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 : Name. H.Vishal
ROLL NUMBER : MM16B023

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: To solve the problem of partial observability, it is not enough to consider information only at a single time step. Rather, maintaining a sequence history of the events helps to make better decisions. RNN is a popular framework used by the Deep Learning community to learn temporal dependencies. Therefore, one way to modify the DQN architecture adapt to partial observability with noisy observations is by combining LSTM with the standard DQN framework. The resultant Deep Recurrent Q-Network (DRQN) moves the temporal integration into the agent itself and relates information across data frames. Furthermore, by masking loss for first half of each sample set in every batch, more meaningful information can be sent through back-propagation. This DRQN when trained with partial observations, can have policies that can be generalized to represent complete observations.

   With regards to the experience replay, we shouldn't be sampling batches at random from the experience to develop temporal relations. Instead, we should be able to draw an experience of fixed length N (important hyper-parameter). This can be implemented by storing episodic histories in the replay buffer and drawing sample sets of fixed length from a random episode batch, thereby retaining random sampling and ensuring that the experience samples have some sort of connectivity.

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: Method **1**: Probabilistic Density Models

- Fit density model $P_\theta(\mathbf{s}) = \frac{N(\mathbf{s})}{n}$ to states $\mathcal{D}$ so far, where n is tot no. of visits
- Take a step i and observe s_i
- Fit new model $P_{\theta'}(\mathbf{s}) = \frac{N(\mathbf{s})+1}{n+1}$ to D $\cup$ s_i
- Use $P_\theta(\mathbf{s})$ and $P_{\theta'}(\mathbf{s})$ to estimate $\hat{N}(\mathbf{s})$ and compute $r_i(s) = \tau \times \frac{1}{\left(\hat{N}(\mathbf{s}) + \epsilon\right)^{1/2}}$

$P_\theta(\mathbf{s})$ and hence $\hat{N}(\mathbf{s})$ can be high even for new states!
The model described in Bellemare, et al. (2016) also assigns a diminishing probability to yet unseen states making $\hat{N}(\mathbf{s}) \neq 0 \ \forall \ \mathbf{s}$

Method **2**: Counting with hashes

The idea is to compress a state $\mathbf{s}$ into a k-bit code via $\phi(\mathbf{s})$ and then count $N(\phi(\mathbf{s}))$. The way how $\phi(\mathbf{s})$ is computed is mentioned in Tang et al. 2017. But the important point to note is, the hashing scheme is chosen to be locality preserving, i.e. similar states have a similar hash code. Therefore, by assigning the intrinsic reward function $r_i(\mathbf{s}) = f\big(N(\phi(\mathbf{s}))\big)$, states not visited even once can be assigned a intrinsic reward.

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: Consider a first order MDP. If at any state s, $b(s)$ be the belief state vector, z is the observable and a is the action to be taken, then next belief state vector can be calculated as the probability of being in state s' after $< b, a, z >$ i.e.

$$b(s') = Pr(s'|b, a, z) = \sum_s Pr(s|b, a, z).Pr(s'|b, a, z, s)$$

Applying Bayes' Theorem,

$$\implies b(s') = \sum_s b(s) Pr(s'|a, z, s) = \sum_s b(s) \frac{Pr(z|s')Pr(s'|a, s)}{Pr(z|a, s)}$$

$$\implies b(s') = \frac{\sum_s b(s) O(s', z) T(s, a, s')}{\sum_s \sum_{s'} b(s) O(s', z) T(s, a, s')}$$

where O and T are the observation and transition probabilities.

From the above equations, it is evident that it is not the belief and the action alone that determines the next belief state. Given the action and observation, we might be able to tell what the next state is, but an important factor to consider is that the observation itself has a probability associated with it. If we are considering an MDP that is kth order Markov, we'd essentially be considering sequences of observations that contain k entries and it is more likely that the O function is not 1 or 0 somewhere in between. Therefore, even for a deterministic environment, knowing $\langle$observation, action, observation, ...$\rangle$ doesn't always ascertain the future state.

Also, there is a possibility that the actions are non-deterministic. It adds a level of uncertainty as the observations don't allow the agent to determine with certainty the transitioned state after taking an action. This stochasticity will automatically make $b(s)$ smooth after k iterations, and any noise in the sensor outputs only makes the problem of state determination harder. Therefore, $b(s)$ after k updates might not be a one-hot vector but rather depend on the initial belief b and the observation and transition probabilities O, T $\implies$ the claim given in the question is **invalid**.

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: Policy through Q-MDP is obtained as follows:

$$\text{score}(a) = \sum bel(s) Q(s, a)$$

The policy is then obtained by picking a that maximizes score(a). This is obtained assuming we know information about the states. But given the uncertainty in POMDP, there might have been a better action to pick in terms of the total return. Therefore, the policy obtained is optimal only for the MDP corresponding to the POMDP, not the POMDP itself as the partial observability is not taken into account while obtaining the optimal policy. If we want it to optimal, we must know all the states of the POMDP with certainty, or in other words, $bel(s)$ must be 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: First off, A3C allows usage of on-policy methods which DQN doesn't. But the most important advantage of A3C over DQN is not having a replay memory buffer which can be computationally expensive and memory intensive otherwise. In A3C, the updates are asynchronous through different agents operating in parallel on multiple instances of the environment setting. At a particular time step, the agents operating in parallel de-correlate the training samples which represent diverse data, and therefore after experiencing a whole lot of different states, learning tends to be robust and stabilized (stationary distributions). Also, by asynchronously launching data, the process of data collection becomes faster.

