精打細算

——利用RL輔助消費決策

109034533 李珩慈

目錄 Content

問題定義5W1H

What: 建立**產品組合推薦**系統

Why: 提升**購物決策**的品質

Who: 精打細算的**消費者**

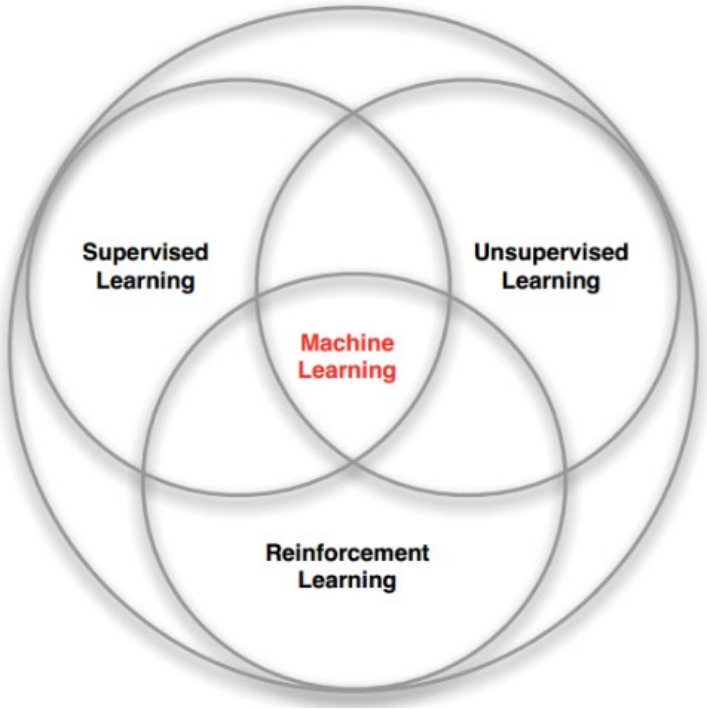
Where: 實體、線上**超市**

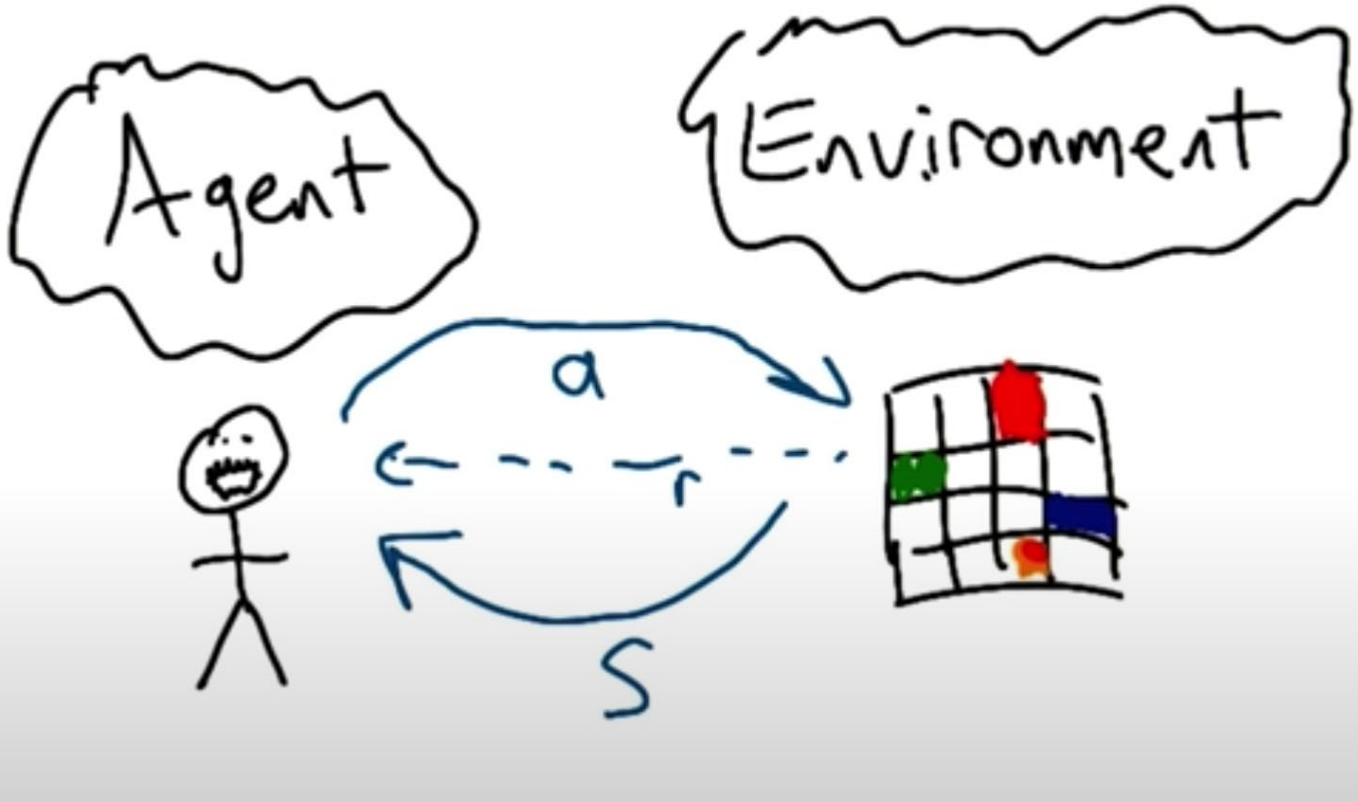
When: 準備做出**消費**行為時

How: 建立**強化學習模型**，協助消費者迅速判斷

*什麼是強化學習 (Reinforcement Learning)？

- **State:** 特定時間點agent 身處的狀態
- **Action:** agent 進行的動作
- **Reward:** environment 給予agent 所做action 的獎勵

