Artificial-Intelligence

homework3

0816023 張紳濡

1 Introduction

使用 minimax, Expectimax, Qlearning, DQN 進行 pacman Agent 訓練

2 Minimax

```
def Minimax(agent, depth, gameState):
    if gameState.isLose() or gameState.isWin() or depth == self.depth:
        return self.evaluationFunction(gameState)
    if agent == 0:
        tmp = float(-1e9)
        for State in gameState.getLegalActions(agent):
            tmp = max(Minimax(1, depth, gameState.getNextState(agent, State)),tmp)
        return tmp
    else:
        nextagent = agent + 1
        if gameState.getNumAgents() == nextagent:
            nextagent = 0
            depth += 1
        tmp = float(1e9)
        for State in gameState.getLegalActions(agent):
            tmp = min(Minimax(nextagent, depth, gameState.getNextState(agent, State)),tmp)
        return tmp
val = float(-1e9)
for State in gameState.getLegalActions(0):
    tmp = Minimax(1, 0, gameState.getNextState(0, State))
    if tmp > val:
        val = tmp
        next = State
return next
```

先判斷是否為中止狀態，若不是判斷 agent number 是否為 0(為 pacman) 是的話取 max，不是的話 (ghost) 取 min

3 Expectimax

```python
def Expectimax(agent, depth, gameState):
    if gameState.isLose() or gameState.isWin() or depth == self.depth:
        return self.evaluationFunction(gameState)
    if agent == 0:
        tmp = float(-1e9)
        for State in gameState.getLegalActions(agent):
            tmp = max(Expectimax(1, depth, gameState.getNextState(agent, State)),tmp)
        return tmp
    else:
        nextagent = agent + 1
        if gameState.getNumAgents() == nextagent:
            nextagent = 0
            depth += 1
        tmp = 0
        for State in gameState.getLegalActions(agent):
            tmp += Expectimax(nextagent, depth, gameState.getNextState(agent, State))
        return tmp/float(len(gameState.getLegalActions(agent)))
val = float(-1e9)
for State in gameState.getLegalActions(0):
    tmp = Expectimax(1, 0, gameState.getNextState(0, State))
    if tmp > val:
        val = tmp
        next = State
return next
```

與 minmax 大致上相通，但在 ghost 行動那邊因為 ghost 行動是 random 不是 optimal，所以在決定 min 時改成以 child node 的平均值代替

4 Q-learning

4.1 Value Iteration

```python
def computeQValueFromValues(self, state, action):
    """
      Compute the Q-value of action in state from the
      value function stored in self.values.
    """
    "*** YOUR CODE HERE ***"
    # Begin your code
    val = 0
    for trans_prob in self.mdp.getTransitionStatesAndProbs(state, action):
        t,p = trans_prob
        reward = self.mdp.getReward(state, action, t)
        gamma = self.discount
        val += p*(reward+gamma * self.getValue(t))
    return val
```

計算 value 的公式為 $\sum \mathrm{p} * (\mathrm{reward} + \mathrm{gamma} * \mathrm{value})$

```python
    if (self.mdp.isTerminal(state)):
        return None
    val = util.Counter()
    for i in self.mdp.getPossibleActions(state):
        val[i] = self.computeQValueFromValues(state, i)
    return val.argMax()
```

在決定行動時，則計算整個矩陣內最大值得行動

4.2 Q-learning

```python
def computeValueFromQValues(self, state):
    """
      Returns max_action Q(state,action)
      where the max is over legal actions.  Note that if
      there are no legal actions, which is the case at the
      terminal state, you should return a value of 0.0.
    """
    "*** YOUR CODE HERE ***"
    # Begin your code
    values = []
    for action in self.getLegalActions(state):
        values.append(self.getQValue(state, action))
    if len(values) != 0:
        tmp = float(-1e9)
        for val in values:
            tmp = max(val,tmp)
        return tmp
    else:
        return 0.0
```

一樣計算最大值

4.3 epsilon-greedy

```python
if (util.flipCoin(self.epsilon)):
    action = random.choice(legalActions)
else:
    action = self.computeActionFromQValues(state)
return action
# End your code
```

