HW3 110652036 邱立昕

What is the difference between On-policy and Off-policy?

On-policy，要學習的 Agent 跟與環境互動的 Agent 是同一個 Agent。

留言

Off-policy，要學習的 Agent 跟與環境互動的 Agent 並不是同一個 Agent，意味著要學習的那個 Agent 是在一邊看著另一個 Agent 與環境互動，以它們互動的狀況來進行學習。使用這個方法要特別注意兩個 Agent 之間的差距。好比你看著 Joradn 空間停留三秒，但是你上場的時候根本學不起來

Briefly explain value-based, policy-based and Actor-Critic. Also, describe the value function $V^{\pi}(S)$

Policy-based 訓練一個 Actor，負責執行動作。

Value-based 訓練一個 Critic，負責對目前的狀況給予評價。

Actor-Critic 上述兩個混合使用

value function:給定一個 actor-π，看到某一個 observation(state)-s

，然後評估接下來一直到遊戲結束，我們會得到的 reward 有多大。

這個期望值即為$V^{\pi}(S)$

What is the difference between Monte-Carlo (MC) based approach and Temporal-difference (TD) approach for estimating $V^{\pi}(S)$

Monte-Carlo

讓 Critic 觀察目前的 actor-π的行為，讓 actor-π與環境互動，然後統計 actor-π會得到的 reward

舉例來說，它在看到sa之後會得到的 reward-Ga，注意到，這邊所統計的 reward 是一直到遊戲結束的 reward 總合，這樣子機器才有辦法看的長遠。

因此，機器要學習的就是當看到sa的時候，其$V\hat{}\pi(s)$要跟Ga愈接近愈好。

Temporal-difference

我們只看整個互動的其中一小段，某一個st採取什麼樣的at而得到多少的rt..

假設我們已經知道$V\pi(st)$，它跟下一個時間點st+1 之間差了一個rt即$V\pi(s$

$t)+rt=V\pi(st+1)$機器要學習的就是，$V\pi$

$(st)-V\pi(st+1)$要愈接近rt愈好。

Describe State-action value function Q and the relationship between Vπ (s, a) in Q-learning. π (S)

State-action value function Q:

輸入為 state 與 action 的 pair 不同於 $V\pi(s)$ 僅考慮 state 之後計算期望的 reward， $Q\pi(s,a)$ 考慮 state 與 action 來計算接下來可能得到的 reward 輸出為 scalar，告讓你接下來會得到的 reward 有多大 。

如果你的 action 可以窮舉的話就可以讓 $Q\pi$ 的輸入單純的只有 state，而輸出的部份就可以帶 action，每一個 dimension 都帶有一個 action，每一個 action 會有多少的 reward，但再次的說明，這只針對 discrete action 可行。

Q-Learning

$$\pi'(s) = arg\max_a Q^\pi(s,a)$$

$$V^{\pi'}(s) \geq V^\pi(s), \text{ for all state s}$$

$$V^\pi(s) = Q^\pi\big(s,\pi(s)\big)$$

$$\leq \max_a Q^\pi(s,a) = Q^\pi\big(s,\pi'(s)\big)$$

$$V^\pi(s) \leq Q^\pi\big(s,\pi'(s)\big)$$

$$= E_{\pi'}[r_{t+1} + V^\pi(s_{t+1})|s_t = s]$$

$$\leq E_{\pi'}[r_{t+1} + Q^\pi\big(s_{t+1},\pi'(s_{t+1})\big)|s_t = s]$$

$$= E_{\pi'}[r_{t+1} + r_{t+2} + V^\pi(s_{t+2})|s_t = s]$$

$$\leq E_{\pi'}[r_{t+1} + r_{t+2} + Q^\pi\big(s_{t+2},\pi'(s_{t+2})\big)|s_t = s]$$

$$\ldots\ldots \leq V^{\pi'}(s)$$

Created with EverCam.
http://www.camdemy.com

Describe following tips Target Network, Exploration and Replay Buffer using in Q-learning.

Target Network：

用於穩定訓練過程，常見的 Q-learning 算法，如 DQN，使用一個神經網絡來估

計 function Q。訓練過程中，會使用 Target Network 來計算目標 Q 值。Target Network 是原始 Q 網絡的一個複製品，在訓練過程中定期更新。這樣做的目的是降低訓練中的目標 Q 值估計和實際 Q 值之間的相互影響，從而提高訓練的穩定性和收斂性。

Exploration：

指 agent 在學習過程中採取未知行動或探索新的環境，以便獲取更多信息並改進策略。在 Q-learning 中，Exploration 通常通過使用 ε -greedy 策略實現，即以 1- ε 的概率選擇最佳已知動作，以 ε 的概率隨機選擇一個動作來進行探索，控制 ε 的值是一個重要的超參數，可以影響 agent 在學習過程中的探索和利用之間的平衡。