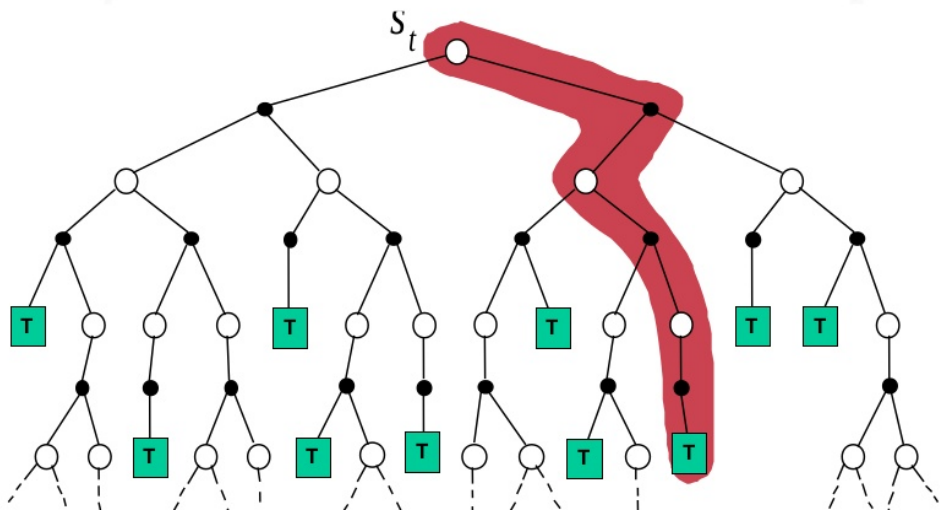




資料處理

*Monte Carlo Learning

依據經驗求解最佳策略，透過函式獲得Reward並依樣本平均回報解決RL問題



*如何計算?

$$V(a) \leftarrow V(a) + \alpha * (G - V(a))$$

學習率　R的總和

期末的生活品質

靠隊友躺分 R + 1

熬夜做報告 R -1

第一天　你的選擇：靠隊友躺分

V1(躺分) = 0 + 0.1*(1-0)
= 0.1

更新表

第二天　你的選擇：靠隊友躺分

• 建構資料

資料處理

2 x 2 x 2 x 3

= 24種選擇組合

商品價格　起始V值

Ingredient	Product	QMerged_label	Real_Cost	V_0
1	1	11	10	0
1	2	12	6	0
2	1	21	8	0
2	2	22	11	0
3	1	31	3	0
3	2	32	7	0
4	1	41	8	0
4	2	42	5	0
4	3	43	1	0

- **State:** 每一項食材
- **Action:** 選擇食材的特定品牌產品
- **Reward:** 是否有超過預算，無R_T+1；有則R_T -1

