CSCI567 Machine Learning (Spring 2021)

Sirisha Rambhatla

University of Southern California

April 16, 2021

Outline

Outline

Mulit-armed bandits: motivation

Imagine going to a casino to play a slot machine

- invariably it takes your money like a "bandit".

Of course there are many slot machines in the casino

- like a bandit with multiple arms (hence the name)
- **if I can play for 10 times, which machines should I play?**

Formal setup

There are K **arms** (actions/choices/...)

The problem proceeds in rounds between the environment and a learner: for each time $t = 1, \ldots, T$

- the environment decides the reward for each arm $r_{t,1}, \ldots, r_{t,K}$
- the learner picks an arm $a_t \in [K]$
- the learner observes the reward for arm a_t, i.e., r_{t,a_t}

Importantly, *learner does not observe rewards for arms not selected!*

This kind of limited feedback is now usually referred to as bandit feedback

Objective

Maximizing total rewards $\sum_{t=1}^{T} r_{t,a_t}$ seems natural

But the absolute value of rewards is not meaningful, instead we should compare it to some benchmark. A classic benchmark is

$$\max_{a \in [K]} \sum_{t=1}^{T} r_{t,a}$$

i.e. the largest reward one can achieve by always playing a fixed arm

So we want to minimize

$$\max_{a \in [K]} \sum_{t=1}^{T} r_{t,a} - \sum_{t=1}^{T} r_{t,a_t}$$

This is called the **regret**: *how much I regret for not sticking with the best fixed arm in hindsight?*

Balancing exploration vs. exploitation

A simple modification of "Greedy" leads to the well-known:

Upper Confidence Bound (UCB) algorithm

For $t = 1, \ldots, T$, pick $a_t = \mathrm{argmax}_a \ \mathrm{UCB}_{t,a}$ where

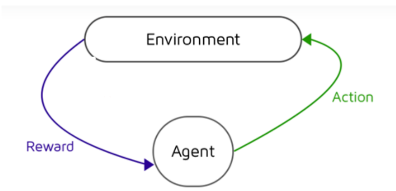
$$\mathrm{UCB}_{t,a} \triangleq \hat{\mu}_{t-1,a} + 2\sqrt{\frac{\ln t}{n_{t-1,a}}}$$

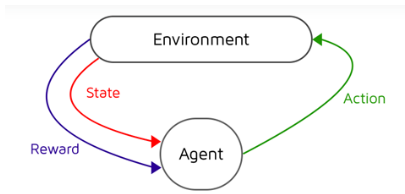
- the first term in $\mathrm{UCB}_{t,a}$ represents exploitation, while the second (bonus) term represents exploration
- the bonus term is large if the arm is not pulled often enough, which encourages exploration (adaptive due to the first term)
- a parameter-free algorithm, and *it enjoys optimal regret!*

Outline

Motivation

Multi-armed bandit is among the simplest decision making problems with limited feedback (**Bandit Feedback**).





It's often too simple to capture many real-life problems. One thing it fails to capture is the "state" of the learning agent, which has impacts on the reward of each action.

- e.g. for Atari games, after making one move, the agent moves to a different state, with possible different rewards for each action

Reinforcement learning

Reinforcement learning (RL) is one way to deal with this issue.

Huge recent success when combined with deep learning techniques

- Atari games, poker, self-driving cars, etc.

The foundation of RL is **Markov Decision Process (MDP)**, a combination of Markov model and multi-armed bandit

Markov decision process

An MDP is parameterized by five elements

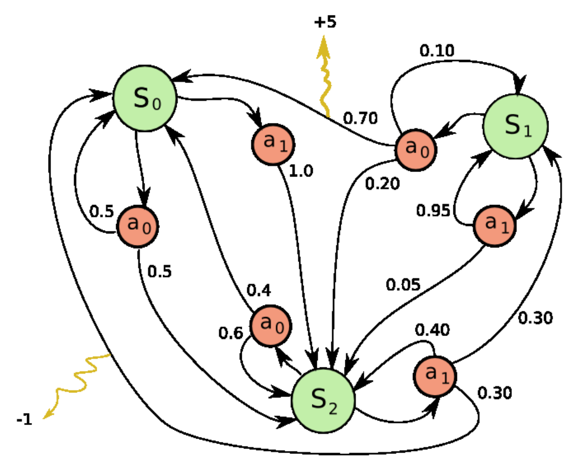
- $\mathcal{S}$: a set of possible states
- $\mathcal{A}$: a set of possible actions
- P: transition probability, $P_a(s, s')$ is the probability of transiting from state s to state s' after taking action a (Markov property)
- r: reward function, $r_a(s)$ is (expected) reward of action a at state s
- $\gamma \in (0, 1)$: discount factor, informally, reward of 1 from tomorrow is only counted as γ for today

Different from Markov models the state transition is influenced by the taken action.

