PSG College of Technology

15Z017 Machine Learning

Reinforcement Learning (Batch 8)

COMPUTER SCIENCE & ENGINEERING

PSG COLLEGE OF TECHNOLOGY

(Autonomous Institution)

Done by

Deepthishree G S 18Z312

Harshini S 18Z320

Iswaryaa G P 18Z323

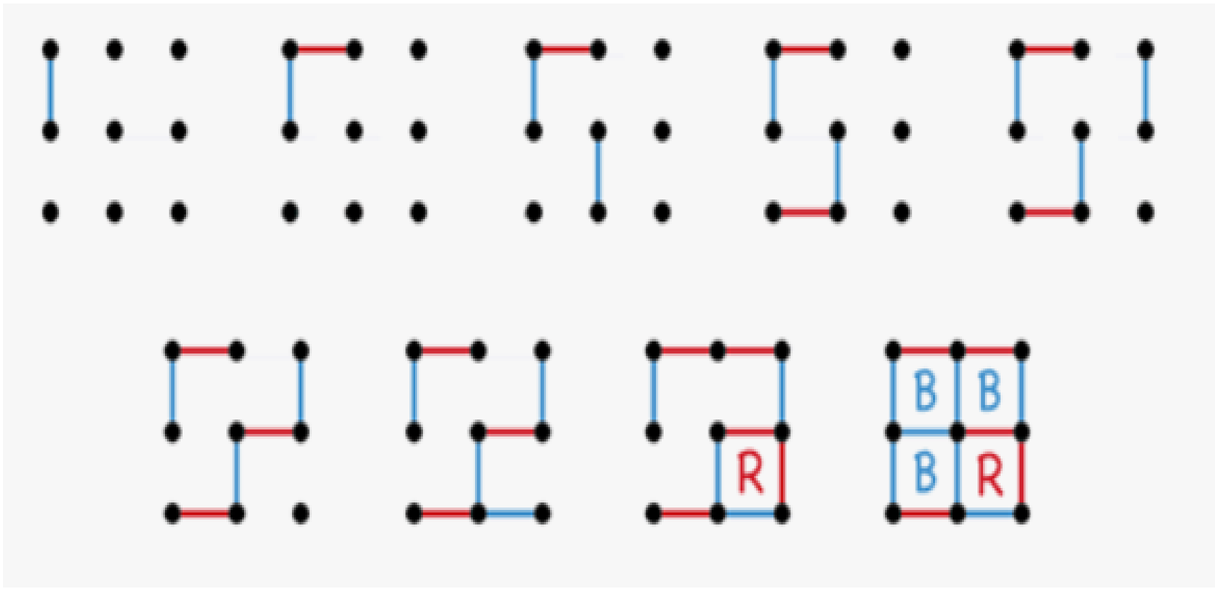
Janani R 18Z324

Swetha M 18Z360

DOTS AND BOXES
REINFORCEMENT LEARNING

Introduction

Reinforcement Learning is an area of machine learning in which suitable actions will be taken by the agent based on the rewards given at each stage by the environment. The problem statement is to make the agent learn the playing of a simple pen and pencil game **Dots and Boxes** using Q-learning algorithm. The goal is to make the agent learn the game first by playing around 10K games with itself after which its performance is gauged. The major goal is to deepen the understanding of Q Learning Algorithm by building it from scratch.



Tools Used

- IDE: Jupyter Notebook
- Language: Python
- Libraries: BitArray, Numpy, Random, Json (to save the table)

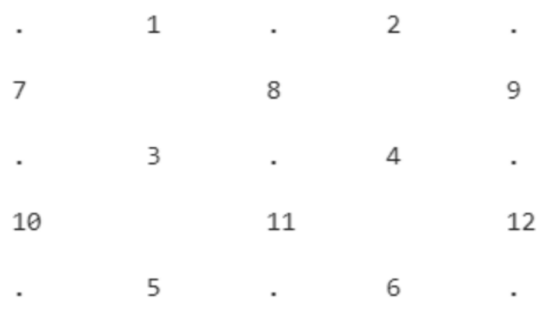
Problem Objective

The problem objective is

- To *train an agent* to play a simple pen and pencil game Dots and Boxes using a reinforcement learning (***Q-learning Algorithm***)
- To make the agent learn the game from the *rewards and penalties*
- To check its *performance* by playing against human and random agents
- To *deepen the understanding* of Q Learning algorithm by building it from scratch