資料處理

Product	V_0(a)
a1	0
a2	0
b1	0
b2	0
c1	0
c2	0
d1	0
d2	0
d3	0

選擇：a1, b1, c1, d1

TotalCost = $29 < 30

因此 $R_T = +1$

$$V_1(a1) \leftarrow V_0(a1) + \alpha * (G - V_0(a1))$$

$$\Longrightarrow V_1(a1) \leftarrow 0 + 0.5 * (1 - 0)$$

$$= 0.5$$

V_1(a)
0.5
0
0.5
0
0.5
0
0.5
0
0

選擇：a1, b2, c2, d1

TotalCost = $36 < 30

因此 $R_T = -1$

$$V_2(a1) \leftarrow V_1(a1) + \alpha * (G - V_1(a1))$$

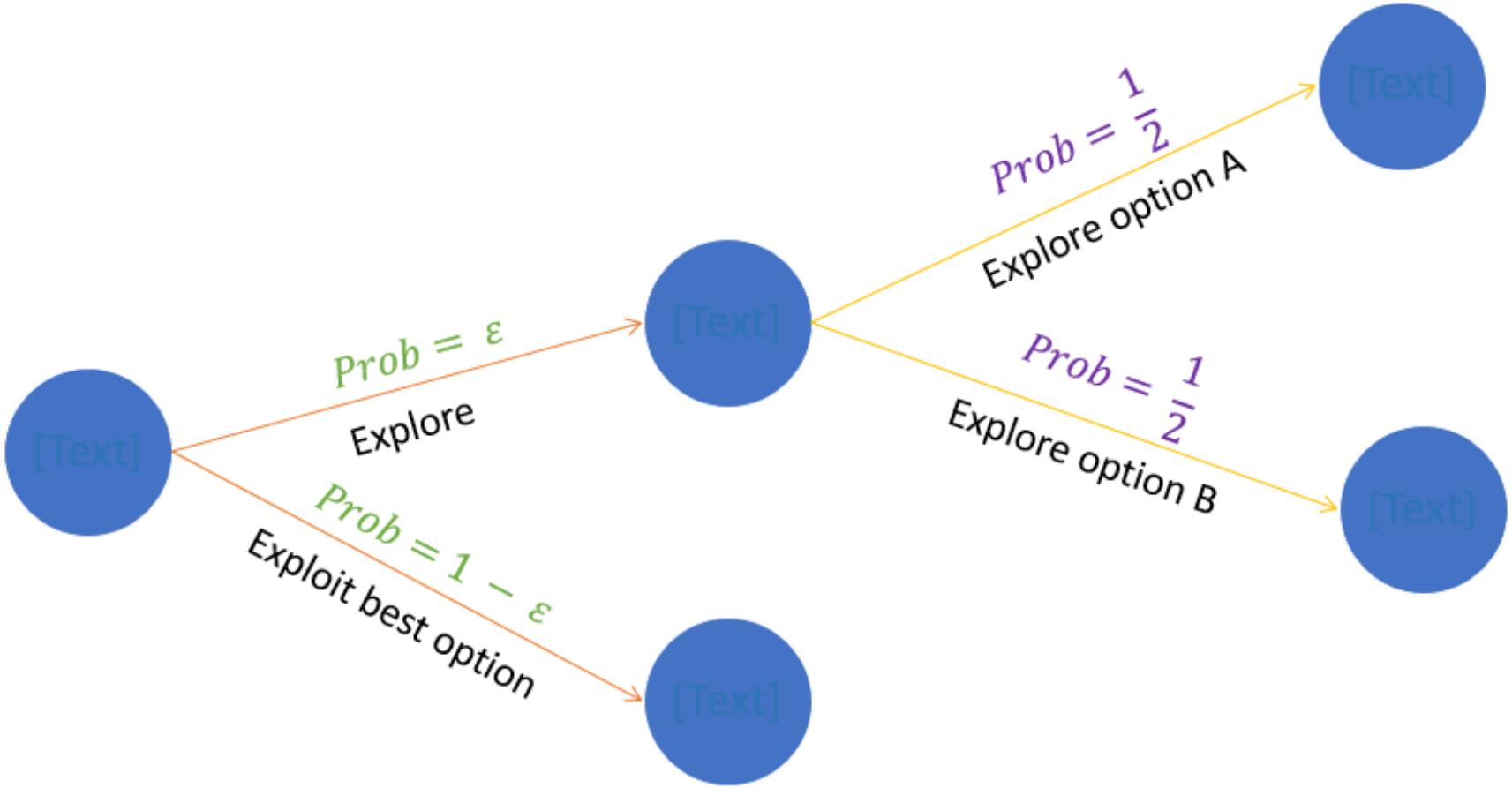
$$\Longrightarrow V_2(a1) \leftarrow 0.5 + 0.5 * (-1 - 0.5)$$

$$\Longrightarrow V_2(a1) = 0.5 - 0.75$$

V_2(a)
-0.25
0
0.5
-0.5
0.5
-0.5
-0.25
0
0

*如何選擇Action？ **Epsilon-greedy**

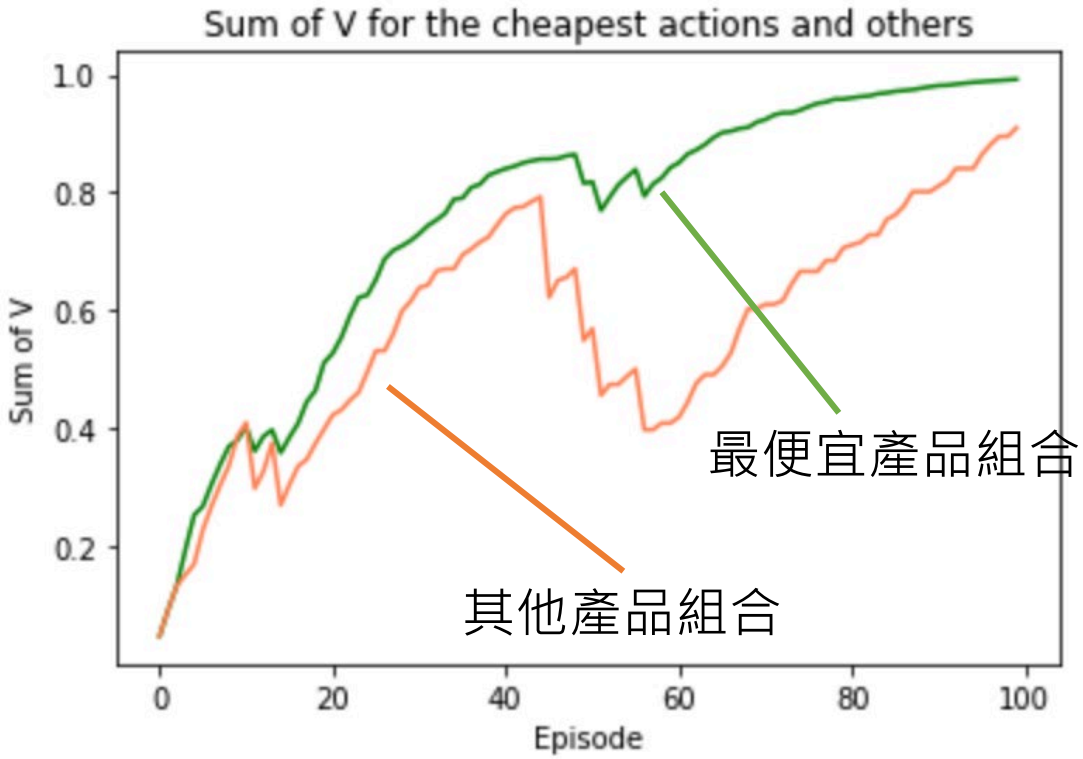
ε = 0.9 時，有 10%的機率會選擇V值最大的行為，
有 90%的機率會隨機選擇行為。

