DLP LAB7

0756138 黃伯凱

May 30, 2019

1 Introduction

這一次的LAB要實作Reinforcement Learning，用TD learning讓Agent學會怎麼玩2048。2048是一個只需要操作上下左右的遊戲，而且分數也都是2的次方數，比較容易實作出來。

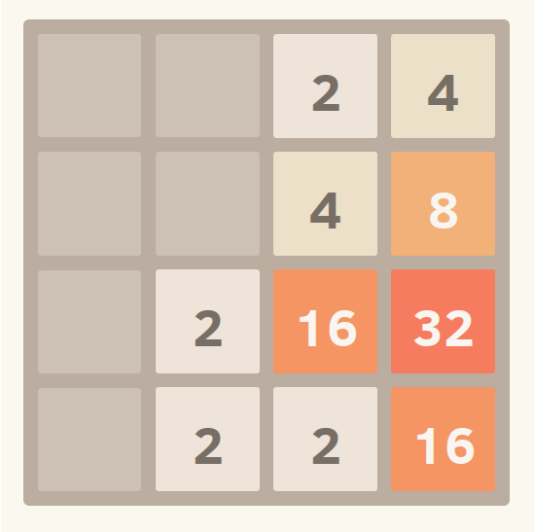

Figure 1: 2048 Game

2 Reinforcement Learning

因為RL的目的是要讓Agent自己學會做某件特定的事情，所以Agent必須學會在不同情況下必須要作出什麼樣的動作。每一種不同的情況都稱為**State**，Agent的決定的動作稱為**Action**。就2048為例，Environment每一次產生一個新的tile之後就是一個新的State，而Agent就要決定接下來要做出什麼Action。

RL的學習方法則是讓Agent從什麼都不會的情況下開始玩遊戲(Random Action)，通過Reward和Value的期望值來學習在不同的State底下做出什麼Action會比較好。所以讓Agent玩越多場遊戲、更新越多次weight，Agent就可以把遊戲玩得越來越好。

Agent怎麼決定Action是RL的一大課題，這一次的LAB要用Temporal Difference Learning讓Agent決定Aciton。又Temporal Difference Learning與Monte Carlo Learning有很大的關係，所以我們先來討論Monte Carlo Learning再看Temporal Difference Learning跟它有什麼樣的不同。

2.1 Monte Carlo Learning

Monte Carlo (MC) Learning的更新公式如下：

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)] \tag{1}$$

公式中的G_t是從S_t一直到整個episode結束後得到的reward總和，$V(S_t)$是在State S_t期望得到的Value。有了$V(S_t)$之後，每次Agent都可以參考下一個action預計得到的reward會是多少再決定下一個action。根據這個公式就可以看出MC必須要等到整個episode都結束才可以更新，所以學習速度會比較慢一些。

2.2 Temporal Difference Learning

希望不要向MC一樣要等到整個episode結束才能更新weight，於是參考了DP中的bootstrapping方法。Temporal Difference (TD) Learning 利用後續State的狀態來更新前面State的Value期望值，更新公式如下：

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] \tag{2}$$

公式中的$R_{t+1} + \gamma V(S_{t+1})$是TD Learning的目標，與MC中的$G_t$相對應。他們的差異就只是TD利用了bootstrapping的方法來更新。

TD Learning參考了DP和MC的做法，和MC不同的是它不用等到整個episode結束才可以更新。但因為MC是用G_t更新、TD是根據每一個State來更新，所以**MC會有高Bias、低Variance，TD會有低Bias、高Variance**。

3 State / AfterState Network

TD有兩種training的方法，分為TD(0)-state和TD(0)-afterstate。整體的演算法如下：

```
function PLAY GAME
  score ← 0
  s ← INITIALIZE GAME STATE
  while IS NOT TERMINAL STATE(s) do
    a ← argmax_{a'∈A(s)} EVALUATE(s, a')
    r, s', s'' ← MAKE MOVE(s, a)
    SAVE RECORD(s, a, r, s', s'')
    score ← score + r
    s ← s''
  for (s, a, r, s', s'') FROM TERMINAL DOWNTO INITIAL do
    LEARN EVALUATION(s, a, r, s', s'')
  return score

function MAKE MOVE(s, a)
  s', r ← COMPUTE AFTERSTATE(s, a)
  s'' ← ADD RANDOM TILE(s')
  return (r, s', s'')
```

