Machine Data and Learning

SPRING SEMESTER 2020

Team 64

Rishabh Daga (2018101015) and Anubhav Sharma (2018114007)

ASSIGNMENT 2, March 7

TASK-1

Libraries used :

- numpy
- os

Description of algorithm :

- **Value iteration algorithm**

Procedure Value_Iteration(S,A,P,R,θ)
    Inputs
        S is the set of all states
        A is the set of all actions
        P is state transition function specifying $P(s'|s,a)$
        R is a reward function $R(s,a,s')$
        θ a threshold, $\theta>0$
    Output
        $\pi[S]$ approximately optimal policy
        $V[S]$ value function
    Local
        real array $V_k[S]$ is a sequence of value functions
        action array $\pi[S]$
    assign $V_0[S]$ arbitrarily
    $k \leftarrow 0$
    repeat
        $k \leftarrow k+1$
        for each state s **do**
            $V_k[s] = max_a \sum_{s'} P(s'|s,a)\ (R(s,a,s')+\ \gamma V_{k-1}[s'])$
    until $\forall s\ |V_k[s]-V_{k-1}[s]| < \theta$
    for each state s **do**
        $\pi[s] = argmax_a \sum_{s'} P(s'|s,a)\ (R(s,a,s')+\ \gamma V_k[s'])$

23: **return** π, V_k

- ## Markov's decision process :

task_1.py

```python
    iteration = 0

    while iteration == 0 or delta >= given_delta:

        f.write("iteration={}\n".format(iteration))
        iteration += 1

        utility = numpy.copy(utility_next)

        delta = 0

        for i in range(len(utility)): #md_healthoutputs/
            for j in range(len(utility[0])): #no of arrows
                for k in range(len(utility[0][0])): #stamina

                    action = "-1"

                    dodge = recharge = shoot = -1e15

                    if i > 0:

                        # dodge
                        if k == 2:
                            dodge = dodge_reward + gamma * (0.8 * (0.8 * utility[i][min(j+1,3)][k-1] + 0.2 * utility[i][j][k-1]) + 0.2 * (0.8 * utility[i][min(j+1,3)][k-2] + 0.2 * utility[i][j][k-2]))
                        elif k == 1:
                            dodge = dodge_reward + gamma * (0.8 * utility[i][min(j+1,3)][k-1] + 0.2 * utility[i][j][k-1])

                        # recharge
                        recharge = recharge_reward + gamma * (0.8 * utility[i][j][min(k+1,2)] + 0.2 * utility[i][j][min(k,2)])

                        # shoot
                        if j > 0 and k > 0:
                            shoot = shoot_reward + gamma * (0.5 * (utility[i-1][j-1][k-1] + utility[i][j-1][k-1]))
                            if i == 1:
                                shoot += 5

                        utility_next[i][j][k] = max(dodge, recharge, shoot)

                        if max(dodge, recharge, shoot) == dodge:
                            action = "DODGE"
                        elif max(dodge, recharge, shoot) == recharge:
                            action = "RECHARGE"
                        elif max(dodge, recharge, shoot) == shoot:
                            action = "SHOOT"

                    delta = max(delta, utility[i][j][k] - utility_next[i][j][k])

                    f.write("({0},{1},{2}):{3}=[{4:.3f}]\n".format(i,j,k,action,round(utility_next[i][j][k],3)))
        f.write("\n\n")
```

Final observation:

The policy is reasonably obvious, it's better to have more stamina and arrows, and that the Dragon has less health. The actions (shown below) are quite close to the obvious greedy policy (e.g. things like - shoot if you have arrows and dragon is close to dying, dodge if you need arrows or if you have lesser stamina and the dragon is still not close to dying and recharge definitely when you have 0 stamina) since the value function of the game is very smooth.

iteration=118

(0,0,0):-1=[0.000]

(0,0,1):-1=[0.000]

(0,0,2):-1=[0.000]

(0,1,0):-1=[0.000]

(0,1,1):-1=[0.000]

(0,1,2):-1=[0.000]

(0,2,0):-1=[0.000]

(0,2,1):-1=[0.000]

(0,2,2):-1=[0.000]

