CS6700 : Reinforcement Learning

Programming Assignment #2 Nirav Bhavsar (CS17S016)

Q1 on Puddle Gridworld :

- I have created below puddle gridworld, where start state is any one of grey box with equal probability, puddle in middle, goal is any one of pink boxes and black box is wall or boundary. In this reward for moving one step is 0, expect when moved to blue is -1, green is -2, red is -3 and to pink is $+10$. Also the environment is stochastic, so it will take correct action with 0.9 probability and any three of incorrect action with 0.1/3 probability for each action.
- There is also westerly wind blowing, which will move you one more step to east with 0.5 probability. So for example if you are in some state s and take action a and move to next state s' and your reward for that movement is suppose -1 and wind also below you away one more step to east in state s'' and suppose reward for that movement is -2 then, the enviromment for state s and action a will return you next state as s'' and immediate reward as -3. Also, assuming this can happen even if on taking action a you move on goal state s' and wind can take you to other state s'' which is not a goal state.

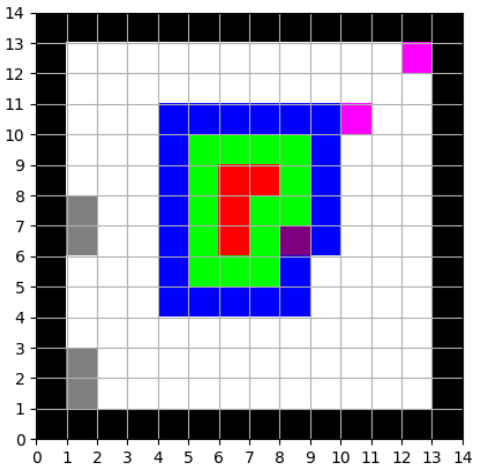


Figure 1: Puddle Gridworld: Start State: Grey, Goal: Pink, Puddle in middle

Q - Learning

- We now implement Q-learning, by considering any one of pink states as goal. I ran each problem for 500 episodes and compute the averages over 50 independent runs. In each problem discount rate γ is set to 0.9, exploration parameter ϵ to 0.1 and learning rate α to 0.1.
- While in case of problem three, Westerly wind was turned off and learning rate α was set to 0.2 and epsilon to 0.3, because i observed it will take very large number of steps initially as it will avoid going into puddle and thus will not be able to find goal. Thus by setting little higher learning rate and higher exploration parameter, will speed up learning process.
- I have below Q-learning update :

$$Q(s,a) = Q(s,a) + \alpha[r + \gamma max_{a'} Q(s',a') - Q(s,a)]$$

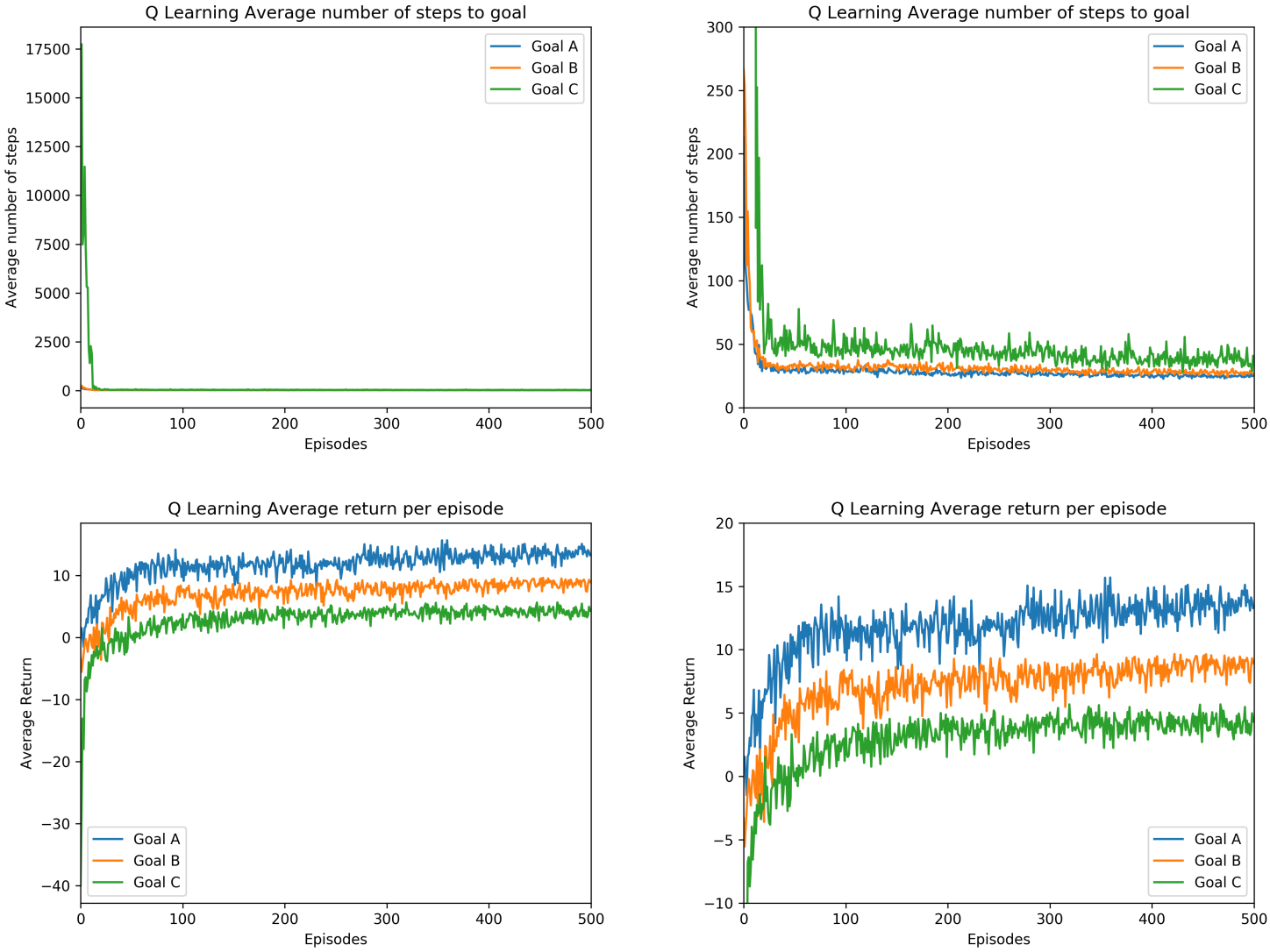
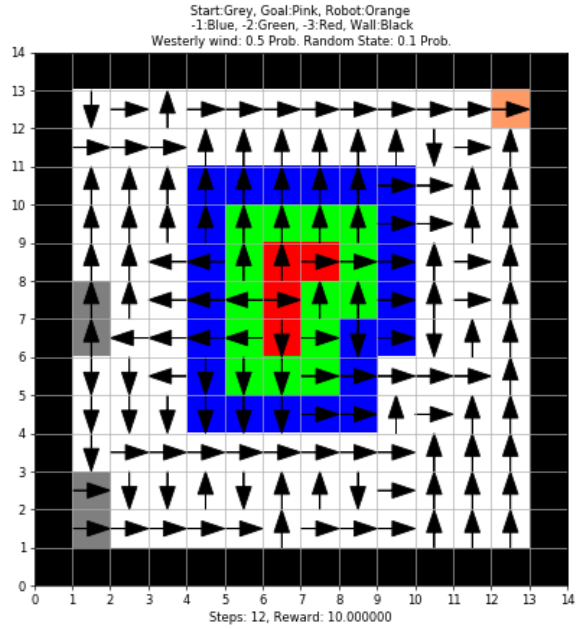
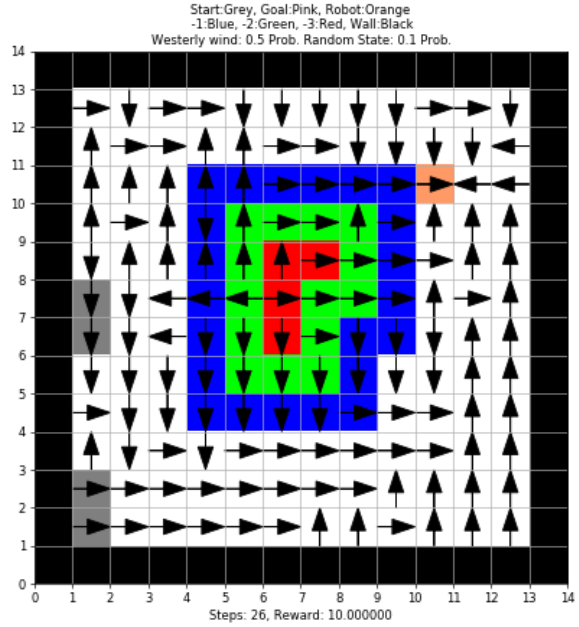


