ECE 2795: REINFORCEMENT LEARNING HOMEWORK ASSIGNMENT 3

Due September 25, 2023

1 ACTOR-CRITIC ALGORITHM (50PT)

In the first part of the homework we will work with the Actor-Critic algorithm using the advantage function. Read Chapter 13, Sections 5 to 7 of the RL book (`http://incompleteideas.net/book/RLbook2018.pdf`).

One-step Actor–Critic (episodic), for estimating $\pi_\theta \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^d$ (e.g., to $\mathbf{0}$)
Loop forever (for each episode):
  Initialize S (first state of episode)
  $I \leftarrow 1$
  Loop while S is not terminal (for each time step):
    $A \sim \pi(\cdot|S,\boldsymbol{\theta})$
    Take action A, observe S', R
    $\delta \leftarrow R + \gamma \hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$ (if S' is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat{v}(S,\mathbf{w})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} I \delta \nabla \ln \pi(A|S,\boldsymbol{\theta})$
    $I \leftarrow \gamma I$
    $S \leftarrow S'$

For this problem, we will use a Gaussian policy:

$$\pi(a|s,\theta) = \frac{1}{\sigma\sqrt{2\pi}} \exp(-\frac{(a-\mu(s,\theta))^2}{2\sigma^2}) \tag{1}$$

where the variance σ is constant, and the mean of the policy μ is expanded as a function approximation. To estimate the advantage function, we also need to expand the v-function as we did for the baselines. The attached Jupyter notebook proposes to use Gaussian radial bassis function approximations fro the policy and the value -function, but you can use your own if you tryied something different to solved the homework assignemnt 2.

1. Fill in the routine *actorcritic.py* to implement the actor-critic method for the mountain car problem.
2. Write the routines implementing the gradient $\nabla_\theta \log \pi_\theta$ and the updates for the policy parameters and the value function.
3. For each episode, plot the car's position in the horizontal axis versus time.
4. Upon convergence, plot the value function and the mean of the policy as a function of the position and the velocity. Use a colormap instead of a 3D plot. Comment your graphs.
5. Plot a colormap of the q-function as a function of the position and velocity and the action as a constant, for $action = -1$, and $action = 1$
6. Is this algorithm faster than reinforce? Explain why

2 Q-LEARNING (50PT)

Another type of Reinforcement Learning method is concerned with learning the optimal Value function and Q-value functions of the environment first and then obtain the polic. Then, using the (q)value functions, we can obtain an optimal policy by selecting the action that maximizes the q-funciton. This has some advantages over policy gradient techniqes, in that for discrete state/action systems, these algorithms are guaranteed to converge to the optimal solution from any starting guess. Read Chapter 6 of the RL book (`http://incompleteideas.net/book/RLbook2018.pdf`).

We will be implementing Q-learning in this homework.

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal

Fill in the functions in the second part of the Jupiter notebook, implementing the greedy policy and the q-learning update, and answer the following questions: Hint: be careful of the terminal step of an episode in Qlearning. What does the update look like?

1. (ϵ-Greedy Policy) Run Qlearning on MAP3. Create a plot where the x axis is the episode number and y axis is the total reward for the episode. Compare this with the output of the *evaluate_greedy_policy()* function. Is there a difference, and if so, why?
2. (Optimal Q values) Run Q learning on MAP2. Compute the Qvalue for the start state. (Hint, you know the optimal policy already, just need to add a few numbers to get the discounted return).

   a. Create a plot of the Qvalue for the start state during training (x axis is episodes, y axis is the Q-value of the start state). Also add a horizontal line at the optimal Q value. How many iterations does it take to converge on the true Qvalue?

   b. Save the Qvalue table at some earlier time before the Qvalues converge. You may find that even using Qvalues that are not converged can perform well when using the greedy policy. Why?
3. (Visualization) Run Q learning on MAP4. Create a matrix that is the same size as the map where each entry is the maximum Q value over actions for that state. Create the matrix after training for 0, 10, 100, 500, 1000 episodes. use *matplotlib.pyplot.imshow()* to visualize theses matrices.
4. (Reading check) Q learning seems very similar in structure/equations to Value iterations. Why use Q-learning at all? (The answer to this is why value iteration is not considered a Reinforcement Learning algorithm)