(0,3,0):-1=[0.000]

(0,3,1):-1=[0.000]

(0,3,2):-1=[0.000]

(1,0,0):RECHARGE=[-85.180]

(1,0,1):DODGE=[-73.629]

(1,0,2):DODGE=[-64.272]

(1,1,0):RECHARGE=[-59.045]

(1,1,1):SHOOT=[-47.164]

(1,1,2):SHOOT=[-41.446]

(1,2,0):RECHARGE=[-46.269]

(1,2,1):SHOOT=[-34.227]

(1,2,2):SHOOT=[-28.346]

(1,3,0):RECHARGE=[-40.024]

(1,3,1):SHOOT=[-27.903]

(1,3,2):SHOOT=[-21.942]

(2,0,0):RECHARGE=[-171.473]

(2,0,1):DODGE=[-161.012]

(2,0,2):DODGE=[-152.537]

(2,1,0):RECHARGE=[-147.803]

(2,1,1):DODGE=[-137.043]

(2,1,2):SHOOT=[-126.147]

(2,2,0):RECHARGE=[-123.457]

(2,2,1):SHOOT=[-112.390]

(2,2,2):SHOOT=[-101.182]

(2,3,0):RECHARGE=[-105.311]

(2,3,1):RECHARGE=[-94.015]

(2,3,2):SHOOT=[-82.575]

(3,0,0):RECHARGE=[-249.625]

(3,0,1):DODGE=[-240.151]

(3,0,2):DODGE=[-232.476]

(3,1,0):RECHARGE=[-228.188]

(3,1,1):DODGE=[-218.443]

(3,1,2):SHOOT=[-208.575]

(3,2,0):RECHARGE=[-206.139]

(3,2,1):RECHARGE=[-196.116]

(3,2,2):SHOOT=[-185.966]

(3,3,0):RECHARGE=[-183.460]

(3,3,1):RECHARGE=[-173.150]

(3,3,2):SHOOT=[-162.710]

(4,0,0):RECHARGE=[-320.400]

(4,0,1):DODGE=[-311.820]

(4,0,2):DODGE=[-304.870]

(4,1,0):RECHARGE=[-300.987]

(4,1,1):DODGE=[-292.162]

(4,1,2):SHOOT=[-283.225]

(4,2,0):RECHARGE=[-281.019]

(4,2,1):DODGE=[-271.941]

(4,2,2):SHOOT=[-262.749]

(4,3,0):RECHARGE=[-260.480]

(4,3,1):RECHARGE=[-251.143]

(4,3,2):SHOOT=[-241.688]

TASK-2

Final Observations

Subtask 1: New Step Rewards

The Convergence of this function is faster than was before (in task 1), the lowered step costs and the **favored shoot action** allows the model to learn the kill technique much faster, which it later slowly changes to better action combinations involving dodge and recharge, improving the score further. There is more incentive to be greedy and shoot at the beginning.

iteration=99
(0,0,0):-1=[0.000]
(0,0,1):-1=[0.000]
(0,0,2):-1=[0.000]
(0,1,0):-1=[0.000]
(0,1,1):-1=[0.000]
(0,1,2):-1=[0.000]
(0,2,0):-1=[0.000]
(0,2,1):-1=[0.000]
(0,2,2):-1=[0.000]
(0,3,0):-1=[0.000]
(0,3,1):-1=[0.000]
(0,3,2):-1=[0.000]

(1,0,0):RECHARGE=[-10.317]

(1,0,1):DODGE=[-7.291]

(1,0,2):DODGE=[-4.839]

(1,1,0):RECHARGE=[-3.470]

(1,1,1):SHOOT=[-0.357]

(1,1,2):SHOOT=[1.141]

(1,2,0):RECHARGE=[-0.123]

(1,2,1):SHOOT=[3.032]

(1,2,2):SHOOT=[4.573]

(1,3,0):RECHARGE=[1.514]

(1,3,1):SHOOT=[4.689]

(1,3,2):SHOOT=[6.251]

(2,0,0):RECHARGE=[-28.809]

(2,0,1):DODGE=[-26.016]

