DRL HW3 說明報告

▼ 1. 作業概述

多臂機器人問題（Multi-Armed Bandit, MAB）是強化學習中的經典問題之一，其命名來自賭場中的拉霸機概念：面對多台報酬機率未知的拉霸機，玩家每次只能選擇一台進行操作，目的是在有限次的操作中獲得最大的總報酬。這個問題本質上是在探索（嘗試新選項以獲得更多資訊）與利用（選擇目前看起來最佳的選項以獲得最大報酬）之間取得平衡，適用於如廣告投放、推薦系統、臨床試驗等各類決策場景。

本作業將針對四種常見的 MAB 演算法進行探討與比較，分別為 Epsilon-Greedy、UCB(Upper Confidence Bound)、Softmax 以及 Thompson Sampling。每個演算法的內容將包含數學公式推導、幫助理解或解析該算法的 ChatGPT 提示語、實作程式碼與結果圖表展示，並針對演算法表現進行時間與空間效率的分析。藉由本次作業，我們不僅能實際操作並觀察各種策略的學習行為，也能深入理解它們在不同情境下對探索與利用所做的取捨。

▼ 2. 算法公式 (含 Latex 程式碼)

Formulas of Four Multi-Armed Bandit Algorithms

1. Epsilon-Greedy Algorithm

Description: Epsilon-Greedy balances exploration and exploitation. With probability $1-\varepsilon$, it selects the action with the highest estimated reward. With probability ε, it explores randomly.

$$A_t = \begin{cases} \arg\max_a Q_t(a), & \text{with probability } 1-\varepsilon \\ \text{randomly select } a \in \mathcal{A}, & \text{with probability } \varepsilon \end{cases}$$

Explanation:

- A_t: the action selected at time t.
- $Q_t(a)$: estimated reward for action a at time t.
- ε: exploration probability.
- $\mathcal{A}$: set of all possible actions.

2. UCB (Upper Confidence Bound) Algorithm

Description: UCB selects actions based on the upper confidence bound, balancing current reward estimates with uncertainty.

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

Explanation:

- $Q_t(a)$: average reward of action a.
- $N_t(a)$: number of times action a has been selected before time t.
- t: current round.
- c: exploration constant.

3. Softmax Algorithm

Description: Softmax assigns a probability to each action using the softmax function, allowing smoother exploration based on reward estimates.

$$P(a) = \frac{\exp(Q_t(a)/\tau)}{\sum_{b \in \mathcal{A}} \exp(Q_t(b)/\tau)}$$

Explanation:

- $P(a)$: probability of selecting action a.
- $Q_t(a)$: estimated reward of action a.
- τ: temperature parameter; higher values encourage more exploration.
- $\mathcal{A}$: set of all possible actions.

4. Thompson Sampling Algorithm

Description: Thompson Sampling uses Bayesian inference to maintain uncertainty and samples from posterior distributions to select actions.

$$\theta_a \sim p(\theta_a \mid \text{history}), \quad A_t = \arg\max_a \theta_a$$

Explanation:

- θ_a: a sampled parameter from the posterior distribution for action a.
- $p(\theta_a \mid \text{history})$: posterior distribution of action a based on past observations.
- A_t: action selected with the highest sampled value.

```
\documentclass{article}
\usepackage{amsmath}
\usepackage{enumitem}
\usepackage{titlesec}
\usepackage{geometry}
\geometry{margin=1in}

% Reduce space after section titles
\titlespacing*{\section}{0pt}{0pt}{6pt}
\titleformat{\section}{\normalfont\Large\bfseries}{\thesection.}{0.5em}{}

% Indent all paragraphs
\setlength{\parindent}{2em}
\setlength{\parskip}{0pt}

\title{Formulas of Four Multi-Armed Bandit Algorithms}
\author{}
\date{}
\begin{document}

\maketitle\vspace{-1em}

% ---------------------- 1. Epsilon-Greedy ----------------------
\section{Epsilon-Greedy Algorithm}

\textbf{Description:} Epsilon-Greedy balances exploration and exploitation. With probability $1 - \varepsilon$

\[
A_t =
\begin{cases}
\arg\max_a Q_t(a), & \text{with probability } 1 - \varepsilon \\
\text{randomly select } a \in \mathcal{A}, & \text{with probability } \varepsilon
\end{cases}
\]

\textbf{Explanation:}
\begin{itemize}[leftmargin=2em]
  \item $A_t$: the action selected at time $t$.
  \item $Q_t(a)$: estimated reward for action $a$ at time $t$.
  \item $\varepsilon$: exploration probability.
  \item $\mathcal{A}$: set of all possible actions.
\end{itemize}

% ---------------------- 2. UCB ----------------------
```

```
\section{UCB (Upper Confidence Bound) Algorithm}

\textbf{Description:} UCB selects actions based on the upper confidence bound, balancing current reward es

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

\textbf{Explanation:}
\begin{itemize}[leftmargin=2em]
  \item $Q_t(a)$: average reward of action $a$.
  \item $N_t(a)$: number of times action $a$ has been selected before time $t$.
  \item $t$: current round.
  \item $c$: exploration constant.
\end{itemize}

% ---------------------- 3. Softmax ----------------------
\section{Softmax Algorithm}

\textbf{Description:} Softmax assigns a probability to each action using the softmax function, allowing smoot

\[
P(a) = \frac{\exp(Q_t(a)/\tau)}{\sum_{b \in \mathcal{A}} \exp(Q_t(b)/\tau)}
\]

\textbf{Explanation:}
\begin{itemize}[leftmargin=2em]
  \item $P(a)$: probability of selecting action $a$.
  \item $Q_t(a)$: estimated reward of action $a$.
  \item $\tau$: temperature parameter; higher values encourage more exploration.
  \item $\mathcal{A}$: set of all possible actions.
\end{itemize}

% ---------------------- 4. Thompson Sampling ----------------------
\section{Thompson Sampling Algorithm}

\textbf{Description:} Thompson Sampling uses Bayesian inference to maintain uncertainty and samples from

\[
\theta_a \sim p(\theta_a \mid \text{history}), \quad A_t = \arg\max_a \theta_a
\]

\textbf{Explanation:}
\begin{itemize}[leftmargin=2em]
  \item $\theta_a$: a sampled parameter from the posterior distribution for action $a$.
  \item $p(\theta_a \mid \text{history})$: posterior distribution of action $a$ based on past observations.
  \item $A_t$: action selected with the highest sampled value.
\end{itemize}

\end{document}
```

▼ 3. 算法解析 (含 ChatGPT Prompt)

> 📝 請詳細解釋四種 Multi-Armed Bandit 演算法的核心概念與策略差異：
>
> 1. Epsilon-Greedy：說明它如何在探索與利用之間做平衡，以及 epsilon 參數對選擇行為的影響。
> 2. UCB（Upper Confidence Bound）：介紹它如何根據每個選項的「平均報酬」和「不確定性」來做選擇，並強調它的探索特性。
> 3. Softmax：解釋它如何根據每個選項的相對價值來分配選擇機率，以及溫度參數對探索強度的調控作用。
> 4. Thompson Sampling：描述它如何透過機率分布來進行決策，並使用過往觀察資料不斷更新模型來平衡探索與利用。

3-1. Epsilon-Greedy

核心概念：

Epsilon-Greedy 是最基本的探索與利用策略之一。它會以固定機率 `ε` 隨機選擇一個動作（探索 explore），其餘時間（1−ε）則選擇當前估計報酬最高的動作（利用 exploit）。

