature T , as shown here:

$$P_t(a) \propto \frac{\exp(Q_t(a)/T)}{\sum_{i=1}^n \exp(Q_t(i)/T)} \quad (2)$$

- When T is high, all the arms have an equal probability of being selected and when T is low, the arm that has the maximum average reward will have a high probability of being selected.
- So, we set T to a high number in the initial rounds, and after a series of rounds we reduce the value of T . This means that in the initial round we explore all the arms equally and after a series of rounds, we select the best arm that has a high probability.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Softmax exploration example

Let's understand this with a simple example. Say we have four arms, arm 1 to arm 4. Suppose we pull arm 1 and receive a reward of 1. Then the average reward of arm 1 will be 1 and the average reward of all other arms will be 0, as *Table 6.8* shows:

arm	Q
arm 1	1
arm 2	0
arm 3	0
arm 4	0

Table 6.8: Average reward for each arm



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Softmax exploration example

Now, if we convert the average reward to probabilities using the softmax function given in equation (1), then our probabilities look like the following:

arm	Q	probability
arm 1	1	0.475
arm 2	0	0.174
arm 3	0	0.174
arm 4	0	0.174

Table 6.9: Probability of each arm

As we can observe, we have a 47% probability for arm 1 and a 17% probability for all other arms. But we cannot assign a high probability to arm 1 by just pulling arm 1 once.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Softmax exploration example

So, we set T to a high number, say $T = 30$, and calculate the probabilities based on equation (2). Now our probabilities become:

arm	Q	probability
arm 1	1	0.253
arm 2	0	0.248
arm 3	0	0.248
arm 4	0	0.248

Table 6.10: Probability of each arm with $T=30$

As we can see, now all the arms have equal probabilities of being selected. Now we explore the arms based on this probability and over a series of rounds, the T value will be reduced, and we will have a high probability to the best arm.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Softmax exploration example

Let's suppose after some 30 rounds, the average reward of all the arms is

arm	Q
arm 1	0.84
arm 2	0.24
arm 3	0.0
arm 4	0.13

Table 6.11: Average reward for each arm after 30+ rounds



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Softmax exploration example

We learned that the value of T is reduced over several rounds. Suppose the value of T is reduced and it is now 0.3 ($T=0.3$); then the probabilities will become:

arm	Q	probability
arm 1	0.84	0.775
arm 2	0.24	0.104
arm 3	0.0	0.047
arm 4	0.13	0.072

Table 6.12: Probabilities for each arm with T now set to 0.3

As we can see, arm 1 has a high probability compared to other arms. So, we select arm 1 as the best arm and explore the non-best arms – [arm 2, arm 3, arm 4] – based on their probabilities in the next rounds.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Softmax exploration example

- Thus, in the initial round, we don't know which arm is the best arm.
- So instead of assigning a high probability to the arm based on the average reward, we assign an equal probability to all the arms in the initial round with a high value of T and over a series of rounds, we reduce the value of T and assign a high probability to the arm that has a high average reward.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



```
#import necessary libraries
```

```
import gym
```

```
import gym_bandits
```

```
import numpy as np
```

```
#create the environment
```

```
env = gym.make("BanditTwoArmedHighLowFixed-v0")
```

```
#print(env.action_space.n)
```

```
print(env.p_dist)
```

```
env.reset()
```



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



#define the softmax function to choose an arm in each round

def softmax(T):

denom = sum([np.exp(i/T) for i in Q])

probs = [np.exp(i/T)/denom for i in Q]

arm = np.random.choice(env.action_space.n, p=probs)

return arm

#initialize the count, sum_rewards and Q of all the arms

count = np.zeros(2)

sum_rewards = np.zeros(2)

Q = np.zeros(2)



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



#initialize the parameters

num_rounds = 100

T = 50

#generate the rounds

for i in range(num_rounds):

 arm = softmax(T)

 next_state, reward, done, info = env.step(arm)

 count[arm] += 1

 sum_rewards[arm] += reward

 Q[arm] = sum_rewards[arm]/count[arm]

 T = T*0.99



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



#print the average reward of all the arms

```
print(Q)
```

#print the optimal arm

```
print('The optimal arm is arm {}'.format(np.argmax(Q)+1))
```



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Upper confidence bound (UCB)

- The confidence interval denotes the interval within which the true value lies.
- So, in our setting, the confidence interval denotes the interval within which the true mean reward of the arm lies.

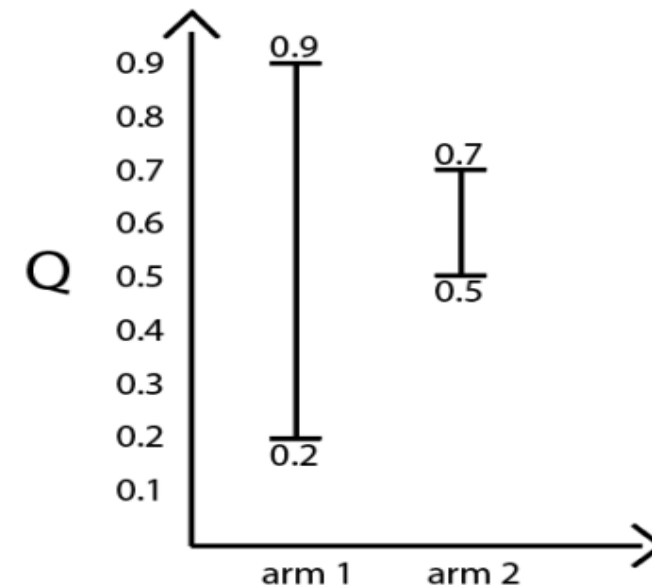


Figure 6.2: Confidence intervals for arms 1 and 2



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Upper confidence bound

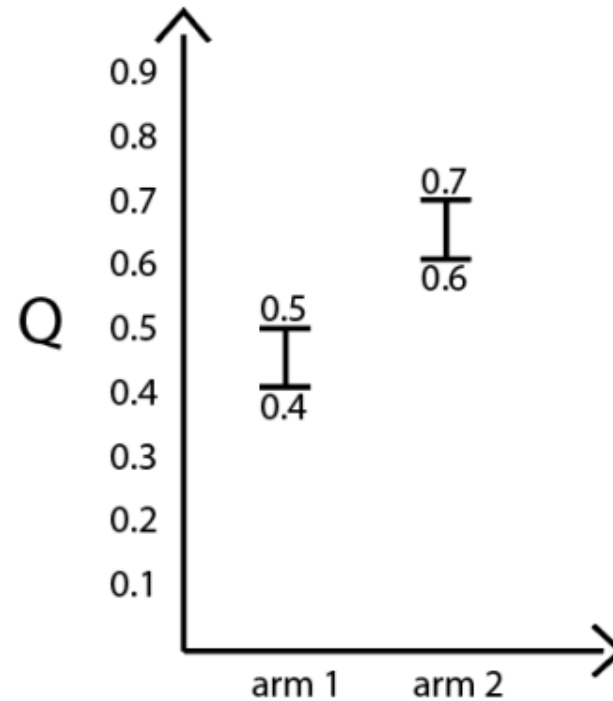


Figure 6.3: Confidence intervals for arms 1 and 2 after several rounds



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Upper confidence bound

- As we play the game for several rounds by selecting the arm that has a high UCB, our confidence interval of both arms will get narrower and denote a more accurate mean value.
- For instance, as we can see in Figure 6.3, after playing the game for several rounds, the confidence interval of both the arms becomes small and denotes a more accurate mean value.
- The confidence interval of both arms is small and we have a more accurate mean, and since in UCB we select arm that has the highest UCB, we select arm 2 as the best arm.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Upper confidence bound

Let $N(a)$ be the number of times arm a was pulled and t be the total number of rounds, then the upper confidence bound of arm a can be computed as:

$$\text{UCB}(a) = Q(a) + \sqrt{\frac{2 \log(t)}{N(a)}} \quad (3)$$

We select the arm that has the highest upper confidence bound as the best arm:

$$a^* = \arg \max_a \text{UCB}(a)$$



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Upper confidence bound

The algorithm of UCB is given as follows:

1. Select the arm whose upper confidence bound is high
2. Pull the arm and receive a reward
3. Update the arm's mean reward and confidence interval
4. Repeat steps 1 to 3 for several rounds



