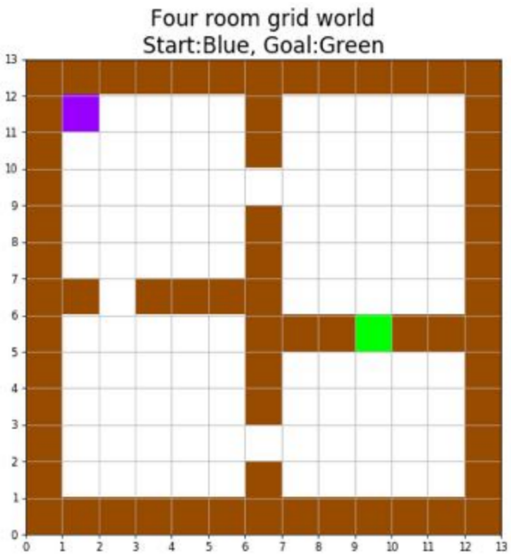
CS6700 : Reinforcement Learning
Programming Assignment-3 Report
HRL and DQN

Name: Pragnesh Rana **Roll number:** ME17S301

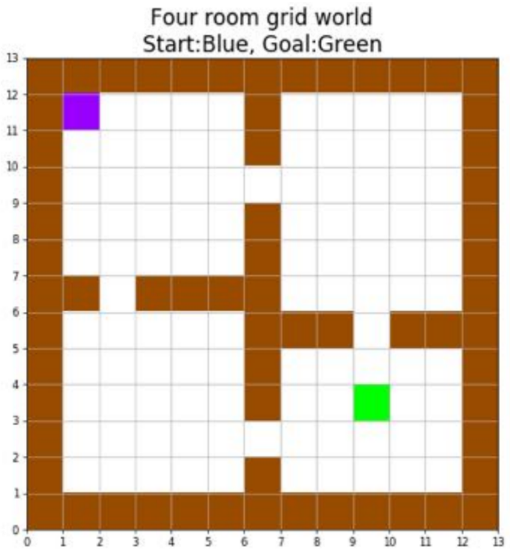
- Part-1 is on Hierarchical Reinforcement Learning
- Part-2 is on Deep Reinforcement Learning

1. Hierarchical Reinforcement Learning

1.1. Grid World of Four Rooms:



(a) Grid world with goal G1



(b) Grid world with goal G2

Figure 1: Puddle World with different colour: Grey:Start, Pink:Terminal, Middle:Puddle

In Puddle World-C, westerly wind also blowing which forces the agent to move in east direction with probability of 0.5. Suppose agent wants to go in desired state-a with probability 0.9 and reward will be 0 due to transition but due to wind it may happen that agent may end up in the another state-a' with probability 0.5 and assume such state has higher negative reward then wind misguides the agent.

1.2. Q-learning:

The goal is to reach the terminal state with highest possible reward. ran the code for 500 episodes and average steps and return has been computed for 50 runs. For each case discount rate γ is taken as 0.9, learning rare α is taken as 0.1 and exploration parameter is taken as ε= 0.3.

Q-learning update Rule: [2]

$$Q(S,A) = Q(S,A) + \alpha[r + \gamma \max_{a} Q(S,a) - Q(S,A) \tag{1}$$

Epsilon-greedy policy has been used for the exploration. Initially the epsilon was set as 0.1 but as it takes may steps to compute the terminal state. so epsilon value has been boosted to 0.3.

Figure: **??**, shows optimal policy obtained after 500 episodes. It clear from the figure-**??** that agent will never go in the puddle in world-A whereas agent will stay away from the puddle in world-B. Even though world-C has terminal state in the centre, age will try to agent avoid the puddle as much as possible and finally by taking one or least step it will find a way to terminal state.

1.3. SARSA:

SARSA stand for State-Action-Reward-State-Action. It is an off policy algorithm. The goal is to reach the terminal state with highest possible reward. Other detail and parameters setting is same as Q-learning,

SARSA update Rule: [2]

$$Q(S,A) = Q(S,A) + \alpha[r + \gamma Q(S', A') - Q(S,A) \tag{2}$$

The main difference between Q-learning and SARSA is that in Q-learning action is chosen in greedy fashion whereas in SARSA is chosen based on certain policy. [**?**]. From the the figure:**??**, it clear that SARSA also takes a while (around 200) steps in search of optimal policy which also stabilize after around 400 steps. The behaviour of return obtained in different world similar as of Q-Learning but actual values are slightly higher side in case of SARSA.

From the fig-**??**, it clear that for the world-A and B, average number of steps stabilize reduces quite rapidly whereas, in case of world-C it takes more steps around 25000+ initially which reduces to least after around 180 episodes. Average number of steps in world-c fluctuates due to less its terminal state in puddle world and due to wind.

Figure: **??**, shows optimal policy obtained after 500 episodes for SARSA. It clear from the figure-**??** agent will try stay away from the puddle in case of world-A. But for the world-B and C, as their terminal state is near and in the puddle zone they will be negative reward initially but final policy is obtain in such away that they negative reward will be minimal. From the figure-**??** and **??**, shows that SARSA learns the safe policy whereas Q-learning learns optimal policy.

