CS6700 : Reinforcement Learning
Written Assignment #2

Deadline: 31 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 : Pranav Aurangabadkar.
ROLL NUMBER : ED18S007

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 parameter update equation according to REINFORCE is
$\Delta\Theta_n = \alpha_n (R_n - b_n) \frac{\partial ln\Pi(a_n,\Theta_n)}{\partial\Theta}$ ... baseline $b_n = 0$
Policy is $\pi(a;\mu,\sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(a-\mu)^2}{2\sigma^2}}$ with parameters mean μ and variance $\nu = \sigma^2$.

With respect to μ $\quad \frac{\partial ln\Pi(a_t,\mu_t,\nu_t)}{\partial\mu_t} = \frac{\partial}{\partial\mu_t}\left(-\frac{(a_t-\mu_t)^2}{2\nu_t}\right) = \frac{a_t-\mu_t}{\nu_t}$.

With respect to ν $\quad \frac{\partial ln\Pi(a_t,\mu_t,\nu_t)}{\partial\nu_t} = \frac{\partial}{\partial\nu_t}\left(-ln\left(2\pi\nu_t\right)^{0.5}\right) + \frac{\partial}{\partial\nu_t}\left(-\frac{[a_t-\mu_t]^2}{2\nu_t}\right)$.

i.e $\quad \frac{\partial ln\Pi(a_t,\mu_t,\nu_t)}{\partial\nu_t} = -\frac{1}{2\nu_t} + \frac{[a_t-\mu_t]^2}{2\nu_t^2} = \frac{1}{2\nu_t}\left(\frac{[a_t-\mu_t]^2}{\nu_t} - 1\right)$

With respect to σ $\quad \frac{\partial ln\Pi(a_t,\mu_t,\sigma_t)}{\partial\sigma_t} = \frac{\partial}{\partial\sigma_t}\left(-ln\left(2\pi\right)^{0.5} - ln\sigma_t\right) + \frac{\partial}{\partial\sigma_t}\left(-\frac{[a_t-\mu_t]^2}{2\sigma_t^2}\right)$.

i.e $\quad \frac{\partial ln\Pi(a_t,\mu_t,\sigma_t)}{\partial\sigma_t} = \frac{-1}{\sigma_t} + \frac{(a_t-\mu_t)^2}{\sigma_t^3} = \frac{1}{\sigma_t}\left[\left(\frac{a_t-\mu_t}{\sigma_t}\right)^2 - 1\right]$

Thus the parameter updates are

For mean μ $\quad \mu_{t+1} = \mu_t + \alpha r_t \left[\frac{a_t-\mu_t}{\nu_t}\right]$

For variance $\nu = \sigma^2$ $\quad \nu_{t+1} = \nu_t + \frac{\alpha r_t}{2\nu_t}\left[\frac{(a_t-\mu_t)^2}{\nu_t} - 1\right]$

For standard deviation σ $\quad \sigma_{t+1} = \sigma_t + \frac{\alpha r_t}{\sigma_t}\left[\left(\frac{a_t-\mu_t}{\sigma_t}\right)^2 - 1\right]$

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:

- For large state action space or continuous state action policy representation is compact in the case of Policy Gradient methods whereas we require large number of parameters as compared and complex state action representation in case of value function methods.
- In case of parameterized form i.e value function using function approximator there is no guarantee of convergence except in case of linear parameterization. But in case of policy gradient based method it is guaranteed to converge at least to local mimima.
- Policy Gradients methods model probabilities of actions, it is capable of learning stochastic policies, while value based can't. Also, Policy Gradients can be easily applied to model continuous action space since the policy network is designed to model probability distribution, on the other hand, value based methods have to go through an expensive action discretization process which is undesirable.
- Policy gradient methods also include Actor-Critic approaches which learn both the policy and an associated value function. Thus policy gradient approach is considered better than the value function based approach.

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

Solution: Policy gradient approach drawback is sample inefficiency: since policy gradients are estimated from rollouts the variance is often extreme. Value function based approach can learn from any trajectory sampled from the same environment. In some (s,a) to take optimal actions, it is enough for the value of q(s, a) to be greatest. If the estimated policy is such that it has maximum value for (s,a) pairs which lead to optimality, then it can act as a proxy for the optimal policy. However, the policy obtained using the policy gradient method cannot behave similar to the optimal policy. Therefore, in this circumstance, the value function based approach outperforms the policy gradient method.

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

Solution: The value function method can behave like optimal policy for a set of q(s, a) values given the circumstance in (b). Unlike the value function based method, the policy gradient method can neither converge to the optimal policy nor behave like it. Only an approximation of the optimal policy can be done in the policy gradient method unlike the value function method. Therefore, only in the circumstance of (b) it is possible to find optimal policy (where value function method is used).