Figure 2: Q Learning: Average Steps taken to reach goal and average return per episode averaged over 50 independent runs for all three problems (Figures on right is zoomed version of figure on left by clipping y-axis).

Observation

- For initial few episodes for all problems, agent will take large number of steps (~ 250) to find and reach goal, and after learning from about 50 episodes, number of steps to goal will reduce to ~ 50. While in case of Goal C, initially number of steps to goal is very large ~ 17000, but again after learning from about 50 episodes it will get reduced drastically to ~ 70, this is because goal C is in puddle and agent will avoid going into puddle and thus taking large steps to find goal.
- Similarly intital average return per episode will be very less ~ -35, but after about 50 learning trails it will get stabilized. Also note reward for *goal A is going beyond* 10 it is because of wind, you might take one step, reach goal & get +10 & then wind takes you to east thus landing in same state gives another +10.
- We can observe that average reward for goal A is more compared to other, because goal A is far away from puddle, so agent will always try to stay away from puddle thus avoiding negative rewards, while for goal B is lesser as it is near to puddle & for goal C even lesser as it is in puddle and thus guaranteed to get negative reward, and also because some times agent will go into puddle due to exploration and also due to stochasticity in environment & wind.
- This can also be observed from optimal policies shown on next page, where for goal A agent will never go into puddle. For goal B agent will try to avoid going into puddle, but due to stochasticity in environment it will land into puddle and for goal C it will have to go into puddle atleast for one step to reach goal.
- **Note:** *I have also added video of agent playing for 10 episodes along with optimal policies shown for each state after learning from 3000 episodes for all three goals in "Videos of trained agents" folder and extra plots showing performance for each goal individually is added in"Extra Plots" folder.*





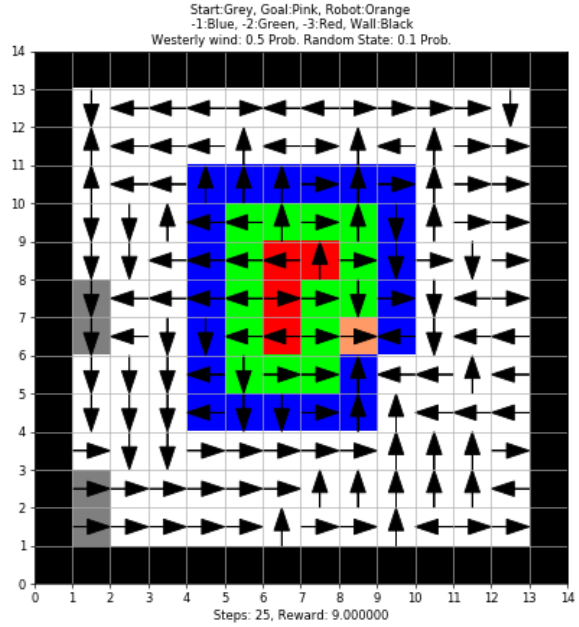


Figure 3: Q Learning: Optimal policy achieved after learning from 3000 episodes for all three problems.

