MS&E338 Reinforcement Learning Lecture 11 - May 7, 2018

Actor-Critic Methods, Experience Replay, and Bernoulli Bandits

Lecturer: Ben Van Roy *Scribe: Jiaming Zeng and Jamie Kang*

1 Policy Gradient

Policy gradients operate by nudging θ to improve the performance over time. As a recap from last time, we define the following policy gradient. In the following definition, θ is the parameter for value iteration and ζ is parameter for the policy.

$$V(\zeta) = \sum_{s\in S} \rho(s) V_{\pi_\zeta}(s) \tag{1}$$

$$\mu_\zeta(s) = E\Big[\sum_{t=0}^{\tau-1} \mathbb{1}(s_t = s) | \pi_\zeta\Big] \tag{2}$$

$$\nabla_\zeta V(\zeta) = \sum_{s\in S} \mu_\zeta(s) \sum_{a\in A} \nabla_\zeta \pi_\zeta(a|s) Q_{\pi_\zeta}(s,a) \tag{3}$$

Equation (1) is the weighted average of the rewards for an episode. Equation (2) is the expected number of times we visit state s. And Equation (3) is the gradient of $V(\zeta)$ after applying the chain rule. In (3), $\mu_\zeta(s)$ weights the states by how often they are visited, $\nabla_\zeta \pi_\zeta(a|s)$ tells us how to change ζ if we increase action a. The final term, $Q_{\pi_\zeta}(s,a)$ is one policy iteration over π_ζ.

In the stochastic approximation algorithm, we used an unbiased estimate for value of Q that gives us an approximation of the parametrized Q function.

$$\hat{Q}_t = \sum_{k=t}^{\tau-1} r_{k+1} \approx Q_{\pi_\theta}(s_t, a_t)$$

1.1 Actor-Critic Methods

In actor-critic methods, we modify the gradient expression. $\tilde{Q}_\theta$ is the critic that evaluates the actor, π, parametrized with ζ. Instead of using $\hat{Q}_t$, we use a parametrized value function to approximate it. Hence, we are approximating

$$\tilde{Q}_\theta \approx Q_{\pi_\zeta}$$

The following algorithm is a hybrid of the Q-learning methods and the actor-critic thinking.

Algorithm 1 actor-critic method for real-time Q-learning

```
Initialize for $\theta$, $\zeta$
for $l = 0, 1, 2, ...$ do
    Observe $s_0$
    for $t = 0, 1, 2, ...$ do
        Select $a_t \sim \pi_\zeta(\cdot|s_t)$
        Observe $r_{t+1}, s_{t+1}$
        $\theta := \theta + \gamma_l \nabla_\theta \tilde{Q}_\theta(s_t, a_t)(r_{t+1} + \max_{a\in A} \tilde{Q}_\theta(s_{t+1}, a) - \tilde{Q}_\theta(s_t, a_t))$
        $\zeta := \zeta + \gamma'_l(\frac{\nabla_\zeta \pi_\zeta(a_t|s_t)}{\pi_\zeta(a_t|s_t)})\tilde{Q}_\theta(s_t, a_t)$
```

There are three main motivations for the actor-critic method:

- Variance reduction of $\hat{Q}_t$. However, we do sacrifice by having some bias.
- Works better for real-time Q-learning with a huge action space. We parametrize the policy for action.
- Inspiration comes from neural science model, and there is a decomposition into actor and critic of θ and ζ.

As an improvement upon the current algorithm, we observe the following:

$$\begin{aligned}\sum_{a\in A}\nabla_\zeta \pi_\zeta(a|s) &= \nabla_\zeta \sum_{a\in A}\pi_\zeta(a|s)\\ &= \nabla_\zeta 1\\ &= 0\end{aligned}$$

Hence, we see that $Q_{\pi_\zeta}(s,a)+c$ would give us the same gradient expression. Therefore, we can modify equation (3) above with various offsets that can help with the convergence of the algorithm. One common offset is to subtract out the maximum Q value achievable from the actions, as follows:

$$\nabla_\theta V(\zeta) = \sum_{s\in S}\mu_\zeta(s)\sum_{a\in A}(\nabla_\zeta \pi_\zeta(a|s))(\tilde{Q}_\theta(s,a) - \max_{a'\in A}\tilde{Q}_\theta(s,a'))$$