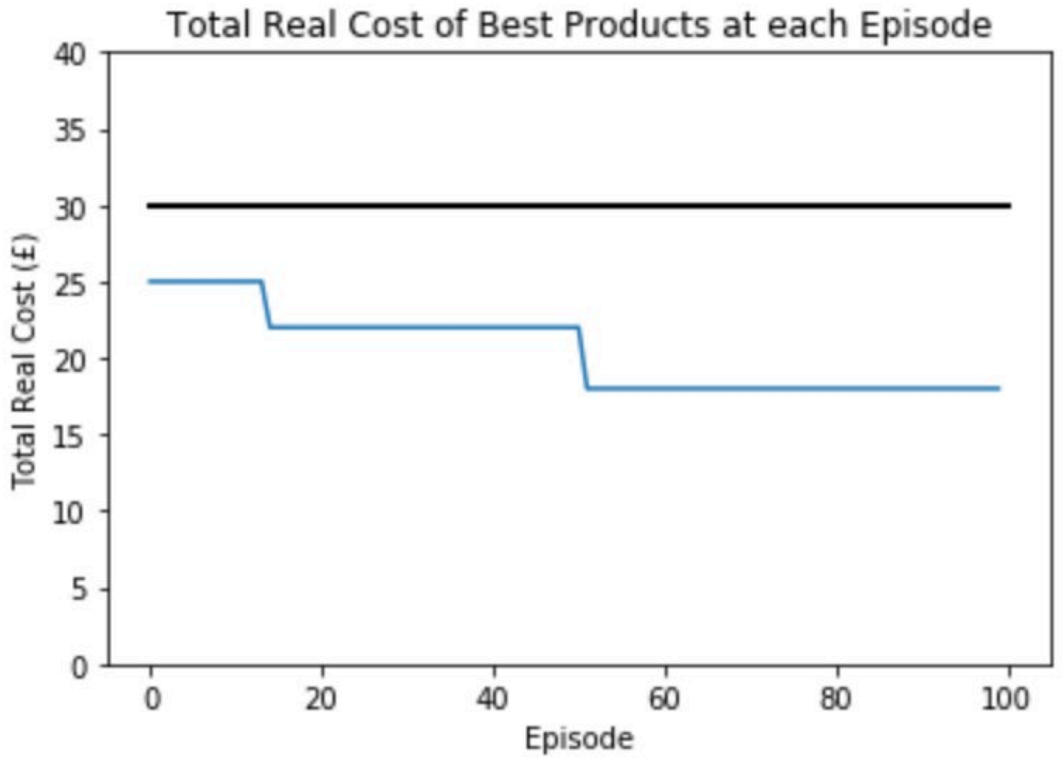


• 初始模型

```
if(budget >= episode2['Real_Cost'].sum()):
        Return = 1
else:
        Return = -1
```