Sarsa

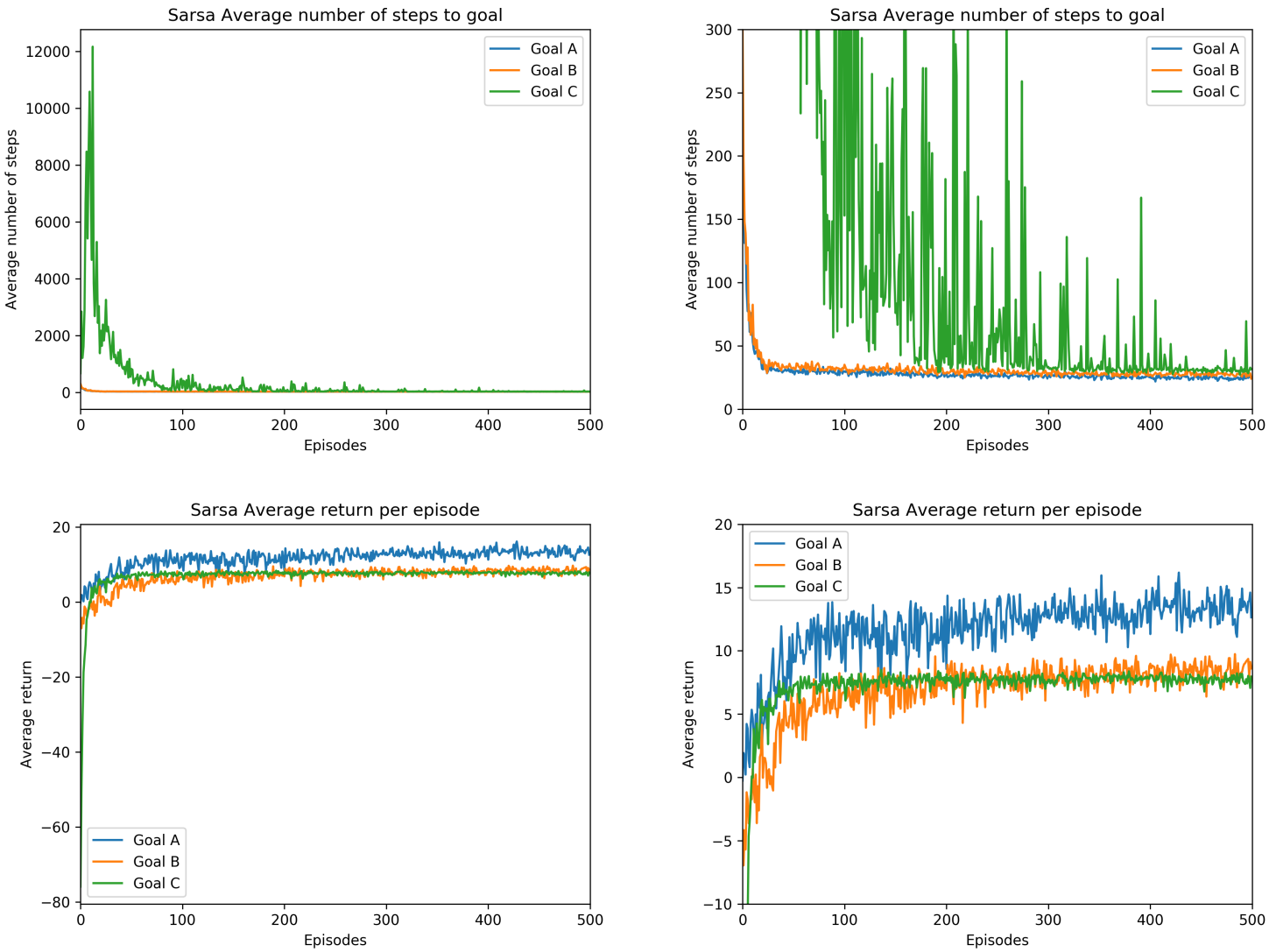


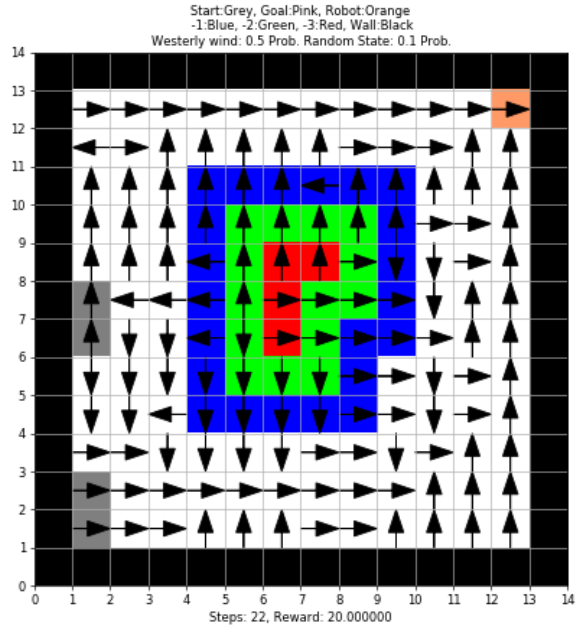
Figure 4: Sarsa: Average Steps taken to reach goal and average return per episode averaged over 50 independent runs for all three problems (Figures on right is zoomed version of figure on left by clipping y-axis).