The downside of A3C arises from the intrinsic nature of asynchronous updates. Different processing elements (PEs) have a common objective, but as some of the PEs make updates, the target function shifts. So, there will be some workers operating on an incorrect objective function. This leads to reduced performance.

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: Let us consider a finite number of rooms to which the mail needs to be delivered to. We suppose that the cost of delivering mails between 2 rooms of a particular floor is the euclidean distance between them. An agent, the travelling mailman has to find the cheapest way of delivering the mail to all rooms. So, the goal of the agent is to learn by hierarchical RL, the optimal policy.

Since it is a multi-storey building, we assume either escalators, lifts or stairs

are present to navigate between floors. Additionally, there might be corridors or hallways present in each floor that may be a central point connecting rooms.

Sub-problem description:

- Starting state is a funnel state that helps the agent navigate b/w floors
- There are n_i rooms that the agent has to visit in floor i
- a room reached by the agent is considered to be a final state if the agent has visited all the rooms in that floor so far
- the possible actions of the agent are to navigate between rooms, or to and fro b/w rooms and funnel states.
- an action is picked according to the ϵ-greedy method and stochastic effects are added
- the reward received by the agent in a given state (room) is considered to be the euclidean distance between the current room and the funnel state (say lift) which can help the agent navigate between rooms.
- After delivering to all the rooms in a particular floor, the state of the agent is reset to the starting funnel state

Ant-Q learning, a popular travelling salesman heuristic method can be adopted to learn the optimal policy for the sub-problem. As soon as the agent enters the building, it is positioned at one of the funnel states that can help it navigate between floors after completing delivery to all the rooms of a floor. Each funnel state would then correspond to an option (hard-coded). The agent through various trail runs explores and finds the various bottleneck states like corridors in the room making it solve the sub-problem progressively easier. Solutions of the sub-problems can be encapsulated as options and the agent can be kept moving one floor up (primitive action) solving sub-problems until it delivers to the final room in the final floor and receives a grand reward. But if the agent moves one floor up without covering all the rooms of a floor, a negative reward is assigned. On the environment, action and the option space defined, intra-option Q-Learning can be performed to update $Q(s, o)$ and recover the optimal policy from the Q function as $\pi(o) = max_o Q(s, o)$

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

Solution: Consider a situation where the office rooms are sparsely located or partitioned. It is possible that the most optimal solution for the agent is to take

the lift in one part of the building, deliver to all the rooms vertically , then walk across the hall in the top floor and deliver to the rooms in the other part of the building while descending down. If we are enforcing hierarchy, this can never be achieved.

7. (6 marks) This question may require you to refer to this 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 ?
   (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 ?

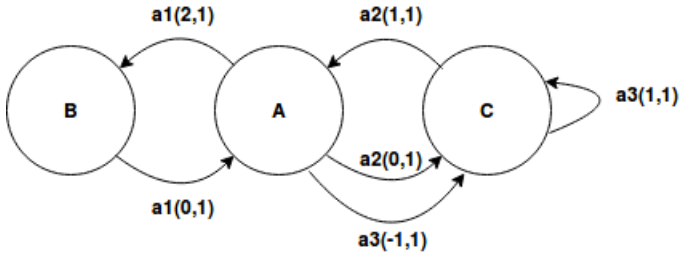

Figure 1
Notation : action(reward, transition probability). Example : a1(3, 1) refers to action a1 which results in a transition with reward +3 and probability 1

Solution: There are in total **5** different uni-chain policies

Uni-chain policies			
Policy no.	action at B	action at A	action at C
1	a1(0,1)	a1(2,1)	-
2	-	a2(0,1)	a2(1,1)
3	-	a2(0,1)	a3(1,1)
4	-	a3(-1,1)	a2(1,1)
5	-	a3(-1,1)	a3(1,1)

Average reward is calculated using

$$\rho^\pi(x) = \lim_{N\to\infty} \frac{E(\sum_{t=0}^{N-1} R_t^\pi(x))}{N}$$

For uni-chain policies, average reward is independent of the state, i.e.

$$\rho^\pi(x) = \rho^\pi(y) = \rho^\pi$$

Uni-chain policies		
Policy no.	Recurrent class	ρ^π
1	A,B	**1**
2	A,C	1/2
3	C	**1**
4	A,C	0
5	C	**1**

A policy π^* is gain-optimal if $\rho^{\pi^*}(x) \geq \rho^\pi(x)$ over all policies π and states x

$\Longrightarrow$ For the given setting, policies 1,3,5 are gain-optimal.