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 : H Vishal
ROLL NUMBER : MM16B023

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 $\pi(a;\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(a-\mu)^2}{2\sigma^2}}$. Derive the parameter update conditions according to the REINFORCE procedure (assume baseline is zero).

Solution:

$$\text{Update rule:} \quad \theta_{t+1} \leftarrow \theta_t + \alpha_t (r_t - \cancelto{0}{b_t}) \nabla_{\theta_t} ln(\pi_t(a_t))$$

$$\theta_t = \begin{pmatrix} \mu_t \\ \sigma_t \end{pmatrix}$$

Mean

$$\frac{\partial\, ln(\pi_t(a_t))}{\partial \mu_t} = -\frac{1}{2}\cancelto{0}{\frac{\partial\, ln(2\pi\sigma_t^2)}{\partial \mu_t}} - \frac{1}{2}\frac{\partial}{\partial \mu_t}\frac{(a_t-\mu_t)^2}{\sigma_t^2}$$

$$\implies \frac{\partial\, ln(\pi_t(a_t))}{\partial \mu_t} = \frac{(a_t-\mu_t)}{\sigma_t^2}$$

Variance

$$\frac{\partial\, ln(\pi_t(a_t))}{\partial \sigma_t} = -\frac{1}{2}\frac{\partial\, ln(2\pi\sigma_t^2)}{\partial \sigma_t} - \frac{1}{2}\frac{\partial}{\partial \sigma_t}\frac{(a_t-\mu_t)^2}{\sigma_t^2}$$

$$\implies \frac{\partial\, ln(\pi_t(a_t))}{\partial \sigma_t} = -\frac{1}{\sigma_t} + \frac{(a_t-\mu_t)^2}{\sigma_t^3}$$

Updates

$$\mu_{t+1} = \mu_t + \alpha_t r_t \frac{(a_t - \mu_t)}{\sigma_t^2},$$

$$\sigma_{t+1} = \sigma_t - \alpha_t r_t \left\{ \frac{1}{\sigma_t} - \frac{(a_t - \mu_t)^2}{\sigma_t^3} \right\}$$

$$\implies \boxed{\begin{pmatrix} \mu_{t+1} \\ \sigma_{t+1} \end{pmatrix} = \begin{pmatrix} \mu_t + \alpha_t r_t \dfrac{(a_t - \mu_t)}{\sigma_t^2} \\ \sigma_t - \dfrac{\alpha_t r_t}{\sigma_t} \left[1 - \left(\dfrac{a_t - \mu_t}{\sigma_t} \right)^2 \right] \end{pmatrix}}$$

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: Value function approach finds deterministic policies by picking actions that maximizes the value function for various states. There is no such maximization involved in the case of the policy gradient (PG) approach where only the parameters are learned. This provides scope to learn stochastic policies, and since the given scenario presents a non-existent optimal policy, the PG approach would work better as stochasticity would enable a wider search to arrive at a policy closer to the optimal one. Also, in continuous space, value function methods can become computationally expensive. Working directly with probability distributions can avoid this problem.

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

   Solution: If distributions are not easy to sample from and the optimal policy is difficult to be parameterized, value function based approach would be better. Also, PG method has the potential risk of getting stuck at local optima. If the optimal policy is such that in a significant number of states it is sufficient

to have Q value maximum to pick the optimal action, value function based approach is more likely to arrive at a closer to optimal policy.

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

Solution: As described above, if it suffices for the optimal policy to have Q values maximum over state action pairs to identify the optimal action, the value function based approach would converge to optimality. But however approximate the policy learned through PG method is, it cannot find 'the optimal policy' as in the case of value function method, although extremely small state-action spaces would be suitable for quick convergence.

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:

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

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

Solution: The complete DQN parameter update is,

$$\Delta\theta = \alpha E_{<s,a,s',r>\sim D}\Big[\big(r + \gamma V(s', \theta^-) - V(s, \theta)\big)\nabla_\theta V(s, \theta)\Big]$$

Corrections: Samples are drawn from replay memory D to make use of experience replay. If target function and the estimated value function have the same weight parameters, we'd essentially be shooting at a moving target. To address this issue, target function is assigned a different weight parameter θ^- to bring in stability to the DQN setup.