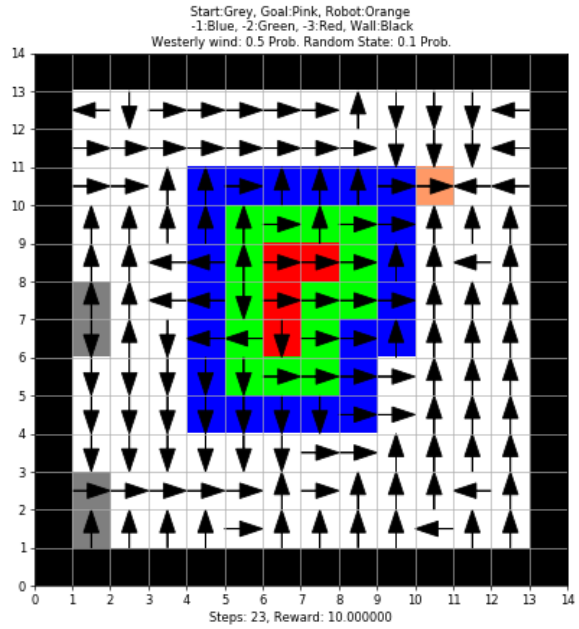
Observations :

- In case of Sarsa also initially for few episodes agent will take large number of steps (~ 250) to find and reach goal, and after learning from about 100 episodes, number of steps to goal will reduce to ~ 50. While in case of Goal C, initially number of steps to goal is very large ~ 12000, but again after learning from about 100 episodes it will get reduced drastically to ~ 200 but it will still oscillate and will stabilies only after learning from 450 episodes, this is because goal C is in puddle and agent will avoid going into puddle and thus taking large steps to find goal. Similarly, intital average return per episode will be very less ~ -70, but after about 50 learning trails it will get stabilized.

- *Q-learning will try to learn optimal policy but Sarsa will try to learn safe poilcy.* Particularly Sarsa will try to move away from puddle even if it little more steps to reach goal. Thus it can be observed that it will take little more time to stabilize compared to Q-learning. Hence we see more oscillation for goal C and it stabilizes only after ~ 450 learning trials.

- We can also observe that average reward for goal A is more compared to other, because goal A is far away from puddle, so agent will always try to stay away from puddle thus avoiding negative rewards, while for goal B & C is almost similar, this is because both are near puddle and for goal B it will take long path by staying away from puddle and for goal C it will also try to stay away from puddle but eventually it will have to enter puddle and suffer negative reward to reach goal. This can also be observed from optimal policies shown below and from video files as mentioned above.





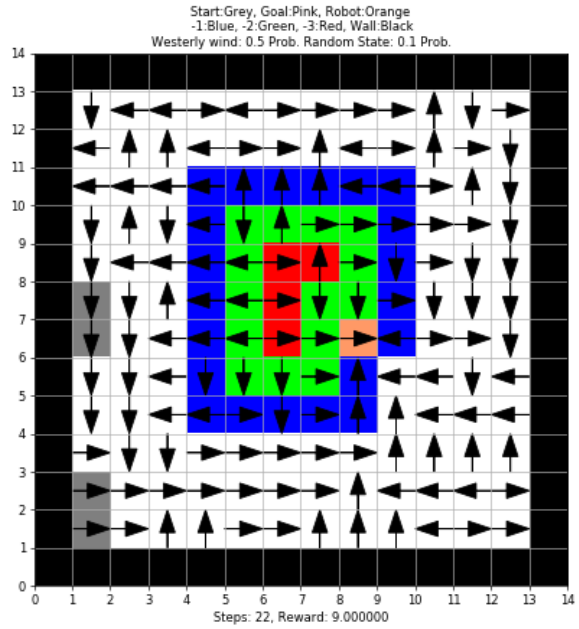
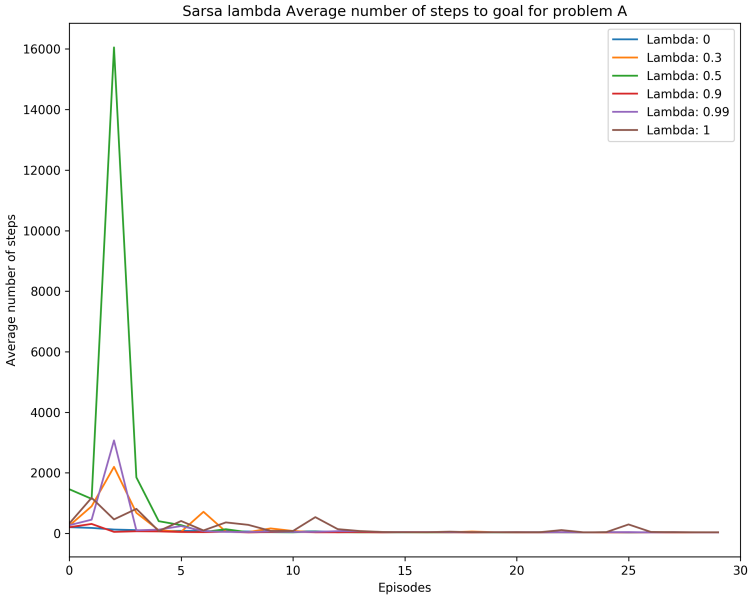
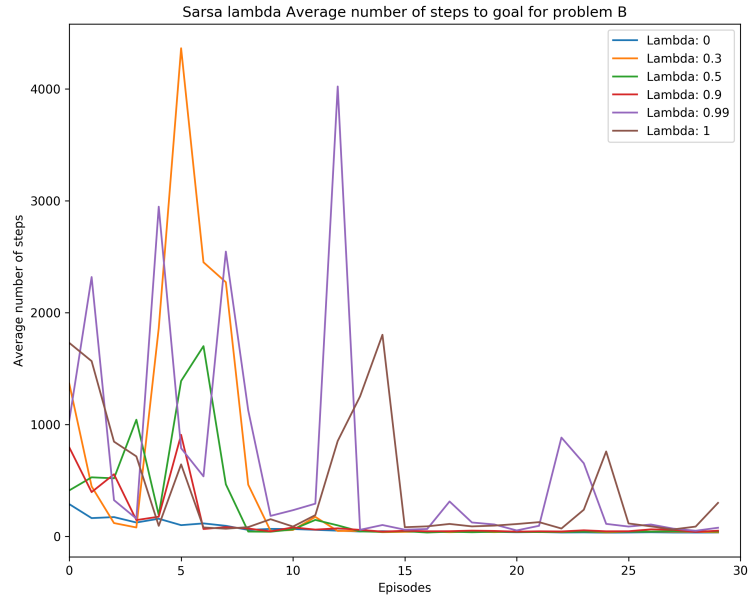


Figure 5: Sarsa: Optimal policy achieved after learning from 3000 episodes for all three problems.