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Implementation

#import necessary libraries

```
import gym
```

```
import gym_bandits
```

```
import numpy as np
```

#create the envt and print the no. of actions, prob distbn of the arms

```
env = gym.make("BanditTwoArmedHighLowFixed-v0")
```

```
print(env.action_space.n)
```

```
print(env.p_dist)
```

```
env.reset()
```


#initialize count, sum-of-rewards and average reward of all arms to zero

```
count = np.zeros(2)
```

```
sum_rewards = np.zeros(2)
```

```
Q = np.zeros(2)
```

#define a function for UCB

```
def UCB(i):
```

```
    ucb = np.zeros(2)
```

```
    if i < 2:
```

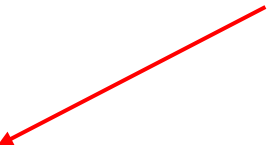
```
        return i
```

```
    else:
```

```
        for arm in range(2):
```

```
            ucb[arm] = Q[arm] + np.sqrt((2*np.log(sum(count))) / count[arm])
```

```
    return (np.argmax(ucb))
```

$$UCB(a) = Q(a) + \sqrt{\frac{2\log(t)}{N(a)}}$$




**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



#initialize the parameters

```
num_rounds = 100
```

#generate several rounds

```
for i in range(num_rounds):
```

```
    arm = ucb(i)
```

```
    next_state, reward, done, info = env.step(arm)
```

```
    count[arm] += 1
```

```
    sum_rewards[arm] += reward
```

```
    Q[arm] = sum_rewards[arm]/count[arm]
```



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



#print the average reward of both arms

```
print(Q)
```

#print the optimal arm

```
print('The optimal arm is arm {}'.format(np.argmax(Q)+1))
```



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Thompson sampling

- Thompson sampling (TS) is another interesting exploration strategy to overcome the exploration-exploitation dilemma and it is based on a beta distribution.
- So, before diving into Thompson sampling, let's first understand the beta distribution. The beta distribution is a probability distribution function and it is expressed as:

$$f(x) = \frac{1}{B(\alpha, \beta)} x^{\alpha-1} (1-x)^{\beta-1}$$

Where $B(\alpha, \beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha + \beta)}$ and $\Gamma(.)$ is the gamma function.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Beta distribution

- The shape of the distribution is controlled by the two parameters α and β . When the values of α and β are the same, then we will have a symmetric distribution.

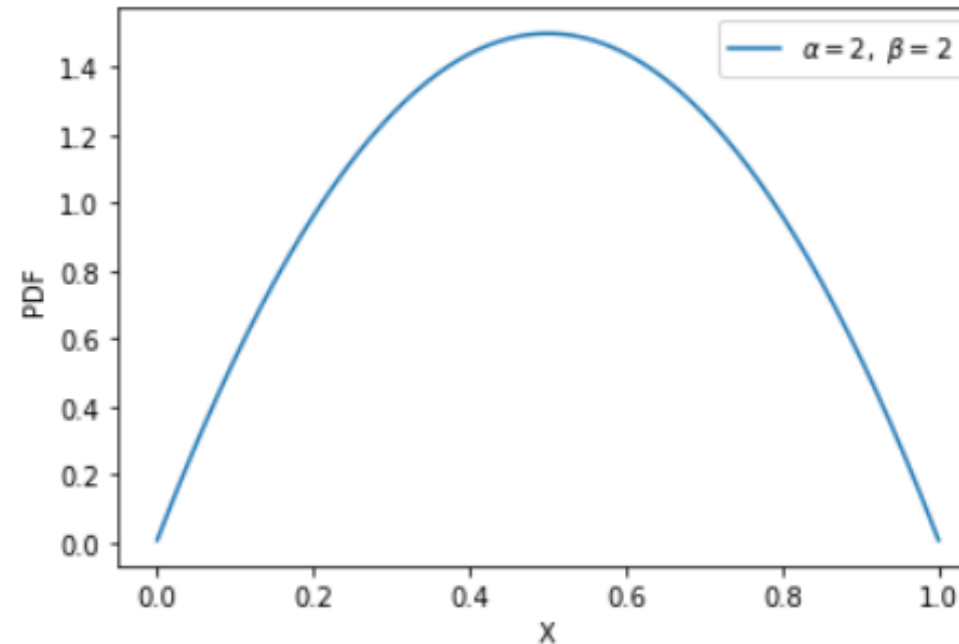


Figure 6.4: Symmetric beta distribution



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Beta distribution

- When the value of α is higher than β then we will have a probability closer to 1 than 0. For instance, as Figure 6.5 shows, since the value of $\alpha=9$ and $\beta=2$, we have a high probability closer to 1 than 0:

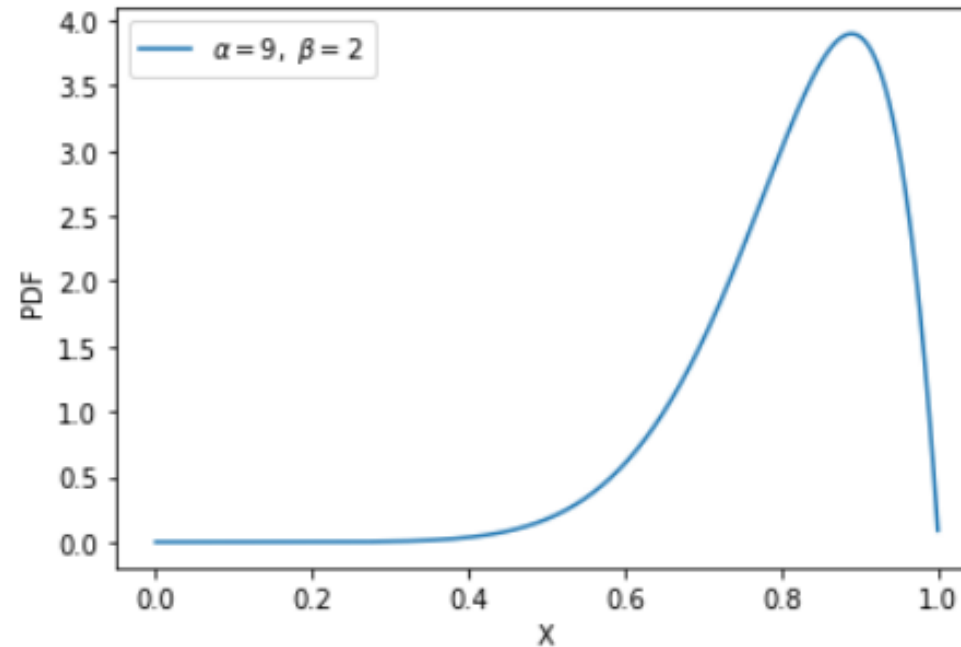


Figure 6.5: Beta distribution where $\alpha > \beta$



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Beta distribution

- When the value of β is higher than α then we will have a high probability closer to 0 than 1. For instance, as shown in the following plot, since the value of $\alpha=2$ and $\beta=9$, we have a high probability closer to 0 than 1:

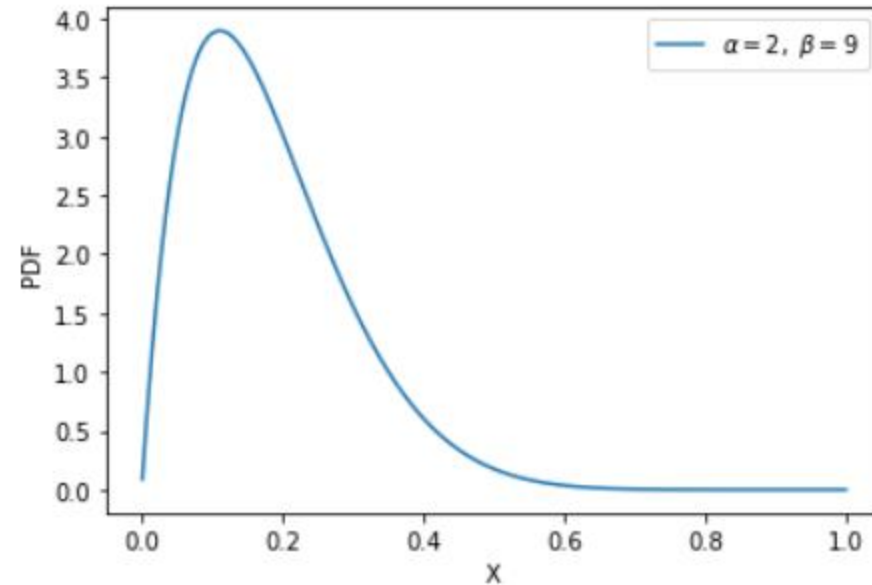


Figure 6.6: Gamma distribution where $\alpha < \beta$



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Beta distribution

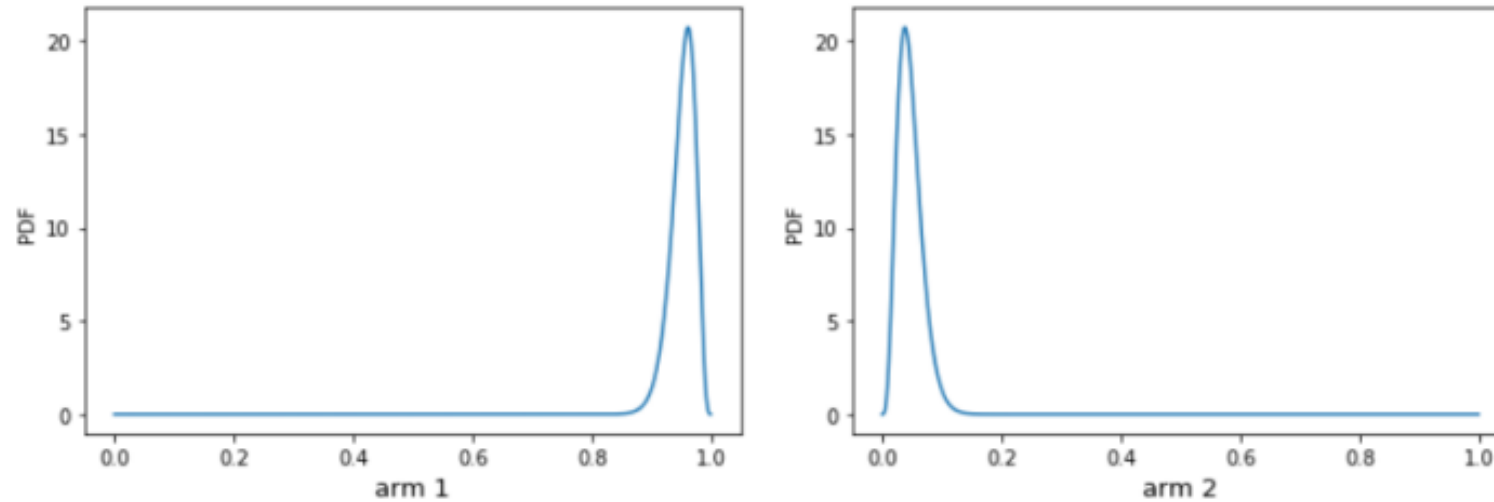


Figure 6.7: True distributions for arms 1 and 2



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Beta distribution

- From Figure 6.7, we can see that it is better to pull arm 1 than arm 2 because arm 1 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 understand which arm is the best arm.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Thompson sampling

- Thompson sampling is a probabilistic method and it is based on a prior distribution.
- Here, we use the beta distribution as a prior distribution. Say we have two arms, so we will have two beta distributions (prior distributions), and we initialize both α and β to the same value, say 3, as Figure 6.10 shows:



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Thompson sampling

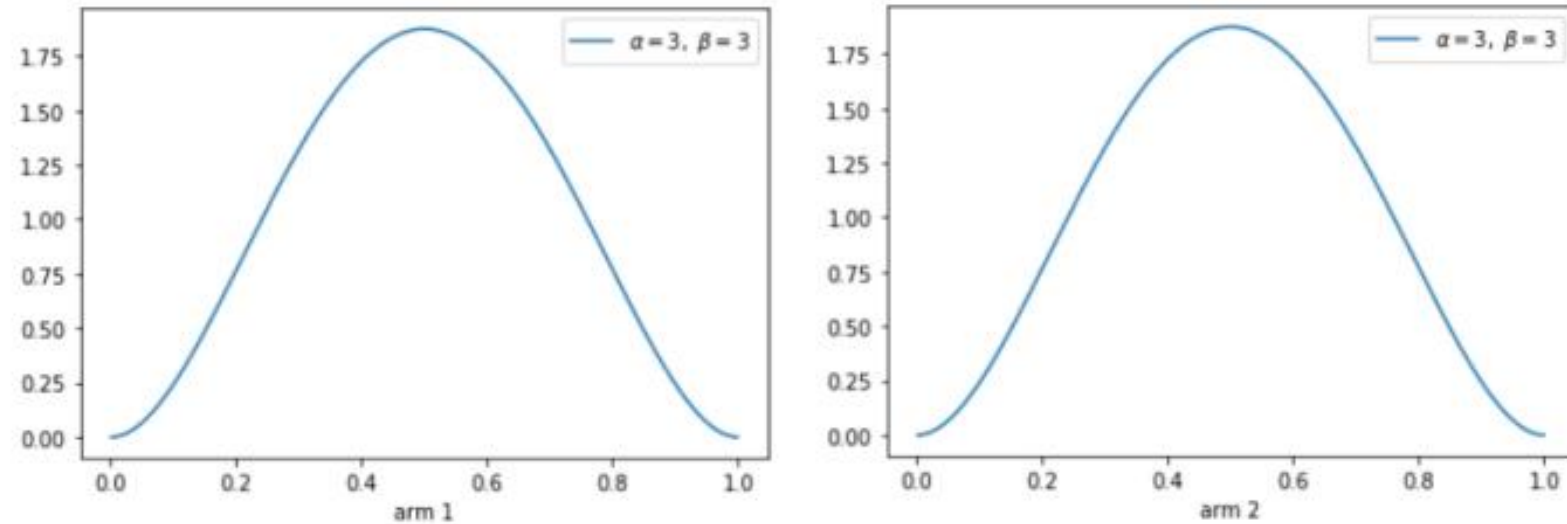


Figure 6.10: Initialized prior distributions for arms 1 and 2 look the same



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Thompson sampling

- In the first round, we just randomly sample a value from these two distributions and select the arm that has the maximum sampled value. Let's say the sampled value of arm 1 is high, so in this case, we pull arm 1.
- Say we win the game by pulling arm 1, then we update the distribution of arm 1 by incrementing the alpha value of the distribution by 1; that is, we update the alpha value as $\alpha = \alpha + 1$.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Thompson sampling

As Figure 6.11 shows, the alpha value of the distribution of arm 1 is incremented, and as we can see, arm 1's beta distribution has slightly high probability closer to 1 compared to arm 2:

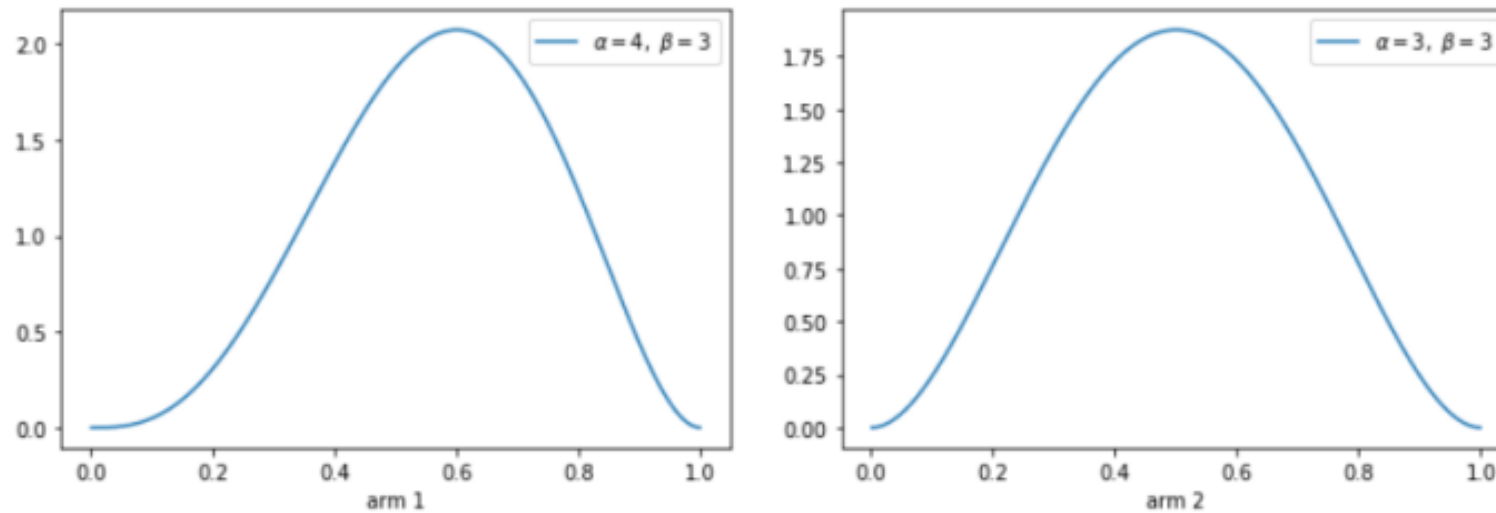


Figure 6.11: Prior distributions for arms 1 and 2 after round 1



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Thompson sampling

- As Figure 6.12 shows, the alpha value of arm 1's distribution is incremented, and arm 1's beta distribution has a slightly high probability close to 1