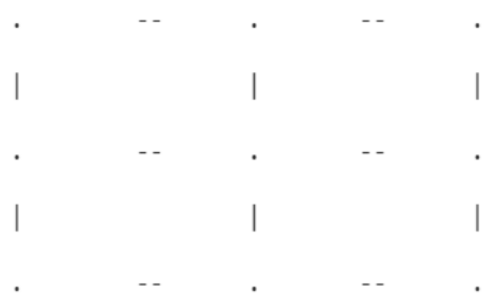
Assumptions

- 3x3 grid with 4 boxes.
- 2 player game
- Player with the greater number of boxes wins. There is a possibility of tie (unlike a few variations of the game)
- Assign Line numbers from 1 to 12 to accept input and represent the board state.



```
.      1      .      2      .
7             8             9
.      3      .      4      .
10            11            12
.      5      .      6      .
```

State: 000000000000



```
.     --     .     --     .
|            |            |
.     --     .     --     .
|            |            |
.     --     .     --     .
```

State: 111111111111

- Board state is represented as ***a bit array***.

  Eg: 000000000000 is the initial state and 010101010101 is an intermediary state with 6 lines drawn

- ***Number of states = 2 power 12 = 4096***
- Action is a ***number from 1 to 12*** indicating legal moves from a state.

  Eg: in the above mentioned state 1, 12 possible actions {1, 2, 3, ... 12 } and state 2, only six possible actions {1, 3, 5, 7, 9, 11}

Problem Design

- ***Task*****:** Play Dots and Boxes
- ***Performance*****:** Win-rate against itself, humans and a random agent
- ***Training Experience*****:** Play 10000 Games against itself
- ***Target Function*****:** Policy : State -> Action
- ***Target Function representation*****:** Q Table

  Q(state, action)=ImmediateReward + DR * V(NextState)

where DR is discount rate, V is a value function which maximises QValue for each state, ImmediateReward, NextState are indicated by environment

Policy is to choose the action with maximum Q value for the given state

Design Choices

- ***Learning Rate***=0.2 [Each update of QTable affects learning slowly]
- ***Discount Rate***=0.8 [Importance of future rewards compared to immediate rewards]
- ***Reward:*** 200 for winner (*long term reward*), 100 for each box filled (*short term reward* - only 4 boxes need to be filled), 50 for tie (more common in this game)
- ***Penalty***: 200 for loser, 100 for each box opponent fills
- ***Score***: 2 for each box completed
- ***Exploration*** (50 %) using random moves and visiting states with least visit count and then ***Exploitation*** using Q Table

Techniques used

- Sequences of moves are updated in **reverse chronological order**. This improves training efficiency. [*Reference: Tom Mitchell Book Chapter 13*]
- Q TABLE usually has *states as rows and actions as columns* and the entries are the corresponding Q values. Since in a game, only legal moves are the actions, the number of legal moves for each state is different. So, a **hash table** is constructed with ***keys as states and values as action:qvalue pairs for QTABLE***.
- A **visit table** maintains the count of visits for each state. This help us explore by visiting the states not visited often (An assumption for Q Learning *Convergence* in *TOM MITCHELL book* is '*Each state should be visited infinite times*.')
- **Discount rate** is used to strike a balance between immediate and future rewards. The ultimate aim is to maximise the discounted cumulative reward.
- **Learning rate** is used to update the Q Table slowly and not to completely alter whatever is learnt till now.

Challenges Faced

- ***4x4 and bigger board states*** had many board states (2 power 24) . This made it infeasible as Cloud RAM ran out of space. So, we had to stick with 3x3 board
- To ***decide the right reward, penalty, learning rate, discount rate*** etc. It took some amount of trial and error.
- The ***user interface of the game was not attractive***. Sometimes, that reduced the performance of the human players.
- The agent showed different performances which depended on who made the first move. When it ***didn't make the first move, most games got tied***.

Training details

- The agent played **10000** games with itself.
- Training involved playing a game and at the end of the game, updating the QTable in *reverse chronological order* with rewards and penalties properly set.
- The moves for the first 1000 games were *random (Exploration)*
- The moves for the next 4000 games were based on the visit count of the states. (Exploration). This made sure the *less visited states were visited*. (simple agent)
- The moves for the next 5000 games made *exploitation of the Q Learning Table*.
- The training took about ***15 minutes***. The state of the Q Table was ***saved and backed up*** in a JSON file.

Performance of the agent

- The agent played games against a random agent with *no intelligence*.
    - The results were *really good*.
    - Among *1000* games, the agent had an outstanding ***win rate of 0.921, lose rate of 0.015 and a fraction of 0.064 games were tied***.
- When the agent played against a human opponent (with intelligence)
    - Among *10* games, the ***agent won 0.3 of the games, lost 0.2 and tied 0.5 of the games***

Scope of Improvement

- The ***symmetry of the board*** could be used to update multiple areas of the QTable.
- The *board size of 3x3 is too small* to judge the actual performance of the Q Learning algorithm.
- Different Q Learning agents (of different performance and training) can be made to *play against each other* to improve the overall system.

