Contents

Chapter 1

Bandits

The **multi-armed bandits** (MAB) setting is a simple but powerful setting for studying the basic challenges of RL. In this setting, an agent repeatedly chooses from a fixed set of actions, called **arms**, each of which has an associated reward distribution. The agent's goal is to maximize the total reward it receives over some time period.

States	Actions	Rewards
None	Finite	Stochastic

In particular, we'll spend a lot of time discussing the **Exploration-Exploitation Tradeoff**: should the agent choose new actions to learn more about the environment, or should it choose actions that it already knows to be good?

Example 1.0.1: Online advertising

Let's suppose you, the agent, are an advertising company. You have K different ads that you can show to users; For concreteness, let's suppose there's just a single user. You receive 1 reward if the user clicks the ad, and 0 otherwise. Thus, the unknown *reward distribution* associated to each ad is a Bernoulli distribution defined by the probability that the user clicks on the ad. Your goal is to maximize the total number of clicks by the user.

examples
nical trials,
ce, etc.

In this chapter, we will introduce the multi-armed bandits setting, and discuss some of the challenges that arise when trying to solve problems in this setting. We will also introduce some of the key concepts that we will use throughout the book, such as regret and exploration-exploitation tradeoffs.

1.1 Multi-Armed Bandits

Remark 1.1.1: Namesake

The name "multi-armed bandits" comes from slot machines in casinos, which are often called "one-armed bandits" since they have one arm (the lever) and take money from the player.

Let K denote the number of arms. We'll label them $1, \dots, K$ and use *superscripts* to indicate the arm index; since we seldom need to raise values to a power, this won't cause much confusion. For simplicity, we'll assume rewards are *bounded* between 0 and 1. Then each arm has an unknown reward distribution $\nu^k \in \triangle([0,1])$ with mean $\mu^k = \mathbb{E}_{r\sim\nu^k}[r]$.

Formally speaking, the agent's interaction with the MAB environment can be described by the following process:

```
for timestep in range(0, T):
    # Agent chooses an arm
    k = agent.choose_arm()

    # Environment generates a reward
    r = env.generate_reward(k)

    # Agent observes the reward
    agent.observe_reward(k, r)
```

What's the optimal strategy for the agent? Convince yourself that the agent should try to always pull the arm with the highest expected reward $\mu^\star := \max_{k\in[K]} \mu^k$.

The goal, then, is to minimize the **regret**, defined below:

Definition 1.1.1: Regret

The agent's **regret** after T timesteps is the difference between the total reward it observes and the total reward it *would* have received if it had always pulled the optimal arm:

$$\text{Regret}_T := \sum_{t=0}^{T-1} \mu^\star - \mu^{a_t} \tag{1.1}$$

Often we consider the **expected regret** $\mathbb{E}[\text{Regret}_T]$, where the randomness comes from the agent's strategy.

Ideally, we'd like to asymptotically achieve **zero regret**, i.e. $\mathbb{E}[\text{Regret}_T] = o(T)$.

Define Big O notation in appendix

1.1.1 Pure exploration (random guessing)

A trivial strategy is to always choose arms at random (i.e. "pure exploration"). What is the expected regret of this strategy?

$$\begin{aligned}
\mathbb{E}[\text{Regret}_T] &= \sum_{t=0}^{T-1} \mathbb{E}[\mu^\star - \mu^{a_t}] \\
&= T(\mu^\star - \bar{\mu}) > 0 \\
\text{where} \quad \bar{\mu} &:= \mathbb{E}[\mu^{a_t}] = \frac{1}{K}\sum_{k=1}^{K} \mu^k
\end{aligned}$$

1.1.2 Pure greedy

How might we improve on pure exploration? Instead, we could try each arm once, and then commit to the one with the highest observed reward. We'll call this the **pure greedy** strategy.

How does the expected regret of this strategy compare to that of pure exploration? For concreteness, suppose there's just two arms, with Bernoulli reward distributions given by $\mu^1 > \mu^2$.