- Budget = 30
- Alpha = 0.1
- Epilson = 0.5
- Num_episodes = 100





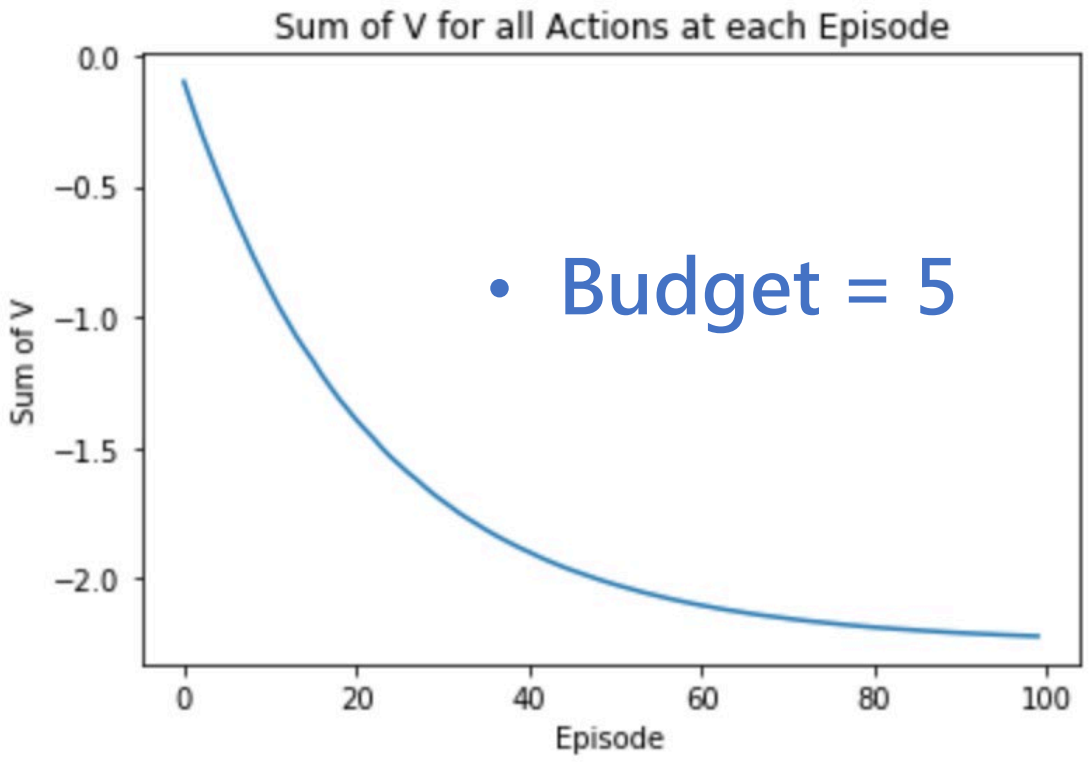
- 原始碼範例

```
def MCModelv1(data, alpha, e, epsilon, budget, reward):
    # Define the States
    Ingredients = list(set(data['Ingredient']))
    # Initialise V_0
    V0 = data['V_0']
    data['V'] = V0
    output = []
    output1 = []
    output2 = []
    actioninfull = []
    #Interate over the number of episodes specified
    for e in range(0,e):

        episode_run = []
        #Introduce epsilon-greedy selection, we randomly select the first episode as V_0(a) = 0 for all actions
        epsilon = epsilon
        if e == 0:
            for i in range(0,len(Ingredients)):
                episode_run = np.append(episode_run,np.random.random_integers(low = 1, high = sum(1 for p in data.iloc[:, 0] if p == i+1 ), size = None))
            episode_run = episode_run.astype(int)

        else:
            for i in range(0,len(Ingredients)):
                greedyselection = np.random.random_integers(low = 1, high =10)
                if greedyselection <= (epsilon)*10:
                    episode_run = np.append(episode_run,np.random.random_integers(low = 1, high = sum(1 for p in data.iloc[:, 0] if p == i+1 ), size = None))
                else:
                    data_I = data[data['Ingredient'] == (i+1)]
                    MaxofVforI = data_I[data_I['V'] == data_I['V'].max() ]['Product']
                    #If multiple max values, take first
                    MaxofVforI = MaxofVforI.values[0]
                    episode_run = np.append(episode_run, MaxofVforI)

                episode_run = episode_run.astype(int)
```