Figure 2: Game Engine pseudo code

PLAY GAME的function會玩一次遊戲直到遊戲結束，也就是一個episode。EVALUATE(s,a')是回傳在State s的情況下做出Action a'時的Reward r，其中a'是在State s情況下所有可行的Action。

MAKE MOVE(s,a)會回傳在State s情況下執行Action a後的State s'與Reward

r，以及在State s'情況下Environment隨機產生一個tile後的State s''。再由SAVE RECORD記錄下來每一個State s與其相對應的Action a、State s'、Reward r和新的State s''。

在遊戲結束之後，會進到LEARN EVALUATION的function，這一個function就是TD Learning在更新weight的地方，它的做法會是從後面倒著更新回去，更新完之後這一個episode就正式結束。所以當我們的episode數量很大的時候，這一個Agent可以更新weight更多次並把遊戲玩得更好。

code中的EVALUATE和LEARN EVALUATION會因兩種不同方法而做法不同，下面兩個章節再討論這兩種不同的做法。

3.1 TD(0)-State

TD(0)-state的pseudo code如下：

TD(0)-state

function EVALUATE(s, a)
  $s', r \leftarrow$ COMPUTE AFTERSTATE(s, a)
  $S'' \leftarrow$ ALL POSSIBLE NEXT STATES(s')
  return $r + \Sigma_{s'' \in S''} P(s, a, s'')V(s'')$

function LEARN EVALUATION(s, a, r, s', s'')
  $V(s) \leftarrow V(s) + \alpha(r + V(s'') - V(s))$

Figure 3: TD(0)-State pseudo code

這邊的COMPUTE AFTERSTATE會先return State s執行Action a後的Reward r以及State s'。接下來是算出所有State s'有可能得到的新State s''並拿去算期望的value。這一個演算法的s''要考慮到所有Environment可能產生出來的State，所以它在return的時候要考慮到每一個s''的機率才乘上s''的Value $V(s'')$。所以EVALUATE這一個function可以算出做出什麼Action可以得到更高的期望分數。

LEARN EVALUATION和Eq.2基本上一模一樣，因為這個演算法在EVALUATE的時候就已經把各種可能的s''都考慮進去了，所以在update的時候就不用再做這一個步驟了。

3.2 TD(0)-Afterstate

TD(0)-afterstate的pseudo code如下：

TD(0)-afterstate

function EVALUATE(s, a)
  $s', r \leftarrow$ COMPUTE AFTERSTATE(s, a)
  return $r + V(s')$

function LEARN EVALUATION(s, a, r, s', s'')
  $a_{next} \leftarrow \underset{a' \in A(s'')}{\text{argmax}}\, EVALUATE(s'', a')$
  $s'_{next}, r_{next} \leftarrow COMPUTE\ \ AFTERSTATE(s'', a_{next})$
  $V(s') \leftarrow V(s') + \alpha(r_{next} + V(s'_{next}) - V(s'))$

Figure 4: TD(0)-Afterstate pseudo code

這邊的EVALUATE比上一個單純一些，不用考慮到Environment隨機產生的State。只需要算出reward以及State s'的期望分數就可以了。

LEARN EVALUATION會用到episode的record，要把每一個state s相對應的s''都拿出來丟進EVALUATE選出再下一個最佳的Action a_{next}。算出s''執行完a_{next}的s'_{next}和r_{next}後，就可以利用TD Learning的更新公式更新我們的$V(s')$期望值。

根據Fig.2，LEARN EVALUATION是從後面更新到前面的。這是因為遊戲進行的時候，Agent不會知道實際走哪一步會比較好，但是遊戲結束的那一個state可以確定它的value一定是0。所以倒著更新weight可以更正Agent學錯的Value，也可以讓Agent學對的Value更有更高的期望值。