VARIABLES

- TABLE : hashtable, key: state(4096), value: action-qvalue pairs // permanent for all training and test

- GAME EPISODE: we have to store, memory for player 1 and 2 {state, action, nextstate, reward}

- SCORE of each player after each box is filled -- > decide winner

- Current player = player 1 or player 2

- CURRENT board STATE

ALGORITHMS

learner:

```
- init QTABLE
- while 10000 games:
    play game
- save QTABLE
- playwithHuman
```

game:

```
- init board state
- init players as p1,p2 or p1,human
- init current player = p1
- init box [0 0 0 0]
- while not final state:
    - if currentplayer != 'human' :
        currentplayer.make_move
    - else:
        accept input
    - update currentplayer.memory
    - Check if new box formed, update player score
    - else
```

```
        -   toggle the currentplayer
        - update board state with current move
    - update QTABLE with rewards and penalties
```

update memory (player, {state, action, nextstate, reward})

```
    - for player's Memory table
        - create new row
        - add values for each column
```

update Qtable

```
    - at the end of game
    - for each player update Qtable depending on whether he is winner or loser or tie
    - for each entry in memorytable(s,a,ns,rwd) in reverse:
        - UPDATE Q value as
            Q(s,a) = (1-LR)*Q(s,a)+LR*( rwd+ DR * MAX( Q( ns,all a's )))
```

Contribution

Name	**Contribution**
Deepthishree G S	Designed the implementation for functions related to boardstate, transition, print BoardState
Harshini S	Implementation of game and rules
Iswaryaa GP	Implemented learner function and measured performance by playing against a random agent and humans.
Janani R	Implemented MakeMove, BestAction (random agent, qlearner, simple agent)
Swetha M	Inititalised, updated Memory and Qtable

Code:

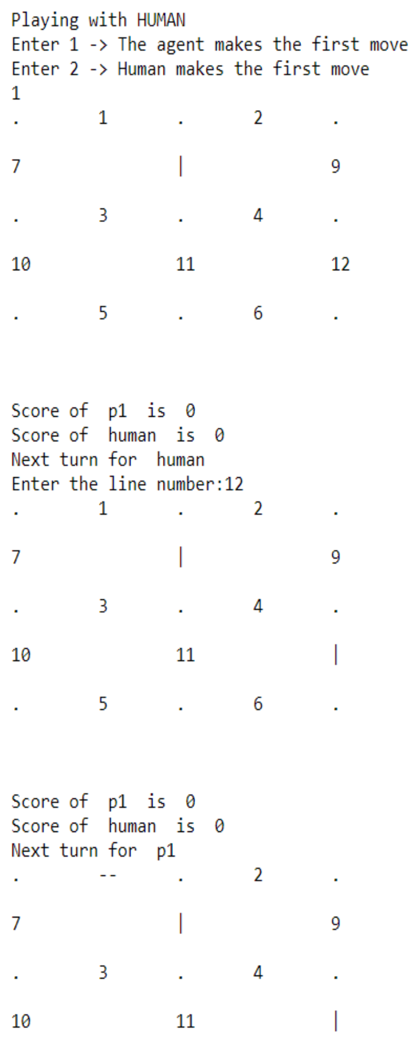
PYTHON NOTEBOOK in QLearning_for_DOTS_and_BOXES.ipynb

Github repository:

Repository

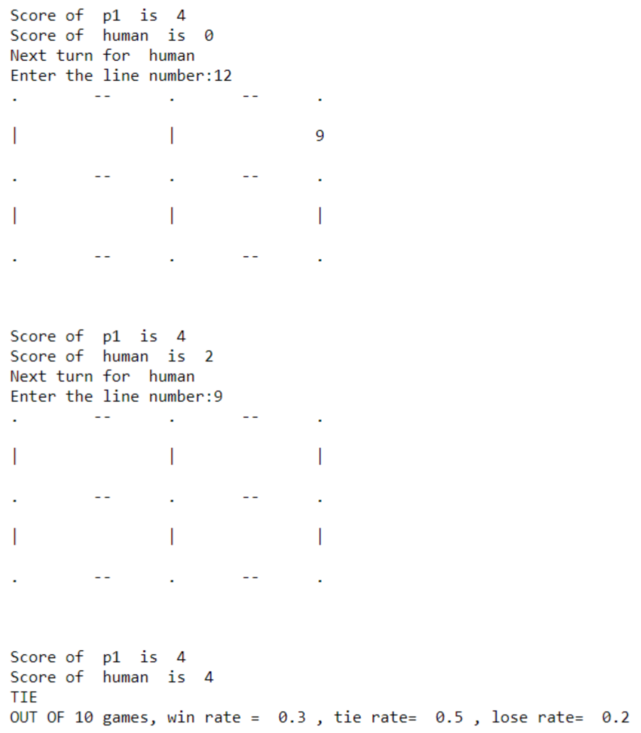
Sample Output:

Against Human:



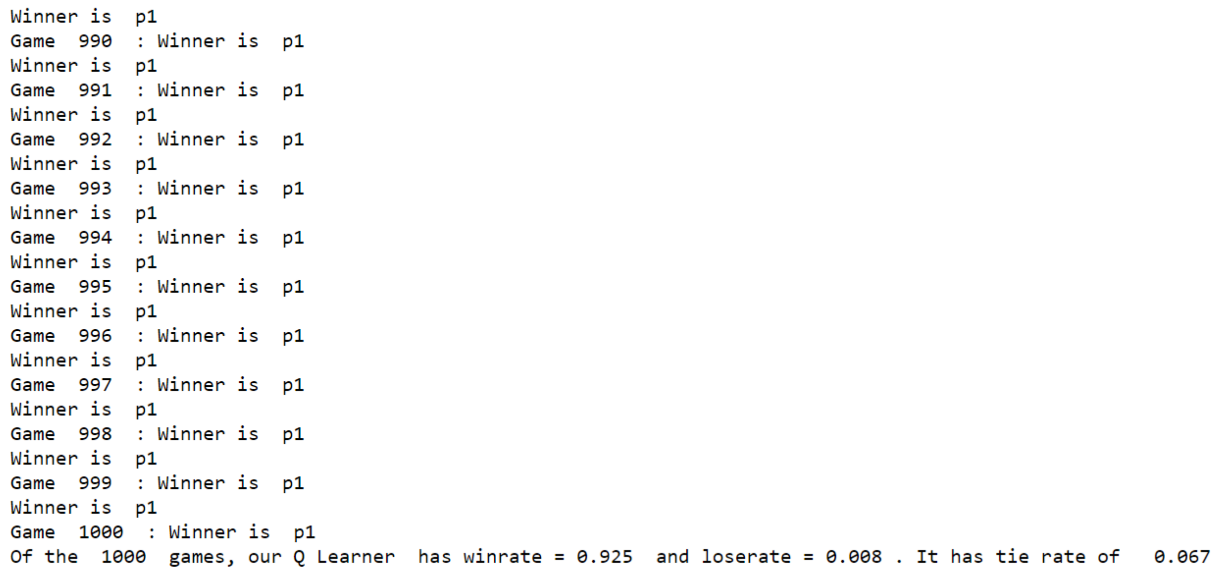
```
Playing with HUMAN
Enter 1 -> The agent makes the first move
Enter 2 -> Human makes the first move
1
.        1       .       2       .

7                |               9

.        3       .       4       .

10               11              12

.        5       .       6       .



Score of  p1  is  0
Score of  human  is  0
Next turn for  human
Enter the line number:12
.        1       .       2       .

7                |               9

.        3       .       4       .

10               11              |

.        5       .       6       .



Score of  p1  is  0
Score of  human  is  0
Next turn for  p1
.        --      .       2       .

7                |               9

.        3       .       4       .

10               11              |
```

Result:



```
Score of  p1  is  4
Score of  human  is  0
Next turn for  human
Enter the line number:12
.        --       .       --      .

|                 |               9

.        --       .       --      .

|                 |               |

.        --       .       --      .


Score of  p1  is  4
Score of  human  is  2
Next turn for  human
Enter the line number:9
.        --       .       --      .

|                 |               |

.        --       .       --      .

|                 |               |

.        --       .       --      .


Score of  p1  is  4
Score of  human  is  4
TIE
OUT OF 10 games, win rate =  0.3 , tie rate=  0.5 , lose rate=  0.2
```

Against Random Agent:



```
Winner is  p1
Game  990  : Winner is  p1
Winner is  p1
Game  991  : Winner is  p1
Winner is  p1
Game  992  : Winner is  p1
Winner is  p1
Game  993  : Winner is  p1
Winner is  p1
Game  994  : Winner is  p1
Winner is  p1
Game  995  : Winner is  p1
Winner is  p1
Game  996  : Winner is  p1
Winner is  p1
Game  997  : Winner is  p1
Winner is  p1
Game  998  : Winner is  p1
Winner is  p1
Game  999  : Winner is  p1
Winner is  p1
Game  1000  : Winner is  p1
Of the  1000  games, our Q Learner  has winrate = 0.925  and loserate = 0.008 . It has tie rate of   0.067
```

References

1. Machine Learning - Tom Mitchell (Chapter 13)
2. Q-learning for a simple board game
3. Reinforcement Learning and 9 examples of what you can do with it. | by Jair Ribeiro | Towards Data Science
4. Bellman Optimality Equation in Reinforcement Learning