Mihir Kelkar

6 November 2014

CMSC 678 Homework 7

Q Learning Grid Implementation

All the iterations for my q learning algorithm have been conducted using the epsilon greedy approach. The parameters that I used in the experiments are as follows:

Learning Rate : 0.1

Discount Rate 0.9

Sampled Episodes : 1000

Most of the elementary experiments were conducted with epsilon set to 0.1. This means that about 10% of the time the move decided by the greedy algorithm is not taken into account. This noise contributes to the random exploration probability of neighbourhood cells.

The optimal solution for the 15 * 15 grid is approximately 28 steps. In case of a purely greedy approach the learning algorithm converges somewhere in the range of 28 - 35; never actually settling on a consistent value. Eventually the newer values for q backup and converge to the optimal value once again. The behaviour maybe called periodic but I don't think it has symmetric behaviour due to random stepping.

In case of the noise induced greedy approach , the algorithm usually converges between 28 and 72 steps. The increased randomness however increases the time taken before optimal value is hit.

```
import random
from random import randint
class ValueIteration:
    def __init__(self):
        self.Actions=['N','S','E','W']
        self.gamma=0.9
        self.epsilon=1.0
        self.learning_rate = 0.1
        self.grid=[[0.0 for row in range(0,15)] for col in range(0,15)]
#        for row in range(1,15):
#            for col in range(1,15):
#                self.grid[row][col]=0

        self.maxEpisodes=100

    def Epsilon_greedy_nbr(self, row, col):
        q_max=-99999.99;
        q_max_move=[]
        possible_moves=[]

        if row>0:
            possible_moves.append('N')
            if q_max<self.grid[row-1][col]:
                q_max=self.grid[row-1][col]
                q_max_move=['N']
            elif q_max==self.grid[row-1][col]:
                q_max_move.append('N')

        if row<14:
            possible_moves.append('S')
            if q_max<self.grid[row+1][col]:
                q_max=self.grid[row+1][col]
                q_max_move=['S']
            elif q_max==self.grid[row+1][col]:
                q_max_move.append('S')

        if col>0:
            possible_moves.append('W')
            if q_max<self.grid[row][col-1]:
                q_max=self.grid[row][col-1]
                q_max_move=['W']
            elif q_max==self.grid[row][col-1]:
                q_max_move.append('W')

        if col<14:
            possible_moves.append('E')
            if q_max<self.grid[row][col+1]:
                q_max=self.grid[row][col+1]
                q_max_move=['E']
            elif q_max==self.grid[row][col+1]:
                q_max_move.append('E')

        explorationProbability=random.randint(1,10)
        if explorationProbability/10.0 > self.epsilon:
            for move in q_max_move:
                possible_moves.remove(move)

            if possible_moves==[]:
                return [random.choice(q_max_move),q_max]

            randomMove=random.choice(possible_moves)

            if randomMove=='N':
                QVal=self.grid[row-1][col]
            elif randomMove=='S':
                QVal=self.grid[row+1][col]
            elif randomMove=='E':
                QVal=self.grid[row][col+1]
            else:
                QVal=self.grid[row][col-1]
            return [randomMove,QVal]

        return [random.choice(q_max_move),q_max]


    def EpsilonGreedyLearn(self):
        episode=1
        while episode<=self.maxEpisodes:
            row=1
            col=1
            steps=0
            while True:
                nextState = self.Epsilon_greedy_nbr(row,col)
```

```
                        steps+=1

                        if nextState[0]=='N':
                            newRow= row-1
                            newcol=col
                        elif nextState[0]=='S':
                            newRow=row+1
                            newcol=col
                        elif nextState[0]=='E':
                            newcol=col+1
                            newRow=row
                        else:
                            newcol=col-1
                            newRow=row



                        if newRow<0:
                            self.grid[row][col] = self.grid[row][col] + self.learning_rate * -1
                        elif newRow>14:
                            self.grid[row][col] = self.grid[row][col] + self.learning_rate * -1
                        elif newcol<0:
                            self.grid[row][col] = self.grid[row][col] + self.learning_rate * -1
                        elif newcol>14:

                            self.grid[row][col] = self.grid[row][col] + self.learning_rate * -1

                        else:
                            if newRow == 14 and newcol==14:
                                self.grid[row][col] = self.grid[row][col] + self.learning_rate * (10 - self.grid[row][col])
                                break

                            futureMove=self.Epsilon_greedy_nbr(newRow,newcol)

                            self.grid[row][col] = self.grid[row][col] + self.learning_rate * (-1 + 
(self.gamma*futureMove[1]) - self.grid[row][col])
                            row=newRow
                            col=newcol
                    print steps
                    episode+=1

        learner=ValueIteration()
```

```
learner.EpsilonGreedyLearn()
print learner.grid
```