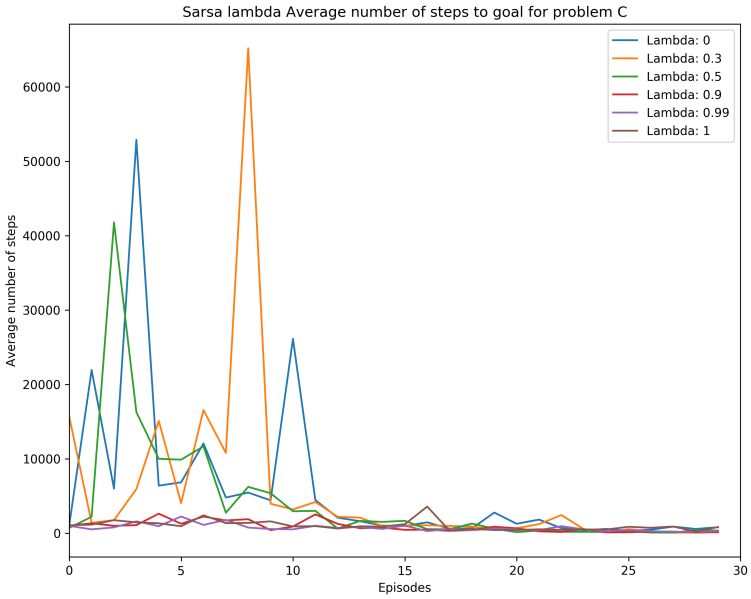
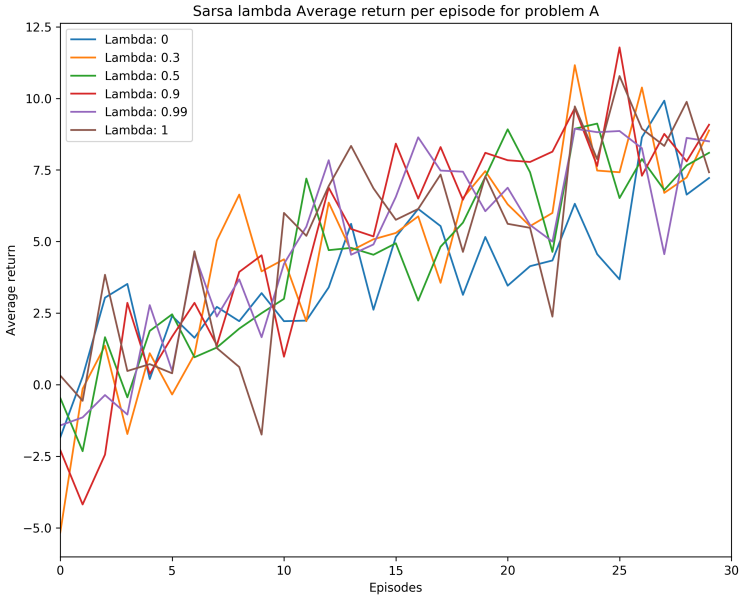
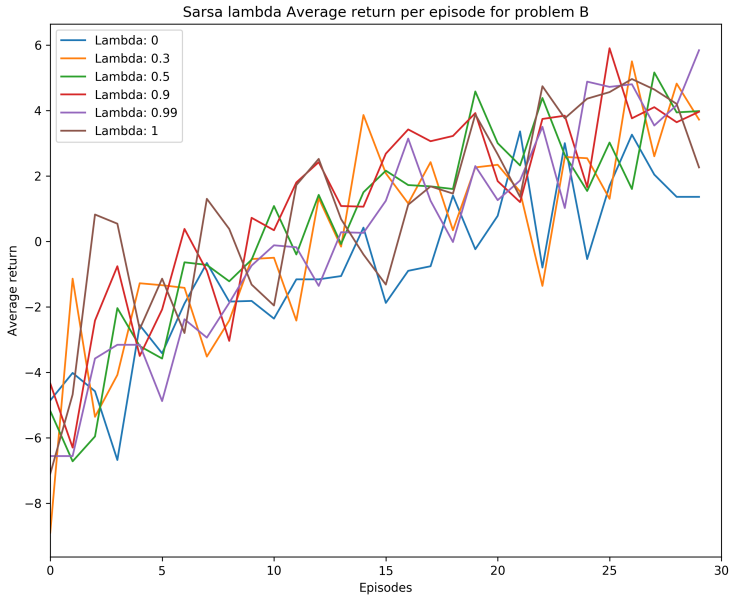


Figure 6: Sarsa lambda: Average Steps to reach goal averaged over 50 independent runs after 30 learning trials.





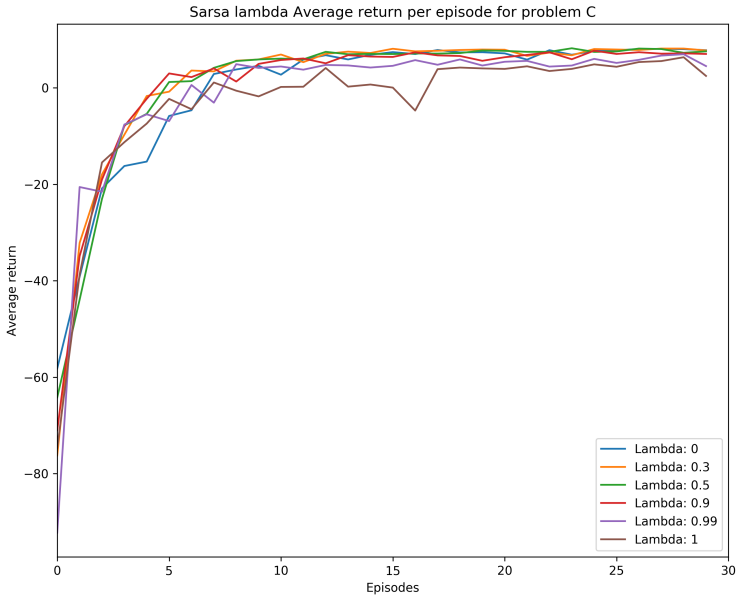


Figure 7: Sarsa lambda: Average return per episode averaged over 50 independent runs after 30 learning trials.

