

CSCI567 Machine Learning (Fall 2020)

Prof. Haipeng Luo

U of Southern California

Nov 12, 2020

1 / 43

Outline

- 1 Review of last lecture
- 2 Multi-armed Bandits
- 3 Reinforcement learning

3 / 43

Administration

HW5 should be graded by the end of the week.

Quiz 2 coverage:

- non-MC: SVM, boosting, clustering, HMM, MLE and EM
- MC: all other topics, with the focus on materials after Quiz 1

Quiz 2 logistics:

- same as Quiz 1
- make sure to go to the assigned breakout room
- *submit before 7:30pm, no exception*

2 / 43

Review of last lecture

Outline

- 1 Review of last lecture
- 2 Multi-armed Bandits
- 3 Reinforcement learning

4 / 43

Hidden Markov Models

Model parameters:

- initial distribution**

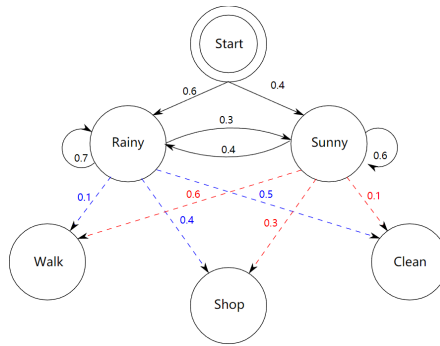
$$P(Z_1 = s) = \pi_s$$

- transition distribution**

$$P(Z_{t+1} = s' \mid Z_t = s) = a_{s,s'}$$

- emission distribution**

$$P(X_t = o \mid Z_t = s) = b_{s,o}$$



Baum–Welch algorithm

Step 0 Initialize the parameters (π, A, B)

Step 1 (E-Step) Fixing the parameters, **compute forward and backward messages for all sample sequences**, then use these to compute $\gamma_s^{(n)}(t)$ and $\xi_{s,s'}^{(n)}(t)$ for each n, t, s, s' .

Step 2 (M-Step) Update parameters:

$$\pi_s \propto \sum_n \gamma_s^{(n)}(1), \quad a_{s,s'} \propto \sum_n \sum_{t=1}^{T-1} \xi_{s,s'}^{(n)}(t), \quad b_{s,o} \propto \sum_n \sum_{t: x_t=o} \gamma_s^{(n)}(t)$$

Step 3 Return to Step 1 if not converged

Viterbi Algorithm

Viterbi Algorithm

For each $s \in [S]$, compute $\delta_s(1) = \pi_s b_{s,x_1}$.

For each $t = 2, \dots, T$,

- for each $s \in [S]$, compute

$$\delta_s(t) = b_{s,x_t} \max_{s'} a_{s',s} \delta_{s'}(t-1)$$

$$\Delta_s(t) = \operatorname{argmax}_{s'} a_{s',s} \delta_{s'}(t-1)$$

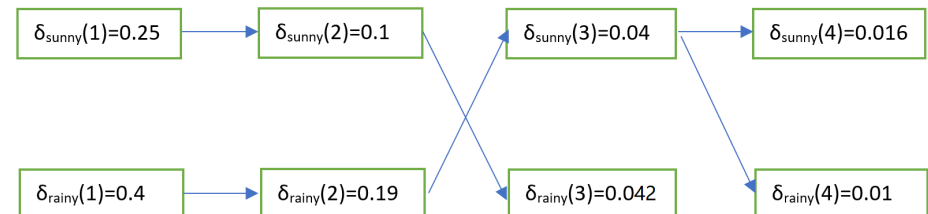
Backtracking: let $z_T^* = \operatorname{argmax}_s \delta_s(T)$.

For each $t = T, \dots, 2$: set $z_{t-1}^* = \Delta_{z_t^*}(t)$.

Output the most likely path z_1^*, \dots, z_T^* .

Example

Arrows represent the “argmax”, i.e. $\Delta_s(t)$.



The most likely path is **“rainy, rainy, sunny, sunny”**.

Viterbi Algorithm with missing data

Viterbi Algorithm with partial data $x_{1:T_0}$

For each $s \in [S]$, compute $\delta_s(1) = \pi_s b_{s,x_1}$.

For each $t = 2, \dots, T$,

- for each $s \in [S]$, compute

$$\delta_s(t) = \begin{cases} b_{s,x_t} \max_{s'} a_{s',s} \delta_{s'}(t-1) & \text{if } t \leq T_0 \\ \max_{s'} a_{s',s} \delta_{s'}(t-1) & \text{else} \end{cases}$$

$$\Delta_s(t) = \operatorname{argmax}_{s'} a_{s',s} \delta_{s'}(t-1).$$

Backtracking: let $z_T^* = \operatorname{argmax}_s \delta_s(T)$.

For each $t = T, \dots, 2$: set $z_{t-1}^* = \Delta_{z_t^*}(t)$.

Output the most likely path z_1^*, \dots, z_T^* .

