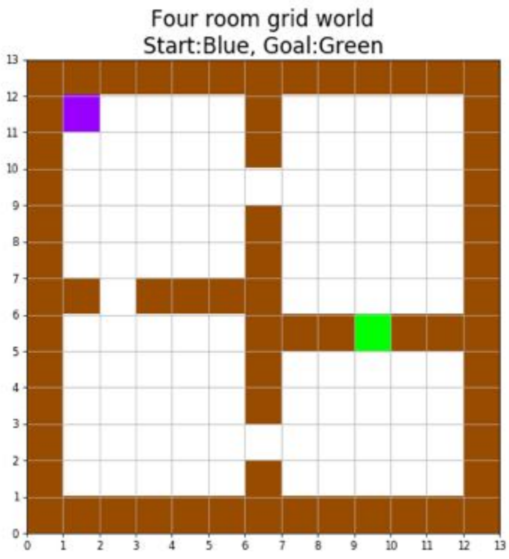
CS6700 : Reinforcement Learning

Programming Assignment #3 Nirav Bhavsar (CS17S016)

Note: Due to file upload size limit I have compressed this pdf. Original report and extra plots can be found on this link.

Q1 on Four room Gridworld :

- We will implement SMDP and Intra-Option Q-learning on below four room gridworld, where start state is blue box i.e., top left of room 1 or center of room 4, goal is any one of green box and brown's are wall. In this reward for moving one step is 0 and for reaching goal state is +1. If the agent hits the wall, it remains in the same state and gets reward of 0. The discount factor γ is taken as 0.9. Also the environment is stochastic, so it will take correct action with 2/3 probability and any three of incorrect action with 1/9 probability for each action.



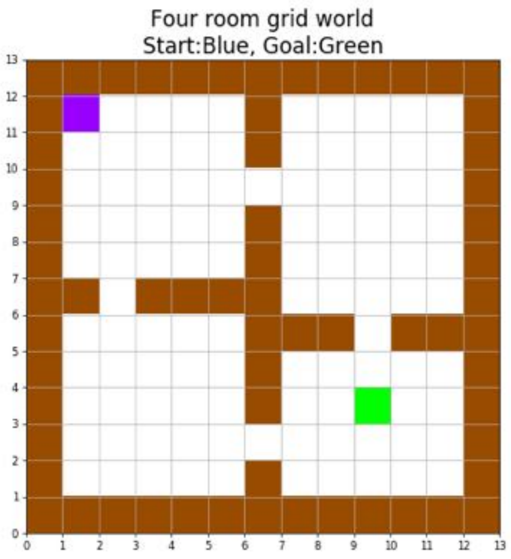


Figure 1: Four room Gridworld

- There are 8 multi-step options: each one leading to a particular room's hallway i.e., for each of the four rooms we have two built-in hallway options which is designed to take the agent from anywhere within the room to one of the two hallway cells leading out of the room. A hallway option's policy π follows a shortest path within the room to its target hallway while minimizing the chance of stumbling into the other hallway. The termination condition $\beta(s)$ for each hallway option is zero for states s within the room and 1 for states outside the room, including the hallway states. The initiation set $\mathcal{I}$ comprises the states within the room plus the non-target hallway state leading into the room. These options are deterministic and Markov, and that an option's policy is not defined outside of its initiation set.

- For simplicity, I have assumed that for each room hallway option 1 would take agent from current room to hallway state which comes first in clockwise direction i.e., for room 1 Hallway option 1 would take to hallway state which is on right side leading to room 2. Similarly hallway option 2 for each room would take agent from current room to hallway state which comes first in anti-clockwise direction i.e., for room 1 Hallway option 2 would take to hallway state which is on bottom side leading to room 4.

- To learn hallway option optimal policy π for each room, I used Q-learning by randomising the start state to any point in the room and changing the goal to the hallway state. I saved those 8 policy's which would be used in implementing SMDP and Intra-Option Q-learning. The hallway option 1 and 2 optimal policy learned and state values obtained by taking $V(s) = \underset{a \in A}{argmax}\ Q(s, a)$ for each state is shown in below figures.

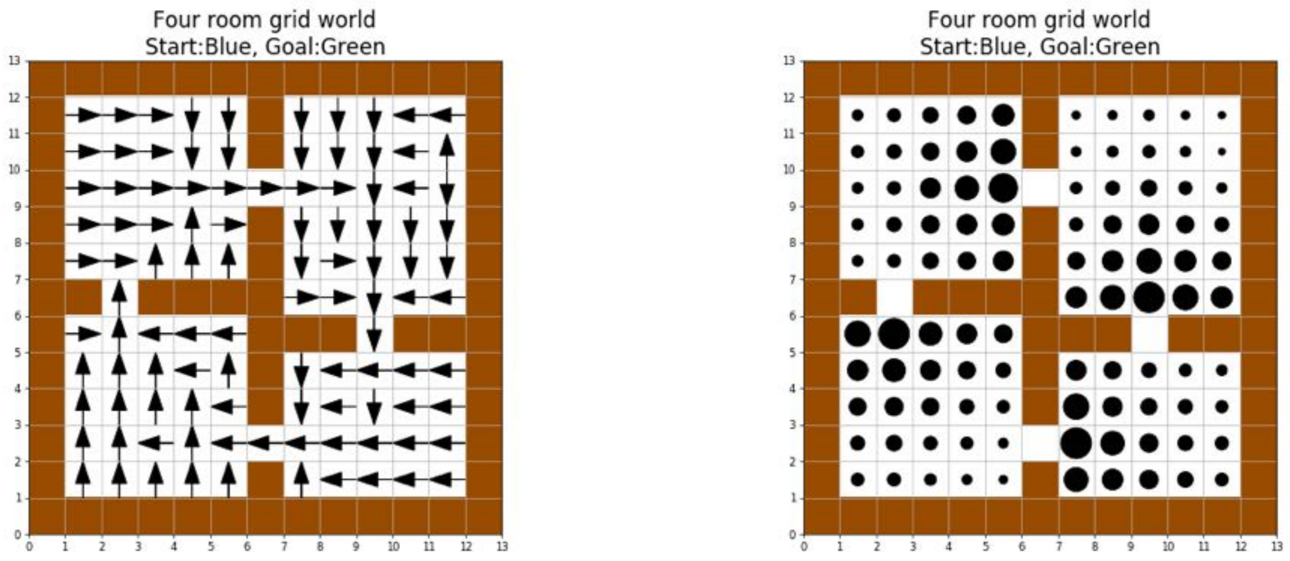


Figure 2: Q-Learning: Learned a) Optimal policy and b) State-Values for **Hallway Option 1 (clockwise)** for each room by setting hallway state as goal.

