Reinforcement Learning

Notes

Jingye Wang

wangjy5@shanghaitech.edu.cn

Spring 2020

Contents

3 Bandit Algorithms

The large part of this section was done with references [1, 4, 7, 8, 9].

3.1 Bandit Models

As a special case of Reinforcement Learning, bandit algorithms share some important concepts of that.

Reward: *A reward R_t is a scalar feedback signal which indicates how well agent is doing at step t.*

Reinforcement learning is based on the reward hypothesis that *all goals can be described by the maximization of expected cumulative reward.* Furthermore, Depends on how well we know the reward, there are three types of feedback.

Bandit Feedback: *the agent only knows the reward for the chosen arm;*

Full Feedback: *the agent knows the rewards for all arms that could have been chosen;*

Partial Feedback: *apart from the chosen arm, the agent knows rewards for some arms.*

Besides the feedback, the rewards model also varies, such as *IID rewards, adversarial rewards, constrained adversarial rewards, and stochastic rewards.*

3.2 Stochastic Bandits

The basic settings of stochastic bandits are *bandit feedback* and *IID reward*. Also, we assume that per-round rewards are bounded. A multi-armed bandit $< \mathcal{A}, \mathcal{R} >$ are defined as follows:

- $\mathcal{A}$: a known set of m actions;
- $\mathcal{R}^a(r) = \mathbb{P}(r|a)$: an *unknown* probability distribution over rewards r;
- a_t: the action selected by the agent at time t and $a_t \in \mathcal{A}$;
- r_t: the reward generated by the environment at time t and $r_t \sim \mathcal{R}^{a_t}$;
- $\mathbb{E}\left[\sum_{\tau=1}^{t} r_\tau\right]$: the goal we want to maximize.

For convenience, we define

- $Q(a) = \mathbb{E}[r_t|a_t = a]$: the mean reward for the action a;
- $V^* = \max_{a\in\mathcal{A}} Q(a)$: the optimal reward for the action a;
- $L_\tau = \mathbb{E}[V^* - Q(a_\tau)]$: the regret for the round τ, indicating the opportunity loss.

With those definitions, the goal of the multi-armed bandit $< \mathcal{A}, \mathcal{R} >$ is equivalent to minimize

$$L_t = \mathbb{E}\left[\sum_{\tau=1}^{t}(V^* - Q(a_\tau))\right].$$

That is to say, minimize the total regret we might have by not selecting the optimal action up to the time step t. Another way to formulate the regret is about counting:

$N_t(a) = \sum_{\tau=1}^{t} I_{a_\tau=a}$: the number of selections for the action a after the end of the round t;

$\Delta_a = V^* - Q(a)$: the difference in the value between the action a and the optimal action a^*;

Then the regret follows that

$$L_t = \sum_{a \in \mathcal{A}} \mathbb{E}[N_t(a)]\Delta_a,$$

which is called *Regret Decomposition Lemma*. Depending on how the regret converges, we have

Linear regret: *The regret L_t over t rounds is s.t.*

$$\lim_{t \to \infty} \frac{L_t}{t} = 1.$$

Sublinear regret: *The regret L_t over t rounds is s.t.*

$$\lim_{t \to \infty} \frac{L_t}{t} = 0.$$

Logarithmic asymptotic regret: *The performance of any bandit algorithm is determined by similarity between optimal arm and other arms. Asymptotic total regret is at least logarithmic in number of steps*:

$$\lim_{t \to \infty} L_t \geq \log t \sum_{a | \Delta_a > 0} \frac{\Delta_a}{KL(\mathcal{R}^{\dashv} || \mathcal{R}^*)}.$$

The lower bound is also known as *Lai-Robbins* lower bound. Such lower bound can be obtained with advance knowledge of the gap.

The Exploration-Exploitation Dilemma: *As the action-values are unknown, we must both try actions to learn the action-values (explore), and prefer those that appear best (exploit).*

3.3 Greedy Algorithms

Greedy algorithm tends to take the best action most of the time, while doing exploration occasionally.

Greedy Action: *Define the greedy action at time t as*

$$a_t^{\varepsilon} = \arg\max_{a \in \mathcal{A}} Q(a).$$

If $a_t = a_t^{\varepsilon}$, we are making exploitation, otherwise, we are making exploration, i.e. selecting an arm arbitrarily.

