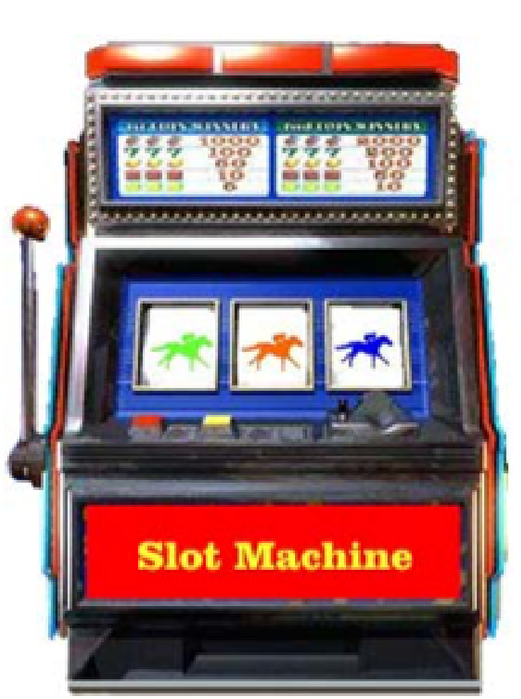
MAB Problem

MULTI-ARMED BANDIT (MAB)

Contents

01. The MAB problem
02. The epsilon-greedy method
03. The upper confidence bound algorithm
04. The Thompson sampling algorithm
05. Applications of MAB
06. Finding the best advertisement banner using MAB
07. Contextual bandits