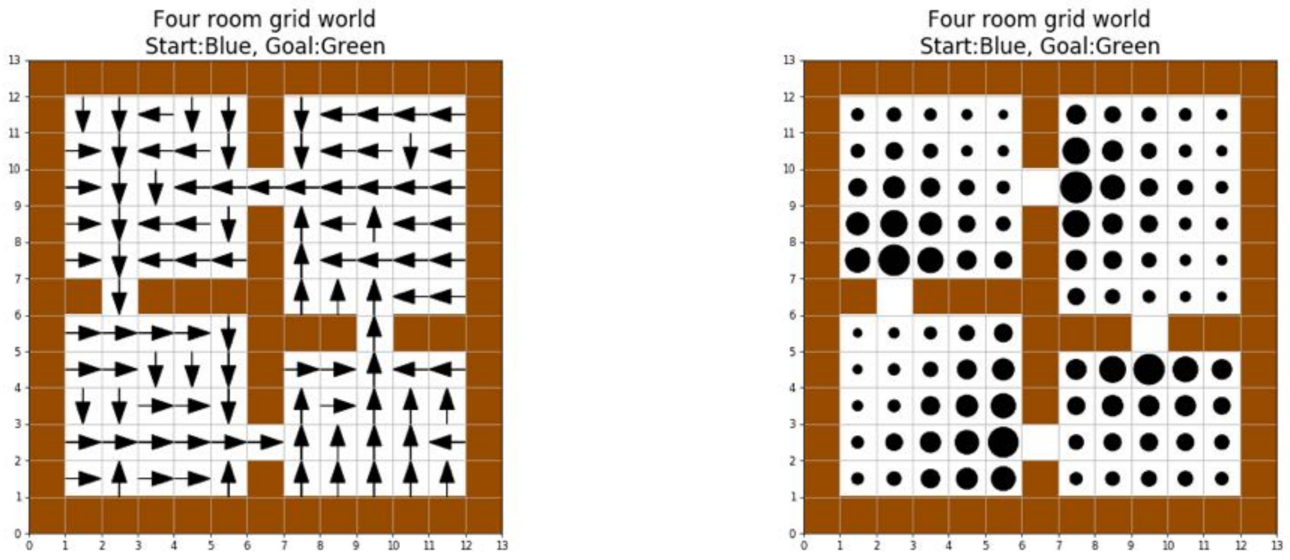


Figure 3: Q-Learning: Learned a) Optimal policy and b) State-Values for **Hallway Option 2 (anti-clockwise)** for each room by setting hallway state as goal.

SMDP Q-Learning [1]

- SMDP is an MDP augmented by the options where execution of option o which starts in some state $s \in \mathcal{I}$, follows option policy π and jumps to the state s' in which o terminates. Based on this experience, the one-step SMDP Q-learning updates option-value function $Q(s, o)$ by

$$Q(s, o) = Q(s, o) + \alpha[r + \gamma^k \max_{o' \in \mathcal{O}_{s'}} Q(s', o') - Q(s, o)]$$

  where k denotes the number of time steps elapsing between s and s', r denotes the cumulative discounted reward over this time

- One drawback to SMDP learning methods is that they need to execute an option to termination before they can learn about it. Because of this, they can only be applied to one option at a time. To overcome this limitation and to learn about an option before the option terminates intra-option methods are used.

Intra-Option Q-Learning [2]

- Intra-option is an off-policy learning methods because they learn about the consequences of one policy while actually behaving according to another, potentially different policy. They can learn about many different options without ever executing those options from the same experience. Hence they are potentially more efficient than SMDP methods because they extract more training examples from the same experience.

- On experiencing $< s, a, r, s' >$, for every option o that picks action a in state s, intra-option Q-learning performs the following update:

$$Q_{\mathcal{O}}(s,o) = Q_{\mathcal{O}}(s,o) + \alpha(s,o)[r + \gamma \tilde{Q}_{\mathcal{O}}(s',o) - Q_{\mathcal{O}}(s,o)]$$

$$\tilde{Q}_{\mathcal{O}}(s',o) = (1-\beta(s'))Q_{\mathcal{O}}(s',o) + \beta(s') \max_{o' \in \mathcal{O}_{s'}} Q_{\mathcal{O}}(s',o')$$

Learning Curves: Average steps and Average return

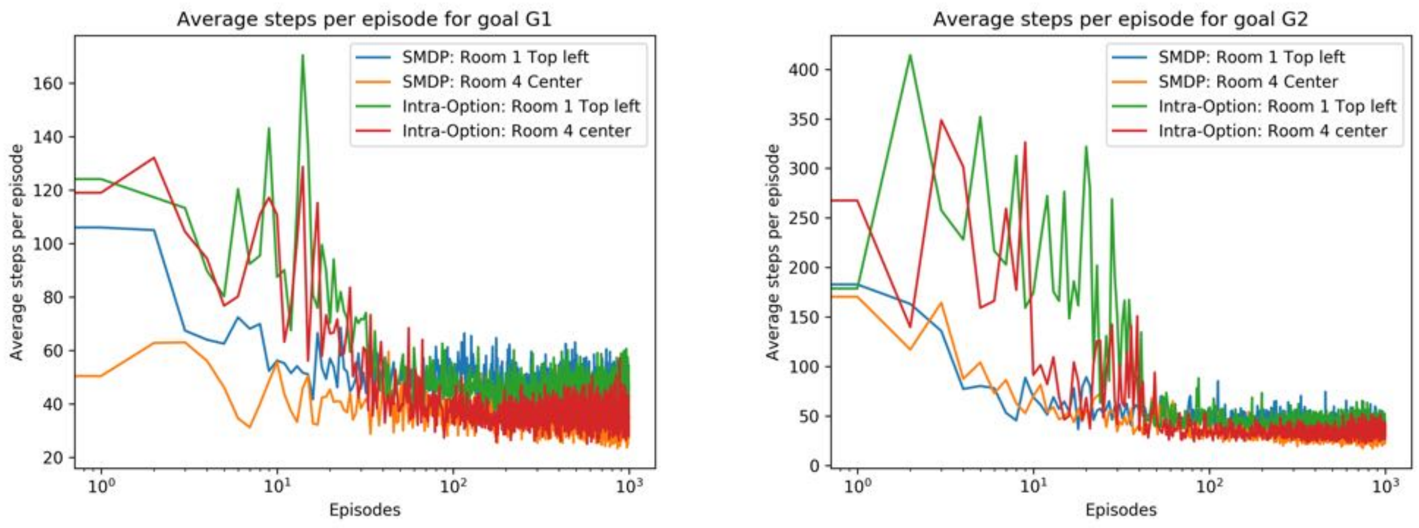


Figure 4: Plot comparing average steps taken per episode by SMDP and Intra-Option Q-Learning with different initial state for Goal 1 and 2. Episodes on x-axis are in **log-10 scale** and average is taken over 10 iterations.

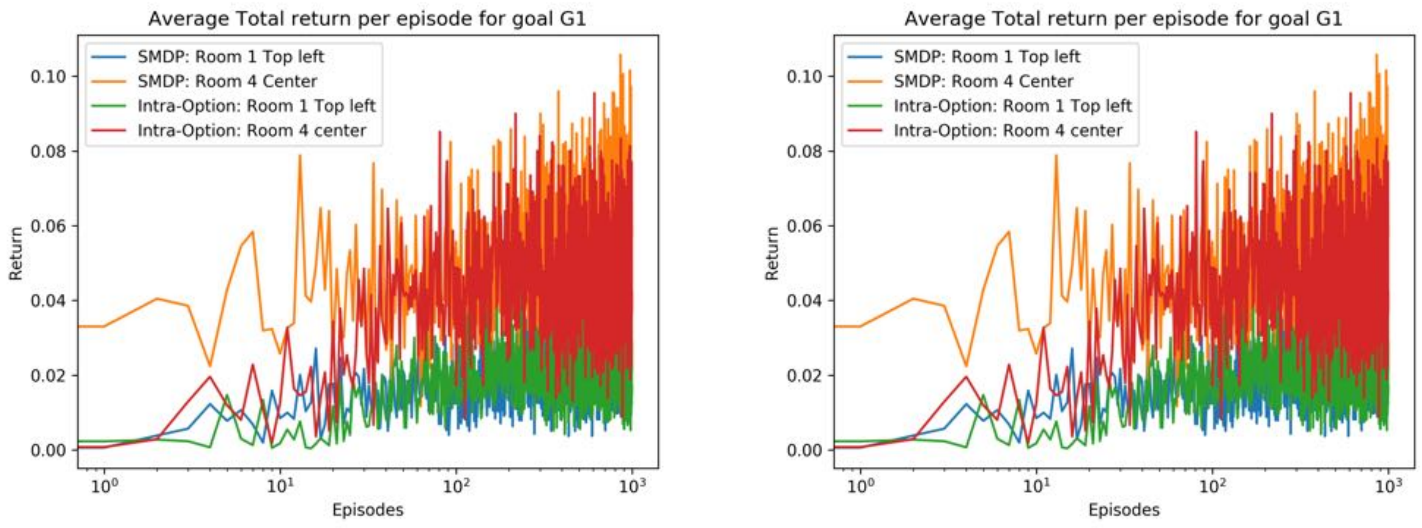


Figure 5: Plot comparing average total return per episode by SMDP and Intra-Option Q-Learning with different initial state for Goal 1 and 2. **Each primitive action is considered as a single step**. Episodes on x-axis are in **log-10 scale** and average is taken over 10 iterations.