Usually $Q(a)$ is estimated by Monte-Carlo evaluation:

$$\hat{Q}_t(a) = \frac{1}{N_{t-1}(a)} \sum_{\tau=1}^{t-1} r_\tau I(a_\tau = a).$$

ε-greedy algorithm: *Rather than making exploitation every round, we make exploitation with probability $1 - \varepsilon$, or make exploration with probability ϵ, which usually is a small number.*

Algorithm 1: ε-greedy algorithm

Initialization: $\hat{Q}(a) \leftarrow 0,\ N(a) \leftarrow 0,\ \forall a \in \mathcal{A}$;

Repeat:

$$a' \leftarrow \begin{cases} \arg\max_a \hat{Q}(a) & \text{with probability } 1-\varepsilon; \\ \text{a random action} & \text{with probability } \varepsilon; \end{cases}$$

$r \leftarrow bandit(a')$;

$N(a') \leftarrow N(a') + 1$;

$\hat{Q}(a') \leftarrow \hat{Q}(a') + \frac{1}{N(a')}(r - \hat{Q}(a'))$;

In practice, it is useful to initialize $\hat{Q}(a)$ with high values. Such trick is called *optimistic initialization*.

Decaying ε_t-greedy algorithm: *The greedy degree ε in decaying version is not a constant. Rather, we have a decay schedule for ε_t as*

$$\varepsilon_t = \min\left\{1, \frac{c|\mathcal{A}|}{d^2 t}\right\},$$

where $c > 0$ and $d = \min_{a|\Delta a>0} \Delta_i$.

For decaying ε-greedy algorithm, it usually has a better performance than the vanilla version, however, the schedule for ε_t requires advance knowledge of gaps Δ_a.

3.4 UCB Algorithms

In ε-greedy algorithm, we make exploration purely randomly, which may waste the opportunity to try out other options. To avoid such inefficient exploration, UCB algorithm allows us to pick the arm with the highest upper bound with high probability.

UCB algorithm: *Estimate an upper confidence bound $\hat{U}_t(a)$ for each $Q(a)$, i.e.*

$$Q(a) \leq \hat{Q}_t(a) + \hat{U}_t(a).$$

Then select actions with max upper confidence bound, i.e.

$$a_t = \arg\max_{a \in \mathcal{A}} \{\hat{Q}_t(a) + \hat{U}_t(a)\}.$$

In other words, UCB algorithm favors exploration of actions with a strong potential to have an optimal value. When we don't have any prior knowledge, the bound can be determined by *Hoeffding's Inequality*. It follows that

$$P(Q(a) > \hat{Q}_t(a) + \hat{U}_t(a)) \leq e^{-2N_t(a)U_t^2(a)}.$$

Since we want to pick a bound so that with high chances the true mean $Q(a)$ is blow the sample mean $\hat{Q}_t(a)$ + the upper confidence bound $\hat{U}_t(a)$, the right-hand side of the inequality should be a small probability, for example, a tiny threshold p, therefore solving for $\hat{U}_t(a)$ we have

$$\hat{U}_t(a) = \sqrt{\frac{-\log p}{2N_t(a)}}.$$

The upper bound $\hat{U}_t(a)$ is a function of $N_t(a)$ which means a larger number of trials $N_t(a)$ should give us a smaller bound $\hat{U}_t(a)$. Further, we want the small bound to have a high probability, which we can achieve by designing the p.

UCB1: *As we want to make more confident bound estimation with more rewards observed, the threshold p should be reduced in time. A reasonable idea is setting $p = t^{-4}$ and then we get UCB1 algorithm*

$$\hat{U}_t(a) = \sqrt{\frac{2\log t}{N_t(a)}},$$

$$a_t^{UCB1} = \arg\max_{a \in \mathcal{A}} \left\{ \hat{Q}(a) + \sqrt{\frac{2\log t}{N_t(a)}} \right\}.$$

The regret bound of UCB1 is

$$\lim_{t\to\infty} L_t \leq 8\log t \sum_{a|\Delta_a > 0} \Delta_a.$$

Algorithm 2: UCB1 algorithm

Initialization: $\hat{Q}(a) \leftarrow 0,\ N(a) \leftarrow 0,\ \forall a \in \mathcal{A}$;
For each $a \in \mathcal{A}$:
    $N(a) \leftarrow 1$;
    $\hat{Q}(a) \leftarrow bandit(a)$;
Repeat:
    $a' \leftarrow \arg\max_a \left(\hat{Q}(a) + c \cdot \sqrt{\frac{2\log t}{N(a)}}\right)$;
    $r \leftarrow bandit(a')$;
    $N(a') \leftarrow N(a') + 1$;
    $\hat{Q}(a') \leftarrow \hat{Q}(a') + \frac{1}{N(a')}(r - \hat{Q}(a'))$;

An intuitive explanation of why UCB works better than greedy algorithm lies in how it handle exploitation *v.s* exploration: when $\hat{Q}_t(a)$ is large, specifically, $\hat{Q}_t(a) \gg \hat{U}_t(a)$, it indicates a high expected reward and leads to exploitation; when $N_t(a)$ is small, concretely, $\hat{Q}_t(a) \ll \hat{U}_t(a)$, it indicates the action may have a great potential and leads to exploration.

