# CSCI567 Machine Learning (Fall 2020) 

Prof. Haipeng Luo<br>U of Southern California

Nov 12, 2020

## Administration

HW5 should be graded by the end of the week.

## 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


## 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


## 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


## 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


## Outline

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

## Outline

(1) Review of last lecture

## Hidden Markov Models

Model parameters:

- initial distribution

$$
P\left(Z_{1}=s\right)=\pi_{s}
$$

- transition distribution

$$
P\left(Z_{t+1}=s^{\prime} \mid Z_{t}=s\right)=a_{s, s^{\prime}}
$$

- emission distribution

$$
P\left(X_{t}=o \mid Z_{t}=s\right)=b_{s, o}
$$



## Baum-Welch algorithm

Step 0 Initialize the parameters $(\boldsymbol{\pi}, \boldsymbol{A}, \boldsymbol{B})$

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 $\xi_{s, s^{\prime}}^{(n)}(t)$ for each $n, t, s, s^{\prime}$.

Step 2 (M-Step) Update parameters:

$$
\pi_{s} \propto \sum_{n} \gamma_{s}^{(n)}(1), \quad a_{s, s^{\prime}} \propto \sum_{n} \sum_{t=1}^{T-1} \xi_{s, s^{\prime}}^{(n)}(t), \quad b_{s, o} \propto \sum_{n} \sum_{t: x_{t}=o} \gamma_{s}^{(n)}(t)
$$

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 $t=2, \ldots, T$,

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

$$
\delta_{s}(t)=b_{s, x_{t}} \max _{s^{\prime}} a_{s^{\prime}, s} \delta_{s^{\prime}}(t-1)
$$

$$
\Delta_{s}(t)=\underset{s^{\prime}}{\operatorname{argmax}} a_{s^{\prime}, s} \delta_{s^{\prime}}(t-1)
$$

Backtracking: let $z_{T}^{*}=\operatorname{argmax}_{s} \delta_{s}(T)$.
For each $t=T, \ldots, 2$ : set $z_{t-1}^{*}=\Delta_{z_{t}^{*}}(t)$.
Output the most likely path $z_{1}^{*}, \ldots, z_{T}^{*}$.

## 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 $t=2, \ldots, T$,

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

$$
\begin{aligned}
\delta_{s}(t) & = \begin{cases}b_{s, x_{t}} \max _{s^{\prime}} a_{s^{\prime}, s} \delta_{s^{\prime}}(t-1) & \text { if } t \leq T_{0} \\
\max _{s^{\prime}} a_{s^{\prime}, s} \delta_{s^{\prime}}(t-1) & \text { else }\end{cases} \\
\Delta_{s}(t) & =\underset{s^{\prime}}{\operatorname{argmax}} a_{s^{\prime}, s} \delta_{s^{\prime}}(t-1) .
\end{aligned}
$$

Backtracking: let $z_{T}^{*}=\operatorname{argmax}_{s} \delta_{s}(T)$.
For each $t=T, \ldots, 2$ : set $z_{t-1}^{*}=\Delta_{z_{t}^{*}}(t)$.
Output the most likely path $z_{1}^{*}, \ldots, z_{T}^{*}$.

## 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


## 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:

## 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


## 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:

## Examples

## Amazon/Netflix/MSN recommendation systems:

- a user visits the website


## Examples

## Amazon/Netflix/MSN recommendation systems:

- a user visits the website
- the system recommends some products/movies/news stories


## 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


## 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:

## 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


## 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)


## 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


## Mulit-armed bandits: motivation

Imagine going to a casino to play a slot machine


## Mulit-armed bandits: motivation

Imagine going to a casino to play a slot machine

- it robs you, like a "bandit" with a single arm



## Mulit-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


## Mulit-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)



## Mulit-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?



## Applications

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

- recommendation systems, each product/movie/news story is an arm


## 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)



## 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



## 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)



## Formal setup

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

## Formal setup

There are $K$ arms (actions/choices/...)
The problem proceeds in rounds between the environment and a learner:

## 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 $r_{t, 1}, \ldots, r_{t, K}$


## 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 $a_{t} \in[K]$


## 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., $r_{t, a_{t}}$


## 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!

## 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

What is the goal of this problem?

## Objective

What is the goal of this problem?
Maximizing total rewards $\sum_{t=1}^{T} r_{t, a_{t}}$ seems natural

## 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,

## 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.

## 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

## 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

$$
\max _{a \in[K]} \sum_{t=1}^{T} r_{t, a}-\sum_{t=1}^{T} r_{t, a_{t}}
$$

## 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?

## Environments

How are the rewards generated by the environments?

## Environments

How are the rewards generated by the environments?

- they could be generated via some fixed distribution


## 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


## 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


## 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 $\operatorname{Ber}\left(\mu_{a}\right)$, that is, $r_{t, a}$ is 1 with prob. $\mu_{a}$, and 0 with prob. $1-\mu_{a}$, independent of anything else.