Different from Multi-armed bandit, the reward depends on the state.

Example

3 states, 2 actions



Policy

A **policy** $\pi : \mathcal{S} \to \mathcal{A}$ indicates which action to take at each state.

If we start from state $s_0 \in \mathcal{S}$ and act according to a policy π, the discounted rewards for time $0, 1, 2, \ldots$ are respectively

$$r_{\pi(s_0)}(s_0), \quad \gamma r_{\pi(s_1)}(s_1), \quad \gamma^2 r_{\pi(s_2)}(s_2), \quad \cdots$$

where $s_1 \sim P_{\pi(s_0)}(s_0, \cdot), \quad s_2 \sim P_{\pi(s_1)}(s_1, \cdot), \quad \cdots$

If we follow the policy forever, the total (discounted) reward is

$$\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_{\pi(s_t)}(s_t)\right]$$

where the randomness is from $s_{t+1} \sim P_{\pi(s_t)}(s_t, \cdot)$.

Note: the discount factor allows us to consider an infinite learning process

Optimal policy and Bellman equation

First goal: knowing all parameters, *how to find the optimal policy*

$$\operatorname*{argmax}_{\pi} \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_{\pi(s_t)}(s_t)\right] \quad ?$$

We first answer a related question: *what is the maximum reward one can achieve starting from an arbitrary state s?*

$$\begin{aligned} V(s) &= \max_{\pi} \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_{\pi(s_t)}(s_t) \;\middle|\; s_0 = s\right] \\ &= \max_{a \in \mathcal{A}} \left(r_s(a) + \gamma \sum_{s' \in \mathcal{S}} P_a(s, s') V(s') \right) \end{aligned}$$

V is called the **value function**. It satisfies the above **Bellman equation**: $|\mathcal{S}|$ unknowns, nonlinear, *how to solve it?*

Value Iteration

Value Iteration

Initialize $V_0(s)$ randomly for all $s \in \mathcal{S}$

For $k = 1, 2, \ldots$ (until convergence)

$$V_k(s) = \max_{a \in \mathcal{A}} \left(r_s(a) + \gamma \sum_{s' \in \mathcal{S}} P_a(s, s') V_{k-1}(s') \right) \qquad (\textbf{Bellman upate})$$

Knowing V, the optimal policy π^* is simply

$$\pi^*(s) = \operatorname*{argmax}_{a \in \mathcal{A}} \left(r_s(a) + \gamma \sum_{s' \in \mathcal{S}} P_a(s, s') V(s') \right)$$

Convergence of Value Iteration

Does Value Iteration always find the true value function V? **Yes!**

$$\begin{aligned} |V_k(s) - V(s)| &= \Bigg| \max_{a \in \mathcal{A}} \left(r_s(a) + \gamma \sum_{s' \in \mathcal{S}} P_a(s, s') V_{k-1}(s') \right) \\ &\qquad - \max_{a \in \mathcal{A}} \left(r_s(a) + \gamma \sum_{s' \in \mathcal{S}} P_a(s, s') V(s') \right) \Bigg| \\ &\le \gamma \max_{a \in \mathcal{A}} \left| \sum_{s' \in \mathcal{S}} P_a(s, s') \left(V_{k-1}(s') - V(s') \right) \right| \\ &\le \gamma \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P_a(s, s') \left| V_{k-1}(s') - V(s') \right| \\ &\le \gamma \max_{s''} \left| V_{k-1}(s'') - V(s'') \right| \le \cdots \le \gamma^k \max_{s''} \left| V_0(s'') - V(s'') \right| \end{aligned}$$

So the distance between V_k and V is shrinking *exponentially fast*.

Learning MDPs

Now suppose we do not know the parameters of the MDP

- transition probability P
- reward function r

But we do still assume we can observe the states (in contrast to HMM), how do we find the optimal policy?

