CS6700 : Reinforcement Learning
Written Assignment #2

Deadline: 31st July,2020

- 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 : Prajjwal Kumar
ROLL NUMBER : NA16B115

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: The policy is defined as

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

For checking how does mean and variance updates, we will find how does policy change with respect to mean and variance.

Mean:

$$\frac{\partial \ln\pi(a_n;\mu_n,\sigma_n^2)}{\partial \mu_n} = \frac{\partial}{\partial \mu_n}\left(-\frac{(a_n-\mu_n)^2}{2\sigma_n^2}\right) \tag{2}$$

$$= \frac{a_n-\mu_n}{\sigma_n^2} \tag{3}$$

Variance:

$$\frac{\partial \ln\pi(a_n;\mu_n,\sigma_n^2)}{\partial \sigma_n} = \frac{\partial}{\partial \sigma_n}(-\ln(\sqrt{2\pi\sigma_n}) + \frac{\partial}{\partial \sigma_n}\left(-\frac{(a_n-\mu_n)^2}{2\sigma_n^2}\right) \tag{4}$$

On solving this further, we get,

$$\frac{\partial \ln\pi(a_n;\mu_n,\sigma_n^2)}{\partial \mu_n} = -\frac{1}{\sigma_n} + \frac{(a_n-\mu_n)^2}{\sigma_n^3} \tag{5}$$

$$= \frac{1}{\sigma_n}\left[\left(\frac{a_n-\mu_n}{\sigma_n}\right)^2 - 1\right] \tag{6}$$

Hence, the updates for mean and variance are as follows:

$$\mu_{n+1} = \mu_n + \alpha_n r_n \left(\frac{a_n - \mu_n}{\sigma_n^2} \right) \tag{7}$$

$$\sigma_{n+1} = \sigma_n + \alpha_n r_n \frac{1}{\sigma_n} \left[\left(\frac{a_n - \mu_n}{\sigma_n} \right)^2 - 1 \right] \tag{8}$$

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: From policy gradient method, we already have the probability distributions of actions to be taken in the state. So, we can easily sample the actions from them. While for value function based approach need to use some method for selecting an action and then find the best one over the action space for all actions taken to estimate optimal action. Also, finding a policy by approximating a value function is found to be difficult to calculate.

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

   Solution: We are already given that value function based method uses a function approximator and there is no optimal policy in the class of policies represented by it. So, if it will have an optimal policy in the class, it would be easier to learn optimal behaviour for value function based method than the policy gradient one.

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

   Solution: In case of very small state-action space, then any of the methods might find the optimal policy.

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:

$$\Delta\theta = \alpha(R_{t+1} + \gamma V(S_{t+1}, \theta) - V(S_t, \theta))\nabla_\theta V(S_t, \theta) \tag{9}$$

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

Solution: The given update is not correct. Instead of that, it should be

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

The term $V(S_{t+1}, \theta^-)$ was missing.

We don't want our target to change continuously with the network parameters in order to have a successful training. But, if we don't do that, our target changes every time we update our parameters. So, we need to make the target Q-value parameters(θ^-) independent of estimated ones(θ). The target parameters are only updated at some intervals.

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

Solution: So, the two ways to update are:
1. Updating at once $\rightarrow$ The target network parameters are updated once per main network update.

2. Updating at intervals $\rightarrow$ The target network ($Q^)$ maintained for some time steps and then updated by weighted addition of the weight of target network and main training network.

$$\theta^- = \rho\theta- + (1 - \rho)\theta$$

where ρ is a hyperparameter lying between 0 and 1. This results into gradual conversion of target weights into main network weights and thus, good training.

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:
During training, the agent's experiences are stored at every time step repre-

sented by e_t at time t. It is given by:

$$e_t = (s_t, a_t, r_{t+1}, s_{t+1})$$

And from this data set of experience (replay memory), we randomly sample for the training and this is called experience replay. If the training is done only from consecutive samples of experience instead of random samples, the samples would be highly correlated and would therefore lead to overfitting, thus, inefficient learning. In order to break this correlation in consecutive samples, we use experience replay.
Also, while training a function approximator. it provides better 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 uses the fact that not all experiences are equal i.e. the agent can learn more from one experience than other. So, instead of random sampling, sampling is done in proportion to TD-error(measuring how far is the value from its estimate). So, higher the error, higher will be chances of learning and thus, there will be less time required for achieving the same level of accuracy.

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 version(Policy Gradient method):
So, this version parameterizes the policy π_θ with parameter θ. According to policy π, actions are generated using a softmax function:

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

On solving further, the update equation comes out to be:

$$p(s_t, a_t) \leftarrow p(s_t, a_t) + \beta\delta_t(1 - \pi_t(a_t|s_t))$$

Original version:
The probability update equation is:

$$p(s_t, a_t) \leftarrow p(s_t, a_t) + \beta\delta_t$$

where $\delta_t = r_{t+1} + \gamma V(s_{t-1}) - V(s_t)$ and β is a step size parameter.
Since, this update doesn't have the policy term $1 - \pi_t(a_t|s_t)$, so, it's 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: As per in the paper, there are total 5 conditions necessary for safe-state abstraction conditions - Max Node Irrelevance, Leaf Irrelevance, Result Distribution Irrelevance, Termination, Shielding. So, when value function decomposition is not used, the conditions of Max Node Irrelevance and Leaf Irrelevance are still necessary while other 3 aren't as their only function is to eliminate the need to maintain completion function values for some(sub task, states) pairs.

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: DQN works great with discrete action space. In DQN, we try to learn or estimate a value-action function.

$$Q^*(s,a) = E_s[r + \gamma max Q^*(s^{'}, a^{'})|s, a]$$

This is really simple when we have a list of actions to choose from. But in case of continuous actions, there will be infinite no of actions and estimating all of them will be impossible. Also, computation and individual comparison for all Q-values leads to non-stationary target values and unstable learning. This process is really exhaustive and computationally expensive.

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

Solution: The DDPG algorithm is an off-policy method that learns a Q-function and a policy to iterate over actions. It is basically Q-learning for continuous action spaces. For exploration, noise is added to the action itself since policy is deterministic. Ornstein-Uhlenbeck noise process is used for this.

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 are those states in the state space which are responsible for joining two densely connected regions of the MDP state space. Without them, they might be isolated from each other. But this connection is necessary for achieving a properly defined goal and agent on successful paths has more chances of visiting these states. So, bottlenecks act as useful sub-goals.

So, using bottlenecks result into connection between otherwise isolated regions, they lead to much lesser exploration and thus in some cases, it performs poorly in spaces of higher dimensions as exploration to a higher degree is necessary to achieve optimality in a large state space.

References:

1. https://towardsdatascience.com/4-ways-to-boost-experience-replay-999d9f17f7b6

2. https://arxiv.org/abs/1902.10250

3.https://medium.com/analytics-vidhya/naf-normalized-advantage-function-dqn-for-continuous-control-tasks-b9dcb6ebeab8