We could use this to modify the ζ update in Algorithm 1.

$$\zeta := \zeta + \gamma'_l(\frac{\nabla_\zeta \pi_\zeta(a_t,s_t)}{\pi_\zeta(a_t,s_t)})(\tilde{Q}_\theta(s_t,a_t) - \max_{a'\in A}\tilde{Q}_\theta(s,a'))$$

2 Experience Replay

In recent years, experience replay has come to play a more prominent role. Experience replay is the idea of having a replay buffer that stores the past experiences. Let us define **replay buffer**

$$D = \{(s,a,r,s'), ...\}$$

In extreme cases, a replay buffer would keep all the old data. However, in practice, having a large replay buffer can significantly hurt performance since we need to iterate through all the past experiences.

In experience replay, we apply Q-learning updates on samples (or minibatchs) of experiences. At each iteration, we define an approximate Bellman operator

$$\tilde{F}Q_\theta = \arg\min_{Q_{\bar\theta}} \sum_{(s,a,r,s')\in D}\Big(Q_{\bar\theta}(s,a) - (r + \max_{a'\in A}Q_\theta(s',a'))\Big)^2$$

The approximate value iteration becomes:

$$Q_\theta := \tilde{F}Q_\theta$$

One of the problems is that minimized squared error causes a ton of time between episodes. A lot of work still need to be done in this field.

3 Thompson Sampling

Common exploration schemes we have looked at so far (eg. ϵ-greedy, Boltzmann) can be incredibly inefficient. For an illustrative example, recall the problem where we tried to balance a pole on a cart. Reward shaping may be useful if we have some prior intuition, but it is in general very complex and hard. Exploring intelligently, namely via “Deep Exploration” can therefore make a huge difference, which leads us to our discussion of Thompson Sampling in this and upcoming lectures. More detailed explanations can be found here: `https://web.stanford.edu/~bvr/pubs/TS_Tutorial.pdf`.

Time		1	2	$\cdots$
Action		j	k	$\cdots$
Observation		\$1	\$0	$\cdots$
Prior		Updates		
(α_1, β_1)	$\rightarrow$	(α_1, β_1)	(α_1, β_1)	$\cdots$
$\vdots$				
(α_j, β_j)	$\rightarrow$	$(\alpha_j + 1, \beta_j)$	(α_j, β_j)	$\cdots$
$\vdots$				
(α_k, β_k)	$\rightarrow$	(α_k, β_k)	$(\alpha_k, \beta_k + 1)$	$\cdots$

Table 1: Parameter updates for each coin

3.1 Multi-Armed Bandit

Let's consider a Beta-Bernoulli Bandit problem. There are K different coins we can flip, each showing heads with probability $p_1, ..., p_K \sim$ Beta distribution. If we get heads, we earn \$1. If we get tails, we earn nothing. In other words,

$$r_t = \begin{cases} 1 \text{ w.p. } p_{x_t} \\ 0 \text{ w.p. } 1 - p_{x_t} \end{cases}$$

where x_t is the action selected at time t (i.e. coin flipped at time t). Our goal is:

$$\max \quad \sum_{t=1}^{T} r_t$$

for some large T (i.e. long horizon). We start with a Beta prior distribution. Then for each coin i,

$$\text{Beta density}_{(\alpha_i, \beta_i)} \propto p_i^{\alpha_i - 1}(1 - p_i)^{\beta_i - 1}$$

.

- **Why do we use Beta distribution?** Most prominent reason is that Beta distributions are simple to work with and provide a straightforward intuition. First, note the following update (illustrated in Table 1:

  $$\text{If you flip coin}_i \text{ and get: } \begin{cases} \text{Heads, update } \alpha_i := \alpha_i + 1 \\ \text{Tails, update } \beta_i := \beta_i + 1 \end{cases}$$

  Hence, the distribution's parameters, (α, β), can be interpreted as the numbers of Heads and Tails (or success and failure) respectively. Moreover, Beta $(\alpha = 1, \beta = 1) =$ Unif $(0, 1)$, and thus we initially begin with a uniform distribution. Lastly, as we gain more data, the probability density will concentrate more and more at a single point.