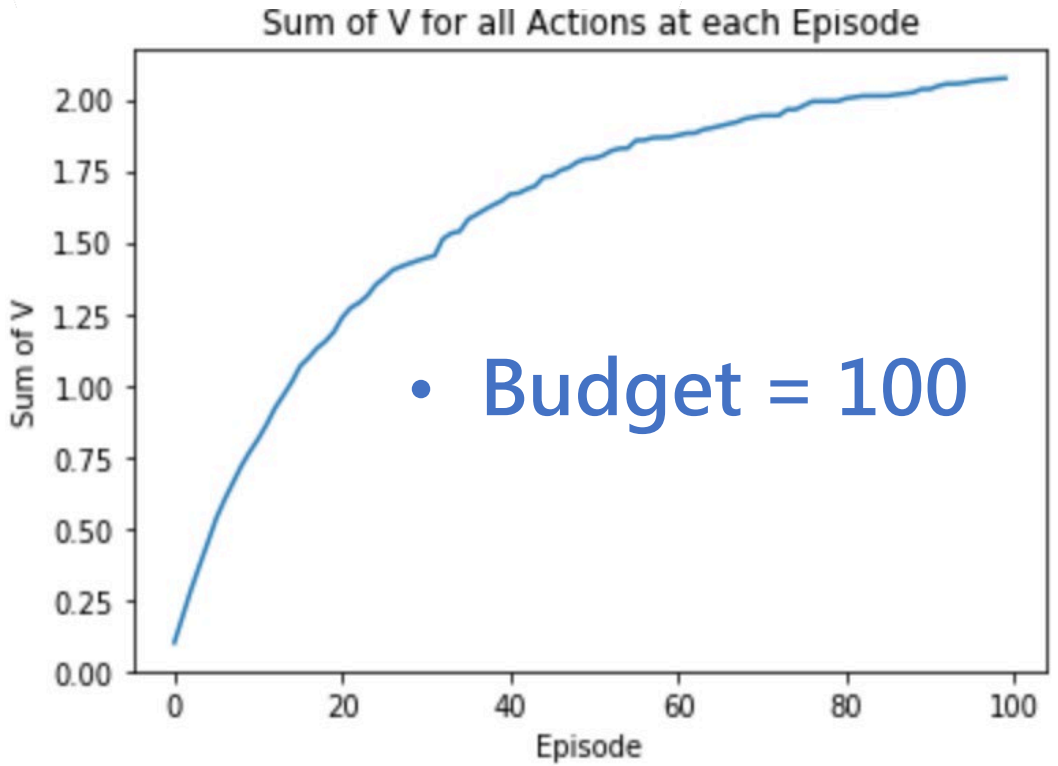
參數調整

• 預算: 非常小&非常高

- V值總和快速趨近於負值，觀察資料發現最便宜的產品組合也無法滿足預算限制



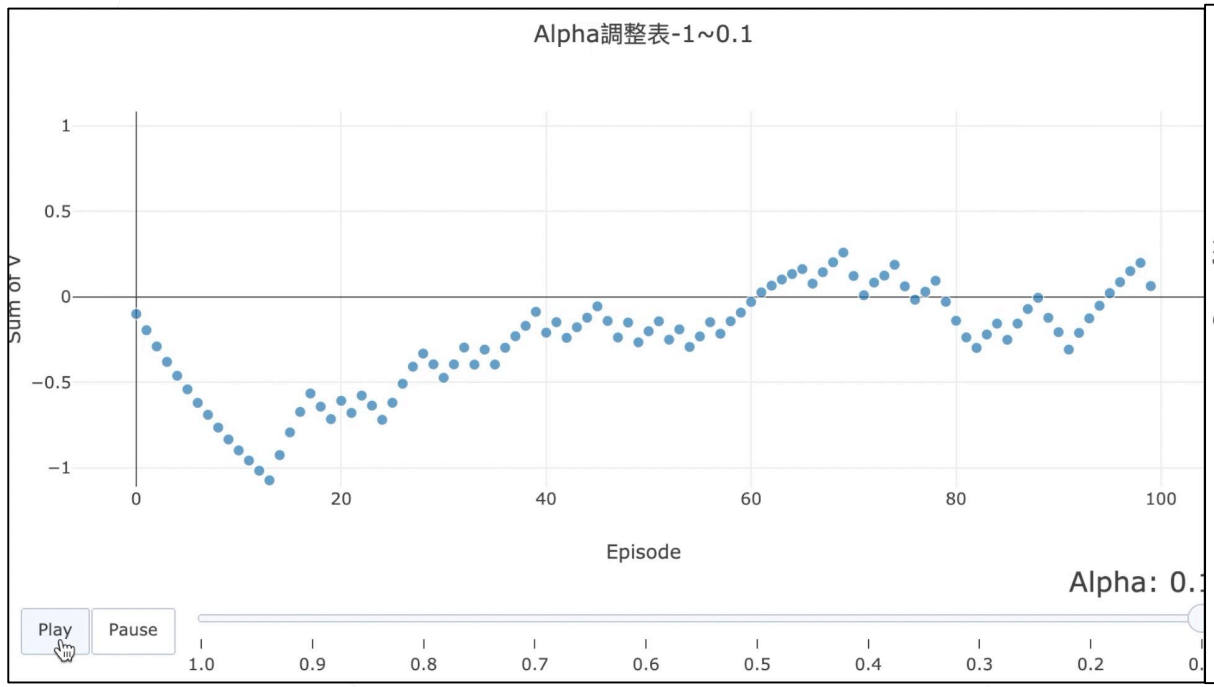
- V值總和皆為正值。由資料了解到最貴的商品組合也不會超過該預算

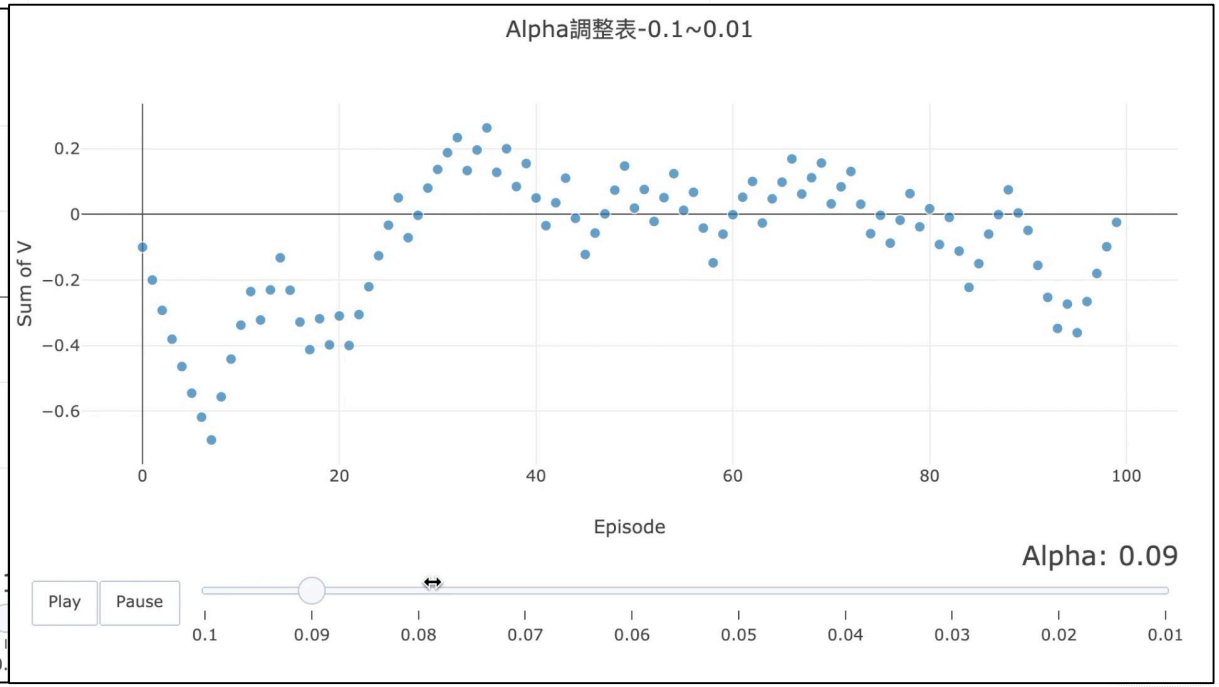


- **Alpha**: 1~0.1~0.01

