Problem set 3 (Soft Policy Iteration)

Due on March 25. Use your favorite programming language & send a report in pdf.

The goal of this problem set is to study the convergence properties of an RL algorithm we covered in class: Soft Policy Iteration. This algorithm performs a sequence of iterations $k = 1, 2, \ldots$ where the following operations are repeated:

1. Execute policy π_k and gather T sample transitions $\big((x_{k,1}, a_{k,1}, r_{k,1}), \ldots, (x_{k,T}, a_{k,T}, r_{k,T})\big)$,
2. using the above samples, build estimate $\widehat{Q}_k$ of the action value function Q^{π_k},
3. update policy $\pi_{k+1}(a|x) = \frac{\pi_k(a|x)\, e^{\eta \widehat{Q}_k(x,a)}}{\sum_{a'} \pi_k(a'|x)\, e^{\eta \widehat{Q}_k(x,a')}}$, where $\eta > 0$ is a stepsize (or learning-rate) parameter.

These updates are soft in the sense that they interpolate between greedy policy updates (corresponding to $\eta = +\infty$) and making no update ($\eta = 0$). The purpose of setting finite values of η is to deal with the effects of errors in approximating Q^{π_k}. As we have seen from the theoretical guarantees presented in class, choosing η to be too large or too small may result in poor performance: small values of η will result in slow convergence, whereas large values of η may result in unstable performance due to overly greedy updates. The purpose of this exercise is to show that these effects may indeed impact the empirical performance of this algorithm.

You are specifically asked to implement soft policy iteration in the MDP used in the previous exercises, and use LSTD with the piecewise linear feature map as a subroutine for approximate policy evaluation. The definitions are recalled at the end of the text for convenience.

Problem: Soft policy iteration

Using the LSTD implementation from the previous problem set, implement a soft policy iteration method where the initial policy is set as the uniform policy $\pi_1(a|x) = 1/|A|$ and the following steps are repeated in each iteration $k = 1, 2, \ldots, K$:

Policy evaluation: Evaluate the current policy π_k using LSTD and T sample transitions, and let $\widehat{V}_k$ denote the result. When generating the trajectories for LSTD, do not reset the initial state to a full queue, but continue where the previous trajectory left off; that is, for each $k > 1$, set $x_{k,1} = x_{k-1,T+1}$ with $x_{k-1,T+1} \sim P(\cdot|x_{k-1,T}, a_{k-1,T})$.

Policy improvement: Define the Q-function estimate as

$$\begin{aligned}\widehat{Q}_k(x,a) = r(x,a) &+ \gamma(1-p)\big(q(a)\,\widehat{V}_k(x-1) + (1-q(a))\,\widehat{V}_k(x)\big) \\ &+ \gamma p\big(q(a)\,\widehat{V}_k(x) + (1-q(a))\,\widehat{V}_k(x+1)\big)\end{aligned}$$

and set the new policy for all (x, a) using the formula

$$\pi_{k+1}(a|x) = \frac{\pi_k(a|x)\, e^{\eta \widehat{Q}_k(x,a)}}{\sum_{a'} \pi_k(a'|x)\, e^{\eta \widehat{Q}_k(x,a')}}.$$

Run the algorithm for a range of M values of the learning rate η spanning the interval $[0.01, 100]$ evenly on a logarithmic scale. (Such a range of values can be generated in numpy using the

command "`numpy.logspace(-2,2,num=M)`".) For each different η_m in the range $m = 1, 2, \ldots, M$, record the total amount of reward gathered during learning: $\widehat{R}_m = \sum_{k=1}^{K} \sum_{t=1}^{T} r_{k,t}$.

Your task is to run the above experiment for the choices $T = 10^5$, $K = 100$, $M = 100$ and plot the rewards as a function of η. (Hint: For a readable plot, it is recommendable to use a logarithmic scale on the η axis, which you can do in matplotlib with the command "`plt.xscale('log')`" after plotting your function, or by using "`plt.semilogx`" instead of "`plt.plot`".) What is your interpretation of the results? Which values of η perform poorly and which ones work well? Is there any improvement over "hard" approximate policy iteration for some value of η?

Remark: Note that the above specification does not exactly match a fully practical implementation of soft PI, since the goal was to keep things relatively simple for didactic purposes. In a practical situation, one is typically not able to calculate the Q-values from the value function estimates since the probabilities p and q may be unknown. Likewise, in a real application, you may not be able to afford a full loop over the state-action space to update the policies. In these cases, it is more practical to parametrize the Q-functions directly and estimate them via approximate DP methods, for instance LSTD for Q-functions (often called "LSTD-Q").

Definitions

The definition of the MDP to be used is the following:

The state space consists of the integers $0, 1, 2, \ldots, N-1$, with $N = 100$, corresponding to the length of the queue.

The action space consists of two actions: action a_{low} corresponding to a low service rate of $q(a_{\text{low}}) = q_{\text{low}} = 0.51$ and action a_{high} corresponding to a high service rate of $q(a_{\text{high}}) = q_{\text{high}} = 0.6$.

The reward function assigns a reward of $r(x, a) = -(x/N)^2 - c(a)$, with $c(a)$ being the cost of action a. This cost is defined for the two actions as $c(a_{\text{low}}) = 0$ and $c(a_{\text{high}}) = 0.01$.

The transition function describes the dynamics of the queue as follows. The arrival rate $p = 0.5$ models the rate at which new requests arrive into the queue and the controlled service rate models the rate at which requests leave the queue. In each round, the queue length increases by 1 with probability p and decreases by 1 with probability $q(a)$ corresponding to the action a taken. Precisely, letting x_t be the current queue length, a_t be the action taken by the agent, we define $I_t \in \{0, 1\}$ as the increment such that $\mathbb{P}[I_t = 1] = p$ and $S_t \in \{0, 1\}$ as the decrement such that $\mathbb{P}[S_t = 1] = q(a_t)$, the queue length is updated in each round as

$$x_{t+1} = \text{trunc}\,(x_t + I_t - S_t)\,,$$

where the "trunc" operator truncates the value to the interval $[0, N-1]$: $\text{trunc}(x) = \min\{N-1, \max\{x, 0\}\}$.

The discount factor will be set as $\gamma = 0.9$.

For LSTD, please use the piecewise linear feature map used in the previous problem set, whose definition is recalled below:

A coarse feature map: an $N/5$-dimensional feature vector with its i-th component defined as

$$\phi_i^{\text{coarse}}(x) = \mathbf{1}\{x \in [5(i-1), 5i-1]\}.$$

This feature map can represent piecewise constant functions on the state space.

A piecewise linear feature map: an $2 \times N/5$-dimensional feature vector with its first $N/5$ entries being equal to the previous feature map (i.e., $\phi_i^{\text{pwl}} = \phi_i^{\text{coarse}}$ for $i = 1, 2, \ldots, N/5$) and the second $N/5$ entries defined as

$$\phi_{N/5+i}^{\text{pwl}}(x) = \mathbf{1}\{x \in [5(i-1), 5i-1]\} \cdot (x - 5(i-1))/5,$$

for $i = 1, 2, \ldots, N/5$. The normalization constant $1/5$ is present to make sure that all features are bounded in $[0, 1]$. This feature map can represent piecewise linear functions on the state space.