CS6700 - Reinforcement Learning - Written Assignment 4

Atul Balaji - EE16B002

May 5, 2019

1 Question 1

We are given a grid-world with different rooms and options to exit the rooms into others. The per-step reward is -1. In addition, the agent can also pick from the four primitive actions. It is also given that the agent does not learn faster with options. The reason for this behaviour may be that the options defined above only allow the agent to learn policies to exit the rooms in minimum number of steps (due to -1 per step reward). As a result, exploration within the room is discouraged and any rewards which may be obtained due to such exploration is not received because of these policies.

2 Question 2

According to the paper on safe state abstraction[3] in MaxQ, there are five conditions for safe state abstraction - Subtask irrelevance, Leaf irrelevance, Result Distribution Irrelevance, Termination, Shielding. Of these, the first two conditions - **Subtask irrelevance** and **Leaf irrelevance** are still necessary when we do not use value function decomposition.
The result distribution irrelevance, termination and shielding conditions can be removed because their only function is to eliminate the need to maintain completion function values for some (subtask, state, subtask) pairs. This is not of importance outside the value function decomposition context.
The first two conditions, however are applicable even outside the value function decomposition context since they only involve eliminating state variables within a subtask.

3 Question 3

3.1 Advantages of A3C over DQN

- Experience replay used in DQN is computationally and memory intensive. In A3C, instead of experience replay, multiple agents are asynchronously executed in parallel, on multiple instances of the environment. Since at any given time step the parallel agents will be experiencing a variety of different states, this will decorrelate the training samples for the agents and the distribution tends to be more stationary, thereby stabilizing the learning.
- It allows the usage of on-policy methods such as SARSA, Actor-Critic, whereas DQN allows only off-policy methods.

3.2 Disadvantage of A3C compared to DQN

Different workers have a common objective function but do not apply parameter updates simultaneously and use a different instances of the environment. As some of the workers make updates, the objective function will keep changing. So, the workers which update later will update on the earlier version of the objective/loss function. The resulting parameter updates will not contribute to learning and are likely to reduce performance. This is a potential issue due to the asynchronous updates.

4 Question 4

Bottlenecks are states in the state space which allow access between two densely connected regions of the MDP state space, which might otherwise be isolated from each other. This connection is essential to achieve an appropriately defined goal. The agent tends to visit the bottlenecks frequently on successful paths to a goal but not on unsuccessful paths. Therefore, they act as useful sub-goals.
Since bottlenecks cut down exploration, in high dimensional spaces, using bottlenecks will result in options that perform poorly, since exploration is required to a high degree to converge quickly to optimality in a large state/action space. Also, the bottleneck approach will work only in cases where the overall goal can be achieved only using the primitive actions alone, without options. If this assumption is not true, the bottleneck method would fail [2].

5 Question 5

Model based methods involve maintaining a model (transition probabilities) of the environment, which is used for learning optimal policies. After a step of updating the model is done, the value functions are calculated using the current version of the model that has been learned. The model is updated when the agent interacts with the environment. Experience (interaction with the environment) can be used to either improve the model or to improve the value function. The model of the world could be used to augment the learning process using **planning algorithms** such as Dyna-Q and Dyna-Q+.
Other methods include **Simulation Based Search** [4], where the next state is determined by taking the current state (in the real environment) and runs multiple simulations starting at this state.
Another way is the **Monte Carlo Tree Search** (MCTS), where given the current state in environment (S_t), we take each possible a and generate simulations starting from (S_t, a). A Q function is calculated for (S_t, a) by averaging returns of the simulated episodes. The action that maximizes is then taken.

6 Question 6

In the QMDP method, the MDP corresponding to the given POMDP is solved and then the Q-function is converted to a value function over belief states, actions.

$$Q_a(b) = \sum_s b(s) Q_{MDP}(s, a) \tag{1}$$

Using the underlying MDP, we get a policy that is optimal for the MDP. This solution is optimal if we knew all the state at all points. But it may not be optimal for the POMDP, since we cannot observe it completely. The policy obtained will be optimal only when the POMDP is completely observable. This means that the belief state vector is a one-hot vector.

7 Question 7

- In the context of planning, exploration means trying actions that improve the model and exploitation means behaving optimally given the current model. In this example, after 3000 steps, a shortcut is created in the maze environment, which makes it easier to reach the goal in a smaller number of steps.
  The Dyna-Q agent does not change its model (if we use greedy policy) or changes it very little, randomly (if we use an ϵ-greedy policy). This means that even if the environment changes, the model used in Dyna-Q does not change.
  In Dyna-Q+, a count is kept of how many time steps have elapsed since a particular (s,a) pair have been tried in a real interaction with the environment. For those pairs for which this time is large, the chance of the model being incorrect is high and the agent is given a reward to encourage it to try these actions.
  Before 3000 time steps, Dyna-Q+ outperforms Dyna-Q because it encourages exploration to rarely visited states and therefore can find the optimal policy faster than Dyna-Q which proceeds greedily. However the difference between the two methods is quite small in this situation, since there are no changes to the environment.
  After 3000 time steps, the Dyna-Q+ agent is able to detect the change in environment and update its model, thereby performing better than Dyna, which is slow to find the new shortcut. This is why the difference in cumulative rewards between them becomes larger, as shown in graph.

- In the Dyna agents, transitions are started in (s,a) pairs selected uniformly at random from all previously experienced pairs. As the agent interacts with the environment, model updates are done, but in a random order. In order to outperform these methods, we can make use of **Prioritized Sweeping**.
  We maintain a queue of every (s,a) pair whose estimated value would change if updated. This would be a priority queue, prioritized by the magnitude of the change (P). If we are in state S and take action A which is given by a policy, let the next state be S' and resulting reward be R. The model for (S,A) is updated using R and S'. The change (P) is given by: $P = |R + \gamma max_a Q(S', a) - Q(S, A)|$.
  If this P is greater than some threshold, then the pair (S,A) is inserted in the queue with the priority P. After this, we apply n planning steps using the state-action pairs present in the priority queue.

References

1. https://towardsdatascience.com/advanced-reinforcement-learning-6d769f529eb3
2. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.21.2063rep=rep1type=pdf
3. https://papers.nips.cc/paper/1770-state-abstraction-in-maxq-hierarchical-reinforcement-learning.pdf
4. https://towardsdatascience.com/model-based-reinforcement-learning-cb9e41ff1f0d