## 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 $\operatorname{Ber}\left(\mu_{a}\right)$, that is, $r_{t, a}$ is 1 with prob. $\mu_{a}$, and 0 with prob. $1-\mu_{a}$, independent of anything else.
- each arm has a different mean $\left(\mu_{1}, \ldots, \mu_{K}\right)$; the problem is essentially about finding the best arm $\operatorname{argmax}_{a} \mu_{a}$ as quickly as possible


## 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

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

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

## 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

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

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

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

## Exploitation only

## Greedy

Pick each arm once for the first $K$ rounds.

## Exploitation only

Greedy
Pick each arm once for the first $K$ rounds.
For $t=K+1, \ldots, T$, pick $a_{t}=\operatorname{argmax}_{a} \hat{\mu}_{t-1, a}$

## Exploitation only

Greedy
Pick each arm once for the first $K$ rounds.
For $t=K+1, \ldots, T$, pick $a_{t}=\operatorname{argmax}_{a} \hat{\mu}_{t-1, a}$

What's wrong with this greedy algorithm?

## Exploitation only

## Greedy

Pick each arm once for the first $K$ rounds.
For $t=K+1, \ldots, T$, pick $a_{t}=\operatorname{argmax}_{a} \hat{\mu}_{t-1, a}$

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)


## Exploitation only

## Greedy

Pick each arm once for the first $K$ rounds.
For $t=K+1, \ldots, T$, pick $a_{t}=\operatorname{argmax}_{a} \hat{\mu}_{t-1, a}$

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)


## Exploitation only

## Greedy

Pick each arm once for the first $K$ rounds.
For $t=K+1, \ldots, T$, pick $a_{t}=\operatorname{argmax}_{a} \hat{\mu}_{t-1, a}$

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!


## The key challenge

All bandit problems face the same dilemma:
Exploitation vs. Exploration trade-off

## 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


## 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


## 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?


## 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.

## A natural first attempt

## Explore-then-Exploit

Input: a parameter $T_{0} \in[T]$

## 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

## 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 $\operatorname{argmax}_{a} \hat{\mu}_{T_{0}, a}$

## 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 $\operatorname{argmax}_{a} \hat{\mu}_{T_{0}, a}$

Parameter $T_{0}$ clearly controls the exploration/exploitation trade-off

## Issues of Explore-then-Exploit

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

## Issues of Explore-then-Exploit

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

- not clear how to tune the hyperparameter $T_{0}$


## 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


## 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


## A slightly better algorithm

## $\epsilon$-Greedy

Pick each arm once for the first $K$ rounds.

## A slightly better algorithm

$\epsilon$-Greedy
Pick each arm once for the first $K$ rounds.
For $t=K+1, \ldots, T$,

- with probability $\epsilon$, explore: pick an arm uniformly at random


## A slightly better algorithm

$\epsilon$-Greedy
Pick each arm once for the first $K$ rounds.
For $t=K+1, \ldots, T$,

- with probability $\epsilon$, explore: pick an arm uniformly at random
- with probability $1-\epsilon$, exploit: pick $a_{t}=\operatorname{argmax}_{a} \hat{\mu}_{t-1, a}$


## A slightly better algorithm

$\epsilon$-Greedy
Pick each arm once for the first $K$ rounds.
For $t=K+1, \ldots, T$,

- with probability $\epsilon$, explore: pick an arm uniformly at random
- with probability $1-\epsilon$, exploit: pick $a_{t}=\operatorname{argmax}_{a} \hat{\mu}_{t-1, a}$


## Pros

- always exploring and exploiting


## A slightly better algorithm

## $\epsilon$-Greedy

Pick each arm once for the first $K$ rounds.
For $t=K+1, \ldots, T$,

- with probability $\epsilon$, explore: pick an arm uniformly at random
- with probability $1-\epsilon$, exploit: pick $a_{t}=\operatorname{argmax}_{a} \hat{\mu}_{t-1, a}$


## Pros

- always exploring and exploiting
- applicable to many other problems


## A slightly better algorithm

$\epsilon$-Greedy
Pick each arm once for the first $K$ rounds.
For $t=K+1, \ldots, T$,

- with probability $\epsilon$, explore: pick an arm uniformly at random
- with probability $1-\epsilon$, exploit: pick $a_{t}=\operatorname{argmax}_{a} \hat{\mu}_{t-1, a}$


## Pros

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


## A slightly better algorithm

## $\epsilon$-Greedy

Pick each arm once for the first $K$ rounds.
For $t=K+1, \ldots, T$,

- with probability $\epsilon$, explore: pick an arm uniformly at random
- with probability $1-\epsilon$, exploit: pick $a_{t}=\operatorname{argmax}_{a} \hat{\mu}_{t-1, a}$