4 Code

我做的是TD(0)-Afterstate，所以下面的code以及weight更新方式都是跟Fig.4一樣。

4.1 Game

```
with player(play_args) as play, rndenv(evil_args) as evil:
    for iteration, ep in enumerate(tqdm(range(stat.total))):
        play.open_episode("~:" + evil.name())
        evil.open_episode(play.name() + ":~")

        stat.open_episode(play.name() + ":" + evil.name())
        game = stat.back()

        idx = 1
        while True:
            who = game.take_turns(play, evil)
            gameOver, move = who.take_action(game.state())

            if gameOver:
                break
            else:
                game.apply_action(move)
            idx += 1
        win = game.last_turns(play, evil)  # The winner
        stat.close_episode(win.name())
        play.close_episode(game.ep_moves, win.name())
        evil.close_episode(win.name())
```

Figure 5: Code Snippet for Game Engine

這一段Code和Fig.2基本上是一樣的，for Loop裡面就是一個PLAY GAME function。while Loop會一直重複直到Game Over，其中第一行***who***是在決定輪到Player還是Environment，而***take_action***會執行不同agent相對應的動作：Environment agent會找到空格並隨機放2或4，Player agent的動作在Fig.10會介紹。在一個episode結束之後，要更新一次weight，在圖中的倒數第二行，這部分會在Fig.11介紹。

4.2 6-tuple Pattern

因為TD Learning要記錄每一個State是長什麼樣子，所以必須要紀錄每一格是什麼數字。假設每一格不會超過2^{16}，並且參考助教建議的6-tuple pattern (Fig.6)去設計。

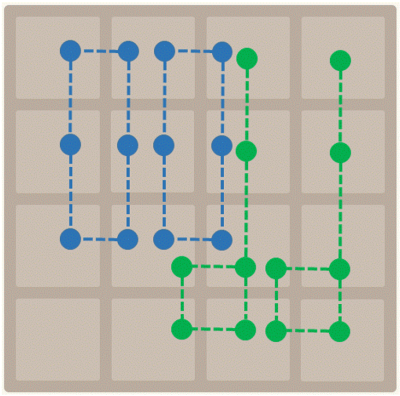

Figure 6: 6-tuple Pattern

所以我也建立了四組feature map，分別紀錄的位置如Fig.7圖中的註解。

```
def init_weights(self):
    self.net += [weight(16**6)] # feature for line [0 1 4 5 8 9] includes 16*16*16*16*16*16 possible
    self.net += [weight(16**6)] # feature for line [1 2 5 6 9 10] includes 16*16*16*16*16*16 possible
    self.net += [weight(16**6)] # feature for line [2 6 10 9 14 13] includes 16*16*16*16*16*16 possible
    self.net += [weight(16**6)] # feature for line [3 7 11 10 15 14] includes 16*16*16*16*16*16 possible
    return
```

Figure 7: Code Snippet for initial 6-tuple Pattern

因為拆成了4個feature map，所以一個state必須要得出四組index才能對應到4個feature map。Fig.8中的idx0∼idx3依序對應Fig.7圖中由上到下的feature map。

```python
# Get the index of feature
def lineIndex(self, board_state):
    idx0 = 0
    idx1 = 0
    idx2 = 0
    idx3 = 0
    for i in range(3):
        for j in range(2):
            idx0 = 16*idx0 + board_state[4*i + j]
            idx1 = 16*idx1 + board_state[4*i + (j+1)]
    for i in range(4):
        idx2 = 16*idx2 + board_state[4*i + 2]
        idx3 = 16*idx3 + board_state[4*i + 3]
        if i>=2:
            idx2 = 16*idx2 + board_state[4*i + 1]
            idx3 = 16*idx3 + board_state[4*i + 2]
    return idx0, idx1, idx2, idx3
```