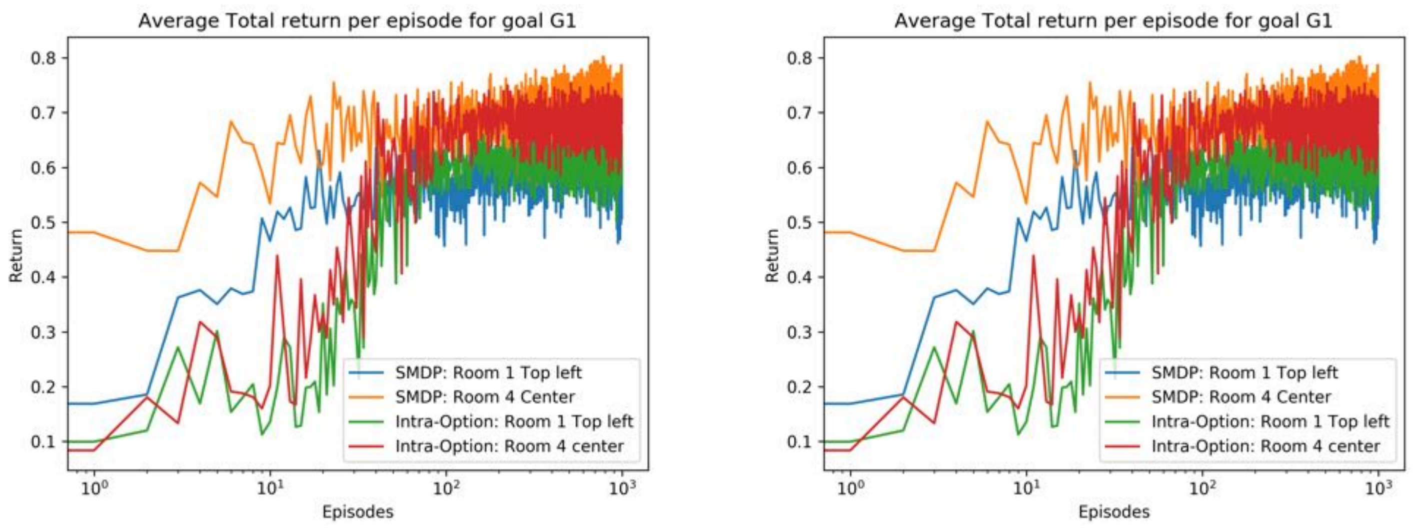


Figure 6: Plot comparing average total return per episode by SMDP and Intra-Option Q-Learning with different initial state for Goal 1 and 2. **Each option is considered as a single step**. Episodes on x-axis are in **log-10 scale** and average is taken over 10 iterations.

Observations

- For goal G1 which is in hallway state:
  - ➔ SMDP Q-learning with initial state as top left state of room 1, we observe that initially it explores and takes more steps to reach goal, but it quickly learns and is able to reach goal on an average of 50 steps after learning from 30 episodes. It is able to learn quickly because the goal state is one of the hallway state which is also the terminal state for option. While when initial state is centre of room 4, we observe that since initial state is near to goal it takes comparatively less number of steps to reach goal. Thus it is able to reach goal in around 40 steps after learning from about 30 episodes.
  - ➔ Intra-Option Q-learning with initial state as top left state of room 1, we observe that since intra-option learns about every applicable options from every experience, so it will be able to learn options correct value faster compared to SMDP. Initially it explores and learns about options but quickly it learns which is the correct option to take and hence we can see it is able to reach goal in around 50 episodes after learning from about 50 episodes. While when initial state is centre of room 4, we observe that since initial state is near goal it takes less number of steps and it is able to reach goal in about 40 episodes after learning from 30 episodes.
  - ➔ Average step taken by Intra-option method is similar to SMDP method but the difference here is that in addition to that it is able to learn options value better and thus it will prefer taking options over primitive actions for this goal, this can also be observed from optimal policy learned in later figures.
- For goal G2:
  - ➔ SMDP Q-learning with initial state as top left state of room 1, we observe that initially the steps taken is more but after about 30 episodes of learning it learns optimal policy and reaches the goal in about 50 steps. It is able to learn quickly as goal state is near hallway state so by reaching hallway state it can reach goal faster as near by goal states will have higher Q-value for some action which leads to goal. While when initial state is centre of room 4, we observe that since initial state is near to goal state it takes less number of steps on average and is able to stabilize learning in about 20 episodes.
  - ➔ Intra-Option Q-learning with initial state as top left state of room 1, we observe that since goal state is now not a terminal state of any option. Intra-option method will explore more to figure out that it should take primitive action near goal states in addition to options in other states and thus it takes more steps to find goal initially. But quickly it learn optimal policy and is able to reach goal in around 50 steps after learning from about 50 episodes. While when initial state is centre of room 4, we observe that it takes less number of steps on average compared to earlier case.

- Similarly trends can also be observed from total discounted return plots which increases quickly by learning from few episodes.

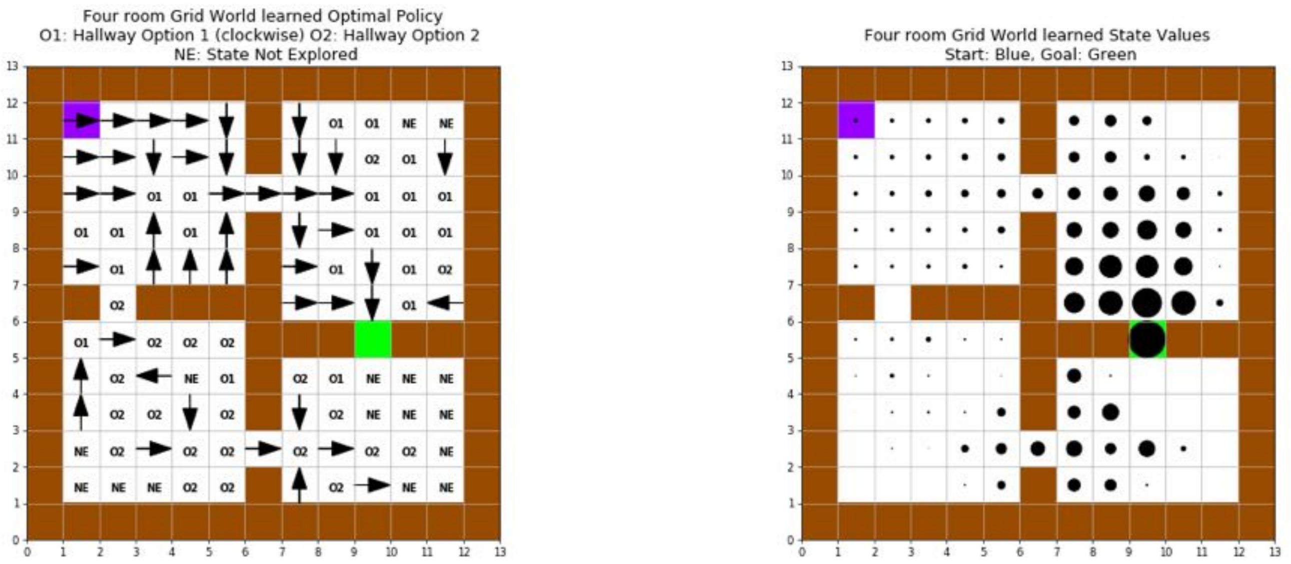


Figure 7: SMDP Q-Learning: Learned a) Optimal policy and b) State-Values for Goal 1 after training for $10K$ episodes. Start state: Blue and Goal: Green