(2,0,2):DODGE=[-23.754]

(2,1,0):RECHARGE=[-22.490]

(2,1,1):SHOOT=[-19.617]

(2,1,2):SHOOT=[-16.737]

(2,2,0):RECHARGE=[-16.054]

(2,2,1):SHOOT=[-13.100]

(2,2,2):SHOOT=[-10.137]

(2,3,0):RECHARGE=[-11.272]

(2,3,1):SHOOT=[-8.257]

(2,3,2):SHOOT=[-5.233]

(3,0,0):RECHARGE=[-45.556]

(3,0,1):DODGE=[-42.975]

(3,0,2):DODGE=[-40.884]

(3,1,0):RECHARGE=[-39.716]

(3,1,1):SHOOT=[-37.061]

(3,1,2):SHOOT=[-34.400]

(3,2,0):RECHARGE=[-33.772]

(3,2,1):SHOOT=[-31.042]

(3,2,2):SHOOT=[-28.306]

(3,3,0):RECHARGE=[-27.720]

(3,3,1):SHOOT=[-24.914]

(3,3,2):SHOOT=[-22.100]

(4,0,0):RECHARGE=[-60.717]

(4,0,1):DODGE=[-58.328]

(4,0,2):DODGE=[-56.393]

(4,1,0):RECHARGE=[-55.312]

(4,1,1):SHOOT=[-52.855]

(4,1,2):SHOOT=[-50.395]

(4,2,0):RECHARGE=[-49.815]

(4,2,1):SHOOT=[-47.288]

(4,2,2):SHOOT=[-44.758]

(4,3,0):RECHARGE=[-44.223]

(4,3,1):SHOOT=[-41.625]

(4,3,2):SHOOT=[-39.023]

Subtask 2: The High Future Discount

The **low discount factor as compared to Part 1 lowered the incentive** to look and try to get the big reward (killing the dragon) in the future. The agent will only look 1 move deep and only try if the dragon has health and it has both arrows and stamina, otherwise it will give up. All moves / futures look relatively equal.

iteration=4

(0,0,0):-1=[0.000]

(0,0,1):-1=[0.000]

(0,0,2):-1=[0.000]

(0,1,0):-1=[0.000]

(0,1,1):-1=[0.000]

(0,1,2):-1=[0.000]

(0,2,0):-1=[0.000]

(0,2,1):-1=[0.000]

(0,2,2):-1=[0.000]

(0,3,0):-1=[0.000]

(0,3,1):-1=[0.000]

(0,3,2):-1=[0.000]

(1,0,0):RECHARGE=[-2.775]

(1,0,1):DODGE=[-2.744]

(1,0,2):DODGE=[-2.442]

(1,1,0):RECHARGE=[-2.358]

(1,1,1):SHOOT=[2.361]

(1,1,2):SHOOT=[2.363]

(1,2,0):RECHARGE=[-2.357]

(1,2,1):SHOOT=[2.382]

(1,2,2):SHOOT=[2.618]

(1,3,0):RECHARGE=[-2.357]

(1,3,1):SHOOT=[2.382]

(1,3,2):SHOOT=[2.619]

(2,0,0):RECHARGE=[-2.778]

(2,0,1):DODGE=[-2.778]

(2,0,2):DODGE=[-2.778]

(2,1,0):RECHARGE=[-2.778]

(2,1,1):DODGE=[-2.778]

(2,1,2):SHOOT=[-2.776]

(2,2,0):RECHARGE=[-2.776]

(2,2,1):RECHARGE=[-2.757]

(2,2,2):SHOOT=[-2.521]

(2,3,0):RECHARGE=[-2.776]

(2,3,1):RECHARGE=[-2.757]

(2,3,2):SHOOT=[-2.519]

(3,0,0):RECHARGE=[-2.778]

(3,0,1):DODGE=[-2.778]

(3,0,2):DODGE=[-2.778]

(3,1,0):RECHARGE=[-2.778]

(3,1,1):DODGE=[-2.778]

(3,1,2):DODGE=[-2.778]

(3,2,0):RECHARGE=[-2.778]

(3,2,1):DODGE=[-2.778]