策略特性：

- **探索比例固定**：探索與利用的比例由 ε 決定，無論當前已知資訊如何都會維持固定比例。
- **簡單易實作**：程式碼短、概念直覺。

ε 的影響：

- ε 越大：更常進行隨機探索，學習較慢但更全面。
- ε 越小：傾向利用既有知識，收斂快但可能卡在次優解。

3-2. UCB（Upper Confidence Bound）

核心概念：

UCB 在選擇動作時不只考慮平均報酬，還會考慮「這個選項被嘗試的次數」以推估其不確定性，並偏好那些未被充分探索的選項。

策略特性：

- **動態調整探索強度**：對嘗試次數少的選項給較高的「信賴上界」，鼓勵探索。
- **隨著時間收斂**：初期積極探索，後期會穩定在最佳解上。

優勢：

- 能更有系統地避免「早期誤判」。
- 探索是根據統計意義來決定，而非隨機。

3-3. Softmax

核心概念：

Softmax 會根據每個選項的相對價值（例如目前估計的平均報酬）計算選擇機率，所有選項都有機會被選到。

策略特性：

- **探索是機率性的**：高價值的選項有較高機率，但不是絕對；低價值選項仍有機會。

- **調整參數 τ（溫度）**：
  - τ 高 → 探索多，機率差距小。
  - τ 低 → 趨向利用，幾乎只選最好的選項。

3-4. Thompson Sampling

核心概念：

Thompson Sampling 使用貝氏推論來處理不確定性。對每個選項維護一個機率模型（如 Beta 分布），每次根據該分布抽樣，並選出抽樣值最高的選項。

策略特性：

- **自然地平衡探索與利用**：不需外部參數控制，探索行為隨學習自動調整。
- **利用觀察數據更新模型**：透過每次回饋來修正成功／失敗的機率估計。

▼ 4. 程式碼與圖表

`hw3.py` 實現四種 Multi-Armed Bandit 的演算法，以下是**詳細分段解析**：

4-1. 全域模擬設定

- 固定亂數種子，確保模擬可重現。
- 設定 10 支拉霸機，模擬時間長度為 1000 步，每種演算法各模擬 200 次。
- `true_rewards` 為每支拉霸機的實際平均報酬。

```
# Multi-Armed Bandit (MAB) Simulation with Four Algorithms
import numpy as np
import matplotlib.pyplot as plt

# --------------------------
# Global Simulation Settings
# --------------------------
np.random.seed(0)
n_arms = 10
n_steps = 1000
n_trials = 200
true_rewards = np.random.normal(0, 1, n_arms)
```

- 每次模擬回傳一個有雜訊的 reward 向量（真實報酬 + 標準差為 1 的隨機變動）。

```
def simulate_bandit():
    return np.random.normal(true_rewards, 1)
```

4-2. 演算法實作 — Epsilon-Greedy

- 以 `epsilon` 機率探索、其餘時間利用。
- 探索與否以 `explore_flags` 紀錄。

```
class EpsilonGreedy:
    def __init__(self, n_arms, epsilon=0.1):
        self.epsilon = epsilon
```

```
        self.counts = np.zeros(n_arms)
        self.values = np.zeros(n_arms)
        self.explore_flags = []

    def select_action(self):
        if np.random.rand() < self.epsilon:
            self.explore_flags.append(1)
            return np.random.randint(len(self.values))
        else:
            self.explore_flags.append(0)
            return np.argmax(self.values)

    def update(self, action, reward):
        self.counts[action] += 1
        self.values[action] += (reward - self.values[action]) / self.counts[action]
```

4-3. 演算法實作 — UCB（Upper Confidence Bound）

- 當前報酬 + 探索項（依選擇次數調整）。
- 確保每個 arm 都至少被選過一次。

```
class UCB:
    def __init__(self, n_arms):
        self.counts = np.zeros(n_arms)
        self.values = np.zeros(n_arms)
        self.total_counts = 0
        self.explore_flags = []

    def select_action(self):
        self.total_counts += 1
        for a in range(len(self.values)):
            if self.counts[a] == 0:
                self.explore_flags.append(1)
                return a
        confidence_bounds = self.values + np.sqrt((2 * np.log(self.total_counts)) / self.counts)
        self.explore_flags.append(0)
        return np.argmax(confidence_bounds)

    def update(self, action, reward):
        self.counts[action] += 1
        self.values[action] += (reward - self.values[action]) / self.counts[action]
```

4-4. 演算法實作 — Softmax

- 利用 softmax 函數將報酬轉為選擇機率。
- 使用固定學習率 `0.1` 更新值。

```
class Softmax:
    def __init__(self, n_arms, tau=0.1):
        self.tau = tau
        self.values = np.zeros(n_arms)
        self.explore_flags = []
```

```
    def select_action(self):
        exp_vals = np.exp(self.values / self.tau)
        probs = exp_vals / np.sum(exp_vals)
        choice = np.random.choice(len(self.values), p=probs)
        self.explore_flags.append(1 if probs[choice] < 1.0 else 0)
        return choice

    def update(self, action, reward):
        self.values[action] += 0.1 * (reward - self.values[action])
```