  In Figure 1 [1] the agent is confident about the mean rewards of actions 1 and 2, which have been played over one thousand times, as depicted by their probability densities concentrated at 0.6 and 0.4 respectively; whereas the agent is still uncertain about the mean reward of action 3, which has been played only three times.

- **How should we pick action (i.e. coin) at each time?** Some most obvious (but not efficient) mechanisms are:

[1] From Figure 2.2 in Daniel J. Russo, Benjamin Van Roy, Abbas Kazerouni, Ian Osband and Zheng Wen (2018), A Tutorial on Thompson Sampling, Foundations and Trends in Machine Learning (`https://web.stanford.edu/~bvr/pubs/TS_Tutorial.pdf`).

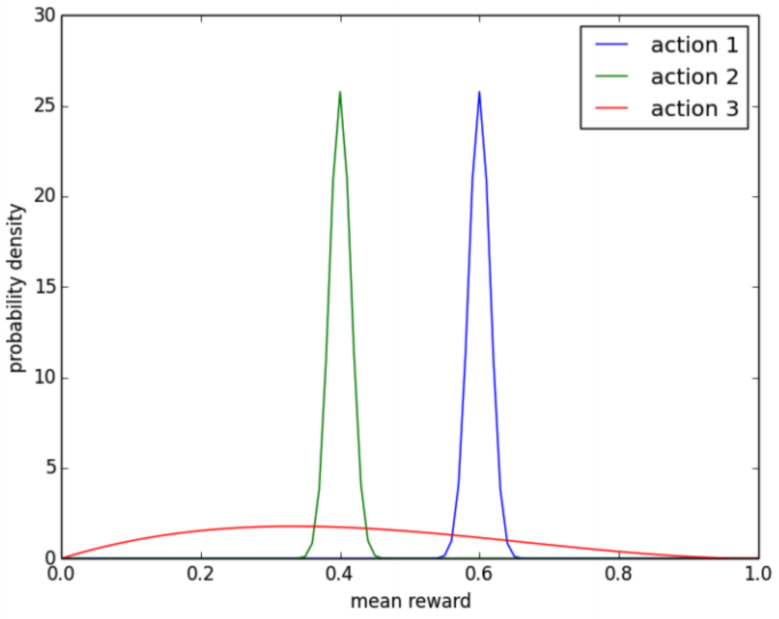

Figure 1: Probability density functions over mean rewards.

1. Greedy algorithm: choose a coin with the highest expectation by taking

$$\arg\max_{x_t} \mathbb{E}[r_t|\mathbb{H}_{t-1}, x_t] = \arg\max_{x_t} \frac{\alpha_{x_t}}{\alpha_{x_t} + \beta_{x_t}}$$

   for Beta-Bernoulli bandits.

   Issue: it can get stuck. There may be a good coin but you may never figure that out. For example, consider:

$$p_1 = 0.5, (\alpha_1 = 1, \beta_1 = 1) \to (\alpha_1 = 1, \beta_1 = 1) \text{ with expected reward } \frac{1}{2} \to \cdots$$

$$p_2 = 0.6, (\alpha_2 = 1, \beta_2 = 1) \to (\alpha_2 = 1, \beta_2 = 2) \text{ with expected reward } \frac{1}{3} \to \cdots$$

   Obviously coin 2 is optimal, but with probability $1 - 0.6 = 0.4$ it will give you tails. If this happens at time 1, then you will likely get stuck and repeatedly flip coin 1, which is suboptimal, and never learn that coin 2 is better.

2. ϵ-greedy: choose a greedy action w.p. $1 - \epsilon$ and choose uniformly w.p. ϵ.

   Issue: this will keep playing coin 1 even when we know that it is bad. We may resolve this by annealing, but it is still inefficient when the action space $|A|$ is very large. For instance, consider the case where we have $1,000,000$ coins and we already know that coin 1 and coin 2 are much better than others. We want to focus effort on something we want to learn (i.e. which one is better between coin 1 and 2), but ϵ-greedy fails to do this in an efficient manner.

3. Boltzmann. *Issue*: it does not factor in how uncertain you are about other coins. Referring back to Figure 1, we see that the agent is pretty certain that action 2 is worse than action 1 and thus there is no need to explore action 2 further. However, Boltzmann will explore actions 2 and 3 equally since they have the same expected reward.

There are two heuristics that can resolve these issues and will be discussed in the next lecture.