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 update equation is incorrect. In DQN
TD target is $[R_{t+1} + \gamma max_a \hat{q}(S_{t+1}, a_t, \theta)]$.
Current value is $\hat{q}(S_t, A_t, \theta)$ and this difference is TD error.
Here our TD target is dependent on function approximation parameters θ which is incorrect.
The missing term in above equation is that of Fixed Q-Targets i.e by fixing the function approximation parameters used to generate target. The correct equation is
$\Delta\theta = \alpha\left(R_{t+1} + \gamma max_a \hat{q}(S_{t+1}, a_t, \theta^-) - \hat{q}(S_t, A_t, \theta)\right)\nabla_\theta \hat{q}(S_t, A_t, \theta)$
where θ^- are parameters that are not changed during training. Practically this is done by saving a copy of θ into θ^- and updated after some time steps.
The update equation can also be stated in terms of state value as
$\Delta\theta = \alpha E_{(s,a,r,s')\sim D}\left(R_{t+1} + \gamma max_a V(S_{t+1}, \theta^-) - V(S_t, \theta)\right)\nabla_\theta V(S_t, \theta)$ where θ^- parameters of the target network, and D is replay buffer.

- (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 the parameters of target network are:

  - A target network Q' is maintained and after every K steps of updates, the parameters of Q-network are cloned into target network Q'. i.e $\theta' = \theta$.

– The target network parameters are updated once per main network update $\theta' = \rho\theta' + (1-\rho)\,\theta$ where $0 < \rho < 1$ is a hyperparameter.

The second method is better than the first. This is because in this case the target values are constrained to change slowly as it updates to the new value of θ' gradually in each step, which improves the stability of learning. The parameters of the target network are thus updated by having them slowly track the Q-Network. However, in the first update the target network parameters are constant for C steps and then updated all at once in the $K^{(th)}$ step.

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:

- At each time step we obtain a state, action, reward, next-state tuple and such tuples are stored in replay buffer. The experiences being obtained have high temporal correlation with previous experience as each action taken influences next state in a sequence which leads to an increase in variance and over-fitting. The main role is we can sample experiences randomly thus breaking this correlation and prevents action values from oscillating and diverging.
- As a result we are able to learn more from individual tuples multiple times, recall rare occurrences, and in general make more efficient use of observed experiences. Thus it is possible to generalize patterns across the state space for training of agent.

   (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:

- While training experience are stored in replay buffer as tuples of state, action, reward, next state. Some states are pretty rare to come by, some states are more important and some actions can be pretty costly. If we sample the batches uniformly then these experiences have very small probability of getting selected. Also buffers are limited in capacity older important experiences may get lost.

- Sampling batches with priorities helps overcome above shortcomings. TD error $\delta_t = R_{t+1} + \gamma max_a \hat{q}(S_{t+1}, a_t, \Theta) - \hat{q}(S_t, A_t, \Theta)$ is used to assign priorities to experiences as more the TD error the more we expect to learn from that experience. This helps to learn and converge 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: Policy gradient method (Modern motivation).
Policy gradient methods use a parameterization of the policy π_θ with parameter θ. The gradient of an objective function $J(\theta)$ is used for learning.
The expected return J is given by
$J(\theta) = \sum_\tau P(\tau;\theta) R(\tau)$ where τ is an arbitrary trajectory.
$\nabla_\theta J(\theta) = \nabla_\theta \sum_\tau P(\tau;\theta) R(\tau) = \sum_\tau \nabla_\theta P(\tau;\theta) R(\tau) = \sum_\tau \frac{P(\tau;\theta)}{P(\tau;\theta)} \nabla_\theta P(\tau;\theta) R(\tau)$
$\nabla_\theta J(\theta) = \sum_\tau P(\tau;\theta) \frac{\nabla_\theta P(\tau;\theta)}{P(\tau;\theta)} R(\tau) = \sum_\tau P(\tau;\theta) \nabla_\theta log P(\tau;\theta) R(\tau)$
The above equation is known as likelihood ratio policy gradient and can be aprroximated with a sample based average as
$\nabla_\theta J(\theta) \approx \frac{1}{n} \sum_{i=1}^{n} \nabla_\theta log P(\tau^i;\theta) R(\tau^i)$ where each τ^i is a sampled trajectory and where $P(\tau^i;\theta) = \prod_{t=0}^{T} \mathbb{P}\left(s_{t+1}^{(i)} \mid s_t^{(i)}, a_t^{(i)}\right) \pi_\theta\left(a_t^{(i)} \mid s_t^{(i)}\right)$ is probability of arbitrary trajectory τ^i
$\nabla_\theta log P(\tau^i;\theta) = \nabla_\theta \prod_{t=0}^{T} \mathbb{P}\left(s_{t+1}^{(i)} \mid s_t^{(i)}, a_t^{(i)}\right) \pi_\theta\left(a_t^{(i)} \mid s_t^{(i)}\right)$
$\nabla_\theta log P(\tau^i;\theta) = \nabla_\theta \sum_{t=0}^{T} log \mathbb{P}\left(s_{t+1}^{(i)} \mid s_t^{(i)}, a_t^{(i)}\right) + \nabla_\theta \sum_{t=0}^{T} log \pi_\theta\left(a_t^{(i)} \mid s_t^{(i)}\right)$
$\nabla_\theta log P(\tau^i;\theta) = \nabla_\theta \sum_{t=0}^{T} log \pi_\theta\left(a_t^{(i)} \mid s_t^{(i)}\right)$
Thus the gradient of the expected return J has gradient term with respect to policy parameter as below
$\nabla_\theta J(\theta) \approx \frac{1}{n} \sum_{i=1}^{n} \sum_{t=0}^{T} \nabla_\theta log \pi_\theta\left(a_t^{(i)} \mid s_t^{(i)}\right) R(\tau^i)$