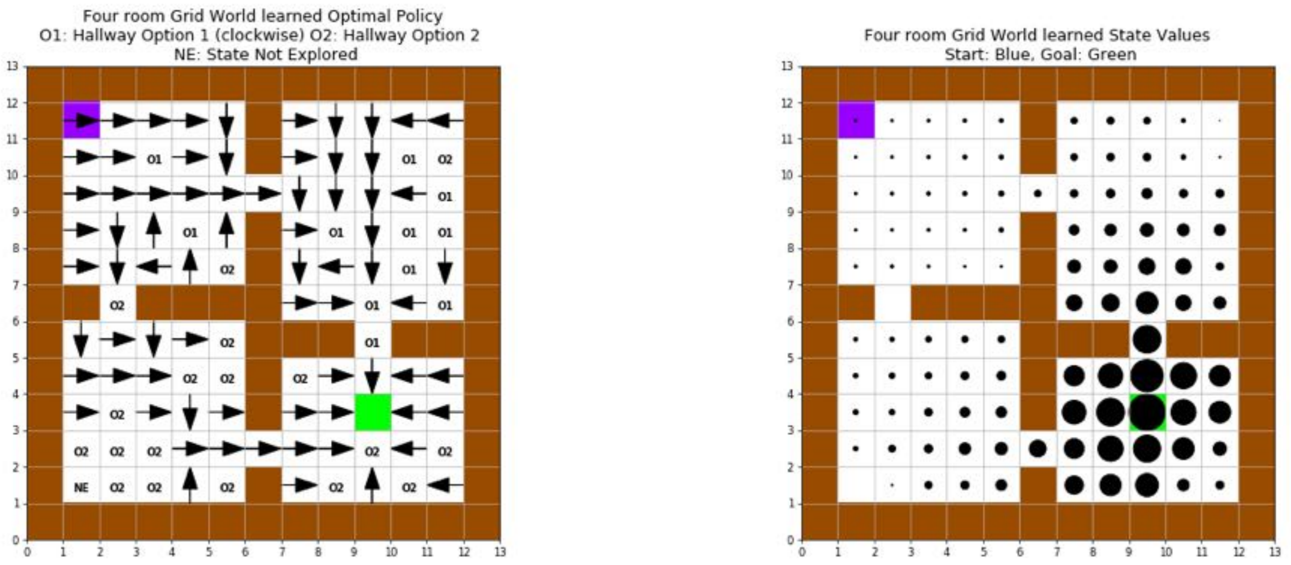


Figure 8: SMDP Q-Learning: Learned a) Optimal policy and b) State-Values of each state for Goal 2 after training for $10K$ episodes. Start state: Blue and Goal: Green

- We can observe that optimal policy learned by SMDP Q-learning has a mix of both primitive actions and options. Because SMDP in general needs more experience to learn options compared to intra-option. Primitive actions would look good compared to options unless an option is terminating near goal state which has higher Q-value learned from previous experiences.

- From state-values also we can observe that it has higher values near goal state and decreases as it moves away from goal, also few far away states have comparitively higher values since it is selecting option in that state which leads to reaching goal state. Also few states are taking primitive action as taking primitive action or opting options boths leads to same return near goal state. Some of the state are not explored as by taking options it is able to reach goal state.

- For goal G2, state near by are selecting primitive actions over options as options would lead it towards hallway state away from goal, and some states are selecting options also because by following option policy it is able to reach goal.

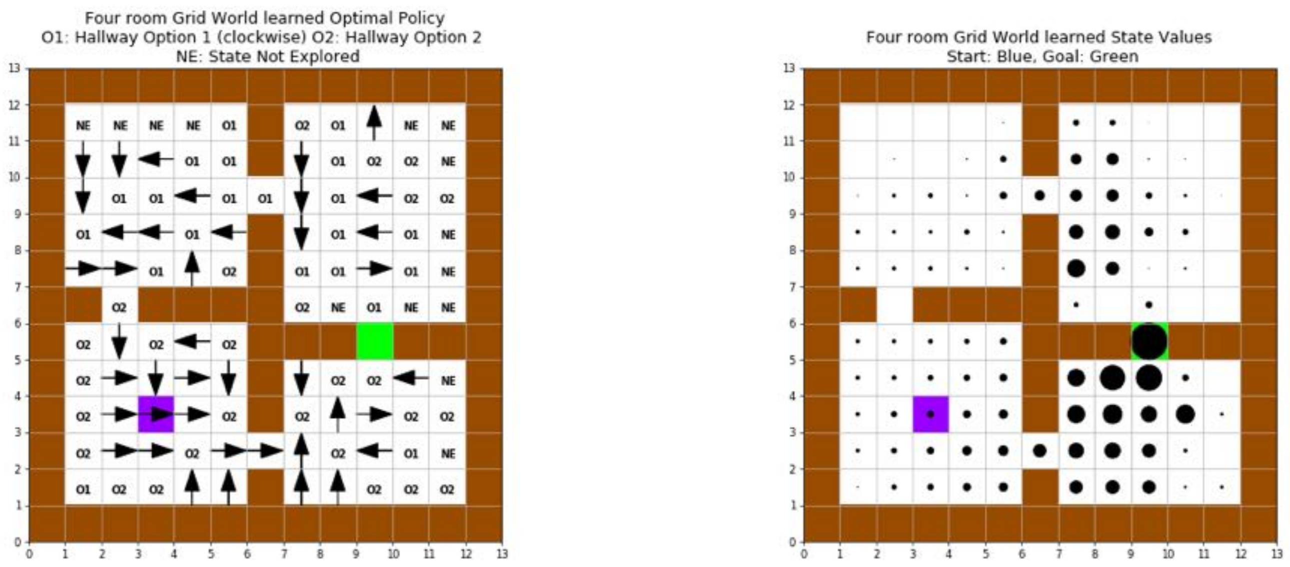


Figure 9: SMDP Q-Learning: Learned a) Optimal policy and b) State-Values for Goal 1 after training for $10K$ episodes. Start state: Blue and Goal: Green