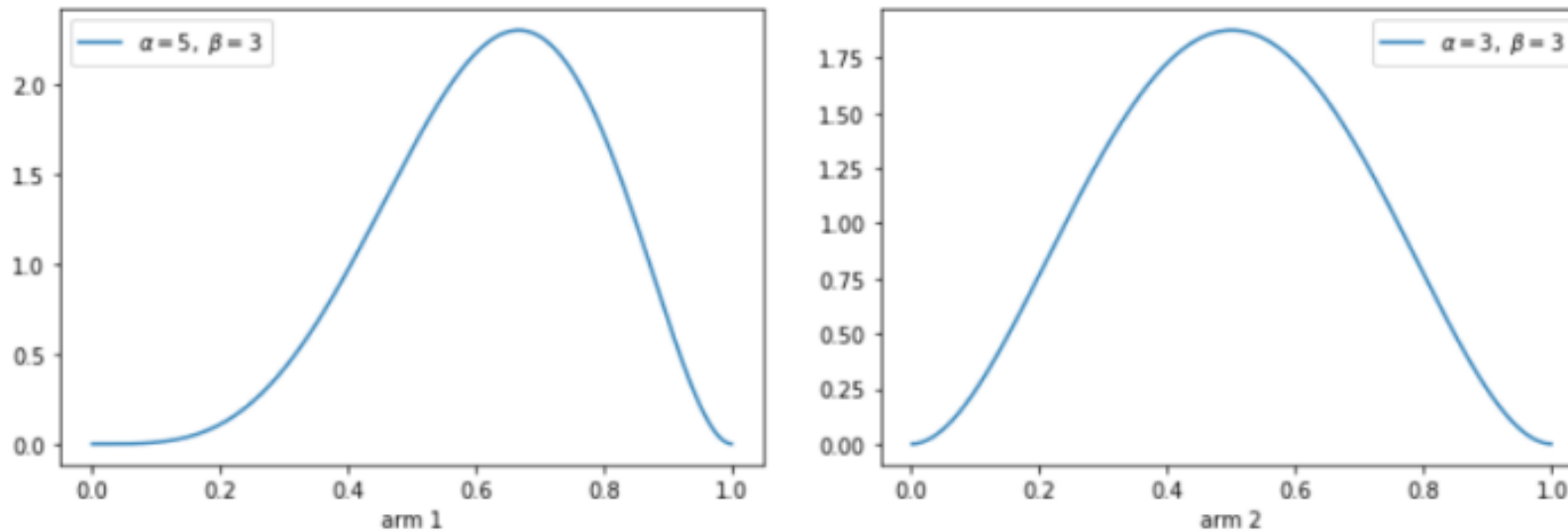


Figure 6.12: Prior distributions for arms 1 and 2 after round 2



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Thompson sampling

- As Figure 6.13 shows, the beta value of arm 2's distribution is incremented and the beta distribution of arm 2 has a slightly high probability close to 0

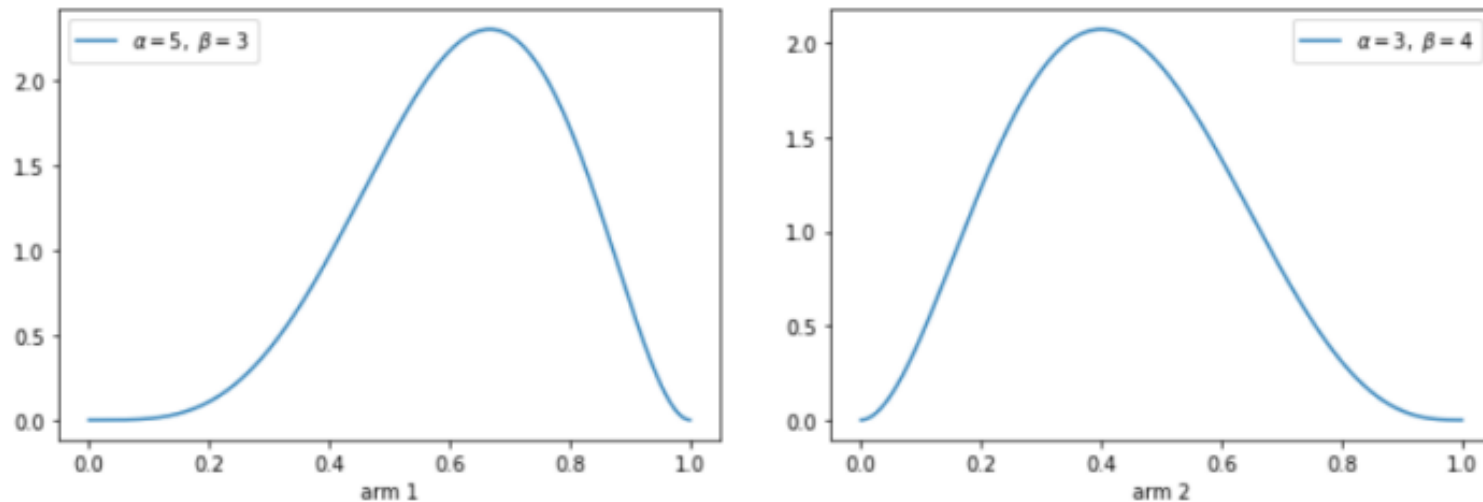


Figure 6.13: Prior distributions for arms 1 and 2 after round 3



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Thompson sampling

- As Figure 6.14 shows, the beta value of arm 2's distribution is incremented by 1 and also arm 2's beta distribution has a slightly high probability close to 0:

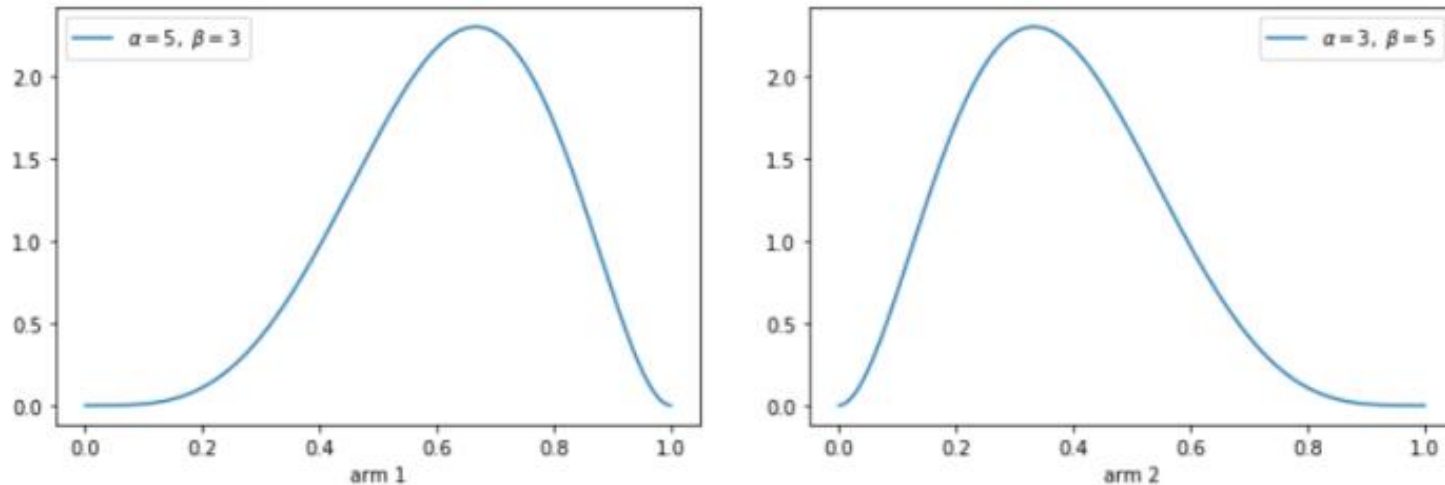


Figure 6.14: Prior distributions for arms 1 and 2 after round 4



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Thompson sampling

If we do this repeatedly for several rounds, then we can learn the true distribution of the arm. Say after several rounds, our distribution will look like Figure 6.15. As we can see, the distributions of both arms resemble the true distributions:

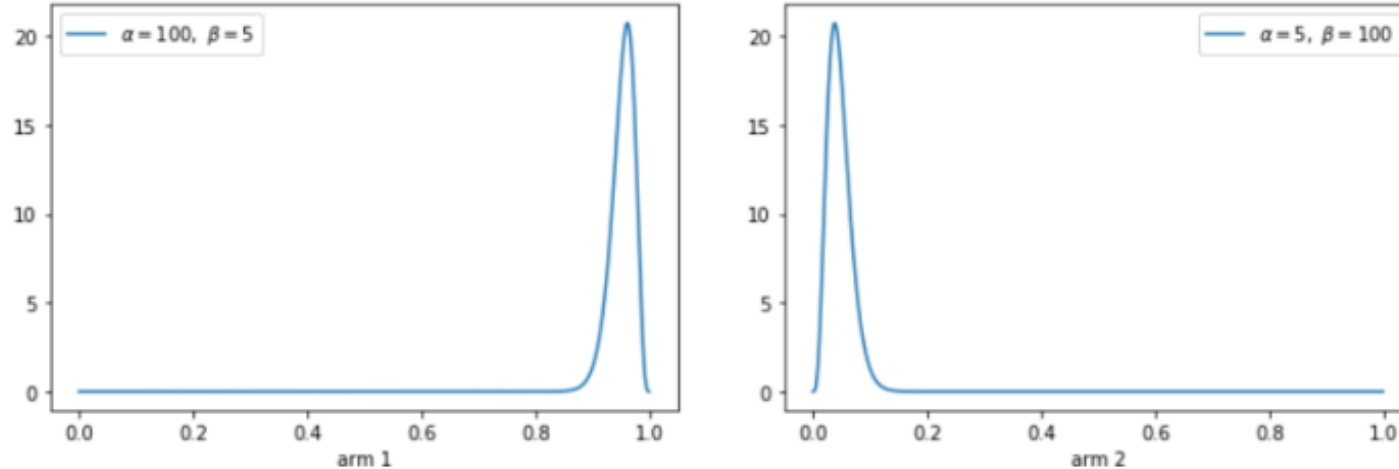


Figure 6.15: Prior distributions for arms 1 and 2 after several rounds



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Thompson sampling

Now if we sample a value from each of these distributions, then the sampled value will always be high from arm 1 and we always pull arm 1 and win the game.

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 $\alpha = \alpha + 1$
5. If we lose the game, then update the beta value of the distribution to $\beta = \beta + 1$
6. Repeat steps 2 to 5 for many rounds



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Applications of MAB

Bandits are widely used for:

- Website optimization
- Maximizing conversion rates
- Online advertisements
- Campaigning



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Finding the best advt banner for a web site

- To find the advertisement banner that is most liked by the users for a product on a web site during beta testing.
- Traditionally done using AB testing.
 - Drawbacks – exploration – exploitation are done at 2 different phases.
 - Disadv – cost is more
- Alternative – frame this problem as MAB.
 - One arm for each of the banner.
 - Adv – can find the best arm/banner with minimum regret



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



#import necessary libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
plt.style.use('ggplot')
```

#create the data set

```
df = pd.DataFrame()
for i in range(5):
    df['Banner_type_'+str(i)] = np.random.randint(0,2,100000)
df.head()
```



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



..... \ / \ /

	Banner_type_0	Banner_type_1	Banner_type_2	Banner_type_3	Banner_type_4
0	0	1	1	0	1
1	1	0	1	0	1
2	0	1	1	0	1
3	1	1	1	0	1
4	0	0	1	1	0

Figure 6.14: Clicks per banner



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



initialize the variables

num_iterations = 100000

num_banner = 5

count = np.zeros(num_banner)

sum_rewards = np.zeros(num_banner)

Q = np.zeros(num_banner)

banner_selected = []



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



define epsilon-greedy exploration strategy

```
def epsilon_greedy_policy(epsilon):
```

```
    if np.random.uniform(0,1) < epsilon:
```

```
        return np.random.choice(num_banner)
```

```
    else:
```

```
        return np.argmax(Q)
```



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



#generate the iterations same as no. of users

for i in range(num_iterations):

 banner = epsilon_greedy_policy(0.5)

 reward = df.values[i, banner]

 count[banner] += 1

 sum_rewards[banner] += reward

 Q[banner] = sum_rewards[banner]/count[banner]

 banner_selected.append(banner)



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



#identify the best banner

```
print( 'The best banner is banner {}'.format(np.argmax(Q)+1))
```

#visualize the count/no. of clicks of each banner

