CS6700 : Reinforcement Learning
Written Assignment #3

Deadline: ??

- This is an individual assignment. Collaborations and discussions are strictly prohibited.
- Be precise with your explanations. Unnecessary verbosity will be penalized.
- Check the Moodle discussion forums regularly for updates regarding the assignment.
- **Please start early.**

AUTHOR : Pranav Aurangabadkar.
ROLL NO : ED18S007

1. (3 marks) Consider the problem of solving POMDPs using Deep Reinforcement Learning. Can you think of ways to modify the standard DQN architecture to ensure it can remember histories of states. Does the experience replay also need to be modified? Explain.

Solution:

- For solving POMDPs using DQN weights θ architecture the Q values can be parameterized prior belief and the action Q(b, a| θ) or action-observation history h and action Q(h, a| θ). If some assumed distribution over the state action space is used then belief b is using the loss function
$L\left(b \mid a, \theta\right) = \left(r + \gamma max_a Q\left(b', a, \theta_i\right) - Q\left(b, a, \theta_i\right)\right)^2$.
- If the agent remembers histories of states the tuple generally represented in MDP with DQN as (s, a, r, s') state, action, reward and next state respectively, where the DQN learns the Q(s,a). If the histories observation- action pair are stored in experience replay the tuples would change to
$\left\{\left(s_i, a_i\right), \left(s_i, a_i, s_{i+1}, a_{i+1}\right), ..., \left(s_i, a_i, s_{i+1}, a_{i+1} ... s_{n-1}, a_{n-1}, s_n, a_n\right)\right\}$ i.e. start initially in state s_i and choose action a_i then next step the state is s_i, a_i, s_{i+1} and action taken is a_{i+1} and so on. DQN now learns Q values of history of observation action pair.

2. (4 marks) Exploration is often ameliorated by the use of counts over the various states. For example, one could maintain a visitation count $N(s)$, for every state and use the same to generate an intrinsic reward $(r_i(s))$ for visiting that state.

$$r_i(s) = \tau \times \frac{1}{N(s)}$$

However, it is intractable to maintain these counts in high-dimensional spaces, since the count values will be zero for a large fraction of the states. Can you suggest a solution(s) to handle this scenario? How can you maintain an approximation of the counts for a large number of states and possibly generalize to unseen states?

Solution: A visitation count $N(s)$, for every state and use the same to generate an intrinsic reward $(r_i(s))$ for visiting that state.

$$r_i(s) = \tau \times \frac{1}{N(s)}$$

- The high dimensional space can be discretized with a hash function ϕ which can be chosen appropriately to generalize across states and distinguish between states.
- We begin by initializing a hash table with state counts $N(s)$ to zero. During each iteration state action tuples are collected using policy π. Then compute and update hash codes using hashing for example SimHash,
- $\phi(s) = sgn(Ag(s))$ where A is standard normal distribution over m collected tuples. Then we update hash table as $n\left(\phi\left(s\right)\right) \leftarrow n\left(\phi\left(s\right)\right) + 1$. Now the intrinsic reward is modified to,

$$r_i(s) = \tau \times \frac{1}{N(\phi(s))}$$

- The approximation of the counts for high dimension would depend on the granularity of the discretization. If prior knowlegde about the state space is known then the hash function can be modified accordingly generalizing better.
- Another method is to use Auto-encoder. A Neural network can be trained to map states to hash values using Auto-encoder. This method is an improvement as even if similar states are mapped to same hash value the Autoencoder reconstructs them perfectly. The Autoencoder can be trained using modified loss function.

3. (5 marks) Suppose that the MDP defined on the observation space is k-th order Markov, i.e. remembering the last k observations is enough to predict the future. Consider using a belief state based approach for solving this problem. For any starting state and initial belief, the belief distribution will localize to the right state after k updates, i.e., the true state the agent is in will have a probability of 1 and the other states will have a probability of 0. Is this statement true or false? Explain your answer.

Solution:

4. (3 marks) Q-MDPs are a technique for solving the problem of behaving in POMDPs. The behavior produced by this approximation would not be optimal. In what sense is it not optimal? Are there circumstances under which it can be optimal?