Sarsa(λ)

- Now we implement same by Sarsa(λ) for values of $\lambda = \{0, 0.3, 0.5, 0.9, 0.99, 1.0\}$, where λ is an eligibility trace parameter when $\lambda = 0$ it behaves like one step TD update and when $\lambda = 1$ it behaves like Monte-Carlo update.
- We can observe that initailly there are oscilliations in average number of steps, but after learning for few episodes it stabilizes. For problem A & B, $\lambda = 0$ & 0.9 performs best and other's oscillate more in intital few episodes. While for problem B, $\lambda = 0.9, 0.99$ & 1 performs best and others oscillate very high in initial few episodes, this is because when λ is low the learning will be slow, thus it will take large steps to find out that goal is present in puddle.
- Similarly, we can observe that average return per episode is low in initially few episodes then it increases with experience as it find out that it should avoid going into puddle. For problem A, $\lambda = 0.9$ performs best and others also perform similarly and all manage to get to same average return of ~ 7.5 after 30 learning trials. For problem B, since it's goal is near puddle, it will some times stumble into puddle thus it's average reward after 30 learning trails is less compared to problem A which is ~ 4 here also $\lambda = 0.9$ performs little better compared to others. While for problem C, since it's goal is in the puddle it will definitely take more episodes to learn, so we can observe that after 30 learning trials almost all for λ, it's average return is ~ 0.

  Note: *I have added video of agent playing for 10 episodes along with optimal policies shown for each state after learning from 3000 episodes for all three goals & for Q-learning and Sarsa in "Videos of trained agents" folder and extra plots showing performance for each goal individually for Q-learning & Sarsa and optimal policies for Sarsa(λ) is added in"Extra Plots" folder.*

Q2 on Policy Gradients

- We now implement chakra and vishamC using policy gradients. We carried out hyperparameters tuning of batch size, learning rate α, discount factor γ and compare it with respect to average returns v/s number of iterations.
- We can observe from plots on next page, that when learning rate α is low ~ 0.1 then there will be less oscillations or fluctuations in the average return as we are giving low weightage to every update and it will be even less if batch size in more ~ 100 compared to when batch size is less ~ 10, but when learning rate α is high ~ 0.9 then learning will be fast provided batch size is sufficiently large ~ 100, otherwise if batch size is less ~ 10, then there will be very high oscillations.
- When discount factor γ is less ~ 0.1 it will learn faster, but it will have oscillation if batch size is small ~ 10, when γ is more ~ 0.9 then it will learn little slower for same setting of α and batch size.
- When batch size is large ~ 100, then learning will be little slow as we are updating parameter θ after every 100 episodes but it will be more stable will less oscillations, and in contrast when batch size is small ~ 10 it will have large oscillations for same settings of other hyperparameters.
- Thus we can learn a policy faster with reasonably good setting of hyperparameter. For example in our problem we can learn faster will following setting of hyperparameters:

$$\alpha = 0.9,\ \gamma = 0.9,\ batchsize = 100\ \&\ iterations = 100$$

- We can observe that with this setting of hyperparameters it is able to reach an average return of about ~ -20 after 50 iterations for chakra and for vishamC we are able to reach average return of about ~ -35 after 50 iterations. Here we have included a simple time-dependent baseline b_t which is just the average of all R_t among the trajectories collected from the previous iteration, where in for first iteration it is set to 0.
- **Note:** More experiments empirical results like iterations, average return, value of parameter θ for different setting of hyperparameters for both problems chakra and vishamC is mentioned in pdf "PG Experiments" placed under "Extra Plots" folder.