Let's let r^1 be the random reward from the first arm and r^2 be the random reward from the second. If $r^1 > r^2$, then we achieve zero regret. Otherwise, we achieve regret $T(\mu^1 - \mu^2)$. Thus, the expected regret is simply:

$$\begin{aligned}
\mathbb{E}[\text{Regret}_T] &= \mathbb{P}(r^1 < r^2) \cdot T(\mu^1 - \mu^2) + c \\
&= (1 - \mu^1)\mu^2 \cdot T(\mu^1 - \mu^2) + c
\end{aligned}$$

Which is still $\Theta(T)$, the same as pure exploration!

Can we do better? For one, we could reduce the variance of the reward estimates by pulling each arm *multiple times*. This is called the **explore-then-commit** strategy.

1.1.3 Explore-then-commit

Let's pull each arm N_{explore} times, and then commit to the arm with the highest observed average reward. What is the expected regret of this strategy?

```
# exploration phase
for k in range(K):
    total = 0
    for i in range(N_explore):
        total += env.generate_reward(k)
    avg_reward[k] = total / N_explore
k_hat = argmax(avg_reward)

# exploitation phase
for t in range(T):
    r = env.generate_reward(k_hat)
```

(Note that the "pure greedy" strategy is just the special case where $N_{\text{explore}} = 1$.)

Let's analyze the expected regret of this strategy by splitting it up into the exploration and exploitation phases.

Exploration phase. This phase takes $N_{\text{explore}}K$ timesteps. Since at each step we incur at most 1 regret, the total regret is at most $N_{\text{explore}}K$.

Exploitation phase. This will take a bit more effort. We'll ultimately prove that:

1. For any total time T,
2. We can choose N_{explore} such that
3. With arbitrarily high probability, the regret is sublinear.

Let $\hat{k} := \arg\max_{k \in [K]} \hat{\mu}^k$ be the arm we choose to "exploit". We know the regret from the exploitation phase is

$$T_{\text{exploit}}(\mu^\star - \mu^{\hat{k}}) \qquad \text{where} \qquad T_{\text{exploit}} := T - N_{\text{explore}}K.$$

So we'd like to bound $\mu^\star - \mu^{\hat{k}} = o(1)$ in order to achieve sublinear regret. How can we do this?

Hoeffding's inequality tells us that, for a given arm k, since the rewards from that arm are i.i.d.,

$$\mathbb{P}\left(|\hat{\mu}^k - \mu^k| > \sqrt{\frac{\ln(2/\delta)}{2N_{\text{explore}}}}\right) \le \delta. \tag{1.2}$$

But note that we can't directly apply this to $\hat{k}$ since it's a random variable. Instead, we need to "uniform-ize" this bound across *all* the arms, i.e. bound the residual across all the arms simultaneously, so that the resulting bound will apply *no matter what* $\hat{k}$ "crystallizes" to.

The *union bound* provides a simple way to do this: The probability of error (i.e. the inequality 1.2) for a *single* arm is δ, so the probability that *any* of the arms is far from the mean is $K\delta$. Exchanging $\delta := K\delta$ and taking the complement of both sides, we have

$$\mathbb{P}\left(\forall k \in [K], |\hat{\mu}^k - \mu^k| \le \sqrt{\frac{\ln(2K/\delta)}{2N_{\text{explore}}}}\right) \ge 1 - \delta$$

Then to apply this bound to $\hat{k}$ in particular, we can apply the useful trick of "adding zero":

$$\begin{aligned}\mu^{k^\star} - \mu^{\hat{k}} &= \mu^{k^\star} - \mu^{\hat{k}} + (\hat{\mu}^{k^\star} - \hat{\mu}^{k^\star}) + \hat{\mu}^{\hat{k}} - \hat{\mu}^{\hat{k}} \\ &= (\mu^{k^\star} - \hat{\mu}^{k^\star}) + \underbrace{(\hat{\mu}^{k^\star} - \hat{\mu}^{\hat{k}})}_{\le 0 \text{ by definition of } \hat{k}} + (\hat{\mu}^{\hat{k}} - \mu^{\hat{k}}) \\ &\le 2\sqrt{\frac{\ln(2K/\delta)}{2N_{\text{explore}}}} \text{ with probability at least } 1 - \delta\end{aligned}$$

rate on this
d and in-
"optimal"
ore

So then setting $N_{\text{explore}} = \sqrt{T}$, for example, achieves sublinear regret, as desired.