Figure 8: Code Snippet for 6-tuple

Fig.9則是算出每一個state相對應的expected value。它會將board旋轉或者轉置，再將每一個處理過後的board weight都加起來變成expected value。

```python
# Get the expected value of state
def lineValue(self, board_state):
    value = 0.0
    for i in range(8):
        board = copy.copy(board_state)
        if (i >= 4):
            board.transpose()
        board.rotate(i)
        idx0, idx1, idx2, idx3 = self.lineIndex(board)
        value += self.net[0][idx0] + self.net[1][idx1] + self.net[2][idx2] + self.net[3][idx3]
    return value
```

Figure 9: Code Snippet for Getting Expected Value

4.3 Agent

Fig.10中take_action function裡面的for Loop實際上就是Fig.4中的EVALUATE function。而整段take_action function其實就是在做Fig.4裡LEARN EVALUATION function中的第一行，用來拿到最佳的Action。

```
class player(weight_agent):

    def __init__(self, options = ""):
        super().__init__("name=weight role=player " + options)
        return

    def __str__(self):
        return 'Player\'s Turn'

    def take_action(self, state):
        expValues = []
        rewards = []
        for op in range(4):
            tmpBoard = board(state)
            # get reward of afterstate
            rewards.append(tmpBoard.slide(op))
            if rewards[-1] == -1:
                # When the action is not allowed (reward==-1),
                # it is impossible to take the action
                expValues.append(-float("inf"))
            else:
                expValues.append(rewards[-1] + self.lineValue(tmpBoard))
        if max(rewards) == -1:
            # if all the reward==-1,
            # then gameover
            return True, action()
        best_move = np.argmax(expValues)
        return False, action.slide(best_move)
```

Figure 10: Code Snippet for Player Agent

4.4 Update Weight

Fig.11中的前兩行是在將整個episode的record整個反轉，這樣才能方便我們backward update weight。

best_action function一樣是在做Fig.4裡LEARN EVALUATION function中的第一行，用來拿到最佳的Action。

for Loop則是依序更新weight，最後一個state的weight直接更新為0，其餘state的作法如同Fig.4裡的LEARN EVALUATION function。可以比照我的註解以及pseudo code裡的notation。

```
# Close episode and Update weight
def close_episode(self, ep, flag = ""):
    episode = ep[2:].copy()
    episode.reverse()

    def best_action(state):  # Return the best action
        expValues = []
        rewards = []
        for op in range(4):
            tmpBoard = copy.copy(state)
            rewards.append(tmpBoard.slide(op))  # get the reward of afterstate
            if rewards[-1] == -1:
                # When the action is not allowed (reward==-1),
                # it is impossible to take the action
                expValues.append(-float("inf"))
            else:
                expValues.append(rewards[-1] + self.lineValue(tmpBoard))
        best_move = np.argmax(expValues)
        return best_move, rewards[best_move]

    for idx in range(1, len(episode), 2):
        if idx == 1:  # Update the last state as 0
            idx0, idx1, idx2, idx3 = self.lineIndex(episode[2][0])
            self.net[0][idx0] = 0
            self.net[1][idx1] = 0
            self.net[2][idx2] = 0
            self.net[3][idx3] = 0
            continue
        sPrime = copy.copy(episode[idx][0])  # State s'
        sPrime2 = copy.copy(episode[idx-1][0])  # State s''
        tmpBoard = copy.copy(sPrime2)
        actionNext, rewardNext = best_action(tmpBoard)  # best action and reward at State s''
        tmpBoard.slide(actionNext)
        sPrime2Next = copy.copy(tmpBoard)  # State s'(next)
        value = rewardNext + self.lineValue(sPrime2Next) - self.lineValue(sPrime)
        self.updateLineValue(board_state=sPrime, value=value)
    return
```