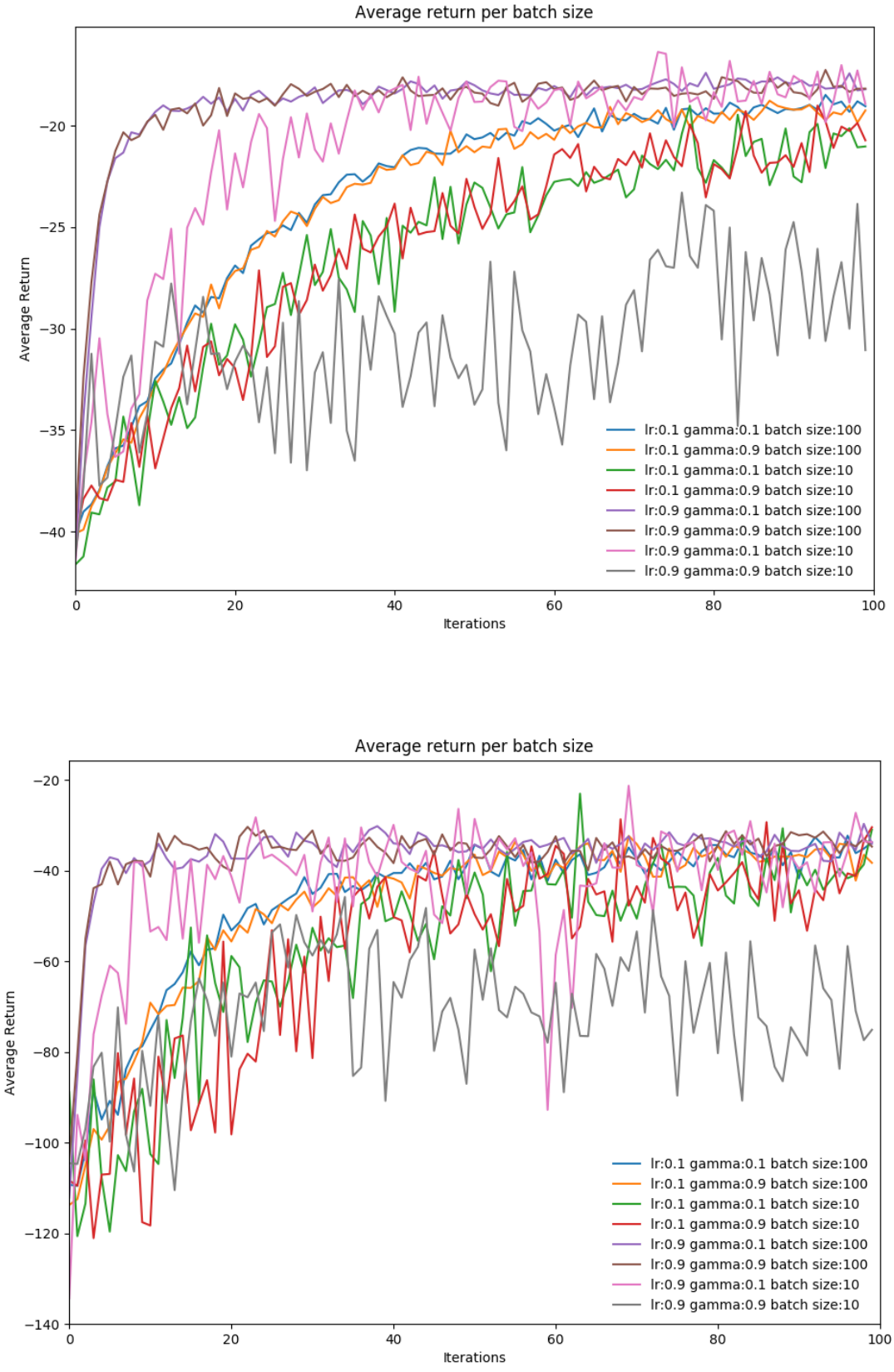


Figure 8: Average return per episode for chakra and vishamC policy gradient method with different hyperparameters. (First figure is for chakra and second is for vishamC)

Visualize a rough value function for the learned policy

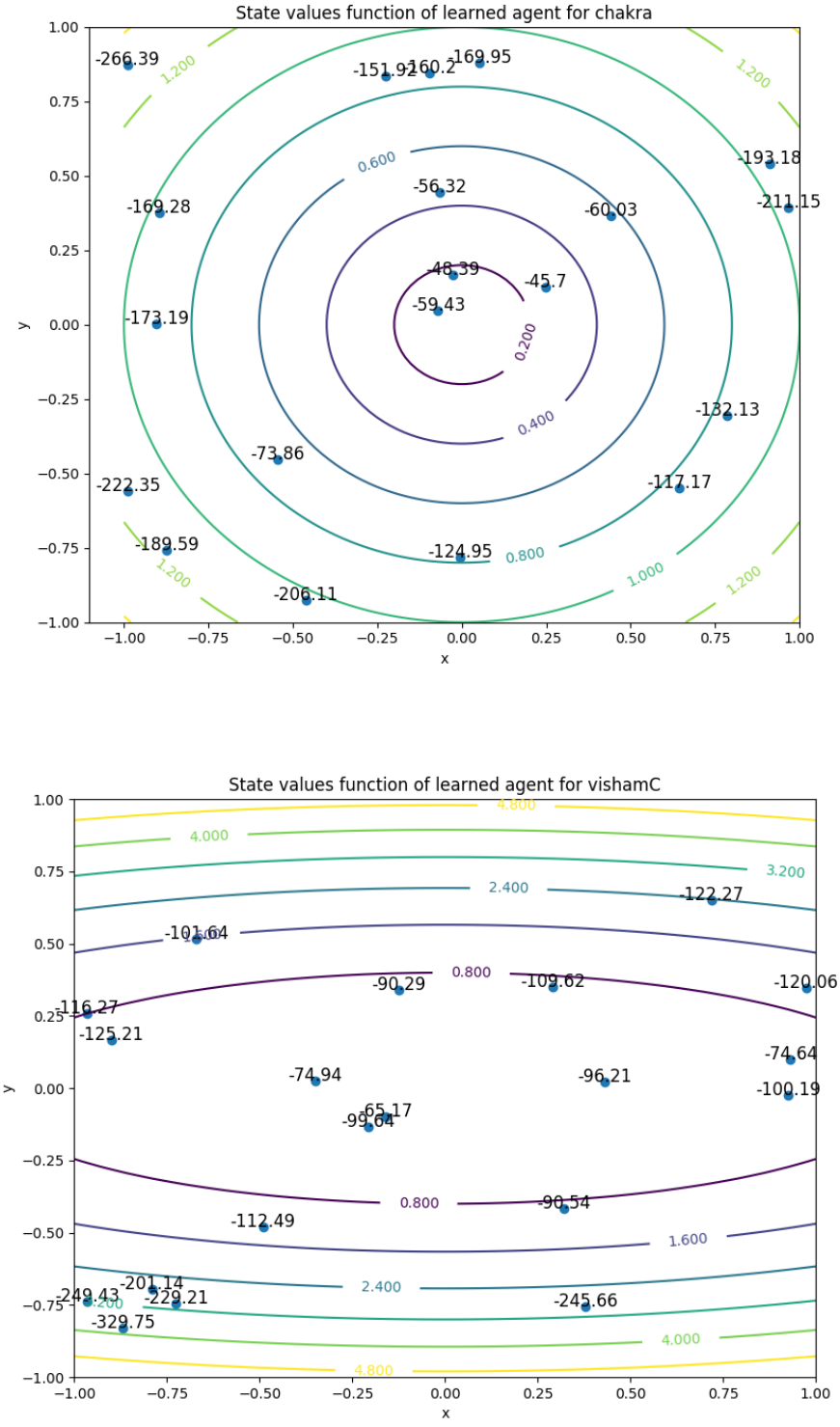


Figure 9: Plot of state value function from the policy learned by agent for both chakra & vishamC

- To visualize state value function for the learned policy through policy gradient method; since we have parameter θ and we how to select action a given state s i.e. by randomly sampling action from the normal distribution whose mean is $\mu = \theta^T s'$, where s' is the vector obtained by appending 1 to the state s. So by randomly selecting any start state and then selecting action according to learned policy π and accumulating return with discount rate $\gamma = 0.9$, we can get the value of state s. In short we use Monte-Carlo method to estimating value functions for given learned policy π.

