CS6700 : Reinforcement Learning
Written Assignment #2

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 : RAHUL. V
ROLL NUMBER : ME16B171

1. (3 points) Consider a bandit problem in which the policy parameters are mean μ and variance σ of normal distribution according to which actions are selected. Policy is defined as $\pi(a;\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{(a-\mu)^2}{2\sigma^2}}$. Derive the parameter update conditions according to the REINFORCE procedure (assume baseline is zero).

Solution: Given policy is :

$$\pi(a;\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\dfrac{(a-\mu)^2}{2\sigma^2}}$$

Update rule according to REINFORCE method is:

$$\theta_{t+1} \leftarrow \theta_t + \alpha_t \cdot (r_t - b_t) \cdot \left. \frac{\partial \ln \pi(a;\theta)}{\partial \theta} \right|_{\theta_t} \tag{1}$$

Here the policy parameters are: $\theta = [\mu, \sigma^2]$
Computing the partial differential of policy w.r.t θ

$$\frac{\partial \ln \pi(a)}{\partial \mu} = \frac{a-\mu}{\sigma^2} \tag{2}$$

$$\frac{\partial \ln \pi(a)}{\partial \sigma} = \frac{(a-\mu)^2}{\sigma^3} - \frac{1}{\sigma} \tag{3}$$

Update conditions are:

$$\begin{bmatrix} \mu_{t+1} \\ \sigma_{t+1} \end{bmatrix} \leftarrow \begin{bmatrix} \mu_t \\ \sigma_t \end{bmatrix} + \alpha_t \cdot r_t \cdot \begin{bmatrix} \frac{a-\mu_t}{\sigma_t^2} \\ \frac{(a-\mu_t)^2}{\sigma_t^3} - \frac{1}{\sigma_t} \end{bmatrix} \tag{4}$$

2. (6 points) Let us consider the effect of approximation on policy search and value function based methods. Suppose that a policy gradient method uses a class of policies that do not contain the optimal policy; and a value function based method uses a function approximator that can represent the values of the policies of this class, but not that of the optimal policy.

   (a) (2 points) Why would you consider the policy gradient approach to be better than the value function based approach?

   Solution: Policy gradient approach directly optimizes for policy unlike value based methods where they learn a value function and then infer a policy from it. Hence policy gradient methods tend to be more stable and less prone to failure.
   Policy gradient methods can also learn stochastic policies where as value function based methods can only represent deterministic policies. Also in high dimensional spaces and continuous action spaces value function based methods are highly inefficient as they need to optimize each time over the action space for action selection

   (b) (2 points) Under what circumstances would the value function based approach be better than the policy gradient approach?

   Solution: Most of the value function based approach are off-policy methods and policy gradient methods are on-policy methods. When the environment is expensive(for example a real environment where a failure would incur huge loses or the environment model is computationally expensive) we would want to make maximum use of the collected data. Which is only possible with off-policy methods. In these circumstances certain value function based methods are better than policy gradient approaches.

   (c) (2 points) Is there some circumstance under which either of the method can find the optimal policy?

   Solution: When the state and action spaces are small, both the methods could possibly approximate the optimal policy accurately.

3. (4 points) Answer the following questions with respect to the DQN algorithm:

   - (2 points) When using one-step TD backup, the TD target is $R_{t+1} + \gamma V(S_{t+1}, \theta)$ and the update to the neural network parameter is as follows:

$$\Delta\theta = \alpha(R_{t+1} + \gamma V(S_{t+1}, \theta) - V(S_t, \theta))\nabla_\theta V(S_t, \theta) \tag{5}$$

Is the update correct ? Is any term missing ? Justify your answer

Solution: No, the update is not correct as it doesn't use a target network and experience replay which are very important for the DQN to stabilize.
The correct update is :

$$\Delta\theta = \alpha \mathbb{E}_{(s,a,r,s')\sim D}(r + \gamma V(s', \theta^-) - V(s, \theta))\nabla_\theta V(s, \theta) \tag{6}$$

Here D is the experience replay buffer and θ_- are the target network parameters.

- (2 points) Describe the two ways discussed in class to update the parameters of target network. Which one is better and why?

Solution: The two ways to update parameters of target networks are:

1. The parameters of the Q network are copied to the target network periodically after every **C** steps
2. Soft Update : The parameters of the target network are slowly moved towards the parameters of Q network after each step.

$$\theta^- = \tau\theta + (1 - \tau)\theta^- \quad \text{where} \quad \tau << 1 \tag{7}$$

The soft update is better as it is more stable compared to the first update because the target is changing very slowly.

4. (4 points) Experience replay is vital for stable training of DQN.

(a) (2 points) What is the role of the experience replay in DQN?