## Pros

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


## Cons

- need to tune $\epsilon$


## A slightly better algorithm

## $\epsilon$-Greedy

Pick each arm once for the first $K$ rounds.
For $t=K+1, \ldots, T$,

- with probability $\epsilon$, explore: pick an arm uniformly at random
- with probability $1-\epsilon$, exploit: pick $a_{t}=\operatorname{argmax}_{a} \hat{\mu}_{t-1, a}$


## Pros

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


## Cons

- need to tune $\epsilon$
- same uniform exploration


## A slightly better algorithm

## $\epsilon$-Greedy

Pick each arm once for the first $K$ rounds.
For $t=K+1, \ldots, T$,

- with probability $\epsilon$, explore: pick an arm uniformly at random
- with probability $1-\epsilon$, exploit: pick $a_{t}=\operatorname{argmax}_{a} \hat{\mu}_{t-1, a}$


## Pros

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


## Cons

- need to tune $\epsilon$
- same uniform exploration

Is there a more adaptive way to explore?

## More adaptive exploration

A simple modification of "Greedy" leads to the well-known:
Upper Confidence Bound (UCB) algorithm

## More adaptive exploration

A simple modification of "Greedy" leads to the well-known:
Upper Confidence Bound (UCB) algorithm
For $t=1, \ldots, T$, pick $a_{t}=\operatorname{argmax}_{a} \mathrm{UCB}_{t, a}$ where

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

## More adaptive exploration

A simple modification of "Greedy" leads to the well-known:
Upper Confidence Bound (UCB) algorithm
For $t=1, \ldots, T$, pick $a_{t}=\operatorname{argmax}_{a} \mathrm{UCB}_{t, a}$ 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


## More adaptive exploration

A simple modification of "Greedy" leads to the well-known:
Upper Confidence Bound (UCB) algorithm
For $t=1, \ldots, T$, pick $a_{t}=\operatorname{argmax}_{a} \mathrm{UCB}_{t, a}$ 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)


## More adaptive exploration

A simple modification of "Greedy" leads to the well-known:
Upper Confidence Bound (UCB) algorithm
For $t=1, \ldots, T$, pick $a_{t}=\operatorname{argmax}_{a} \mathrm{UCB}_{t, a}$ 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,


## More adaptive exploration

A simple modification of "Greedy" leads to the well-known:
Upper Confidence Bound (UCB) algorithm
For $t=1, \ldots, T$, pick $a_{t}=\operatorname{argmax}_{a} \mathrm{UCB}_{t, a}$ 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!


## Upper confidence bound

Why is it called upper confidence bound?

## Upper confidence bound

Why is it called upper confidence bound?

One can prove that with high probability,

$$
\mu_{a} \leq \mathrm{UCB}_{t, a}
$$

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

## Upper confidence bound

Why is it called upper confidence bound?

One can prove that with high probability,

$$
\mu_{a} \leq \mathrm{UCB}_{t, a}
$$

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

Another way to interpret UCB, "optimism in face of uncertainty":

## Upper confidence bound

Why is it called upper confidence bound?

One can prove that with high probability,

$$
\mu_{a} \leq \mathrm{UCB}_{t, a}
$$

so $\mathrm{UCB}_{t, a}$ 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)


## Upper confidence bound

Why is it called upper confidence bound?

One can prove that with high probability,

$$
\mu_{a} \leq \mathrm{UCB}_{t, a}
$$

so $\mathrm{UCB}_{t, a}$ 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)


## Upper confidence bound

Why is it called upper confidence bound?

One can prove that with high probability,

$$
\mu_{a} \leq \mathrm{UCB}_{t, a}
$$

so $\mathrm{UCB}_{t, a}$ 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.

## Outline

## (1) Review of last lecture

## (2) Multi-armed Bandits

(3) Reinforcement learning

- Markov decision process
- Learning MDPs


## Motivation

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


## 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.

## 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


## Reinforcement learning

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

## 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.


## 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

## Markov decision process

An MDP is parameterized by five elements

## Markov decision process

An MDP is parameterized by five elements

- $\mathcal{S}$ : a set of possible states


## Markov decision process

An MDP is parameterized by five elements

- $\mathcal{S}$ : a set of possible states
- $\mathcal{A}$ : a set of possible actions


## 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, $P_{a}\left(s, s^{\prime}\right)$ is the probability of transiting from state $s$ to state $s^{\prime}$ after taking action $a$ (Markov property)


## 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, $P_{a}\left(s, s^{\prime}\right)$ is the probability of transiting from state $s$ to state $s^{\prime}$ after taking action $a$ (Markov property)
- $r$ : reward function, $r_{a}(s)$ is (expected) reward of action $a$ at state $s$


## 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, $P_{a}\left(s, s^{\prime}\right)$ is the probability of transiting from state $s$ to state $s^{\prime}$ 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 $\gamma$ for today


## 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, $P_{a}\left(s, s^{\prime}\right)$ is the probability of transiting from state $s$ to state $s^{\prime}$ 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 $\gamma$ for today

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

## 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, $P_{a}\left(s, s^{\prime}\right)$ is the probability of transiting from state $s$ to state $s^{\prime}$ 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 $\gamma$ 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.