Outline

- 1 Review of last lecture
- 2 Multi-armed Bandits
 - Online decision making
 - Motivation and setup
 - Exploration vs. Exploitation
- 3 Reinforcement learning

Decision making

Problems we have discussed so far:

- start with a training dataset
- learn a predictor or discover some patterns

But many real-life problems are about **learning continuously**:

- make a prediction/decision
- receive some feedback
- repeat

Broadly, these are called **online decision making problems**.

Examples

Amazon/Netflix/MSN **recommendation systems**:

- a user visits the website
- the system recommends some products/movies/news stories
- the system observes whether the user clicks on the recommendation

Playing games (Go/Atari/StarCraft/...) or **controlling robots**:

- make a move
- receive some reward (e.g. score a point) or loss (e.g. fall down)
- make another move...

Two formal setups

We discuss two such problems today:

- multi-armed bandit
- reinforcement learning

13 / 43

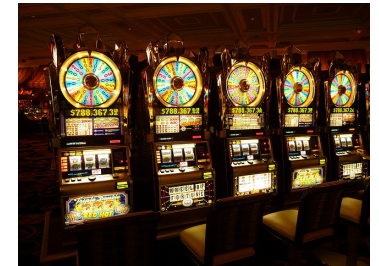
Mult-armed bandits: motivation

Imagine going to a casino to play a slot machine

- it robs you, like a “bandit” with a single arm

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?

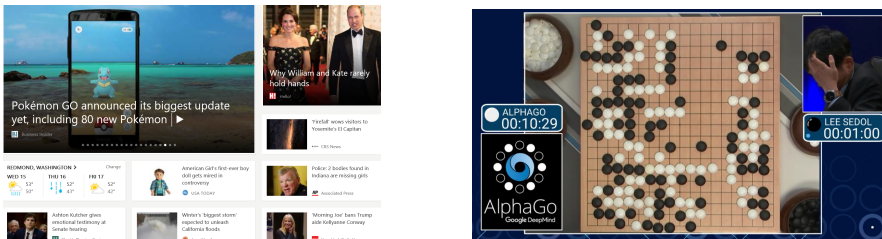


14 / 43

Applications

This simple model and its variants capture many real-life applications

- recommendation systems, each product/movie/news story is an arm (Microsoft MSN indeed employs a variant of bandit algorithm)
- game playing, each possible move is an arm (AlphaGo indeed has a bandit algorithm as one of the components)



15 / 43

Formal setup

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

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

- the environment decides the reward for each arm $r_{t,1}, \dots, r_{t,K}$
- 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

16 / 43

Objective

What is the goal of this problem?

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?*

17 / 43

Environments

How are the rewards generated by the environments?

- they could be generated via some **fixed distribution**
- they could be generated via some **changing distribution**
- they could be generated even **completely arbitrarily/adversarially**

We focus on a simple setting:

- rewards of arm a are i.i.d. samples of $\text{Ber}(\mu_a)$, that is, $r_{t,a}$ is 1 with prob. μ_a , and 0 with prob. $1 - \mu_a$, independent of anything else.
- each arm has a different mean (μ_1, \dots, μ_K) ; the problem is essentially about **finding the best arm** $\arg\max_a \mu_a$ as quickly as possible

18 / 43

Empirical means

Let $\hat{\mu}_{t,a}$ be the **empirical mean** of arm a up to time t :

$$\hat{\mu}_{t,a} = \frac{1}{n_{t,a}} \sum_{\tau \leq t: a_\tau = a} r_{\tau,a}$$

where

$$n_{t,a} = \sum_{\tau \leq t} \mathbb{I}[a_\tau = a]$$

is the **number of times** we have picked arm a .

Concentration: $\hat{\mu}_{t,a}$ should be close to μ_a if $n_{t,a}$ is large

19 / 43

Exploitation only

Greedy

Pick each arm once for the first K rounds.

For $t = K + 1, \dots, T$, pick $a_t = \arg\max_a \hat{\mu}_{t-1,a}$

What's wrong with this greedy algorithm?

Consider the following example:

- $K = 2, \mu_1 = 0.6, \mu_2 = 0.5$ (so arm 1 is the best)
- suppose the alg. first pick arm 1 and see reward 0, then pick arm 2 and see reward 1 (**this happens with decent probability**)
- **the algorithm will never pick arm 1 again!**

20 / 43

The key challenge

All bandit problems face the same **dilemma**:

Exploitation vs. Exploration trade-off

- on one hand we want to **exploit the arms that we think are good**
- on the other hand we need to **explore all arms often enough** in order to figure out which one is better
- so each time we need to ask: *do I explore or exploit? and how?*

We next discuss **three ways** to trade off exploration and exploitation for our simple multi-armed bandit setting.