4-5. 演算法實作 — Thompson Sampling

- 使用 Beta 分布對每個 arm 建立後驗抽樣。
- 根據觀察更新 α、β 值。

```
class ThompsonSampling:
    def __init__(self, n_arms):
        self.alpha = np.ones(n_arms)
        self.beta = np.ones(n_arms)
        self.explore_flags = []

    def select_action(self):
        samples = np.random.beta(self.alpha, self.beta)
        greedy = np.argmax(self.alpha / (self.alpha + self.beta))
        sampled = np.argmax(samples)
        self.explore_flags.append(1 if sampled != greedy else 0)
        return sampled

    def update(self, action, reward):
        if reward > 0:
            self.alpha[action] += 1
        else:
            self.beta[action] += 1
```

4-6. 執行模擬並取得結果

- 為每個 agent 跑 `n_trials` 次模擬。
- 收集 reward 與探索比例。
- `explore_flags` 用於統計探索率。
- 利用 lambda 傳入參數（對 `EpsilonGreedy` 與 `Softmax`）。
- 每個演算法都執行 200 次模擬。

```
# --------------------------
# Simulation Core Function
# --------------------------
def run_simulation(agent_class):
    rewards = np.zeros(n_steps)
    explore_ratios = np.zeros(n_steps)

    for _ in range(n_trials):
        agent = agent_class(n_arms)
        explore_flags = []
```

```
        for t in range(n_steps):
            action = agent.select_action()
            reward = simulate_bandit()[action]
            agent.update(action, reward)
            rewards[t] += reward
            explore_flags.append(agent.explore_flags[-1])

        explore_ratios += np.cumsum(explore_flags) / (np.arange(1, n_steps + 1))

    rewards /= n_trials
    explore_ratios /= n_trials

    return rewards, explore_ratios

# --------------------------
# Execute Simulations
# --------------------------
eg_rewards, eg_explore = run_simulation(lambda n: EpsilonGreedy(n, epsilon=0.1))
ucb_rewards, ucb_explore = run_simulation(UCB)
softmax_rewards, softmax_explore = run_simulation(lambda n: Softmax(n, tau=0.1))
ts_rewards, ts_explore = run_simulation(ThompsonSampling)
```

4-7. 結果視覺化並儲存

- 第一張圖：各演算法的累積報酬。
- 第二張圖：各演算法的探索比例變化。

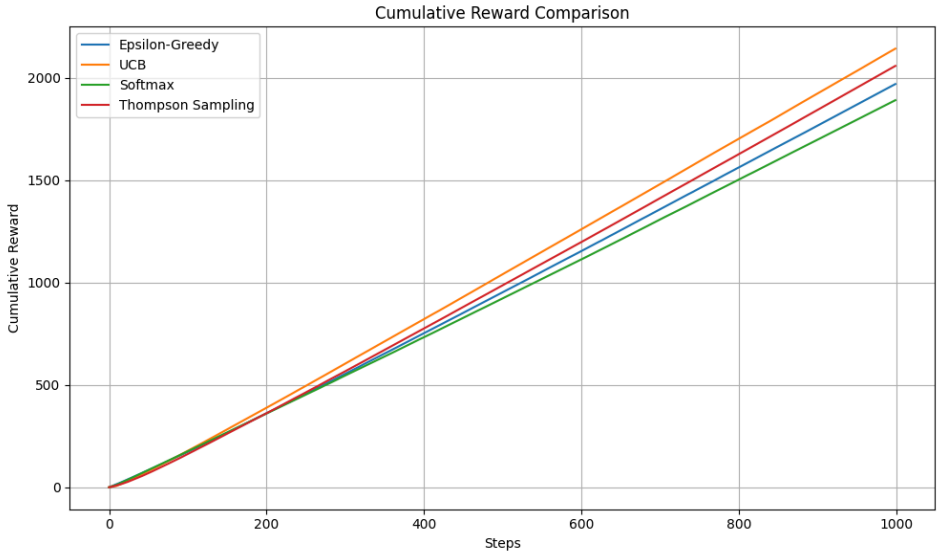
```
# --------------------------
# Plot Results
# --------------------------
plt.figure(figsize=(10, 6))
plt.plot(np.cumsum(eg_rewards), label='Epsilon-Greedy')
plt.plot(np.cumsum(ucb_rewards), label='UCB')
plt.plot(np.cumsum(softmax_rewards), label='Softmax')
plt.plot(np.cumsum(ts_rewards), label='Thompson Sampling')
plt.title('Cumulative Reward Comparison')
plt.xlabel('Steps')
plt.ylabel('Cumulative Reward')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.savefig("cumulative_reward_comparison.png")  # 儲存圖表

plt.figure(figsize=(10, 4))
plt.plot(eg_explore, label='Epsilon-Greedy')
plt.plot(ucb_explore, label='UCB')
plt.plot(softmax_explore, label='Softmax')
plt.plot(ts_explore, label='Thompson Sampling')
plt.title('Explore Ratio Over Time (All Algorithms)')
plt.xlabel('Steps')
plt.ylabel('Explore Ratio')
plt.legend()
plt.grid(True)
```

```
plt.tight_layout()
plt.savefig("explore_ratio_over_time.png") # 儲存圖表
```