## Example

3 states, 2 actions


## Policy

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

## 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 $\pi$, the discounted rewards for time $0,1,2, \ldots$ are respectively

## 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 $\pi$, the discounted rewards for time $0,1,2, \ldots$ are respectively

$$
r_{\pi\left(s_{0}\right)}\left(s_{0}\right), \quad \gamma r_{\pi\left(s_{1}\right)}\left(s_{1}\right), \quad \gamma^{2} r_{\pi\left(s_{2}\right)}\left(s_{2}\right), \cdots
$$

where $s_{1} \sim P_{\pi\left(s_{0}\right)}\left(s_{0}, \cdot\right), \quad s_{2} \sim P_{\pi\left(s_{1}\right)}\left(s_{1}, \cdot\right), \cdots$

## 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 $\pi$, the discounted rewards for time $0,1,2, \ldots$ are respectively

$$
r_{\pi\left(s_{0}\right)}\left(s_{0}\right), \quad \gamma r_{\pi\left(s_{1}\right)}\left(s_{1}\right), \quad \gamma^{2} r_{\pi\left(s_{2}\right)}\left(s_{2}\right), \cdots
$$

where $s_{1} \sim P_{\pi\left(s_{0}\right)}\left(s_{0}, \cdot\right), \quad s_{2} \sim P_{\pi\left(s_{1}\right)}\left(s_{1}, \cdot\right), \cdots$
If we follow the policy forever, the total (discounted) reward is

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

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

## 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 $\pi$, the discounted rewards for time $0,1,2, \ldots$ are respectively

$$
r_{\pi\left(s_{0}\right)}\left(s_{0}\right), \quad \gamma r_{\pi\left(s_{1}\right)}\left(s_{1}\right), \quad \gamma^{2} r_{\pi\left(s_{2}\right)}\left(s_{2}\right), \quad \cdots
$$

where $s_{1} \sim P_{\pi\left(s_{0}\right)}\left(s_{0}, \cdot\right), \quad s_{2} \sim P_{\pi\left(s_{1}\right)}\left(s_{1}, \cdot\right), \cdots$
If we follow the policy forever, the total (discounted) reward is

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

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

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

## Optimal policy and Bellman equation

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

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

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

## Optimal policy and Bellman equation

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

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

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

$$
V(s)=\max _{\pi} \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^{t} r_{\pi\left(s_{t}\right)}\left(s_{t}\right) \mid s_{0}=s\right]
$$

## Optimal policy and Bellman equation

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

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

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

$$
\begin{aligned}
V(s) & =\max _{\pi} \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^{t} r_{\pi\left(s_{t}\right)}\left(s_{t}\right) \mid s_{0}=s\right] \\
& =\max _{a \in \mathcal{A}}\left(r_{s}(a)+\gamma \sum_{s^{\prime} \in \mathcal{S}} P_{a}\left(s, s^{\prime}\right) V\left(s^{\prime}\right)\right)
\end{aligned}
$$

## Optimal policy and Bellman equation

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

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

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

$$
\begin{aligned}
V(s) & =\max _{\pi} \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^{t} r_{\pi\left(s_{t}\right)}\left(s_{t}\right) \mid s_{0}=s\right] \\
& =\max _{a \in \mathcal{A}}\left(r_{s}(a)+\gamma \sum_{s^{\prime} \in \mathcal{S}} P_{a}\left(s, s^{\prime}\right) V\left(s^{\prime}\right)\right)
\end{aligned}
$$

$V$ is called the value function.

## Optimal policy and Bellman equation

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

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

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

$$
\begin{aligned}
V(s) & =\max _{\pi} \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^{t} r_{\pi\left(s_{t}\right)}\left(s_{t}\right) \mid s_{0}=s\right] \\
& =\max _{a \in \mathcal{A}}\left(r_{s}(a)+\gamma \sum_{s^{\prime} \in \mathcal{S}} P_{a}\left(s, s^{\prime}\right) V\left(s^{\prime}\right)\right)
\end{aligned}
$$

$V$ 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 $s \in \mathcal{S}$

## Value Iteration

Value Iteration
Initialize $V_{0}(s)$ randomly for all $s \in \mathcal{S}$
For $k=1,2, \ldots$ (until convergence)

$$
V_{k}(s)=\max _{a \in \mathcal{A}}\left(r_{s}(a)+\gamma \sum_{s^{\prime} \in \mathcal{S}} P_{a}\left(s, s^{\prime}\right) V_{k-1}\left(s^{\prime}\right)\right)
$$

(Bellman upate)