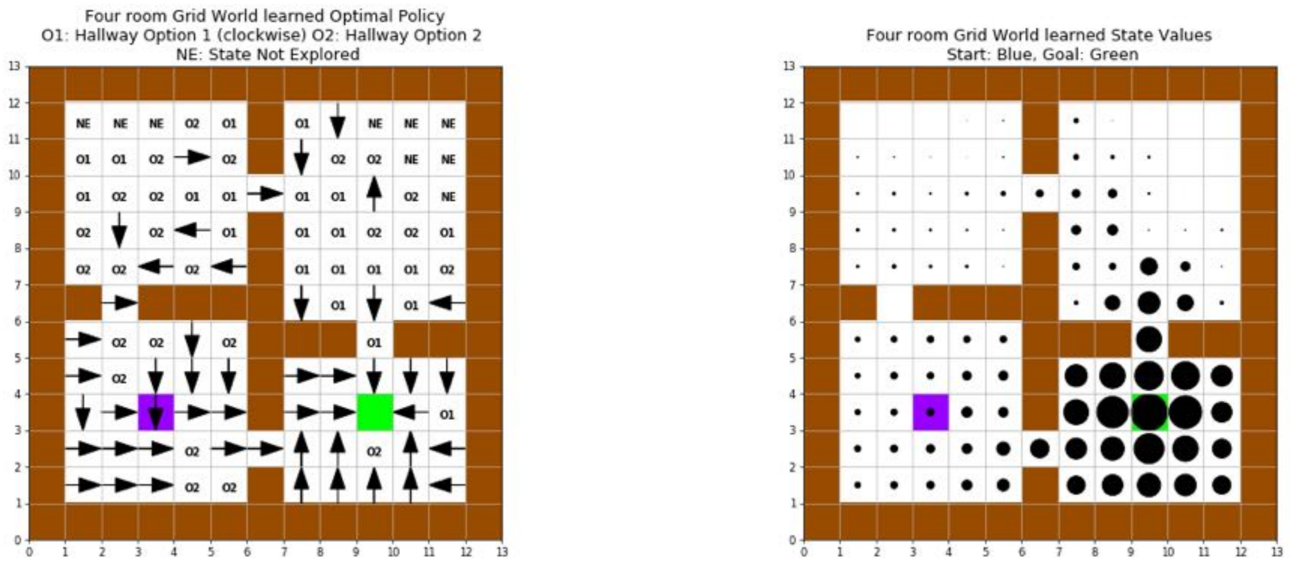


Figure 10: SMDP Q-Learning: Learned a) Optimal policy and b) State-Values for Goal 2 after training for $10K$ episodes. Start state: Blue and Goal: Green

- Similarly in this case also we observe that optimal policy learned by SMDP Q-learning has a mix of both primitive actions and options as in some states primitve action looks better and in some states option look better. From state-value we can observe that due to taking options some states which are far away from goal have higher values. Thus options helps in learning faster particularly when goal state is also terminal state for options.

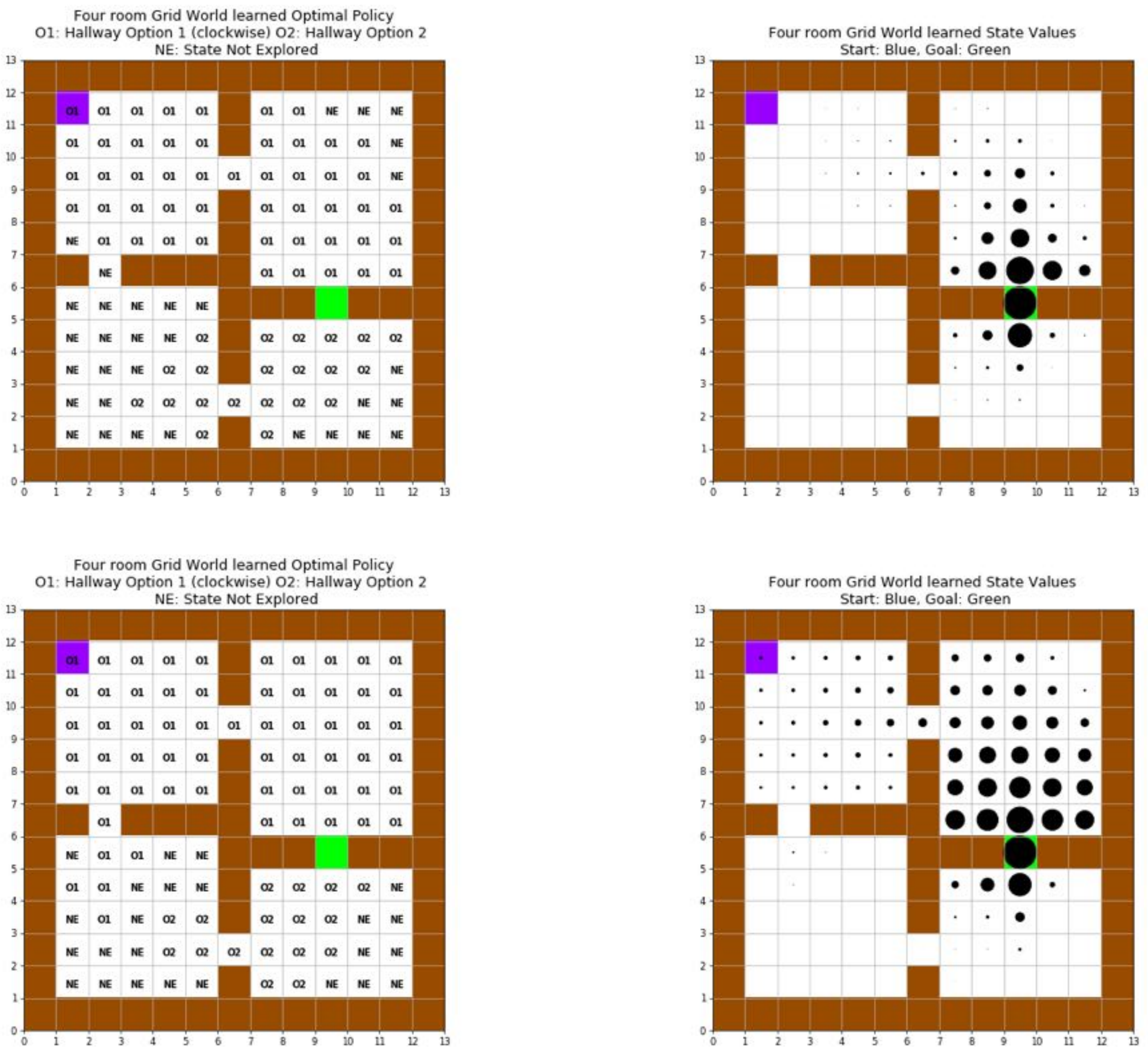


Figure 11: Intra-Option Q-Learning: Learned a) Optimal policy and b) State-Values for Goal 1 after training for **100 & 1000** episodes. Start state: Blue and Goal: Green

- Intra-Option Q-learning learns about every applicable options from every experience. We can observe that optimal policy learned by Intra-Option Q-learning for goal G1 from just 100 episodes picks only options from every state which is correct since goal state is termial state for two options so ideally agent should be picking option which will lead to reaching goal more quickly.
- By learning for more episodes optimal policy doesn't change much only state-values becomes larger as reward is propagated to states which picks that option.
- We can also observe that the options value learned by intra-option method from just 1000 episodes are far better and close to correct that those learned by SMDP methods from 10K episodes which are not correct and thus picks mix of both.