1.4. SARSA-λ:

The goal is to reach the terminal state with highest possible reward which same as before with all the settings. In this case, ran the code for 500 episodes and average steps and return has been computed for 50 runs with different λ values of 0,0.3,0.5,0.9,0.99,1.0 all the obtained result are give in figure-**??**-**??**.where λ denotes eligibility parameter. Epsilon-greedy policy has been used for the exploration with epsilon value has been boosted to 0.3.

For $\lambda = 0$ it behave as as TD update rule whereas for $\lambda = 1$ it behave as Monte-Carlo update. It can be observed from the figure that average return is quite low in intimal stage but with experience it avoids the puddle. With increase in lambda value return increases. For world-A return is quite high as 12-13 with higher lambda values. Where as in case if world-B return is around 7.5 which is also fluctuates lot in both world-A and B. But in case of world-c return is quite low as terminal state in the puddle. For the same reason average steps also fluctuates lot and it takes quite long to stabilize and learn the policy. For all the cases λ = 9,0.99,1 oscillates a lot initially but performs quite well. With low λ learning will be slow as it takes larger steps to find out the goal in the puddle.

The plus point about this algorithm is, it took very less step for world-C comapred to QL and SARSA algorithm. It took around 20000-25000 to average initial step in case of SARSA and QL. where as in this algorithm it took around 10000 maximum steps which made agent to learn policy faster. Average steps further small for λ=0.9 and more.

REMARK:For case of SARSA and Q-learning obtained video is added in result folder.

2. Policy Gradient:

For contentious state and action space. policy gradient algorithm are implemented. [3]

2.1. Part:1 The environment implementation

The chakra and vishamC environment are implemented using python script. All codes are available in **rlpa2 folder**. The goal of task is to move the point to the centre quickly. The difference between two world is reward function. In chakra reward is calculated by distance 0from origin whereas, in vishamC reward is calculated by function as: $0.5x^2 + \gamma 0.5y^2$ where γ=10. All the required function are implemented.

2.2. Part:2 The Rollout Function

All the function including **rollout function is merged in the policy_gradient.py file.** The purpose of the roll out function is to obtain the action that needs to be used in implemented above two world.

2.3. Part:3 Policy Gradient Implementation

As action space is continuous, stochastic policy is used. Along with that multivariate Gaussian distribution with identity matrix is used. Mean is calculated using $\theta^T s'$. Two function $\log \pi(a|s)$ and $\Delta_\theta \log_\pi(a|s)$ are also available in policy_gradient.py file.

2.4. Answers-1 Hyper-parameter Tuning:

After implementation of chakra and VishamC world using policy gradient. parameter tuning of batch size, discounted factor γ and learning rate γ has compared with average return to iterations.

From figure-**??**, it clear that when **batch size** is high around 100 , then learning rate will be less due to up-gradation of hyper parameter θ take place after every 100 episode. Result obtained by this setting is more stable and oscillation be less. When batch size is small, observed oscillation will be larger for same setting. For **discounted factor**, less is good. For value of γ=0.9 the agent learns slower whereas for 0.1 agent learns faster for similar setting of parameters. For lower value of **learning rate** α=0.1, it observed that fluctuation will be less as low weight is given to update. fluctuation will be further less if batch size is high around 100. For high learning rate with high batch size, learning will be faster but higher learning rate with smaller batch size gives high oscillation in the result.

by such observation it is clear that setting of optimal hyper parameters gives after 50 iteration gives return of around -22 in case chakra and around -38 in case of vishamC.

The opbatined optimal hyperparameters are:

$$\alpha = 0.9, \text{batchsize=100}, \gamma = 0.9 \text{ for 100 iterations} \tag{3}$$

2.5. Answers-2: Visualization of value function for learned policy

Visualization of value function for learned policy is done using $\theta^T s'$ as mentioned above by policy gradient method. For given parameter θ and action can be selected from on multivariate normal distribution by random sampling. here s' is obtained by appending 1 to state-vector s. By setting the parameter γ=0.9, following policy π and randomly selecting start state, value state is obtained such method is similar to the Monte-carlo method. Figure:**??**-**??**, obtained after plotting 20 state values obtained by setting parameter to optimal values mentioned above.

For case of **chakra**, the state values are more near the centre which decrease away from the centre. Value are near to each other for same contour which is due to the negative distance from the centrer. For the case of **vishamC**, reward function are obtained as mentioned ahead, which shoes state values are quite skewed. Due to skewness, values are quite near to each other and they also decrease away from the centrer. change is major with Y axis compared to X-axis due to more weightage given by setting γ =10. By similar manner, state-function is also obtained as picking state and action randomly and computing discounted return followed by policy.

2.6. Answers-3: Plotting of policy trajectory after learned policy

In Policy Trajectory **for chakra**, it observed that from the figure-**??** that agent moves towards centre in diagonal direction till it reaches zero x-position. Further it only moves y-direction. Similarly after reaching the y=0 it moves towards centre travelling in x direction. Due to similar values in close region slight oscillations observed in trajectory. In case of **vishamC**, due to more weight toward y skewness is created thus agent moves in y direction after that it starts moving towards centre by moving in x-direction. Transition in such state is not oscillatory due to similar values in close region.

References:

[1] "Gridworldenvs - https://github.com/opocaj92/gridworldenvs,"

[2] *Reinforcement learning: An introduction*.

[3] "Rl course by david silver - https://www.youtube.com/watch?v=2pwv7govuf0list=rdcmucp7jmxsy2xbc3kcae
astart_radio=1t=0,"