So far we have made no assumptions about the reward distribution $\mathcal{R}$, and therefore we have to rely on the Hoeffding's Inequality for a very generalize estimation. If we are able to know the distribution upfront, we would be able to make better bound estimation.

3.5 Thompson Sampling Algorithms

Thompson Sampling algorithm makes use of the posterior to guide exploration. To be specific, at each time step, we want to select action a according to the probability that a is the optimal action:

$$\pi(a|h_t) = P(Q(a) > Q(a'),\ \forall a' \neq a|h_t),$$

where $h_t = a_1, r_1, ..., a_{t-1}, r_{t-1}$ is the history.

Thompson sampling: *Thompson sampling use Bayes' law to compute posterior distribution* $P(\mathcal{R}|h_t)$ *and compute the action-value function* $\hat{Q}(a) = f(\mathcal{R}_a)$*, then it selects the action*

$$a_t^{TS} = \arg\max_{a \in \mathcal{A}} \hat{Q}(a).$$

A naive example for Thompson sampling is for Beta-Bernoulli Bandit. It assumes that the action k being played produces a reward satisfying *Bern*(θ_k), and θ_k follows a *Beta*(α_k, β_k), where θ_k is known but we have α_k, β_k. Let a_t denote the action selected at time t and $r_t \in \{0, 1\}$ denote the corresponding result of action a_t. Then we can update the parameters of the *Beta* distribution

$$(\alpha_k, \beta_k) \leftarrow \begin{cases} (\alpha_k, \beta_k) & \text{if } a_t \neq k, \\ (\alpha_k, \beta_k) + (r_t, 1 - r_t) & \text{if } a_t = k. \end{cases}$$

Algorithm 3: Thompson Sampling algorithm

Initialization: $\alpha_a = |\mathcal{A}|, \beta_a = |\mathcal{A}|, \ \forall a \in \mathcal{A}$;
Repeat:
    For each $a \in \mathcal{A}$:
        Sample $\hat{Q}(a)$ from Beta(α_a, β_a);
    $a' \leftarrow \arg\max_a \hat{Q}(a)$;
    $r \leftarrow bandit(a')$;
    $\alpha_{a'} \leftarrow \alpha_{a'} + r$;
    $\beta_{a'} \leftarrow \beta_{a'} + 1 - r$;

For the details of how it works one can refer to the idea of *Beta Bernoulli Conjugate*. It exploits the power of Bayesian inference to compute the posterior and finally achieves a probability matching. However, for many practical and complex problems, it can be computationally intractable to estimate the posterior distributions with observed true rewards using Bayesian inference. In that case, we may need to approximate the posterior distributions using methods like Gibbs sampling and Laplace approximate.

3.6 Gradient Bandit Algorithms

Previously our bandit algorithms are based on the estimation of action values. while the gradient bandit algorithm is based on a new criterion.

Gradient Bandit Algorithm: *Let* $H_t(a)$ *be a learned preference for taking action* a*, then we have the rules*

$$P(A_t = a) = \frac{e^{H_t(a)}}{\sum_{b=1}^{k} e^{H_t(b)}} = \pi_t(a),$$

and

$$H_{t+1}(a) = H_t(a) + \alpha(r_t - \bar{r}_t)[I(A_t = a) - \pi_t(a)],$$

where $\bar{r}_t = \frac{1}{t}\sum_{i=1}^{t} r_i$ *and the learning rate* $\alpha > 0$.

The algorithm is also known as *stochastic gradient ascent algorithm* since the term $(r_t - \bar{r}_t)[I(A_t = a) - \pi_t(a)]$ is essentially the stochastic gradient of $\mathbb{E}[r_t]$ with respect to $H_t(a)$.

Algorithm 4: Gradient Bandit algorithm

Initialization: $R = 0, H(a) = 0, \forall a \in \mathcal{A}$;
Repeat $t = 1, 2, ...$:
    For each $a \in \mathcal{A}$:
        $\pi(a) = \frac{H(a)}{\sum_{a'} H(a')}$;
    Sample action a' from π;
    $r \leftarrow bandit(a')$;
    For each $a \in \mathcal{A}$:
        $H(a) = H(a) + \alpha(r - R)[I(a' = a) - \pi(a)]$;
    $R \leftarrow \frac{1}{t}[(t-1)R + r]$;

Actually, gradient bandit algorithm has a bad performance when compared with other algorithms. However, the introduction of the gradient and the function, inspire many policy gradient based algorithms in Reinforcement Learning.