```
ax = sns.countplot(banner_selected)  
ax.set(xlabel='Banner', ylabel='Count')  
plt.show()
```



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



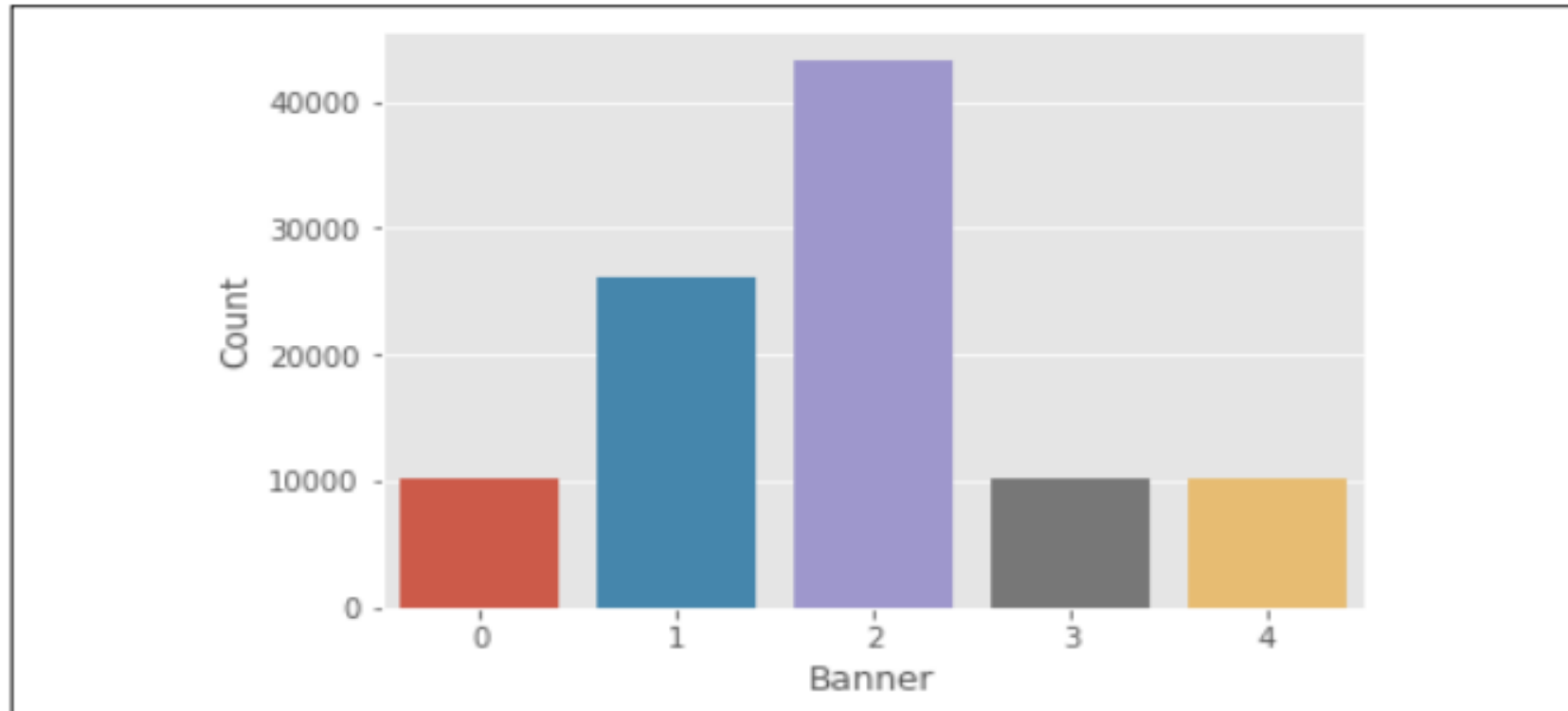


Figure 6.15: Banner 2 is the best advertisement banner



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Contextual bandits

- In the MAB problem, we just perform the action and receive a reward. But with contextual bandits, we take actions based on the state of the environment and the state holds the context.
- For instance, in the advertisement banner example, the state specifies the user behavior and we will take action (show the banner) according to the state (user behavior) that will result in the maximum reward (ad clicks).
- Contextual bandits are widely used for personalizing content according to the user's behavior. They are also used to solve the cold-start problems faced by recommendation systems. Netflix uses contextual bandits for personalizing the artwork for TV shows according to user behavior.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



- The multi-armed bandit algorithm outputs an action but doesn't use any information about the state of the environment (context).
- For example, if you use a multi-armed bandit to choose whether to display cat images or dog images to the user of your website, you'll make the same random decision even if you know something about preferences of the user.
- The contextual bandit extends the model by making the decision conditional on the state of the environment.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013





Multi-armed Bandit



Contextual Bandit



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



- Using contextual bandits, you can :
 - Optimise the decisions based on previous observations
 - Also personalize the decisions
 - Show the image of a cat to a cat-lover, dog image to a dog-lover
 - Also show different images on different times of a day
- The algorithm observes a context, makes a decision, choosing one action from a number of alternative actions, and observes an outcome of that decision. An outcome defines a reward. The goal is to maximize average reward.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



- For example, you can use a contextual bandit to select which news article to show first on the main page of your website to optimize click through rate.
- The context is information about the user:
 - where they come from, previously visited pages of the site, device information, geolocation, etc.
 - An action is a choice of what news article to display.
 - An outcome is whether the user clicked on a link or not.
 - A reward is binary: 0 if there is no click, 1 if there is a click.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Algorithm 1 Contextual Bandit

Require: Number of actions A

Require: Context feature dimensions d

Require: Context generator

Require: Reward function $\rho(a_t, x_t)$

Require: Learning rate α

Require: Bandit model (e.g., LinUCB, Neural Bandit, Decision Tree Bandit)

Initialize the bandit model

for $t = 1$ to T **do**

 Receive the context x_t from the context generator

 Predict the expected reward for each action $\hat{r}(a_t|x_t)$ using the bandit model

 Select the action $a_t = \arg \max_a \hat{r}(a_t|x_t)$

 Receive the reward $r_t = \rho(a_t, x_t)$

 Update the bandit model using the pair (a_t, r_t)

end for

Ensure: The trained bandit model, The cumulative reward



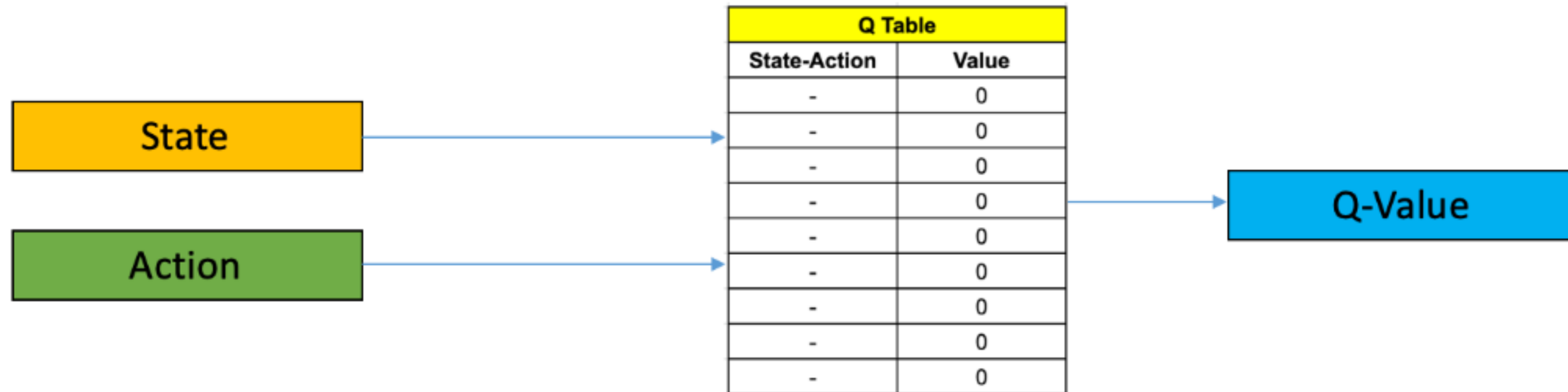
**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013

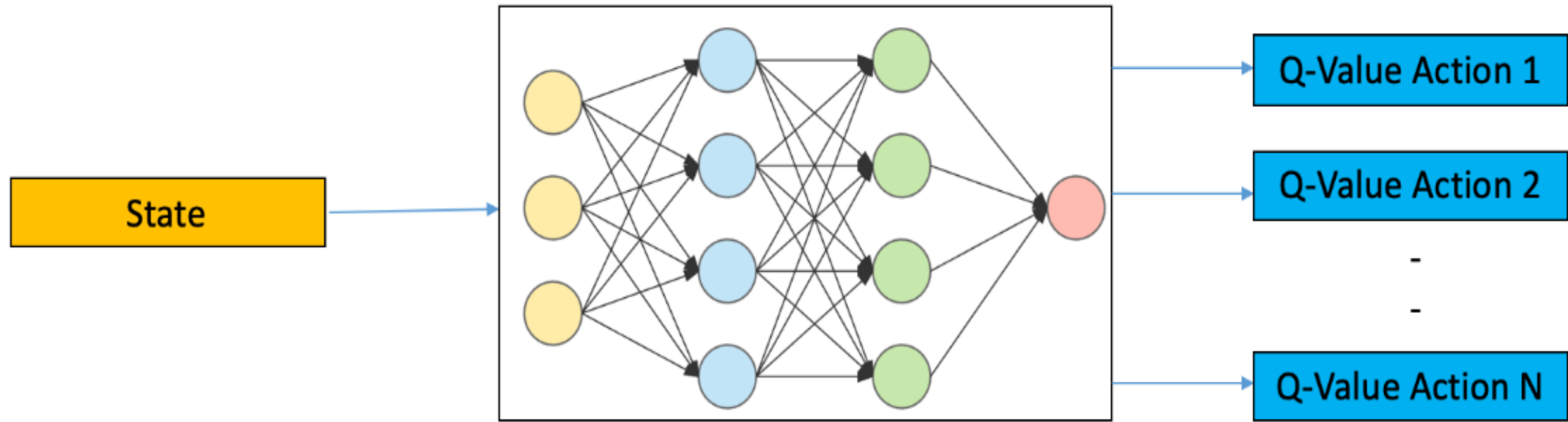


Deep Q learning

- Traditional Q-learning works well with less (s,a) pairs.
- Not suitable for an environment with state space of 1000 states and action space of 10 actions.
- No. of (s,a) pairs = $1000 \times 10 = 10,000$.
- Disadvantage in storing and updating such a large Q-table.
- Hence use function approximators like neural network to approximate the Q-values of a state.



Q Learning



Deep Q Learning



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



- For the grid world environment, if the current state $s = D$ is input to DQN, it gives the 4 Qvalues of this state as in

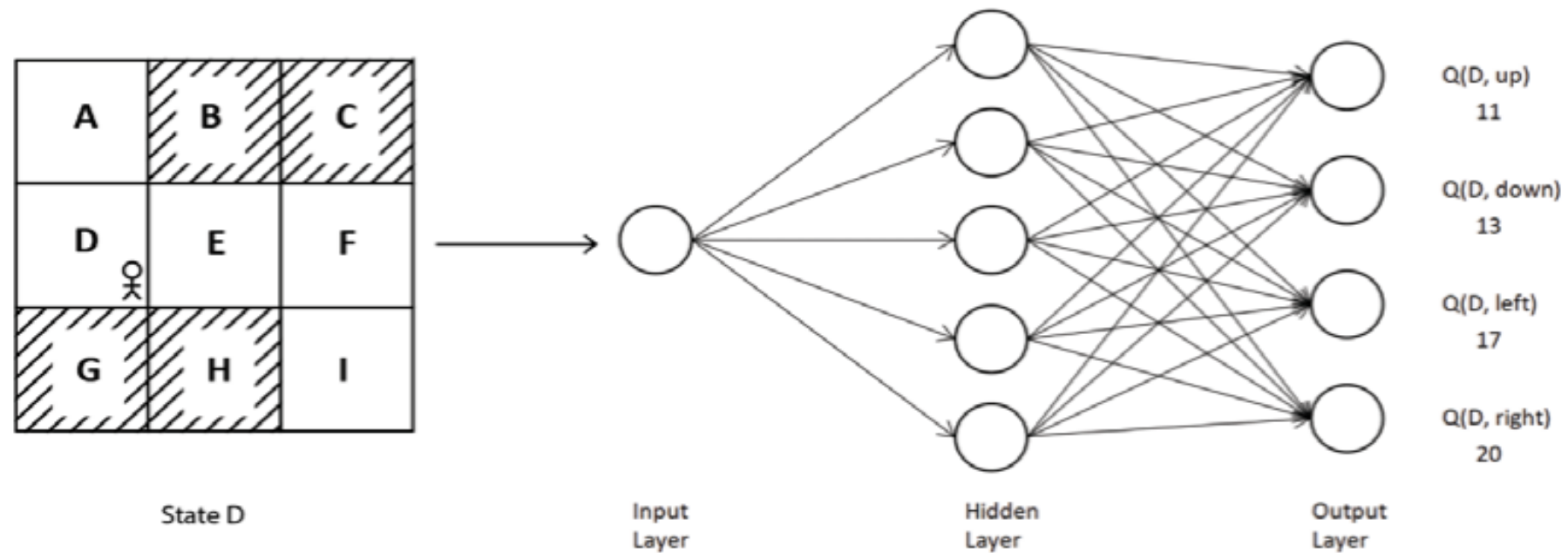


Figure 9.2: Deep Q network



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



- Compute the Q values by approximating them using any function approximator, such as a neural network.
- If we use deep neural network to approximate the Q-table, it is called Deep Q network (DQN).