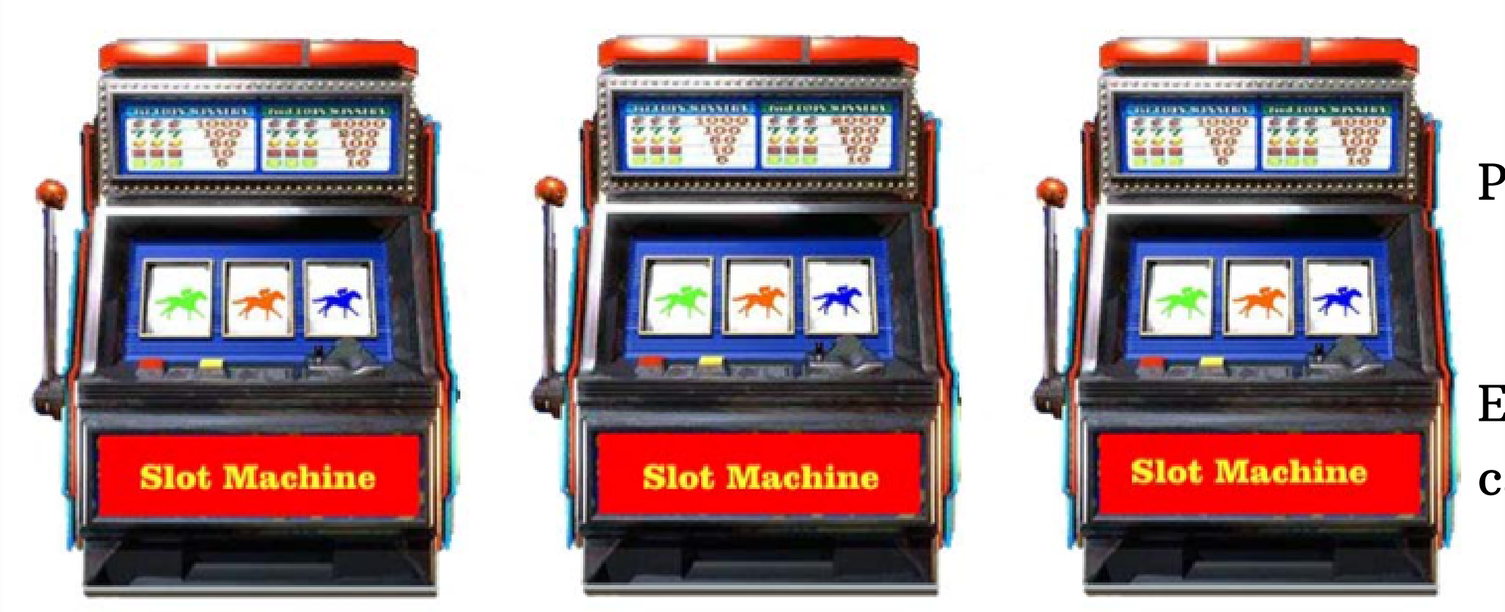


Recap

In the previous sections, we examined the fundamental concepts of reinforcement learning and investigated several fascinating reinforcement learning algorithms. We discussed **dynamic programming** as a **model-based technique**, the **Monte Carlo method** as a **model-free approach**, and introduced the temporal difference method, which merges the advantages of dynamic programming and the Monte Carlo method.

The MAB problem

The Multi-Armed Bandit (MAB) dilemma is a well-known problem in reinforcement learning. In this scenario, each arm of the slot machine gives a reward based on a particular probability distribution when activated. A single slot machine is known as a one-armed bandit, whereas multiple slot machines create a MAB or k-armed bandit, with 'k' denoting the number of slot machines present.



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

Pulling arms for rewards:

- Winning (+1 reward)
- Losing (0 reward)

Each slot machine is called an "arm”

Multi-Armed Bandit (MAB)

As each slot machine gives us the reward from its own probability distribution, our goal is to find out which slot machine will give us the maximum cumulative reward over a sequence of time.So at each time step t, the agent performs an action i.e pulls an arm from the slot machine and receives a reward and goal of our agent is to maximize the cumulative reward.
We define the value of an arm Q(a) as average rewards received by pulling the arm:

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

So the optimal arm is the one which gives us maximum cumulative reward

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

Should we switch between exploring different arms in each round, or should we continue selecting the arm that provided a good reward in the previous rounds?
This situation is commonly referred to as the "exploitation dilemma."

Exploration

Exploitation

Exploration strategies

B/b

Exploration strategies

a

The probability distribution of the arm is [0.8, 0.2]

With arm 1 winning 80% of the time and arm 2 winning 20%, the aim is to determine which arm yields more wins.
Next, exploration strategies for the Multi-Armed Bandit problem will be implemented using Gym environments.

b

Different exploration strategies are used to find the best arm. This tutorial will focus on four common exploration strategies and their implementation to identify the optimal arm.

Exploration Strategies

The goal of our agent is to find the optimal arm and also to minimize the regret which can be defined as the cost of knowing which of the k arms is optimal. Now how do we find the best arm? Whether we should explore all the arms or choose the arm which already gives us a maximum cumulative reward? Here comes exploration-exploitation dilemma. Now we will see how to solve this dilemma using various exploration strategies as follows,

1. Epsilon-greedy policy
2. Softmax exploration
3. Upper Confidence bound algorithm
4. Thomson sampling technique

Epsilon-greedy

Overview: Epsilon-greedy is a simple yet effective exploration strategy. It selects the best arm with probability (1-ε) and explores a random arm with probability ε.

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

```
def epsilon_greedy(epsilon):

    if np.random.uniform(0,1) < epsilon:
        return env.action_space.sample()
    else:
        return np.argmax(Q)
```

Full Code can be found in Lab 6

Softmax Exploration

Overview: Softmax exploration selects arms probabilistically based on their estimated values. The probability of selecting an arm is proportional to its estimated value.

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

Choosing the arm based on the probability of average reward might not be precise in the early stages. To address this issue, we introduce a new parameter known as T, which is referred to as the temperature parameter i.e. tau

```
def softmax(tau):
    total = sum([math.exp(val/tau) for val in Q])
    probs = [math.exp(val/tau)/total for val in Q]

    threshold = random.random()
    cumulative_prob = 0.0
    for i in range(len(probs)):
        cumulative_prob += probs[i]
        if (cumulative_prob > threshold):
            return i
    return np.argmax(probs)
```

Upper Confidence Bound (UCB)

Overview: UCB balances exploration and exploitation by selecting arms based on their upper confidence bounds, which consider both the estimated value and the uncertainty in the estimate.

To ensure the accuracy of mean rewards from arms in a game, a confidence interval is used to represent the range where the true mean value lies.