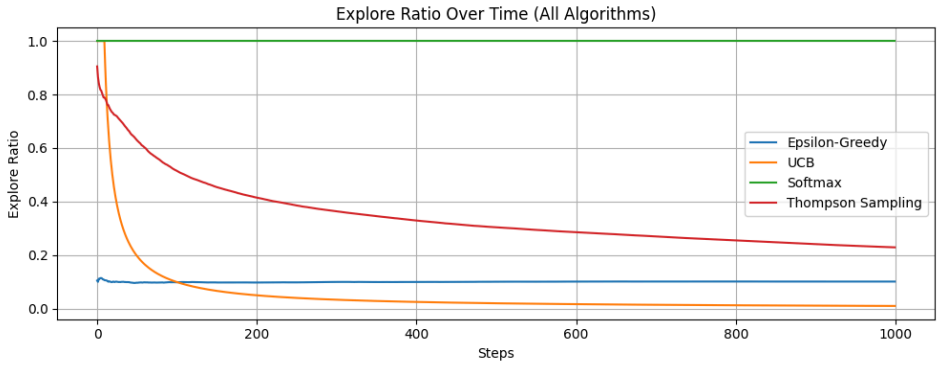
▼ 5. 結果解釋

5-1.第一張圖：累積報酬比較（Cumulative Reward Comparison）



- X軸：步驟數（Steps）
- Y軸：累積報酬（Cumulative Reward）
- 曲線解讀：
  - **UCB** 的報酬曲線最高，代表它在這次模擬中表現最佳。
  - **Thompson Sampling** 次之，也具有不錯的學習能力。
  - **Epsilon-Greedy** 中規中矩。
  - **Softmax** 表現最差，可能與參數設定（`tau=0.1`）過小有關，導致過度 exploitation。

5-2. 第二張圖：探索比例隨時間變化（Explore Ratio Over Time）



- X軸：步驟數（Steps）
- Y軸：探索比例（Explore Ratio）

- 曲線解讀：
  - **Softmax** 幾乎始終保持高探索率（接近 1），這是由 softmax 機率分布造成的特性。
  - **Epsilon-Greedy** 維持固定 ε 探索率（大約 0.1），符合設定。
  - **UCB** 探索率迅速下降，代表它能快速集中在較佳的 arm 上。
  - **Thompson Sampling** 初期探索多，後期逐漸穩定，展現其自適應探索能力。

5-3. 時間與空間層面分析

- 時間面（Temporal Efficiency）：
  - **UCB**：快速收斂，短時間內即可辨識出最佳選項，適合時間成本敏感場景。
  - **Thompson Sampling**：次於 UCB，但也具穩定收斂特性，適合中長期策略學習。
  - **Epsilon-Greedy**：收斂速度一般，受限於固定的 ε 探索率。
  - **Softmax**：收斂緩慢，且因過度探索導致學習效率低。
- 空間面（Exploration-Exploitation Trade-off）：
  - **UCB**：有效掌握每個選項的信賴區間，避免資源浪費。
  - **Thompson Sampling**：利用機率分布優化選擇，能因應不同 reward 結構，策略靈活。
  - **Epsilon-Greedy**：策略簡單但不夠精細，容易在選項數多時性能下降。
  - **Softmax**：探索過度，導致未能有效利用資源。

5-4. 總結與應用建議

演算法	優勢	限制	適用情境
UCB	收斂快、效果佳、理論基礎穩固	初期計算量略高	即時決策、資源有限場域
Thompson Sampling	自適應、穩定性強、具貝氏推論能力	建模略複雜、需瞭解分布假設	不確定性高、報酬結構動態
Epsilon-Greedy	簡單易懂、好實作	固定探索率不夠靈活	教學入門、快速實驗
Softmax	平滑選擇、具有機率性選擇機制	溫度參數難調整、探索過多	模型需引導更平均的選擇策略時