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

Solution: Periodic Update: Replace the weights of $\hat{Q}$ by that of Q after a certain fixed number of steps i.e. $\theta^- = \theta_{k-1}$

Single step update: The target network is updated once per main episode update i.e. $\theta^- = \tau\theta + (1-\tau)\theta$; $\tau \ll 1$. In the second method, the target values change gradually as opposed to changing in a single go like in the first method.

Updating the target network slowly makes learning stable, and therefore, the single step update is better than the periodic update.

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: The agent's experience at each time step $< s_t, a_t, s_{t+1}, r_t >$ is stored in a data set D known as the replay buffer. To tackle the problem of non-stationary distributions, batches are drawn from D, generating a random experience to break any temporal correlations consecutive samples. This makes the behaviour distribution to be averaged over many states and result in smooth learning with reduced bias. Also, since each experience can be used for multiple updates, higher data efficiency can be realized.

   (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: Uniform sampling gives equal importance to all experiences stored in the buffer. But there might be some experiences which are more important, and experience replay would overwrite them with recent transitions due to fixed buffer size. TD error indicates the measure of difference between current value and the next bootstrap estimate. So by giving importance to high TD-error transitions, more informative transitions can be identified. The agent can then learn effectively through these select transitions with very few replays of redundant and less-informative transitions. Therefore by adopting a prioritized experience replay, learning can be made much faster.

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: Modern motivation

$$\pi_t(s,a) = \frac{e^{\phi(s,a)^T\theta}}{\Sigma_b\ e^{\phi(s,b)^T\theta}}$$

$$\theta_{t+1} = \theta_t + \alpha\Big(\phi(s,a) - E_{\pi_\theta}(\phi(s,a)\Big)Q_w(s,a),$$

Original method

$$\pi_t(s,a) = \frac{e^{p(s,a)}}{\Sigma_b\ e^{p(s,b)}}$$

$$p(s_t,a_t) \leftarrow p(s_t,a_t) + \beta\delta_t\ ;\ \ \delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$$

In the modern variance reduction mechanism,

$$\phi(s,a) - E_{\pi_\theta}(\phi(s,a)) = \nabla_\theta\ ln\ \pi_\theta(s,a)$$

The original method uses TD error to update preferences instead of gradients w.r.t policy parameters, and therefore cannot be considered as 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: Sub-task relevance and leaf irrelevance are the 2 abstractions which would still be necessary to preserve the structure of MaxQ graph and maintain hierarchy. The remaining 3 abstractions are not required as their only purpose is to eliminate the need to maintain complete function values, which becomes irrelevant outside the context of value function decomposition. The preserved hierarchy can then be made use of by running a different learning algorithm on top of it.

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: In high dimensional spaces, if discretization is used, the number of states along each dimension over which the policy has to be learnt becomes exponentially huge: Reference

   $$\pi^Q(s) = \text{argmax}_a Q(\phi(s),a;\theta)$$

   The above equation becomes computationally expensive, as a highly iterative process is req if we consider a discretized version of continuous action spaces.

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

Solution: Exploration is ensured by adding noise sampled from a random process $\mathcal{N}$ (say Gaussian noise), which can be controlled independently:

$$a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$$

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: Bottlenecks have a high success rate, i.e. if the agent visits a bottleneck, it is most likely that it is on right path towards the goal. Since bottlenecks most often act as a bridge between densely connected regions, they can be used to define sub goals and make the agent cut down on exploration to save computational cost.

The following 2 scenarios don't root for bottlenecks as sub-goals:
(a) When there are no natural bottlenecks states,

(b) Large state-action spaces require higher degree of exploration, especially in online learning algorithms. Since bottlenecks reduces the exploratory behavior, it will result in options that will perform poorly and take long for convergence.