α 為學習率，將會影響模型收斂到局部可行解或最佳解的速度。
影片中可看出 α 越小，sum of V在Episodes間越平滑，雖然可確保
局部最佳解的機率較小，但也意味著將耗費更多的時間收斂至最佳解。

參數調整





參數調整

- **Epsilon**: 1~0

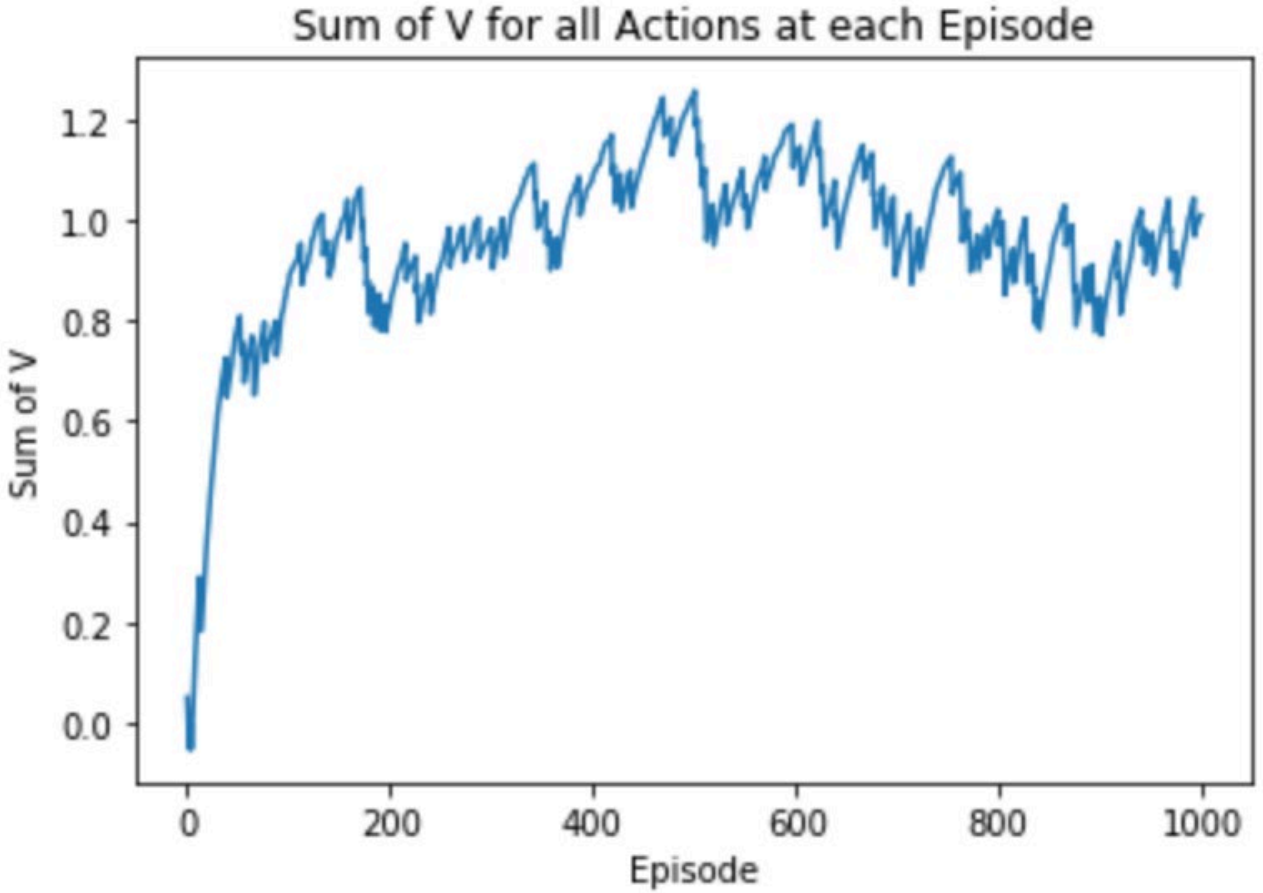
  Epsilon值越大代表有越大機率隨機選擇行為，使得V值總和分布零星，而Epsilon值越小代表有越大機會選擇選擇V值最大行為，因此V值總和越平滑，但需同時**避免落入局部最佳解**