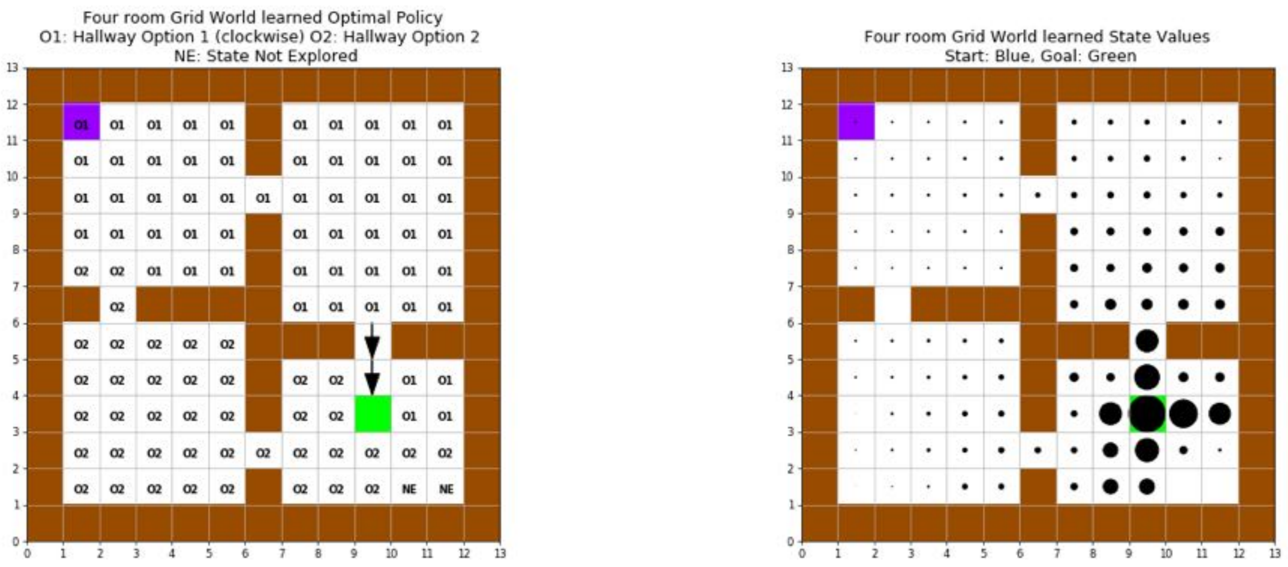


Figure 12: Intra-Option Q-Learning: Learned a) Optimal policy and b) State-Values for Goal 2 after training for **1000** episodes. Start state: Blue and Goal: Green

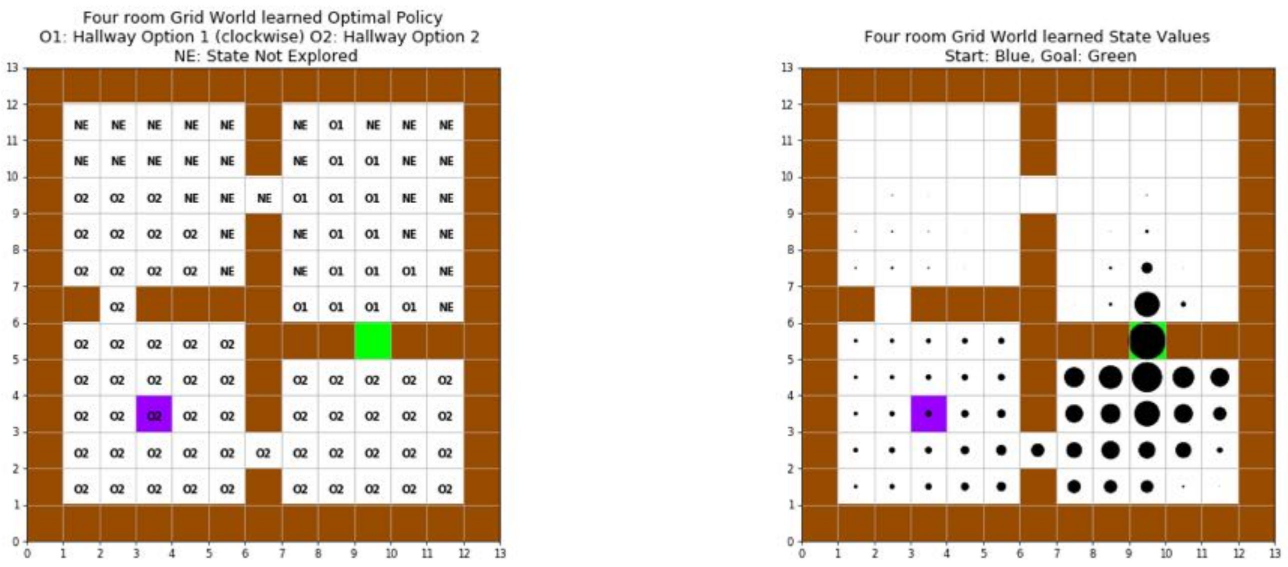


Figure 13: Intra-Option Q-Learning: Learned a) Optimal policy and b) State-Values for Goal 1 after training for **1000** episodes. Start state: Blue and Goal: Green

- Similarly for goal G1, we can observe from optimal policy learned that it is picking options over primitive actions as it has learned that picking option is better as it will lead to goal state.
- For goal G2, we can observe that it is able to learn that to reach goal it has to take primitive action and not take only options, so for states near goal it is picking primitive action and for other states it is picking option. Similar trend is observed from state-values also.

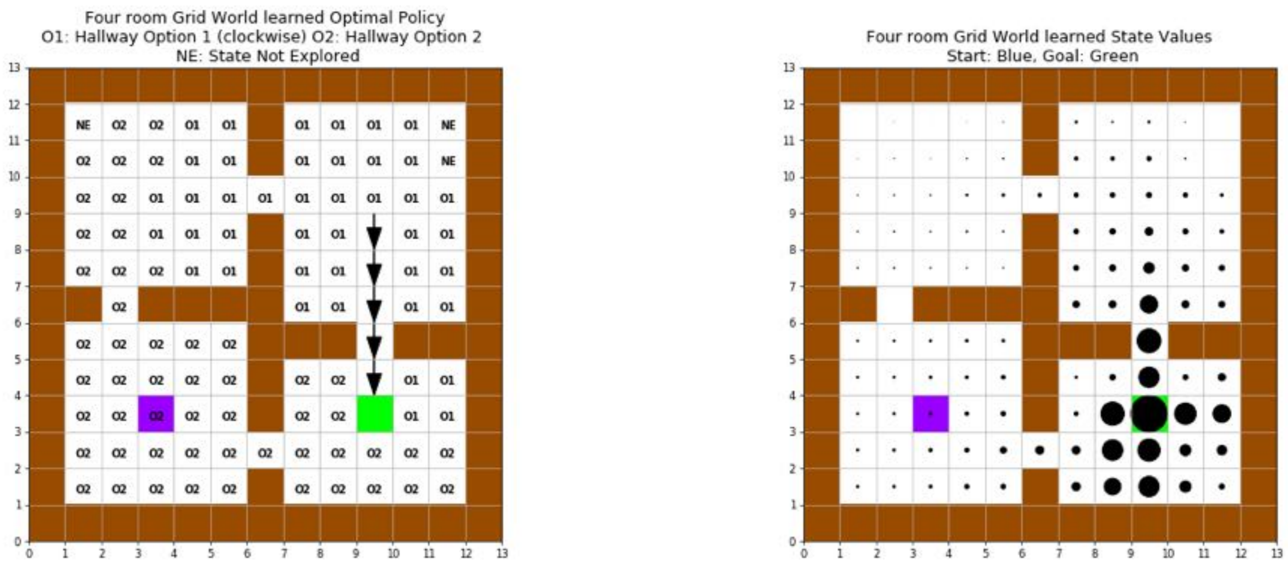


Figure 14: Intra-Option Q-Learning: Learned a) Optimal policy and b) State-Values for Goal 2 after training for **1000** episodes. Start state: Blue and Goal: Green

Q2 on DQN-Cartpole

DQN Brief Overview:

- It is well know that reinforcement learning is unstable or even diverges when a nonlinear function approximator such as a neural network is used to represent the action-value function Q. This is because of the correlations present in the sequence of observations, where small updates to Q may significantly change the policy π and the target values $y_j = r + \gamma \ max_{a'} \ Q(s', a')$.
- DQN [3] made two important modifications to improve the stability with neural networks. First, they stabilize the training of action value function approximation with deep neural networks (CNN) using **experience reply** to update Q function estimates by taking random samples from replay memory, this reduces the correlation between training examples, which smooths out learning and reduces the variance of the updates.
- Second, they use a **separate network for generating the targets** $y_j = r + \gamma \ max_{a'} \ Q(s', a'; \theta_i^-)$ in the Q-learning update. i.e., rather than computing the gradient of the Q function with respect to itself, they clone the network Q to obtain a target network $\hat{Q}$ and use $\hat{Q}$ for generating the Q-learning targets y_j, which reduces correlations with the target and stabilizes the training process.
- The loss function used by DQN to train the network is:

$$L_i(\theta_i) = \underbrace{\mathbb{E}_{s,a,r,s' \sim U(D)}}_{\text{Experience Replay}} \ [\underbrace{(r + \gamma \max_{a'} Q(s', a'; \theta_i^-)}_{\text{Target}} - \underbrace{Q(s, a, \theta_i)}_{\text{Prediction}})^2]$$

  where D is the set of all experiences, γ is the discount factor, θ_i are the network weights at iteration i and θ_i^- are the cloned network weights at iteration i.