Replay Buffer:

是一種記憶機制，用於存儲 agent 在環境中觀察到的狀態、行動、獎勵和下一狀態等信息。在訓練過程中，agent 從 Replay Buffer 中隨機抽樣過去的經驗，並用這些經驗來進行 Q 值的更新和策略的改進。Replay Buffer 可以緩解訓練中的數據相關性問題，並增加訓練的效率和穩定性。

Explain what is different between DQN and Q-learning。

Q-learning：

一種基本的強化學習算法，用於解決馬可夫決策過程 Markov Decision Process，MDP 中的最優控制問題。

通常用於離散的狀態和行動空間，並且可以處理有限狀態空間下的問題。

通過更新 function Q 來學習最佳策略。具體而言，它使用 Bellman equation 來更新 Q，即根據目前的估計 Q 和後繼狀態的最大 Q 來更新目標 Q。

的主要優勢是簡單易懂，容易實現，特別適用於小規模的問題。

DQN（Deep Q-Network）：

基於深度學習的 Q-learning 的擴展，主要用於處理具有高維狀態空間和/或連續行動空間的問題。

使用神經網絡來近似 function Q，這使得它可以處理大型、連續的狀態和行動空間。

目標是通過訓練神經網絡，使其能夠預測在給定狀態下每個可能行動的 Q。

DQN 引入了幾項重要的技術，包括 Target Network、Replay Buffer 和離散化動作空間等，以提高訓練的穩定性和效果。

different：

應用範圍：Q-learning 適用於離散的狀態和行動空間，而 DQN 可以處理連續的、高維的狀態和行動空間。

函數近似：Q-learning 使用表格形式的 function Q，而 DQN 使用神經網絡來近似 function Q。

訓練方法：DQN 引入了深度學習中的技術，例如神經網絡訓練、目標網絡和回放緩存，使得訓練更加穩定和有效。

```
# Begin your code
moves = gameState.getLegalActions(0)
temp = []
for move in moves:
    temp.append((move, self.minimax(gameState.getNextState(0, move), self.depth-1, 1, False)))
move, _ = max(temp, key=lambda item: item[1][1])
return move
# End your code
```

獲取 legalaction 後使用 getnextstate 和 minimax 來找到對應的分數，並返回最大值

```
# Begin your code
moves = gameState.getLegalActions(0)
candidates = []
for move in moves:
    candidates.append((move, self.expectimax(gameState.getNextState(0, move), self.depth-1, 1, Fals
move,  _= max(candidates, key=lambda item: item[1])
return move
```

獲取 legal action 後使用 getnextstate 和來找到對應的分數，並返回最大值

```python
for _ in range(1, self.iterations+1):
    pre_val = self.values.copy()
    for state in self.mdp.getStates():
        if self.mdp.isTerminal(state):
            self.values[state] = 0
            continue
        max_val = -1e9
        for action in self.mdp.getPossibleActions(state):
            All_sum = 0
            for (nextState, prob) in self.mdp.getTransitionStatesAndProbs(state, action):

                All_sum += prob * (self.mdp.getReward(state, action, nextState) + self.discount*pre_val[next
            max_val = max(max_val, All_sum)
        self.values[state] = max_val
```

通過多次迭代計算，將每個狀態的值不斷更新至收斂，直到狀態值不再變化或達到指定的迭代次數

```python
# Begin your code
Q_value = 0
for (nextState, prob) in self.mdp.getTransitionStatesAndProbs(state, action):
    Q_value += prob * (self.mdp.getReward(state, action, nextState) + self.discount*self.values[nextState
return Q_value
# End your code
```

根據 self.values 和 MDP 的轉移函數來計算給定狀態和 Q-value

```python
# Begin your code
#check for terminal
if self.mdp.isTerminal(state):
    return None
actions = self.mdp.getPossibleActions(state)
Q_value = util.Counter()

for action in actions:
    Q_value[action] = self.getQValue(state, action)

return Q_value.argMax()

# End your code
```

檢查狀態是否為終止狀態，如果是終止狀態則返回 None，否則計算每個可能 Q value，並返回具有最高 Q value 作為策略

Describe problems you meet and how you solve them:

這次是作業卡到了期中週，斷斷續續做了很久才做完。Q-learning 實作非常的複雜，很多的 function，和很多的公式都是我去上網搜索才慢慢了解的，還上了不少網課才知道這次作業要做甚麼。DQN 不能用 colab 跑，我的筆電又沒有 nvidia 的顯卡，還特別回家用桌機才能跑 epoch，不過我是使用 wsl 的環境跑，又只能用內顯，所以跑了一整晚才出來結果。