- If θ is the parameter of the neural network, the Q table got from this network is said to be parameterized by this θ .
- We can denote our Q function by $Q_{\theta}(s, a)$.
- Initialize the network parameter θ to some random values.
- Train the network iteratively, to learn optimal values of θ
- The Q table got from this optimal θ , will be optimal to find the best policy for a particular state.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



- How to train the network ? Training data?
- What about the loss function?
- Is it a classification/regression task?



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Training data?

- The transition made by an agent (s, a, r, s') in each time step of an episode is saved in the replay buffer/experience replay.
- Save this data from several episodes in the buffer.
- Train the DQN with the experience/transitions sampled from the buffer.

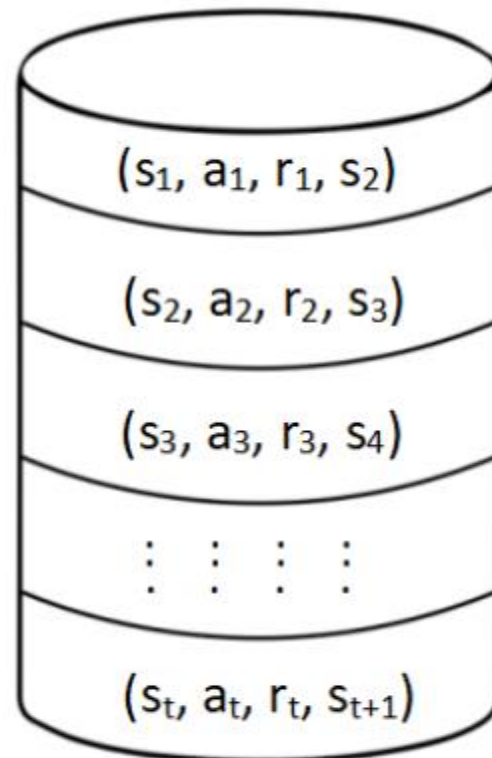


Figure 9.3: Replay buffer



**PRESIDENT
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 4

- To create the replay buffer :
 1. Initialize the replay buffer \mathcal{D} .
 2. For each episode perform *step* 3.
 3. For each step in the episode:
 1. Make a transition, that is, perform an action a in the state s , move to the next state s' , and receive the reward r .
 2. Store the transition information (s, a, r, s') in the replay buffer \mathcal{D} .



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Ex of a replay buffer

Episode 1:

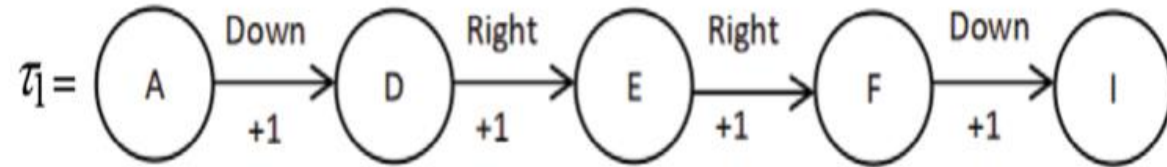


Figure 9.4: Trajectory 1

Episode 2:

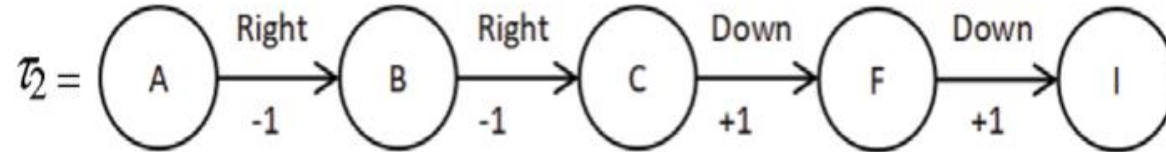


Figure 9.5: Trajectory 2

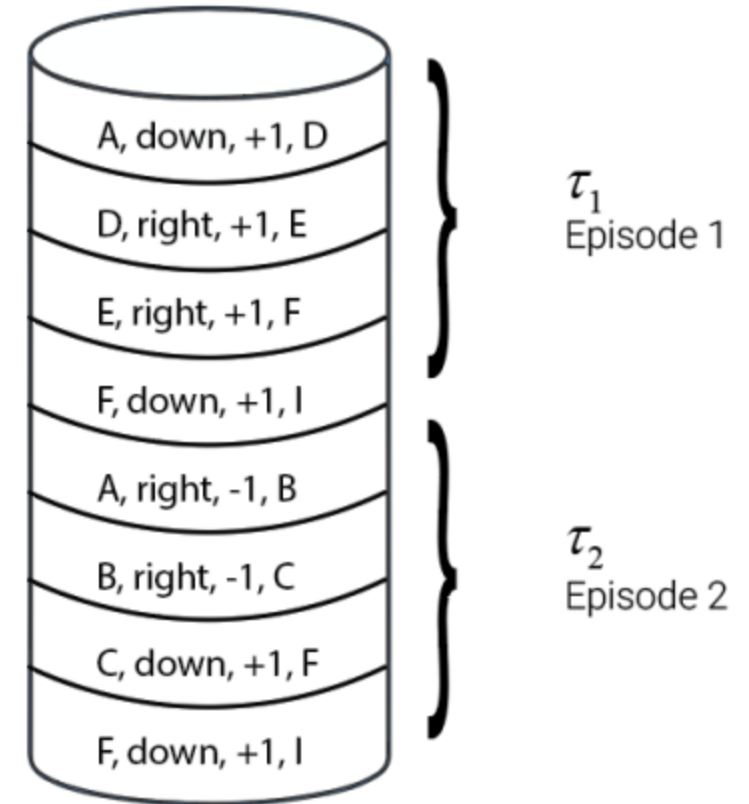


Figure 9.6: Replay buffer



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



- Disadvantage 1:
 - The transitions are stacked sequentially.
 - They are highly correlated.
 - A network trained with this correlated data, easily overfits.
 - Solution : we sample a random mini batch of transitions from the replay buffer and train the network.
- Disadvantage 2:
 - The replay buffer is of limited size. It can store only a fixed amount of the agent's experience.
 - Solution : when the buffer is full we replace the old experience with new experience. A replay buffer is usually implemented as a queue structure (first in first out) rather than a list. So, if the buffer is full when new experience comes in, we remove the old experience and add the new experience into the buffer



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



- The DQN has to approximate the optimal Q values from the mini batch of experience from the replay buffer.
- It's a regression task.
- MSE is the loss function for a regression task.

$$\text{MSE} = \frac{1}{K} \sum_{i=1}^K (y_i - \hat{y}_i)^2$$

Where y is the target value, \hat{y} is the predicted value, and K is the number of training samples.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



- We can train our network by minimizing the MSE between the target Q value and predicted Q value.
- First, how can we obtain the target Q value?
- Our target Q value should be the optimal Q value so that we can train our network by minimizing the error between the optimal Q value and predicted Q value.
- But how can we compute the optimal Q value?
 - Using Bellman equation
$$Q^*(s, a) = E_{s' \sim P} \left[r + \gamma \max_{a'} Q^*(s', a') \right]$$
 - Remove the expectation, by sampling K no. of transitions from the replay buffer and take their average



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Thus, according to the Bellman optimality equation, the optimal Q value is just the sum of the reward and the discounted maximum Q value of the next state-action pair, that is:

$$Q^*(s, a) = r + \gamma \max_{a'} Q^*(s', a') \quad (1)$$

So, we can define our loss as the difference between the target value (the optimal Q value) and the predicted value (the Q value predicted by the DQN) and express the loss function L as:

$$L(\theta) = Q^*(s, a) - Q_\theta(s, a)$$

Substituting equation (1) in the preceding equation, we can write:

$$L(\theta) = r + \gamma \max_{a'} Q(s', a') - Q_\theta(s, a)$$

$$L(\theta) = r + \gamma \max_{a'} \boxed{Q(s', a')} - Q_\theta(s, a)$$



How do we compute this?

using the same DQN parameterized by θ .

$$L(\theta) = r + \gamma \max_{a'} Q_\theta(s', a') - Q_\theta(s, a)$$

both the target value and the predicted Q value are parameterized by θ .



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



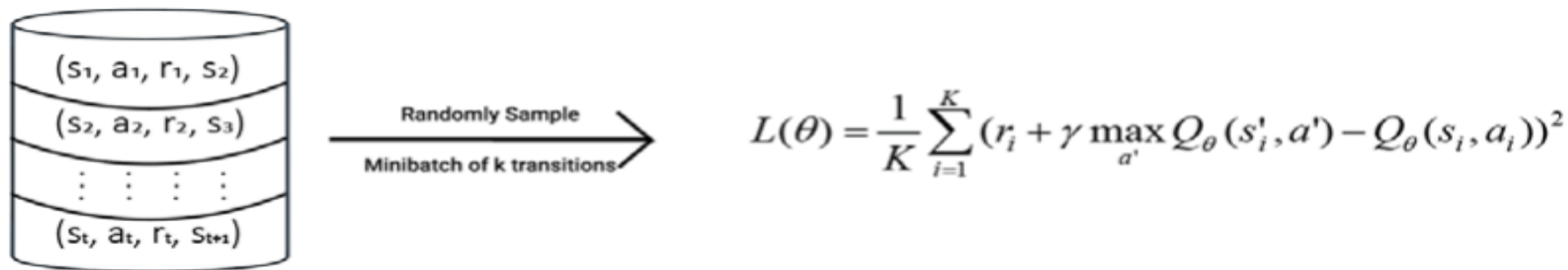


Figure 9.7: Loss function of DQN