Solution: Experience replay is used to address the non-stationary and highly correlation between consequent samples obtained by interacting with the environment. Whereas to prove convergence of a deep neural network we need the samples to be uncorrelated and stationary. By storing the experiences in a replay buffer and sampling uniformly from the buffer making it close to i.i.d sampling and improves convergence.

(b) (2 points) Consequent works in literature sample transitions from the experience replay, in proportion to the TD-error. Hence, instead of sampling transitions using a uniform-random strategy, higher TD-error transitions are sampled at a higher frequency. Why would such a modification help?

Solution: This method is called Prioritised Experience Replay. By sampling at higher frequency from transitions with high TD-error, we are giving more importance to cases where the estimates are poor compared to when the estimates are close the target network. Hence we train the DQN more on transitions with high TD-errors. this would help in quicker and more effective learning compared to uniform-random sampling.

5. (3 points) We discussed two different motivations for actor-critic algorithms: the original motivation was as an extension of reinforcement comparison, and the modern motivation is as a variance reduction mechanism for policy gradient algorithms. Why is the original version of actor-critic not a policy gradient method?

Solution:

- **Reinforcement Comparison:** In this method we update the preferences of each state-action pair by comparing the return get with a baseline and the actions are picked based on these prefences.

$$p_{t+1}(s_t, a_t) = p_t(s_t, a_t) + \beta[G_t - b_t] \tag{8}$$

- **Policy gradient:** In this method we evaluate the gradient of the expected return and update the parameters of the policy.

$$\theta_{t+1} = \theta_t + \alpha A^{\pi}(s_t, a_t)\nabla_{\theta} \log \pi(s_t, a_t; \theta) \tag{9}$$

In the reinforcement comparison method, the $G_t - b_t$ can be replaced by δ_t (TD - error) and extended into an actor-critic by learning the value function along with the policy. As there is no policy gradient involved in this version, it is not a policy gradient method.

6. (4 points) This question requires you to do some additional reading. Dietterich specifies certain conditions for safe-state abstraction for the MaxQ framework. I had mentioned in class that even if we do not use the MaxQ value function decomposition, the hierarchy provided is still useful. So, which of the safe-state abstraction conditions are still necessary when we do not use value function decomposition?

Solution: Dietterich specified **5** conditions for safe-state abstraction for the MaxQ framework.

1. Max Node Irrelevance

2. Leaf Irrelevance

3. Result Distribution Irrelevance

4. Termination

5. Shielding

We can make use of the MaxQ graph without using value function decomposition. However the 5 conditions are for the complete MaxQ framework i.e. MaxQ graph (hierarchy) and MaxQ value function decomposition. **Max Node Irrelevance** and **Leaf Irrevelance** safe-state abstraction condtions are still necessary when ew don't use value function decomposition as these are required to reduce state variables in a subtask.

7. (3 points) Consider the problem of solving continuous control tasks using Deep Reinforcement Learning.

   (a) (2 points) Why can simple discretization of the action space not be used to solve the problem? In which exact step of the DQN training algorithm is there a problem and why?

   Solution:
   Simple discretization of action space cannot be used because the number of actions increases exponentially with the number of degrees of freedom. For example, a 7 degree of freedom human arm with the coarsest discretization $a_i \in \{-k, 0, k\}$ for each joint leads to an action space with dimensionality
   $3^7 = 2187$.
   In DQN, to find the TD target we need to get the max Q value over the action space which in a continuous space requires iterative optimisation process at each step, even if the action space is discretised, we still need to book-keep or evaluate the Q value for a high dimensional action space.

   (b) (1 point) How is exploration ensured in the DDPG algorithm?

   Solution: If the action space is discrete we can ensure exploration by using $\epsilon-$*greedy* algorithm. If the action space is continuous, we can use an exploration policy μ' by adding noise sampled from a noise process $\mathcal{N}$ to our actor policy.

   $$\mu'(s_t) = \mu(s_t|\theta_t^\mu) + \mathcal{N} \tag{10}$$

   $\mathcal{N}$ can be chosen to suit the environment.

8. (3 points) Option discovery has entailed using heuristics, the most popular of which is

to identify bottlenecks. Justify why bottlenecks are useful sub-goals. Describe scenarios in which a such a heuristic could fail.

Solution: a bottleneck is a region in the agent's observation space that the agent tends to visit frequently on successful paths to a goal but not on unsuccessful paths. Let's consider the example of 4 room grid world, if we don't use bottlenecks as sub goals, the agent would spend a lot of time exploring the initial room before it moves to another room which would delay the learning process. However if we used bottlenecks as sub-goals to discover the option we could speed up the learning process.
Using the same example, if there exists another goal state with higher reward in the far corner of the same initial room, and we are using options based on bottlenecks, the agent would take a long time learn that as the option is hindering exploration in the same room. Hence it fails.