21 / 43

A natural first attempt

Explore-then-Exploit

Input: a parameter $T_0 \in [T]$

Exploration phase: for the first T_0 rounds, pick each arm for T_0/K times

Exploitation phase: for the remaining $T - T_0$ rounds, **stick with the empirically best arm** $\arg\max_a \hat{\mu}_{T_0,a}$

Parameter T_0 clearly controls the exploration/exploitation trade-off

22 / 43

Issues of Explore-then-Exploit

It's pretty reasonable, but the **disadvantages** are also clear:

- not clear how to tune the hyperparameter T_0
- in the exploration phase, even if an arm is clearly worse than others based on a few pulls, **it's still pulled for T_0/K times**
- clearly it won't work if the environment is **changing**

23 / 43

A slightly better algorithm

ϵ -Greedy

Pick each arm once for the first K rounds.

For $t = K + 1, \dots, T$,

- **with probability ϵ , explore:** pick an arm uniformly at random
- **with probability $1 - \epsilon$, exploit:** pick $a_t = \arg\max_a \hat{\mu}_{t-1,a}$

Pros

- always exploring and exploiting
- applicable to many other problems
- first thing to try usually

Cons

- need to tune ϵ
- same uniform exploration

Is there a **more adaptive** way to explore?

24 / 43

More adaptive exploration

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

Upper Confidence Bound (UCB) algorithm

For $t = 1, \dots, T$, pick $a_t = \operatorname{argmax}_a \text{UCB}_{t,a}$ where

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

- the first term in $\text{UCB}_{t,a}$ 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!*

25 / 43

Outline

- 1 Review of last lecture
- 2 Multi-armed Bandits
- 3 Reinforcement learning
 - Markov decision process
 - Learning MDPs

27 / 43

Upper confidence bound

Why is it called upper confidence bound?

One can prove that **with high probability**,

$$\mu_a \leq \text{UCB}_{t,a}$$

so $\text{UCB}_{t,a}$ is indeed an upper bound on the true mean.

Another way to interpret UCB, “**optimism in face of uncertainty**”:

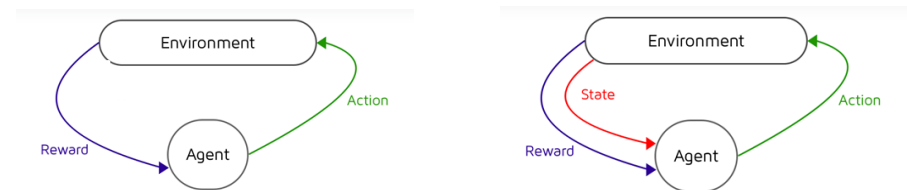
- true environment is unknown due to randomness (**uncertainty**)
- just pretend it's the **most preferable one** among all plausible environments (**optimism**)

This principle is useful for many other bandit problems.

26 / 43

Motivation

Multi-armed bandit is among the simplest decision making problems with limited 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

28 / 43

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** (Lec 10) and **multi-armed bandit**

29 / 43

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

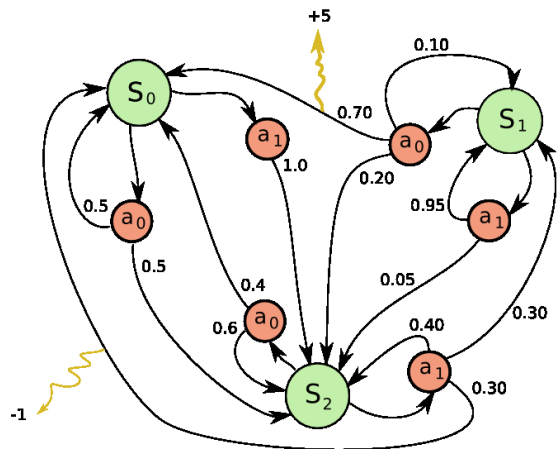
Different from Markov models discussed in Lec 10, the state transition is influenced by the taken action.

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

30 / 43

Example

3 states, 2 actions



31 / 43

Policy

A **policy** $\pi : \mathcal{S} \rightarrow \mathcal{A}$ 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 $0, 1, 2, \dots$ are respectively

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

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

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**

32 / 43

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 ?*

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

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

33 / 43

Value Iteration

Value Iteration

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

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

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

Knowing V , the optimal policy π^* is simply

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

34 / 43

Convergence of Value Iteration

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

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

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

35 / 43

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

36 / 43

Model-based approaches

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

Suppose we have a **sequence of interactions**:

$s_1, a_1, r_1, s_2, a_2, r_2, \dots, s_T, a_T, r_T$, 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.

37 / 43

Model-based approaches

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

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 $t = 1, 2, \dots$,

- **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)

38 / 43

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 $\arg\max_a Q(s, a)$.

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

39 / 43

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:

$$Q(s_t, a_t) \leftarrow (1 - \alpha)Q(s_t, a_t) + \alpha \left(r_t + \gamma \max_{a'} Q(s_{t+1}, a') \right)$$

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

α is like the **learning rate**

40 / 43

Q-learning

The simplest model-free algorithm:

Q-learning

Initialize Q randomly; denote the initial state by s_1 .

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

- **with probability ϵ , explore:** a_t is chosen uniformly at random
- **with probability $1 - \epsilon$, exploit:** $a_t = \operatorname{argmax}_a Q(s_t, a)$
- 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 .

41 / 43

Comparisons

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

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

42 / 43

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**)

43 / 43