Q1: DQN-CartPole Learning curve

- From figure (a), we can observe that initially the agent is exploring and learning about the environment (as i have set initial ϵ to 1 with decay rate of 0.995), so it performs poorly in first 50 episodes. After that agent figures out which is the optimal action to take, hence it starts learning quickly and we can observe that it is able to solve Cartpole task which is getting reward of over 195 for 100 consecutive episodes in around 200 episodes. After 200 episodes agent has learned the optimal policy and it is able to get an average reward of over 195 constantly for the next 100 episodes also.
- Similarly from figure (b), we can observe that for intial 50 episodes agent perfroms poorly as it is exploring the environment. But after 50 episodes the episode length increases rapidly which shows that agent is learning optimal policy and is able to solve Cartpole task in around 200 epsiodes. We can observed that after about 180 episodes the agent is able to get maximum reward of 200 constantly with few fluctuations because of exploration.

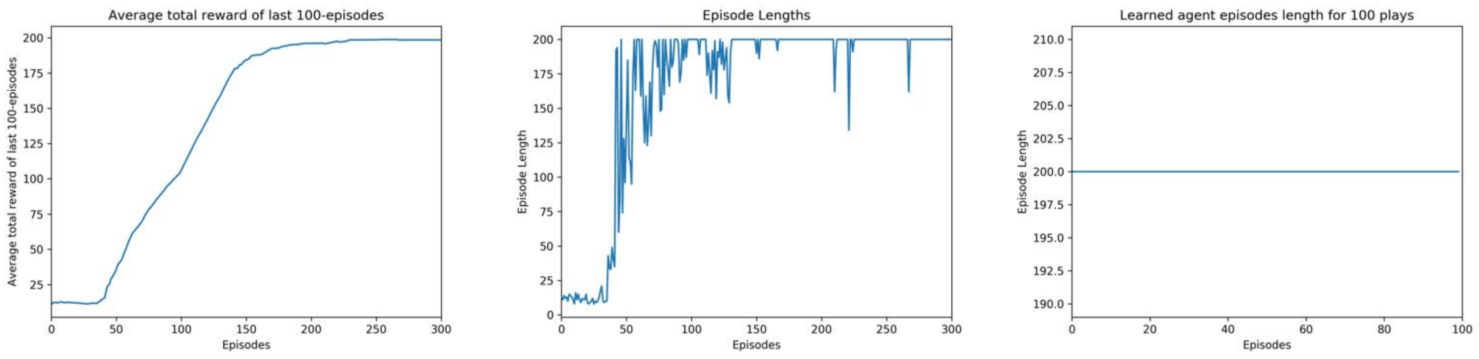


Figure 15: Learning curve over episodes in terms of a) Average total reward of last 100-episodes b) Epsiode lengths and c) Learned agent episode length for 100 episodes

- Figure (c) shows the episode length for 100 episodes after agent has completed learning from 300 episodes. We can observe that since agent is able to learn optimal policy, it is able to get maximum reward of 200 constantly for 100 consecutive episodes.

Q2: Best set of hyperparameters
The following set of hyperparameters worked best for me and I was able to solve CartPole task in around 200 episodes:

Hyperparameter	Value	Description
REPLAY MEMORY SIZE	10000	Number of tuples in experience replay memory
MINIBATCH SIZE	32	Size of minibatch sampled from the experience replay
TARGET UPDATE FREQ	200	Number of steps after which the target network is updated
EPISODES NUM	300	Number of episodes for training
MAX STEPS	200	Maximum number of steps in an episode
DISCOUNT FACTOR	0.999	Discount factor γ used in Q-learning update
INITIAL EPSILON	1	Initial value of ϵ in $\epsilon-$greedy exploration
EPSILON DECAY RATE	0.995	Exponential decay multiplier for epsilon
MIN EPSILON	0.01	Final minimum value of ϵ in ϵ-greedy
HIDDEN1 SIZE	20	Hidden layer 1 size of neural network
HIDDEN2 SIZE	20	Hidden layer 2 size of neural network
HIDDEN3 SIZE	20	Hidden layer 2 size of neural network
LEARNING RATE	0.001	Learning rate for Adam
REGULARIZATION FACTOR	0.0001	Regularization factor to reduce overfitting

Table 1: Set of hyperparameters value which worked best.

Q3: Observations on varying hyperparameters

- Figure (a) compares learning curve in terms of average total reward for different values of initial ϵ from set $\{1, 0.9, 0.7, 0.5\}$. We can observe that if the initial ϵ is high like 1 or 0.9 then agent will first explore and learn about environment and after that it will start learning policy without much fluctuation in learning curve. While if initial ϵ is low like 0.7 or 0.5, then since the agent doesn't explore much initially, it could learn faster, but after some time it could reach state which it hasn't explored and end up taking bad action. Thus we can observe that the learning curve for such initial ϵ is not smooth and there are fluctuation where learning curve goes down and then it learns and goes up again. So in general it is better to start with high initial ϵ to explore the environment first, then gradually decrease ϵ to learn optimal policy.

- Figure (b) compares learning curve in terms of average total reward for different values of hidden layer size from set $\{10, 20, 32, 64, 128\}$. We can observe that if the hidden layer size is less like ≤ 20 then it can learn faster initially as it needs less data to learn few weights, while if the hidden layer size is more like ≤ 128 then it will take time to learn as it needs more data to learn such large number of weights. We can observe that in our problem with hidden layer size of ≤ 64, it is able to learn quickly and smoothly with little fluctuations. So in general, if problem is simple then it is better to use less number of hidden layer size as it will learn faster and reduce over-fitting and if problem is complex then it is better to use large hidden layer size so that agent is able to learn all complex features of that problem.