- As show in above figure, I have plotted 20 state values for corresponding state (x, y) after training parameter θ for 100 iterations with batch size of 100, learning rate $\alpha = 0.9$ & discount rate $\gamma = 0.9$. We can observe

that for state values of chakra near center is more and as we move away from center state value decreases, and also state values which are in same contour region are close to each other. It is expected as the reward which is negative of distance from center would be almost same for those states.

- While for vishamC, since the reward function is a little skewed, as given by: $0.5x^2 + 5y^2$, we can observe that state values are also little skewed. Thus value of states in same contour region are close to each other and value of state decrease as they move away from center. Also state value don't change much if we move little towards x-direction, but changes more as we move little towards y-axis. This is because of the way reward function is defined as more weightage is given to movement along y-axis as compared to x-axis. Whereas in case of chakra equal weightage was given to movement along any direction.
- Similarly, we can find state-action value function also by first randomly selecting state s and action a and then computing discounted return obtained by following learned policy π.

Plotting the policy trajectories for the learned agent

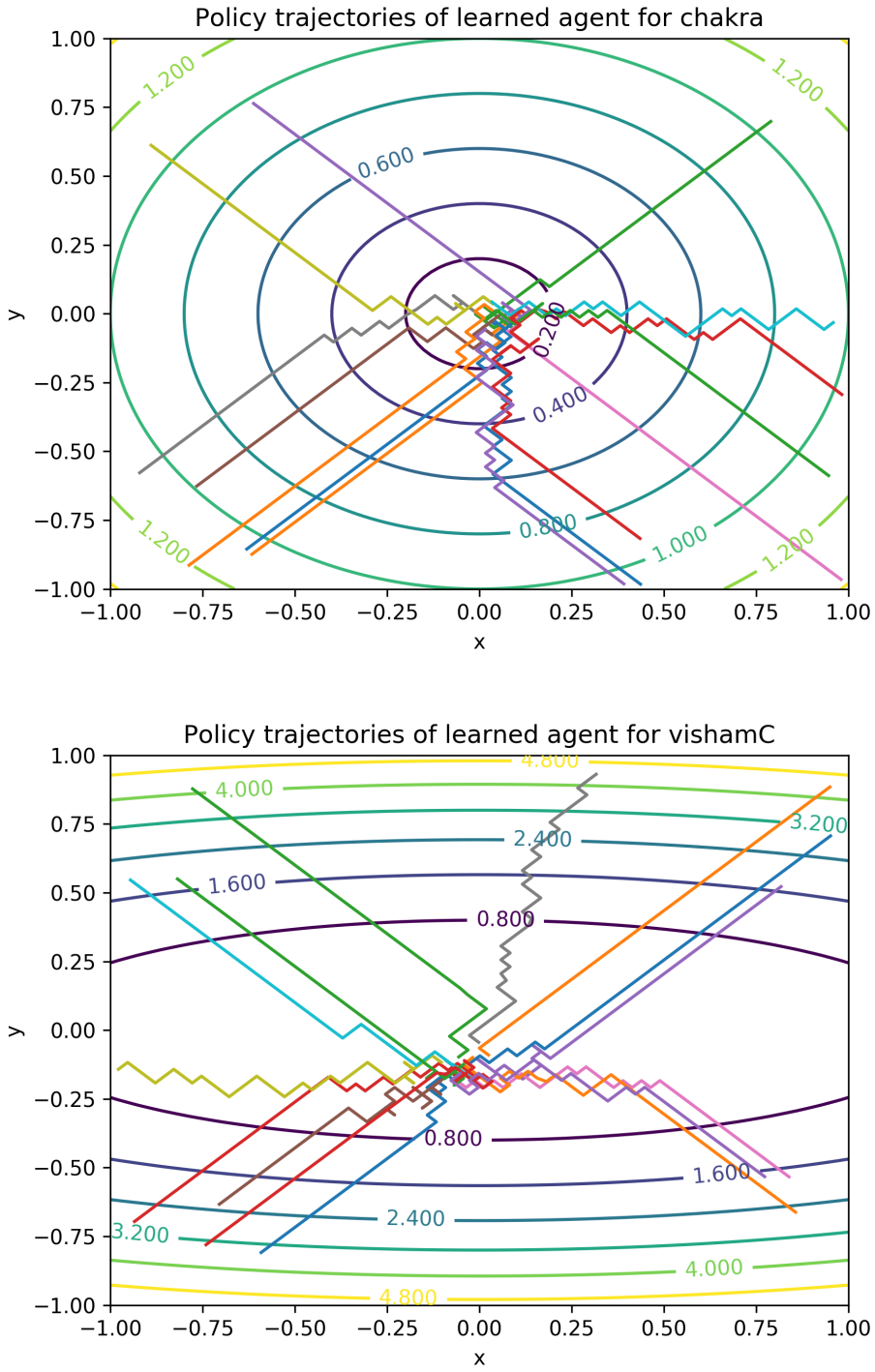


Figure 10: Plot of 15 trajectories from the policy learned by agent for both chakra & vishamC

- In case of chakra, we observe that agent will start moving diagonally toward center and once it has reached $x = 0$ it will start moving towards center by moving along y-direction and will not move along x-direction. Similarly, when agent reaches $y = 0$ it will start moving towards center by moving along x-direction. Also there will be little oscillation when it has to move along only one direction as all closer region will have similar values.
- In case of vishamC, we can observe that agent will start moving diagonally first toward $y = 0$ as movement along y-direction is given more weightage due to skewed reward function, then it will stop moving along that direction and start moving towards center by moving along x-direction. The movement will not be smooth, there will be oscillations along the way, as all regions close to center will have similar values.

References:

1. Richard S. Sutton and Andrew G. Barto, Reinforcement Learning: An Introduction, Second edition
2. Castellini Jacopo. Some GridWorld environments for OpenAI Gym
3. David Silver. Reinforcement Learning Slides