We discuss examples from two families of learning algorithms:

- **model-based** approaches
- **model-free** approaches

Model-based approaches

Key idea: learn the model P and r explicitly from samples

Suppose we have a sequence of interactions: $s_1, a_1, r_1, s_2, a_2, r_2, \ldots, s_T, a_T, r_T$, then the MLE for P and r are simply

$$P_a(s, s') \propto \#\text{transitions from } s \text{ to } s' \text{ after taking action } a$$
$$r_a(s) = \text{average observed reward at state } s \text{ after taking action } a$$

Having estimates of the parameters we can then apply value iteration to find the optimal policy.

Model-based approaches

How do we collect data $s_1, a_1, r_1, s_2, a_2, r_2, \ldots, s_T, a_T, r_T$?

Simplest idea: follow a random policy for T steps. This is similar to explore–then–exploit, and we know this is not the best way.

Let's adopt the ϵ-Greedy idea instead.

A sketch for model-based approaches

Initialize V, P, r randomly

For $t = 1, 2, \ldots,$

- **with probability** ϵ**, explore**: pick an action uniformly at random
- **with probability** $1 - \epsilon$**, exploit**: pick the optimal action based on V
- update the model parameters P, r
- update the value function V (via value iteration)

Model-free approaches

Key idea: do not learn the model explicitly. *What do we learn then?*

Define the $Q : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ function as

$$Q(s, a) = r_a(s) + \gamma \sum_{s' \in \mathcal{S}} P_a(s, s') \max_{a' \in \mathcal{A}} Q(s', a')$$

In words, $Q(s, a)$ is the expected reward one can achieve starting from state s with action a, then acting optimally.

Clearly, $V(s) = \max_a Q(s, a)$.

Knowing $Q(s, a)$, the optimal policy at state s is simply $\operatorname{argmax}_a Q(s, a)$.

Model-free approaches learn the Q function directly from samples.

Temporal difference

How to learn the Q function?

$$Q(s,a) = r_a(s) + \gamma \sum_{s' \in \mathcal{S}} P_a(s,s') \max_{a' \in \mathcal{A}} Q(s',a')$$

On experience $\langle s_t, a_t, r_t, s_{t+1} \rangle$, with the current guess on Q,
$r_t + \gamma \max_{a'} Q(s_{t+1}, a')$ is like a sample of the RHS of the equation.

So it's natural to do the following update:

$$\begin{aligned} Q(s_t,a_t) &\leftarrow (1-\alpha)Q(s_t,a_t) + \alpha\left(r_t + \gamma \max_{a'} Q(s_{t+1},a')\right) \\ &= Q(s_t,a_t) + \alpha \underbrace{\left(r_t + \gamma \max_{a'} Q(s_{t+1},a') - Q(s_t,a_t)\right)}_{\text{temporal difference}} \end{aligned}$$

α is like the learning rate

Q-learning

The simplest model-free algorithm:

Q-learning

Initialize Q randomly; denote the initial state by s_1.

For $t = 1, 2, \ldots,$

- **with probability ϵ, explore**: a_t is chosen uniformly at random
- **with probability $1-\epsilon$, exploit**: $a_t = \text{argmax}_a Q(s_t, a)$
- execute action a_t, receive reward r_t, arrive at state s_{t+1}
- update the Q function

$$Q(s_t,a_t) \leftarrow (1-\alpha)Q(s_t,a_t) + \alpha\left(r_t + \gamma \max_a Q(s_{t+1},a)\right)$$

for some learning rate α.

Comparisons

	Model-based	Model-free
What it learns	model parameters $P, r, \ldots$	Q function
Space	$O(\lvert\mathcal{S}\rvert^2\lvert\mathcal{A}\rvert)$	$O(\lvert\mathcal{S}\rvert\lvert\mathcal{A}\rvert)$
Performance	usually better	usually worse

There are many different algorithms and RL is an active research area.

Summary

A brief introduction to some online decision making problems:

- **Multi-armed bandits**
  - most basic problem to understand **exploration vs. exploitation**
  - algorithms: explore–then–exploit, ϵ-greedy, **UCB**
- **Markov decision process and reinforcement learning**
  - a combination of Markov models and multi-armed bandits
  - learning the optimal policy with a known MDP: **value iteration**
  - learning the optimal policy with an unknown MDP: model-based approach and model-free approach (e.g. **Q-learning**)