## Value Iteration

## Value Iteration

Initialize $V_{0}(s)$ randomly for all $s \in \mathcal{S}$
For $k=1,2, \ldots$ (until convergence)

$$
V_{k}(s)=\max _{a \in \mathcal{A}}\left(r_{s}(a)+\gamma \sum_{s^{\prime} \in \mathcal{S}} P_{a}\left(s, s^{\prime}\right) V_{k-1}\left(s^{\prime}\right)\right)
$$

(Bellman upate)

Knowing $V$, the optimal policy $\pi^{*}$ is simply

$$
\pi^{*}(s)=\underset{a \in \mathcal{A}}{\operatorname{argmax}}\left(r_{s}(a)+\gamma \sum_{s^{\prime} \in \mathcal{S}} P_{a}\left(s, s^{\prime}\right) V\left(s^{\prime}\right)\right)
$$

## Convergence of Value Iteration

Does Value Iteration always find the true value function $V$ ?

## Convergence of Value Iteration

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

## Convergence of Value Iteration

Does Value Iteration always find the true value function V? Yes!
$\left|V_{k}(s)-V(s)\right|$

## Convergence of Value Iteration

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

$$
\begin{aligned}
\left|V_{k}(s)-V(s)\right|= & \mid \max _{a \in \mathcal{A}}\left(r_{s}(a)+\gamma \sum_{s^{\prime} \in \mathcal{S}} P_{a}\left(s, s^{\prime}\right) V_{k-1}\left(s^{\prime}\right)\right) \\
& -\max _{a \in \mathcal{A}}\left(r_{s}(a)+\gamma \sum_{s^{\prime} \in \mathcal{S}} P_{a}\left(s, s^{\prime}\right) V\left(s^{\prime}\right)\right)
\end{aligned}
$$

## Convergence of Value Iteration

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

$$
\begin{aligned}
&\left|V_{k}(s)-V(s)\right|= \mid \max _{a \in \mathcal{A}}\left(r_{s}(a)+\gamma \sum_{s^{\prime} \in \mathcal{S}} P_{a}\left(s, s^{\prime}\right) V_{k-1}\left(s^{\prime}\right)\right) \\
&-\max _{a \in \mathcal{A}}\left(r_{s}(a)+\gamma \sum_{s^{\prime} \in \mathcal{S}} P_{a}\left(s, s^{\prime}\right) V\left(s^{\prime}\right)\right) \\
& \leq \gamma \max _{a \in \mathcal{A}}\left|\sum_{s^{\prime} \in \mathcal{S}} P_{a}\left(s, s^{\prime}\right)\left(V_{k-1}\left(s^{\prime}\right)-V\left(s^{\prime}\right)\right)\right|
\end{aligned}
$$

## Convergence of Value Iteration

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

$$
\begin{aligned}
&\left|V_{k}(s)-V(s)\right|= \mid \max _{a \in \mathcal{A}}\left(r_{s}(a)+\gamma \sum_{s^{\prime} \in \mathcal{S}} P_{a}\left(s, s^{\prime}\right) V_{k-1}\left(s^{\prime}\right)\right) \\
&-\max _{a \in \mathcal{A}}\left(r_{s}(a)+\gamma \sum_{s^{\prime} \in \mathcal{S}} P_{a}\left(s, s^{\prime}\right) V\left(s^{\prime}\right)\right) \\
& \leq \gamma \max _{a \in \mathcal{A}}\left|\sum_{s^{\prime} \in \mathcal{S}} P_{a}\left(s, s^{\prime}\right)\left(V_{k-1}\left(s^{\prime}\right)-V\left(s^{\prime}\right)\right)\right| \\
& \leq \gamma \max _{a \in \mathcal{A}} \sum_{s^{\prime} \in \mathcal{S}} P_{a}\left(s, s^{\prime}\right)\left|V_{k-1}\left(s^{\prime}\right)-V\left(s^{\prime}\right)\right|
\end{aligned}
$$

## Convergence of Value Iteration

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

$$
\begin{aligned}
&\left|V_{k}(s)-V(s)\right|= \mid \max _{a \in \mathcal{A}}\left(r_{s}(a)+\gamma \sum_{s^{\prime} \in \mathcal{S}} P_{a}\left(s, s^{\prime}\right) V_{k-1}\left(s^{\prime}\right)\right) \\
&-\max _{a \in \mathcal{A}}\left(r_{s}(a)+\gamma \sum_{s^{\prime} \in \mathcal{S}} P_{a}\left(s, s^{\prime}\right) V\left(s^{\prime}\right)\right) \\
& \leq \gamma \max _{a \in \mathcal{A}}\left|\sum_{s^{\prime} \in \mathcal{S}} P_{a}\left(s, s^{\prime}\right)\left(V_{k-1}\left(s^{\prime}\right)-V\left(s^{\prime}\right)\right)\right| \\
& \leq \gamma \max _{a \in \mathcal{A}} \sum_{s^{\prime} \in \mathcal{S}} P_{a}\left(s, s^{\prime}\right)\left|V_{k-1}\left(s^{\prime}\right)-V\left(s^{\prime}\right)\right| \\
& \leq \gamma \max _{s^{\prime \prime}}\left|V_{k-1}\left(s^{\prime \prime}\right)-V\left(s^{\prime \prime}\right)\right|
\end{aligned}
$$