(3,2,2):DODGE=[-2.778]

(3,3,0):RECHARGE=[-2.778]

(3,3,1):RECHARGE=[-2.778]

(3,3,2):SHOOT=[-2.777]

(4,0,0):RECHARGE=[-2.778]

(4,0,1):DODGE=[-2.778]

(4,0,2):DODGE=[-2.778]

(4,1,0):RECHARGE=[-2.778]

(4,1,1):DODGE=[-2.778]

(4,1,2):DODGE=[-2.778]

(4,2,0):RECHARGE=[-2.778]

(4,2,1):DODGE=[-2.778]

(4,2,2):DODGE=[-2.778]

(4,3,0):RECHARGE=[-2.778]

(4,3,1):DODGE=[-2.778]

(4,3,2):DODGE=[-2.778]

Subtask 3: Complete Convergence

This change has almost negligible effect as compared to Subtask 2, except on the number of iterations, since the agent has already almost completely discounted the future. The policy will not change as we hone down more on the utility values as the agent himself does not value anything in the future, changing convergence (δ) to 10^{-3} to 10^{-10} are both almost equally good. It just takes a few more iterations to get there.

Decreasing bellman factor only changes number of iterations used without much change in utility value

iteration=11

(0,0,0):-1=[0.000]

(0,0,1):-1=[0.000]

(0,0,2):-1=[0.000]

(0,1,0):-1=[0.000]

(0,1,1):-1=[0.000]

(0,1,2):-1=[0.000]

(0,2,0):-1=[0.000]

(0,2,1):-1=[0.000]

(0,2,2):-1=[0.000]

(0,3,0):-1=[0.000]

(0,3,1):-1=[0.000]

(0,3,2):-1=[0.000]

(1,0,0):RECHARGE=[-2.775]

(1,0,1):DODGE=[-2.744]

(1,0,2):DODGE=[-2.442]

(1,1,0):RECHARGE=[-2.358]

(1,1,1):SHOOT=[2.361]

(1,1,2):SHOOT=[2.363]

(1,2,0):RECHARGE=[-2.357]

(1,2,1):SHOOT=[2.382]

(1,2,2):SHOOT=[2.618]

(1,3,0):RECHARGE=[-2.357]

(1,3,1):SHOOT=[2.382]

(1,3,2):SHOOT=[2.619]

(2,0,0):RECHARGE=[-2.778]

(2,0,1):DODGE=[-2.778]

(2,0,2):DODGE=[-2.778]

(2,1,0):RECHARGE=[-2.778]

(2,1,1):DODGE=[-2.778]

(2,1,2):SHOOT=[-2.776]

(2,2,0):RECHARGE=[-2.776]

(2,2,1):RECHARGE=[-2.757]

(2,2,2):SHOOT=[-2.521]

(2,3,0):RECHARGE=[-2.776]

(2,3,1):RECHARGE=[-2.757]

(2,3,2):SHOOT=[-2.519]

(3,0,0):RECHARGE=[-2.778]

(3,0,1):DODGE=[-2.778]

(3,0,2):DODGE=[-2.778]

(3,1,0):RECHARGE=[-2.778]

(3,1,1):DODGE=[-2.778]

(3,1,2):SHOOT=[-2.778]

(3,2,0):RECHARGE=[-2.778]

(3,2,1):DODGE=[-2.778]

(3,2,2):SHOOT=[-2.778]

(3,3,0):RECHARGE=[-2.778]

(3,3,1):RECHARGE=[-2.778]

(3,3,2):SHOOT=[-2.777]

(4,0,0):RECHARGE=[-2.778]

(4,0,1):DODGE=[-2.778]

(4,0,2):DODGE=[-2.778]

(4,1,0):RECHARGE=[-2.778]

(4,1,1):DODGE=[-2.778]

(4,1,2):SHOOT=[-2.778]

(4,2,0):RECHARGE=[-2.778]

(4,2,1):DODGE=[-2.778]

(4,2,2):SHOOT=[-2.778]

(4,3,0):RECHARGE=[-2.778]

(4,3,1):RECHARGE=[-2.778]

(4,3,2):SHOOT=[-2.778]