6.820 Notes

Lecturer: Pulkit Agrawal

Andrew Liu

Spring 2024

Last updated on Friday 16th February, 2024.

Contents

1 February 8, 2024

1.1 Simple Decision Making

Classic Exploration Exploitation setup

The goal is to learn a policy $\pi(s_{0:t};\theta)$ that maps all the previous information into a best possible action. The objective in reinforcement learning is most generally the maximal sum of rewards over time:

$$\max_{\theta} r_t.$$

Compare this to the objective in normal supervised learning, which is to choose actions by maximizing a probability distribution:

$$\max_{\theta} p(a^{gt}, s_t).$$

If the reward function is unknown and/or not differentiable, then we have to use a different approach to attack these problems.

1.2 Multi-Arm Bandits

The setup here is that we have N arms corresponding to actions $a_1,\ldots,a_N$. The goal is to maximize rewards

$$\sum_{i=1}^{T} r(a_i^t),$$

where at each timestep we choose an action a_i and receive reward $r(a_i)$ according to some unknown reward function.

To measure performance, we want to know how well our model performs relative to the best possible outcome. Therefore, we are interesting in minimizing **regret**:

$$\left\| \sum_i r(a_i^*) - \sum_i r(a_i) \right\|.$$

1.2.1 Explore-First

The Explore-First strategy is pretty straightforward. Up to time K, explore the reward function by sampling each arm equally K/N times. Afterwards, choose the arm with highest average reward $\mu_i = (1/k_i)\sum_{k_i} r(a_i)$.

Under the assumption that $r \in [0,1]$, the worst regret that this could achieve is T. It turns out that the asymptotic performance of this algorithm is $T^{2/3}O(N\log T)^{1/3}$.

1.2.2 Upper Confidence Bound (UCB)

The idea with this algorithm is to instead create some confidence intervals about the reward function, and give an exploration bonus to actions whose confidence intervals are large. We choose actions with policy

$$a_{t+1} = \arg\max_i \mu_i(t) + \sqrt{\frac{4\log t}{k_i}},$$

where the exploration bonus here is scaling with $\sqrt{(1/k_i)}$. Where does this come from? Let's investigate ...