Solution: 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. 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.

5. (3 marks) What are some advantages and disadvantages of A3C over DQN? What are some potential issues that can be caused by asynchronous updates in A3C?

Solution: 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.

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.

6. (6 marks) There are a variety of very efficient heuristics available for solving deterministic travelling salesman problems. We would like to take advantage of such heuristics in solving certain classes of large scale navigational problems in stochastic domains. These problems involve navigating from one well demarcated region to another. For e.g., consider the problem of delivering mail to the office rooms in a multi storey building.

   (a) (4 marks) Outline a method to achieve this, using concepts from hierarchical RL.

Solution:

- HEXQ algorithm can be modified to this problem. The paper titled Model Approximation for HEXQ Hierarchical Reinforcement Learning is used as reference.
- The position of the delivery man in the building is described by three variables, the floor-number, the room-number on each floor and the location-in-room.The delivery man has six primitive actions to move one cell North, South, East, West or pressing Up or Down.
- Task Hierarchy first orders the variables by their frequency of change as the delivery man takes exploratory random actions. Clearly the location-in-room will change most frequently followed by room-number. The floor-number will change rarely.
- It first orders the variables by their frequency of change as the robot takes exploratory ran- dom actions. Clearly the location-in-room will change most frequently followed by room-number. The floor-number will change rarely. HEXQ constructs a three level task hierarchy with one level for each variable. The subtasks are the ways to exit a room and require x smaller MDPs to find the different policies to reach each of the n exit states (d doorways, l lifts and goals). Exits are at the doorways, the lifts and the office rooms location.
- In constructing the second level of subtasks, HEXQ uses the Cartesian prod- uct of the block identifier from the level below and the next variable in the frequency ordering, the room-number, to define an abstract projected state space. The second level partition only contains one block, a whole floor. There are e exit state-action pairs at this level. For example, (state=north-west-room,action=(navigate-to-lift1,press-up)).
- The top level of the task hierarchy is a single MDP with only F abstracted floor states. For this problem HEXQ saves over an order of magnitude in value function storage space and converges an order of magnitude faster than flat Q-learning.

(b) (2 marks) What problems would such an approach encounter?

Solution: The problems encounter include varying coarseness of value function and varying the coarseness of exit states.
$V^*_m(s) = max_a \left[V^*_{m-1}(s) + E^*_m(g,a)\right]$ where $V^*_m(s)$ is the optimum value function for state s in subtask m The action-value function E that is learnt by HEXQ for each subtask.

$$E_m^*(g,a) = \sum_{s'} P^\pi(s' \mid g,a)\left[R_{exit} + V_m^*(s')\right]$$

7. (6 marks) This question may require you to refer to https://link.springer.com/content/pdf/10.1007/BF paper on average reward RL. Consider the 3 state MDP shown in Figure 1. Mention the recurrent class for each such policies. In the average reward setting, what are the corresponding ρ^π for each such policy ? Furthermore, which of these policies are gain optimal ?

   (a) (3 marks) What are the different deterministic uni-chain policies present ?

   Solution:

   (b) (3 marks) In the average reward setting, what are the corresponding ρ^π for each such policy ? Furthermore, which of these policies are gain optimal ?

   Solution:

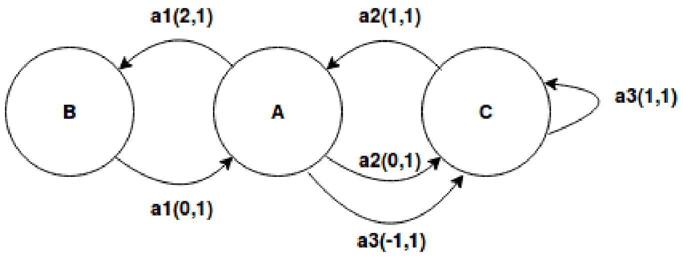

Figure 1: Notation : action(reward, transition probability). Example : a1(3, 1) refers to action a1 which results in a transition with reward +3 and probability 1