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

Upper Confidence Bound (UCB)

- Objective: Balance exploration and exploitation in multi-armed bandit problems.
- Key Idea: Prioritise actions that have high potential but uncertain outcomes.
- Algorithm:
- Initialize: Set each arm's value estimate and confidence bounds.
- Action Selection: Choose the arm with the highest upper confidence bound.
- Update: Update the value estimate and confidence bounds based on the outcome.
- Advantages:
- Efficiently explores less-known actions.
- Avoids overly frequent selection of suboptimal actions.
- Example:
- Suppose you have 3 slot machines (arms) with unknown payout rates.
- UCB would balance between trying new machines and exploiting the best-known one.

```python
def UCB(iters):

    ucb = np.zeros(10)

    #explore all the arms
    if iters < 10:
        return i

    else:
        for arm in range(10):

            # calculate upper bound
            upper_bound = math.sqrt((2*math.log(sum(count))) / count[arm])

            # add upper bound to the Q valyue
            ucb[arm] = Q[arm] + upper_bound

        # return the arm which has maximum value
        return (np.argmax(ucb))
```

Thomson Sampling

Overview: Thomson Sampling is a Bayesian approach that samples arms according to their posterior probabilities of being optimal. It inherently accounts for uncertainty in the estimates.

Thomson Sampling: A Bayesian Approach

- Objective:
  - Select the best arm in a Multi-Armed Bandit problem while balancing exploration and exploitation.
- Approach:
  - Bayesian method that maintains a probability distribution over the true reward of each arm.
  - Samples arms according to their posterior probabilities of being optimal.
- Advantages:
  - Accounts for uncertainty in the estimates by sampling from the posterior distribution.
  - Naturally balances exploration and exploitation.
- Implementation:
  - Maintain a posterior distribution for each arm.
  - Select arm to play based on sampling from these distributions.
- Result:
  - Efficiently learns the best arm over time while exploring other arms.

```python
def UCB(iters):

    ucb = np.zeros(10)

    #explore all the arms
    if iters < 10:
        return i

    else:
        for arm in range(10):

            # calculate upper bound
            upper_bound = math.sqrt((2*math.log(sum(count))) / count[arm])

            # add upper bound to the Q valyue
            ucb[arm] = Q[arm] + upper_bound

        # return the arm which has maximum value
        return (np.argmax(ucb))
```

Contextual bandits

We've recently discovered how to utilize bandits to determine the most effective advertisement banner for individual users. However, it's important to note that each user has their own unique banner preferences; for instance, user A may prefer banner 1 while user B may prefer banner 3, and so forth. To address this, we must tailor advertisement banners to suit each user's preferences. This is where contextual bandits come into play. Unlike in the Multi-Armed Bandit (MAB) problem where we simply take actions and receive rewards, contextual bandits involve taking actions based on the environment's state, with the state providing the necessary context.

Contextual bandits

In the scenario of the advertisement banner, the state defines user behavior, and we respond by displaying the banner based on the state (user behavior) that yields the highest reward (ad clicks). Contextual bandits are commonly employed to tailor content to user behavior and address the cold-start challenges encountered by recommendation systems. Netflix utilizes contextual bandits to customize artwork for TV shows based on user behavior.

Summary

- MAB is a problem where an agent must decide which arm to pull (action to take) to maximize a reward, with each arm having an unknown reward distribution.
- Exploration Strategies:
  - Epsilon-Greedy: Balances exploration and exploitation by selecting the best arm with probability (1-epsilon) and a random arm with probability epsilon.
  - Softmax Exploration: Selects arms based on a probability distribution where the probability of each arm is proportional to its average reward.
  - UCB Algorithm: Selects the arm with the highest upper confidence bound, balancing exploration and exploitation.
  - Thompson Sampling: Samples arms according to their posterior probabilities of being optimal, using a Bayesian approach.
- MAB vs. AB Testing:
  - MAB can be used as an alternative to AB testing for determining the best option among several alternatives, such as finding the best advertisement banner.
- Contextual Bandits:
  - An extension of the MAB problem where the reward for each arm depends on a context or state.

THANK YOU!