Figure 11: Code Snippet for Closing Episode

Fig.12是更新weight的方法，一樣要先將state拆成4個index再將從Fig.11得到的更新值丟進這一個function就可以更新weight了。

```python
# Update Weight
def updateLineValue(self, board_state, value):
    idx0, idx1, idx2, idx3 = self.lineIndex(board_state)
    self.net[0][idx0] += (self.alpha) * (value)
    self.net[1][idx1] += (self.alpha) * (value)
    self.net[2][idx2] += (self.alpha) * (value)
    self.net[3][idx3] += (self.alpha) * (value)
    return
```

Figure 12: Code Snippet for Updating Weight

5 Result

因為來不及在deadline以前跑完100000筆Episodes，所以我先放目前跑到的結果(69000筆)。

```
69000   avg = 35182, max = 106264, ops = 3343 (1718186427)
        32      100.0%  (0.1%)
        128     99.9%   (0.1%)
        256     99.8%   (1.0%)
        512     98.8%   (6.3%)
        1024    92.5%   (20.3%)
        2048    72.2%   (50.2%)
        4096    22.0%   (21.6%)
        8192    0.4%    (0.4%)
```

Figure 13: 69000 Episodes Result

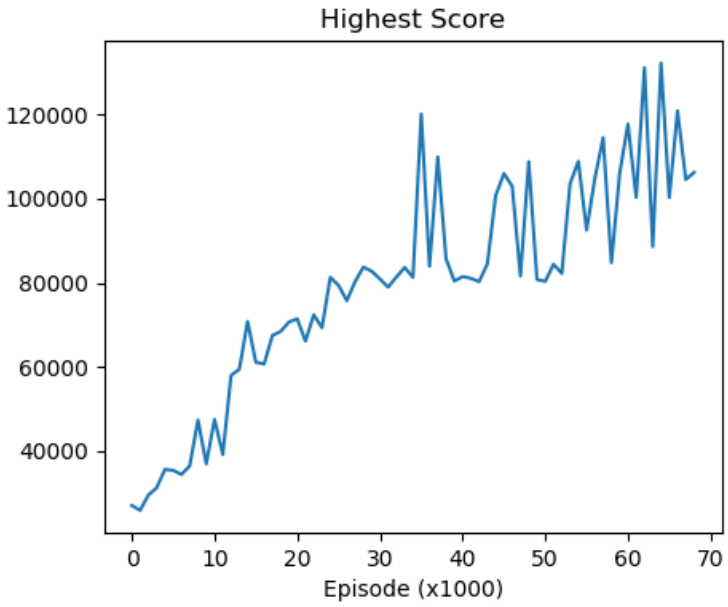

Figure 14: 69000 Episodes Highest Score

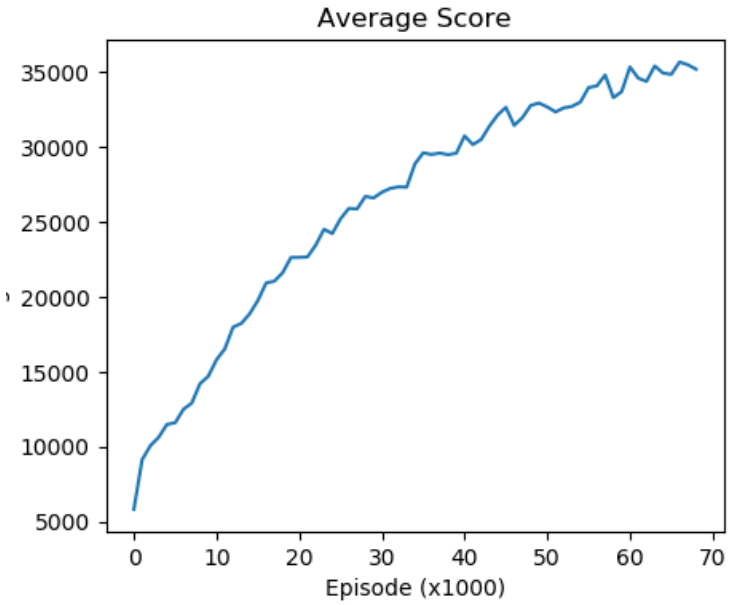

Figure 15: 69000 Episodes Average Score