## Convergence of Value Iteration

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

$$
\begin{aligned}
&\left|V_{k}(s)-V(s)\right|=\mid \max _{a \in \mathcal{A}}\left(r_{s}(a)+\gamma \sum_{s^{\prime} \in \mathcal{S}} P_{a}\left(s, s^{\prime}\right) V_{k-1}\left(s^{\prime}\right)\right) \\
& \quad-\max _{a \in \mathcal{A}}\left(r_{s}(a)+\gamma \sum_{s^{\prime} \in \mathcal{S}} P_{a}\left(s, s^{\prime}\right) V\left(s^{\prime}\right)\right) \mid \\
& \leq \gamma \max _{a \in \mathcal{A}}\left|\sum_{s^{\prime} \in \mathcal{S}} P_{a}\left(s, s^{\prime}\right)\left(V_{k-1}\left(s^{\prime}\right)-V\left(s^{\prime}\right)\right)\right| \\
& \leq \gamma \max _{a \in \mathcal{A}} \sum_{s^{\prime} \in \mathcal{S}} P_{a}\left(s, s^{\prime}\right)\left|V_{k-1}\left(s^{\prime}\right)-V\left(s^{\prime}\right)\right| \\
& \leq \gamma \max _{s^{\prime \prime}}\left|V_{k-1}\left(s^{\prime \prime}\right)-V\left(s^{\prime \prime}\right)\right| \leq \cdots \leq \gamma^{k} \max _{s^{\prime \prime}}\left|V_{0}\left(s^{\prime \prime}\right)-V\left(s^{\prime \prime}\right)\right|
\end{aligned}
$$

## Convergence of Value Iteration

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

$$
\begin{aligned}
&\left|V_{k}(s)-V(s)\right|=\mid \max _{a \in \mathcal{A}}\left(r_{s}(a)+\gamma \sum_{s^{\prime} \in \mathcal{S}} P_{a}\left(s, s^{\prime}\right) V_{k-1}\left(s^{\prime}\right)\right) \\
& \quad-\max _{a \in \mathcal{A}}\left(r_{s}(a)+\gamma \sum_{s^{\prime} \in \mathcal{S}} P_{a}\left(s, s^{\prime}\right) V\left(s^{\prime}\right)\right) \mid \\
& \leq \gamma \max _{a \in \mathcal{A}}\left|\sum_{s^{\prime} \in \mathcal{S}} P_{a}\left(s, s^{\prime}\right)\left(V_{k-1}\left(s^{\prime}\right)-V\left(s^{\prime}\right)\right)\right| \\
& \leq \gamma \max _{a \in \mathcal{A}} \sum_{s^{\prime} \in \mathcal{S}} P_{a}\left(s, s^{\prime}\right)\left|V_{k-1}\left(s^{\prime}\right)-V\left(s^{\prime}\right)\right| \\
& \leq \gamma \max _{s^{\prime \prime}}\left|V_{k-1}\left(s^{\prime \prime}\right)-V\left(s^{\prime \prime}\right)\right| \leq \cdots \leq \gamma^{k} \max _{s^{\prime \prime}}\left|V_{0}\left(s^{\prime \prime}\right)-V\left(s^{\prime \prime}\right)\right|
\end{aligned}
$$

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 $r$


## 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?

## 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

## Model-based approaches

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

Suppose we have a sequence of interactions:
$s_{1}, a_{1}, r_{1}, s_{2}, a_{2}, r_{2}, \ldots, s_{T}, a_{T}, r_{T}$,

## 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
$P_{a}\left(s, s^{\prime}\right) \propto$ \#transitions from $s$ to $s^{\prime}$ after taking action $a$ $r_{a}(s)=$ average observed reward at state $s$ after taking action $a$

## 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
$P_{a}\left(s, s^{\prime}\right) \propto \#$ transitions from $s$ to $s^{\prime}$ after taking action $a$ $r_{a}(s)=$ average observed reward at state $s$ 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 $s_{1}, a_{1}, r_{1}, s_{2}, a_{2}, r_{2}, \ldots, s_{T}, a_{T}, r_{T}$ ?