- Alpha = 0.05
- Budget = 23
- Episodes = 100

參數調整

- **Episode:**

因**調降學習率**，此模型需增加運行的回合數**以達到較良好的學習**，
本模型可能有多種結果的選擇，因此曲線尚有崎嶇的現象 。

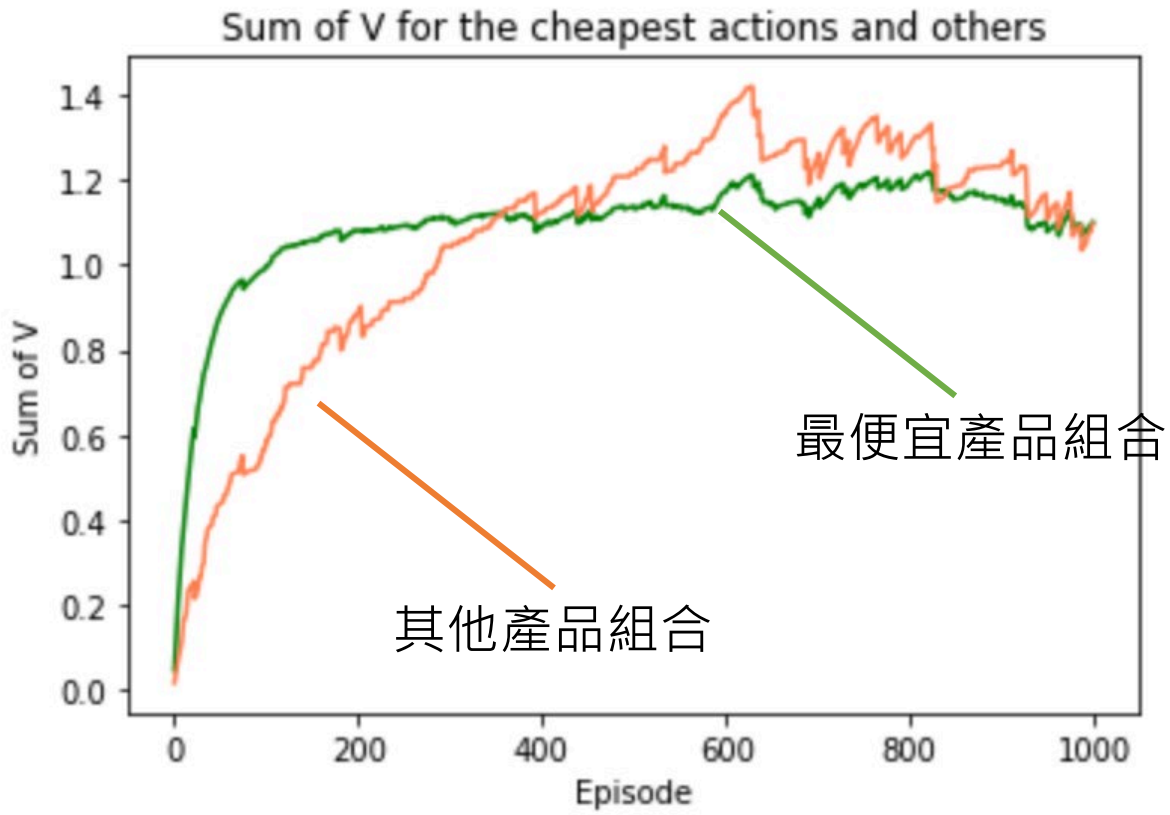


- Episodes = 1000
- Alpha = 0.05
- Epilson = 0.2
- Budget = 23

• 加入個人偏好

參數調整

```
if(budget >= episode2['Real_Cost'].sum()):
        Return = 1 + (episode2['Reward'].sum())/len(Ingredients)
else:
        Return = -1 + (episode2['Reward'].sum())/len(Ingredients)
```



- a2 reward = 0.8
- b2 reward = 0.8

結果

Ingredient	
1	2
2	2
3	1
4	3

結論與未來應用

零售業 利用歷史數據資料，建立個人化的產品組合偏好

公司採購 結合RPA，自動且智慧化選擇廠商的減少流程時間浪費

金融商品 透過數據分析，即時依決策目標 - 風險性和獲利性推薦資金配置組合

資料來源

李宏毅老師
https://www.youtube.com/watch?v=o_g9JUMw1Oc

Coursera
https://www.coursera.org/learn/ai-for-everyone

Terasoft
https://www.terasoft.com.tw/support/tech_articles/reinforcement_learning_a_brief_guide.asp

Kaggle
https://www.kaggle.com/osbornep/sample-data-for-learning-rl-and-monte-carlo

謝謝大家！