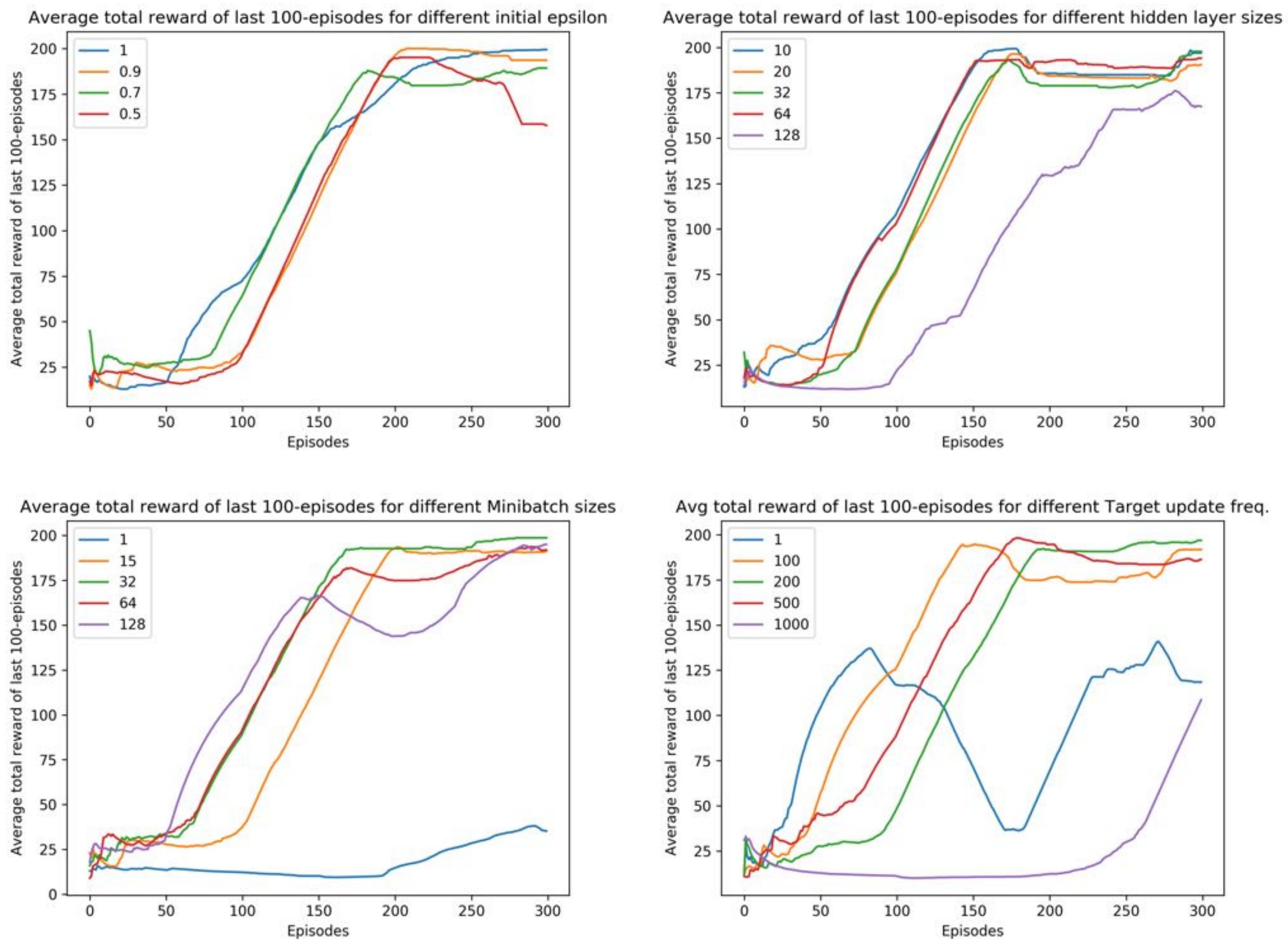


Figure 16: Plots comparing variation in learning curve for different values of hyperparameters a) Initial epsilon b) Hidden layer size c) Minibatch size and d) Target network update frequency

- Figure (c) compares learning curve in terms of average total reward for different values of minibatch sizes from set {1, 15, 32, 64, 128}. We can observe that if the minibatch size is less like 1 then it will take large time to learn, and if minibatch size is more like ≤ 32 then it is able to learn faster and smoothly. But if minibatch size is large like 64 or 128, then it could learn faster initially but with such large batch size it could end up picking bad tuples and thus we see that there are oscillations in their learning curve. So in general it is better to use some intermediate value for batch size so that we can take advantage of previous experience which reduces correlation and smooths out learning.

- Figure (d) compares learning curve in terms of average total reward for different values of target network update frequency from set {1, 100, 200, 500, 1000}. We can observe that if the target update frequency is very high like 1 then there is lot of variation and oscillations in learning curve as the sequence of observations are highly correlated. And if the target update frequency is very low like 1000, then there is very less correlation and thus we observe that there is little variation in learning curve but since update frequency is low it will take more time to learn. When the target update frequency is intermediate like between 100 & 500 then there is less correlation and we can observe that variation in learning curve also decrease when target networks are updates less frequently. So in general it is better to use some intermediate value for target update frequency so that correlation with target is reduced which helps in stabilizing the training process.

Bonus Q4: Variations on removing experience replay and/or the target network

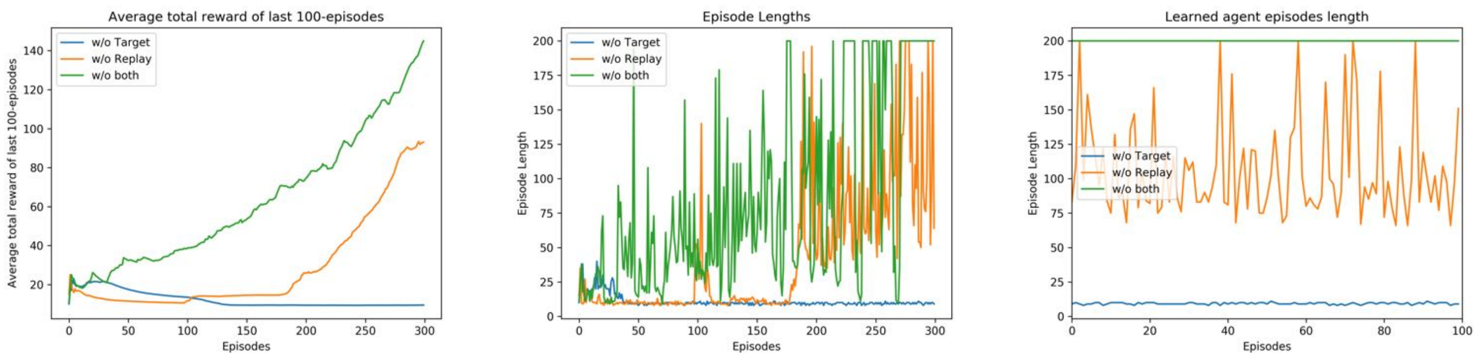


Figure 17: Plot comparing variation in learning curves by removing a) Target network b) Experience replay memory and c) Both experience replay and target network.

- From Figure (a) we can observe that by removing target network and keeping replay memory, we will be using same Q-values which are constantly shifting to compute target from samples taken from replay memory and hence there will be high correlation as a result the network becomes unstable and diverges thus not able to learn decent policy. When only experience replay memory is removed then we will be using only current observation which will be highly correlated to update Q-values as a result there learning process is slowed and thus we can observe that it is able to learn average reward of around 95 in 300 episodes. When both experience replay memory and target networks is removed we observe that learning process is slowed down compared to when both are used but since our problem is simple and we are not using any deep neural network it is able to learn good policy.
- Similarly from figure (b) we can observe that when only target network is removed, the network becomes unstable and it is not able to learn because of high correlation in target which are computed from samples from replay memory. When only replay memory is removed, we observe that there is high variance in episode lengths this is because observation are correlated. When both replay memory and target networks is removed we observe that initially the episode lengths are very low and only after 180 episodes it is able to learn good policy. The reason i think it is able to learn good policy in this case is because this problem is simple and neural network is shallow.
- Figure (c), shows episode length of learned agent after 300 episodes of training. We observe that removing target network gives poor result, removing replay memory gives good result and removing both gives best result.

References:

1. Sutton, Richard S., Doina Precup, and Satinder Singh. "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning." Artificial intelligence 112, no. 1-2 (1999): 181-211.
2. Sutton, Richard S., Doina Precup, and Satinder P. Singh. "Intra-Option Learning about Temporally Abstract Actions." In ICML, vol. 98, pp. 556-564. 1998.
3. Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves et al. "Human-level control through deep reinforcement learning." Nature 518, no. 7540 (2015): 529.
4. Aharutyu. Four room gridworld environment
5. Castellini Jacopo. Some GridWorld environments for OpenAI Gym
6. Viswanathgs. Simple DQN git on GitHub