## 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.

## 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.

## 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 $\epsilon$-Greedy idea instead.
A sketch for model-based approaches
Initialize $V, P, r$ randomly

## 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 $\epsilon$-Greedy idea instead.
A sketch for model-based approaches
Initialize $V, P, r$ randomly
For $t=1,2, \ldots$,

- with probability $\epsilon$, explore: pick an action uniformly at random


## 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 $\epsilon$-Greedy idea instead.
A sketch for model-based approaches
Initialize $V, P, r$ randomly
For $t=1,2, \ldots$,

- with probability $\epsilon$, explore: pick an action uniformly at random
- with probability $1-\epsilon$, exploit: pick the optimal action based on $V$


## 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 $\epsilon$-Greedy idea instead.
A sketch for model-based approaches
Initialize $V, P, r$ randomly
For $t=1,2, \ldots$,

- with probability $\epsilon$, explore: pick an action uniformly at random
- with probability $1-\epsilon$, exploit: pick the optimal action based on $V$
- update the model parameters $P, r$


## 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 $\epsilon$-Greedy idea instead.
A sketch for model-based approaches
Initialize $V, P, r$ randomly
For $t=1,2, \ldots$,

- with probability $\epsilon$, 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.

## Model-free approaches

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

## 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

$$
Q(s, a)=r_{a}(s)+\gamma \sum_{s^{\prime} \in \mathcal{S}} P_{a}\left(s, s^{\prime}\right) \max _{a^{\prime} \in \mathcal{A}} Q\left(s^{\prime}, a^{\prime}\right)
$$

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

## 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

$$
Q(s, a)=r_{a}(s)+\gamma \sum_{s^{\prime} \in \mathcal{S}} P_{a}\left(s, s^{\prime}\right) \max _{a^{\prime} \in \mathcal{A}} Q\left(s^{\prime}, a^{\prime}\right)
$$

In words, $Q(s, a)$ is the expected reward one can achieve starting from
state $s$ with action $a$, then acting optimally.
Clearly, $V(s)=\max _{a} Q(s, a)$.

## 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

$$
Q(s, a)=r_{a}(s)+\gamma \sum_{s^{\prime} \in \mathcal{S}} P_{a}\left(s, s^{\prime}\right) \max _{a^{\prime} \in \mathcal{A}} Q\left(s^{\prime}, a^{\prime}\right)
$$

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

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

$$
Q(s, a)=r_{a}(s)+\gamma \sum_{s^{\prime} \in \mathcal{S}} P_{a}\left(s, s^{\prime}\right) \max _{a^{\prime} \in \mathcal{A}} Q\left(s^{\prime}, a^{\prime}\right)
$$

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?

$$
Q(s, a)=r_{a}(s)+\gamma \sum_{s^{\prime} \in \mathcal{S}} P_{a}\left(s, s^{\prime}\right) \max _{a^{\prime} \in \mathcal{A}} Q\left(s^{\prime}, a^{\prime}\right)
$$

## Temporal difference

How to learn the $Q$ function?

$$
Q(s, a)=r_{a}(s)+\gamma \sum_{s^{\prime} \in \mathcal{S}} P_{a}\left(s, s^{\prime}\right) \max _{a^{\prime} \in \mathcal{A}} Q\left(s^{\prime}, a^{\prime}\right)
$$

On experience $\left\langle s_{t}, a_{t}, r_{t}, s_{t+1}\right\rangle$, with the current guess on $Q$, $r_{t}+\gamma \max _{a^{\prime}} Q\left(s_{t+1}, a^{\prime}\right)$ is like a sample of the RHS of the equation.

## Temporal difference

How to learn the $Q$ function?

$$
Q(s, a)=r_{a}(s)+\gamma \sum_{s^{\prime} \in \mathcal{S}} P_{a}\left(s, s^{\prime}\right) \max _{a^{\prime} \in \mathcal{A}} Q\left(s^{\prime}, a^{\prime}\right)
$$

On experience $\left\langle s_{t}, a_{t}, r_{t}, s_{t+1}\right\rangle$, with the current guess on $Q$, $r_{t}+\gamma \max _{a^{\prime}} Q\left(s_{t+1}, a^{\prime}\right)$ is like a sample of the RHS of the equation.

So it's natural to do the following update:

$$
Q\left(s_{t}, a_{t}\right) \leftarrow(1-\alpha) Q\left(s_{t}, a_{t}\right)+\alpha\left(r_{t}+\gamma \max _{a^{\prime}} Q\left(s_{t+1}, a^{\prime}\right)\right)
$$

## Temporal difference

How to learn the $Q$ function?

$$
Q(s, a)=r_{a}(s)+\gamma \sum_{s^{\prime} \in \mathcal{S}} P_{a}\left(s, s^{\prime}\right) \max _{a^{\prime} \in \mathcal{A}} Q\left(s^{\prime}, a^{\prime}\right)
$$

On experience $\left\langle s_{t}, a_{t}, r_{t}, s_{t+1}\right\rangle$, with the current guess on $Q$, $r_{t}+\gamma \max _{a^{\prime}} Q\left(s_{t+1}, a^{\prime}\right)$ is like a sample of the RHS of the equation.