The ETC algorithm is rather "abrupt" in that it switches from exploration to exploitation after a fixed number of timesteps. In practice, it's often better to use a more gradual transition, which brings us to the *epsilon-greedy* algorithm.

1.1.4 Epsilon-greedy

The **epsilon-greedy** algorithm is a simple modification of ETC that gradually transitions from exploration to exploitation. It works as follows:

```
for t in range(T):
    if random() < epsilon(t):
        # exploration
        k = random_choice(K)
    else:
        # exploitation
        # element-wise division
        k = argmax(total_reward / num_pulls)
    r = env.generate_reward(k)
    total_reward[k] += r
    num_pulls[k] += 1
```

Note that ϵ can vary over time. In particular we might want to gradually *decrease* ϵ as we learn more about the environment over time.

t optimal
n and re-
nalysis /
probability
d

In particular, in the ETC case, we had to set N_{explore} based on the total number of timesteps T. But the epsilon-greedy algorithm actually handles the exploration *automatically*: the regret rate holds for *any* t, and doesn't depend on the final horizon T.

But the way these algorithms explore is rather naive: we've been exploring *uniformly* across all the arms. But what if we could be smarter about it? In particular, what if we could explore more for arms that we're less certain about? This brings us to the **Upper Confidence Bound** (UCB) algorithm.

1.1.5 Upper Confidence Bound (UCB)

Intuitively, we'll estimate *confidence intervals* for the mean of each arm, and then choose the arm with the highest *upper confidence bound.* This operates on the principle of **optimism in the face of uncertainty**: we'll choose the arm that we're most optimistic about.

In particular, we'd like to compute some upper confidence bound M_t^k for arm k at time t and then choose $a_t := \arg\max_{k \in [K]} M_t^k$. But how should we compute M_t^k?

In our regret analysis for ETC, we were able to compute this bound using Hoeffding's inequality. But that required us to know the number of times we'd pulled each arm. In the UCB algorithm, we'll use a different approach: we'll compute a bound based on the *number of times we've pulled each arm so far.*

Let N_t^k denote the number of times arm k has been pulled within the first t timesteps, and $\hat{\mu}_t^k$ denote the sample average of those pulls. That is,

$$\begin{aligned} N_t^k &:= \sum_{\tau=t}^{t-1} \mathbf{1}\{a_\tau = k\} \\ \hat{\mu}_t^k &:= \frac{1}{N_t^k} \sum_{\tau=0}^{t-1} \mathbf{1}\{a_\tau = k\} r_\tau. \end{aligned}$$

However, note that we can't use Hoeffding's inequality to bound $\hat{\mu}_t^k$, since Hoeffding's inequality assumes that the *number of samples* is *fixed.* Here, though, it's random since it depends on the agent's actions, which in turn depend on the random previously observed rewards.

To get around this, we'll need to shift our focus from *time* to *number of samples.* In particular, we'll define $\widetilde{r}_n^k$ to be the nth sample from arm k, and $\widetilde{\mu}_n^k$ to be the sample average of the first n samples from arm k. This satisfies all the assumptions required for Hoeffding's inequality!

So how can we extend our bound on $\widetilde{\mu}_n^k$ to $\hat{\mu}_t^k$? Well, we know $N_t^k \le t$ (which would be the case if we had pulled arm k every time). So we can apply the same trick as last time, where we uniform-ize across all possible values of N_t^k. In particular, we have

$$\mathbb{P}\left(\forall n \le t, |\widetilde{\mu}_n^k - \mu^k| \le \sqrt{\frac{\ln(2t/\delta)}{2n}}\right) \ge 1 - \delta$$