在決定行動時加入隨機值

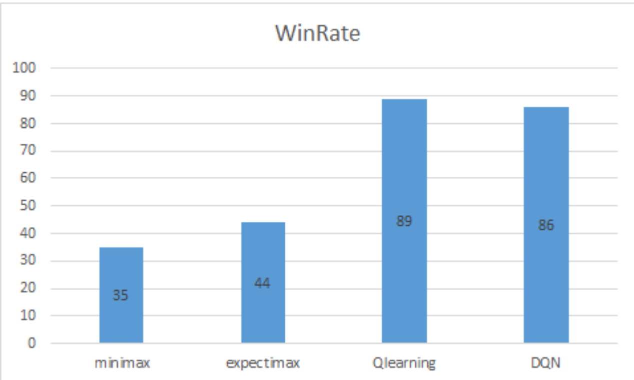
4.4 Approximate Q-learning

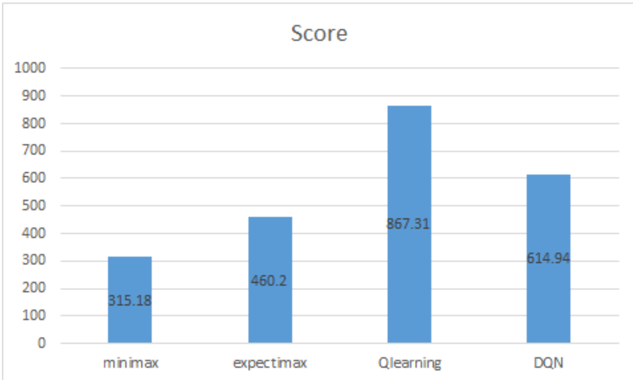
```
def update(self, state, action, nextState, reward):
    """
      Should update your weights based on transition
    """
    "*** YOUR CODE HERE ***"
    # Begin your code
    correction = (reward + self.discount * self.computeValueFromQValues(nextState)) - self.getQValue(state, action)
    features = self.featExtractor.getFeatures(state, action)

    for feature in features:
        self.weights[feature] += self.alpha*correction *features[feature]
    # End your code
```

更新的公式為 $w + \alpha[\text{correction}] * \text{featurre}$, $\text{correction} = \text{reward} + \text{gamma} * \text{value} - \text{Qvalue}$

5 Compare

這邊都使用 smallClassic 進行測試，DQN 與 Q-Learnig 的 train 用 10000,test 的數量用 100,minimax,expecitmax depth 用 3





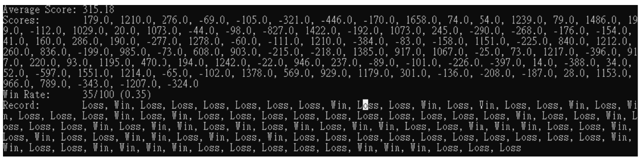


Figure 1: minimax

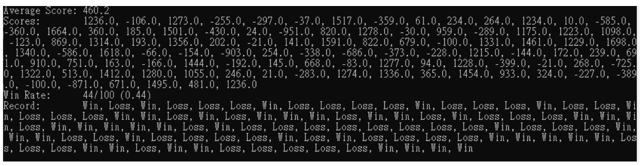


Figure 2: expectimax

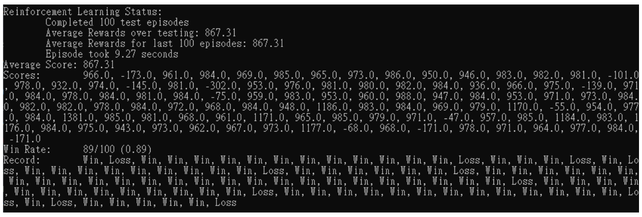


Figure 3: Q-learning

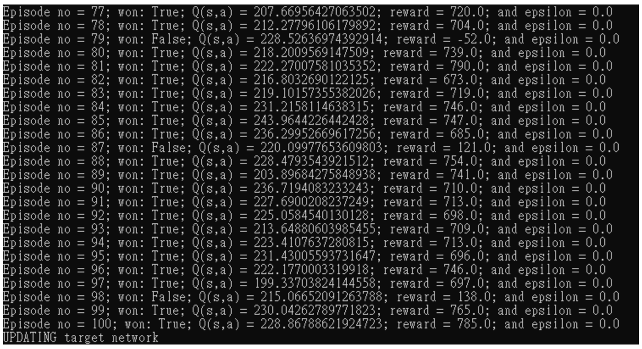


Figure 4: DQN-1

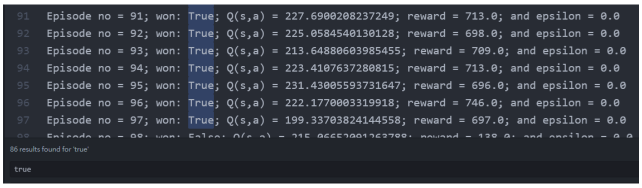


Figure 5: DQN-2

可以發現 Qlearning 與 DQN 的表現明顯比 minimax 與 expectimax 好

6 Problem I meet

- 理論上 DQN 應該要比 Qlearning 的結果好，但實驗結果並非如此，可能需要嘗試更多訓練參數
- 要把數學公式實作，要找到對應參數有點麻煩
- minimax 與 expectimax 跑起來有點花時間，原本想做 1000 筆的比較但跑到一半電腦盪了 QAQ

7 Code

All code can be find in
https://github.com/Sakuya0229/Artificial-Intelligence-HW3