So it's natural to do the following update:

$$
\begin{aligned}
Q\left(s_{t}, a_{t}\right) & \leftarrow(1-\alpha) Q\left(s_{t}, a_{t}\right)+\alpha\left(r_{t}+\gamma \max _{a^{\prime}} Q\left(s_{t+1}, a^{\prime}\right)\right) \\
& =Q\left(s_{t}, a_{t}\right)+\alpha \underbrace{\left(r_{t}+\gamma \max _{a^{\prime}} Q\left(s_{t+1}, a^{\prime}\right)-Q\left(s_{t}, a_{t}\right)\right)}_{\text {temporal difference }}
\end{aligned}
$$

## Temporal difference

How to learn the $Q$ function?

$$
Q(s, a)=r_{a}(s)+\gamma \sum_{s^{\prime} \in \mathcal{S}} P_{a}\left(s, s^{\prime}\right) \max _{a^{\prime} \in \mathcal{A}} Q\left(s^{\prime}, a^{\prime}\right)
$$

On experience $\left\langle s_{t}, a_{t}, r_{t}, s_{t+1}\right\rangle$, with the current guess on $Q$, $r_{t}+\gamma \max _{a^{\prime}} Q\left(s_{t+1}, a^{\prime}\right)$ is like a sample of the RHS of the equation.

So it's natural to do the following update:

$$
\begin{aligned}
Q\left(s_{t}, a_{t}\right) & \leftarrow(1-\alpha) Q\left(s_{t}, a_{t}\right)+\alpha\left(r_{t}+\gamma \max _{a^{\prime}} Q\left(s_{t+1}, a^{\prime}\right)\right) \\
& =Q\left(s_{t}, a_{t}\right)+\alpha \underbrace{\left(r_{t}+\gamma \max _{a^{\prime}} Q\left(s_{t+1}, a^{\prime}\right)-Q\left(s_{t}, a_{t}\right)\right)}_{\text {temporal difference }}
\end{aligned}
$$

$\alpha$ is like the learning rate

## Q-learning

The simplest model-free algorithm:
Q-learning
Initialize $Q$ randomly; denote the initial state by $s_{1}$.

## 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 $\epsilon$, explore: $a_{t}$ is chosen uniformly at random


## 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 $\epsilon$, explore: $a_{t}$ is chosen uniformly at random
- with probability $1-\epsilon$, exploit: $a_{t}=\operatorname{argmax}_{a} Q\left(s_{t}, a\right)$


## 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 $\epsilon$, explore: $a_{t}$ is chosen uniformly at random
- with probability $1-\epsilon$, exploit: $a_{t}=\operatorname{argmax}_{a} Q\left(s_{t}, a\right)$
- execute action $a_{t}$, receive reward $r_{t}$, arrive at state $s_{t+1}$


## 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 $\epsilon$, explore: $a_{t}$ is chosen uniformly at random
- with probability $1-\epsilon$, exploit: $a_{t}=\operatorname{argmax}_{a} Q\left(s_{t}, a\right)$
- execute action $a_{t}$, receive reward $r_{t}$, arrive at state $s_{t+1}$
- update the $Q$ function

$$
Q\left(s_{t}, a_{t}\right) \leftarrow(1-\alpha) Q\left(s_{t}, a_{t}\right)+\alpha\left(r_{t}+\gamma \max _{a} Q\left(s_{t+1}, a\right)\right)
$$

for some learning rate $\alpha$.

## Comparisons

|  | Model-based | Model-free |
| :--- | :---: | :---: |
| What it learns | model parameters $P, r, \ldots$ | $Q$ function |
|  |  |  |
|  |  |  |

## Comparisons

|  | Model-based | Model-free |
| :---: | :---: | :---: |
| What it learns | model parameters $P, r, \ldots$ | $Q$ function |
| Space | $O\left(\|\mathcal{S}\|^{2}\|\mathcal{A}\|\right)$ | $O(\|\mathcal{S}\|\|\mathcal{A}\|)$ |
|  |  |  |

## Comparisons

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

## Comparisons

|  | Model-based | Model-free |
| :---: | :---: | :---: |
| What it learns | model parameters $P, r, \ldots$ | $Q$ function |
| Space | $O\left(\|\mathcal{S}\|^{2}\|\mathcal{A}\|\right)$ | $O(\|\mathcal{S}\|\|\mathcal{A}\|)$ |
| 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
- Markov decision process and reinforcement learning


## Summary

A brief introduction to some online decision making problems:

- Multi-armed bandits
- most basic problem to understand exploration vs. exploitation
- Markov decision process and reinforcement learning


## 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, $\epsilon$-greedy, UCB
- Markov decision process and reinforcement learning


## 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, $\epsilon$-greedy, UCB
- Markov decision process and reinforcement learning
- a combination of Markov models and multi-armed bandits


## 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, $\epsilon$-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


## 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, $\epsilon$-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)