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: The second condition Leaf Irrelevance, a set of state variables is irrelevant for a primitive option of a MAXQ graph if for all states s the expected value of reward function doesn't depend on any of the values of state variables. Thus its value function can be represented compactly. But when we do not use value function decomposition this condition is not necessary. The fourth condition Termination, It applies when a subtask is guaranteed to cause its parent task to terminate in a goal state. In a sense, the subtask is funneling the environment into the set of states described by the goal predicate of the parent task. The fifth condition Shielding condition arises from the structure of the MAXQ graph. These conditions aren't necessary as their only function is to eliminate the need to maintain completion function values for some (subtask, state, subtask) pairs.

According to the paper on safe state abstraction[3] in MaxQ, there are five conditions for safe state abstraction - Subtask irrelevance, Leaf irrelevance, Result Distribution Irrelevance, Termination, Shielding. Of these, the first two conditions - Subtask irrelevance and Leaf irrelevance are still necessary when we do not use value function decomposition.

The first two conditions, however are applicable even outside the value function decomposition context since they only involve eliminating state variables within 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: DQN Algorithm :
Step 1 : Initialize replay memory, action value function and target action value weights.
for episode in e - 1 to M:
Step 2 : Prepare input state.
for time step t - 1 to T:
Step 3: SAMPlE - Choose action A from state S using policy π epsilon greedily. Get reward and next state and store experience tuple in replay memory.
Step 4: LEARN - Get random batch of experiences from buffer.Set target$y_t = r_t + \gamma max_a \hat{q}(s_{t+1}, a, wts_{tgt})$
Update parameters. Every C steps update target wts.
If simple discretization of the action space is used in DQN there is a problem at STEP 4 while learning due to curse of dimensionality: the number of actions increases exponentially with the number of degrees of freedom. If the agent requires fine control of actions then agent requires finer grained disvretization

leading to explosion of number of actions. Such large action spaces are difficult to explore efficiently thus training DQN in this context is likely intractable since DQN relies on finding the action that maximizes the action value function.Additionally, naive discretization of action spaces needlessly throws away information about the structure of the action domain, which may be essential for solving many problems.

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

Solution: Exploration in DDPG is done by constructing a policy $\hat{\mu}$ by adding noise sampled from noise process $\mathbb{N}$ to the actor policy
$\hat{\mu}(s_t) = \hat{\mu}(s_t \mid \theta_t^{\mu}) + \mathbb{N}$
The authors used Ornstein-Uhlenbeck process.
Parameters are θ, μ and σ initialised and then the state x is added with noise dx given by
$dx = \theta\,(\mu - x) + \sigma * [ranarr]$
where ranarr is randomnly generated array of length x
$x = x + dx$.

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 surrogate measures of an option. They act like access states between two densely connected regions of the MDP state space. The agent visits bottlenecks frequently on successfull paths and not on unsuccessfull paths.Thus bottlenecks are useful heuristics as they reduce exploration, help in transfer learning and are reusable.Therefore, they act as useful sub-goals.
- The bottleneck approach will work only in cases where the overall goal can be achieved only using the primitive actions alone, without options. If this assumption is not true, the bottleneck method would fail.