Thus, our loss function can be expressed as:

$$L(\theta) = \frac{1}{K} \sum_{i=1}^K (r_i + \gamma \max_{a'} Q_{\theta}(s'_i, a') - Q_{\theta}(s_i, a_i))^2$$

For simplicity of notation, we can denote the target value by y and rewrite the preceding equation as:

$$L(\theta) = \frac{1}{K} \sum_{i=1}^K (y_i - Q_{\theta}(s_i, a_i))^2$$



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



- If next state is a terminal state then target is just the reward.

$$y_i = \begin{cases} r_i & \text{if } s' \text{ is terminal} \\ r_i + \gamma \max_{a'} Q_{\theta}(s'_i, a') & \text{if } s' \text{ is not terminal} \end{cases}$$

Thus, our loss function is given as:

$$L(\theta) = \frac{1}{K} \sum_{i=1}^K (y_i - Q_{\theta}(s_i, a_i))^2$$

- Train the network by minimising this loss function.
- use gradient descent to find the optimal parameter θ .
- compute the gradient of our loss function $\nabla_{\theta} L(\theta)$ and update our network parameter θ as:

$$\theta = \theta - \alpha \nabla_{\theta} L(\theta)$$



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



• Another issue?

$$L(\theta) = \frac{1}{K} \sum_{i=1}^K (r_i + \gamma \underbrace{\max_{a'} Q_{\theta}(s'_i, a')}_{\text{Compute using } \theta} - \underbrace{Q_{\theta}(s_i, a_i)}_{\text{Compute using } \theta})^2$$

- Both target and predicted are found from the same network which is parameterised using θ .
- this will cause instability in the MSE and the network learns poorly.
- It also causes a lot of divergence during training
- This is because the predicted and target values both depend on the same parameter θ .
- If we update θ , then both the target and predicted values change.
- Thus, the predicted value keeps on trying to be the same as the target value, but the target value keeps on changing due to the update on the network parameter θ .



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



• Solution : use two networks

- **The main network** with parameter θ , to predict the optimal qvalues $Q_{\theta}(s, a)$, by learning the optimal values of θ using gradient descent.
- A **target network** parameterised by θ' called $Q_{\theta'}(s', a')$, to learn the Q-value of the next state-action pair in the target.
- The target network is frozen for a while and then the target network parameter θ' is updated by just copying the main deep Q network parameter θ .
- Freezing the target network for a while and then updating its parameter θ' with the main network, stabilizes the training. So, now our loss function can be rewritten as:

$$L(\theta) = \frac{1}{K} \sum_{i=1}^K (r_i + \gamma \max_{a'} Q_{\theta'}(s'_i, a') - Q_{\theta}(s_i, a_i))^2$$

Thus, the Q value of the next state-action pair in the target is computed by the target network parameterized by θ' , and the predicted Q value is computed by our main network parameterized by θ :

$$L(\theta) = \frac{1}{K} \sum_{i=1}^K (r_i + \gamma \underbrace{\max_{a'} Q_{\theta'}(s'_i, a')}_{\substack{\downarrow \\ \text{Compute} \\ \text{using } \theta'}} - \underbrace{Q_{\theta}(s_i, a_i)}_{\substack{\downarrow \\ \text{Compute} \\ \text{using } \theta}})^2$$



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



1. Initialize the main network parameter θ with random values
2. Initialize the target network parameter θ' by copying the main network parameter θ
3. Initialize the replay buffer \mathcal{D}
4. For N number of episodes, perform *step 5*
5. For each step in the episode, that is, for $t = 0, \dots, T-1$:
 1. Observe the state s and select an action using the epsilon-greedy policy, that is, with probability epsilon, select random action a and with probability $1-\text{epsilon}$, select the action $a = \arg \max_a Q_\theta(s, a)$
 2. Perform the selected action and move to the next state s' and obtain the reward r



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



3. Store the transition information in the replay buffer \mathcal{D}
4. Randomly sample a minibatch of K transitions from the replay buffer \mathcal{D}
5. Compute the target value, that is, $y_i = r_i + \gamma \max_{a'} Q_{\theta'}(s'_i, a')$
6. Compute the loss, $L(\theta) = \frac{1}{K} \sum_{i=1}^K (y_i - Q_{\theta}(s_i, a_i))^2$
7. Compute the gradients of the loss and update the main network parameter θ using gradient descent: $\theta = \theta - \alpha \nabla_{\theta} L(\theta)$
8. Freeze the target network parameter θ' for several time steps and then update it by just copying the main network parameter θ



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013

