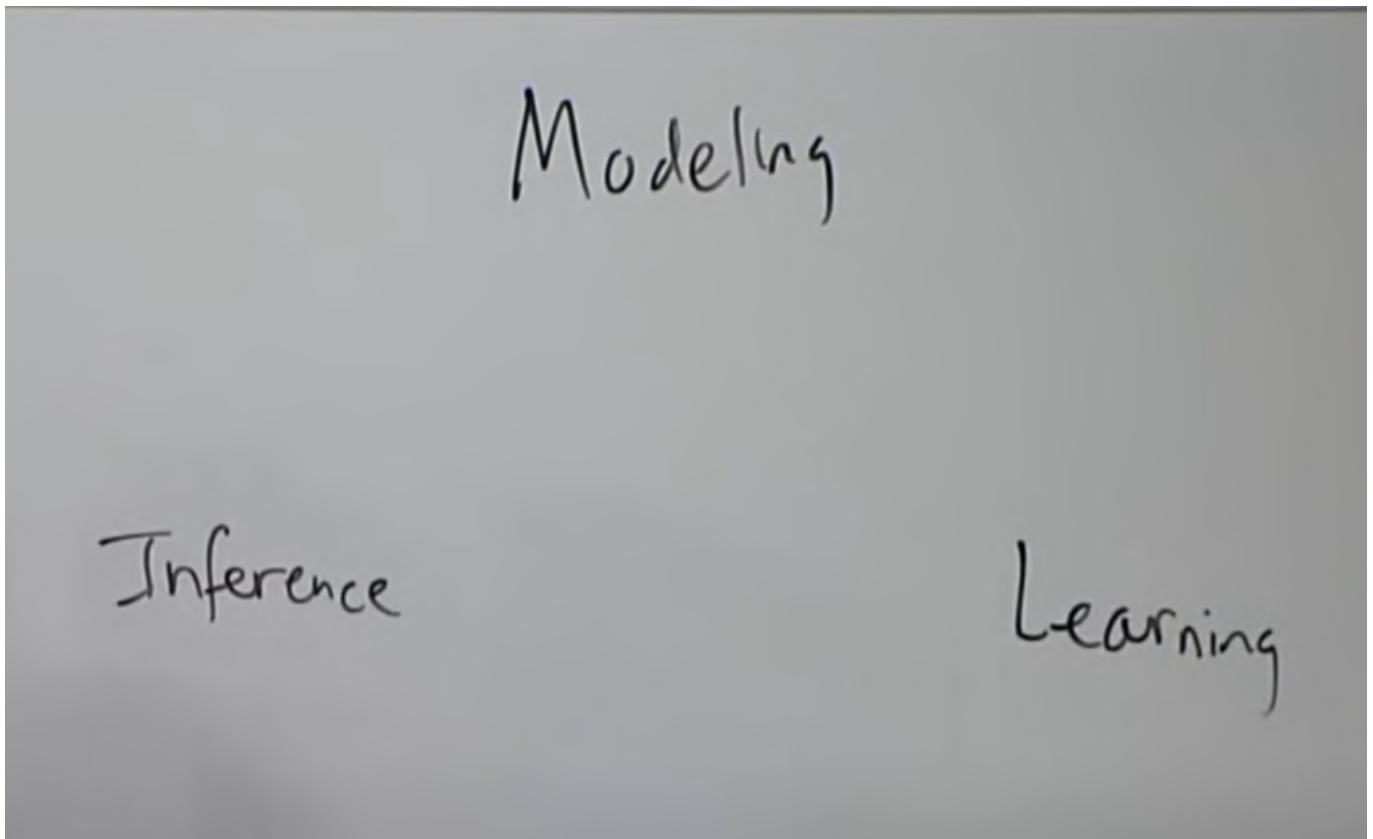


Intro



Models: MDP Inference Algorithms: be able to use model to predict and answer queries

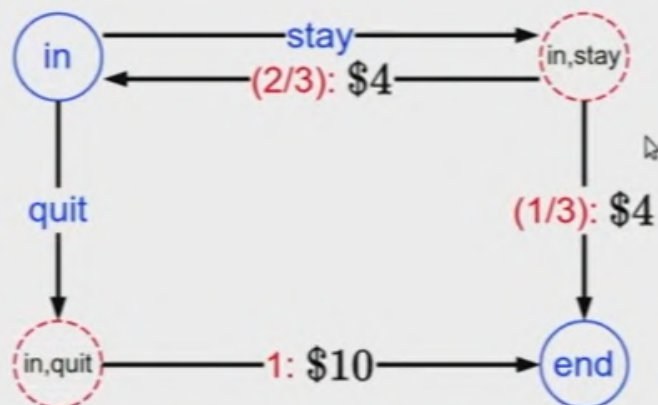
- Value iteration: Compute the optimal policy.
- Policy evaluation: Estimate the value for a particular policy.

Learning: How you actually learn these models

- And this lecture is going to be about learning - Reinforcement learning

Review

Review: MDPs



Definition: Markov decision process

States: the set of states

$s_{\text{start}} \in \text{States}$: starting state

$\text{Actions}(s)$: possible actions from state s

$T(s, a, s')$: probability of s' if take action a in state s

$\text{Reward}(s, a, s')$: reward for the transition (s, a, s')

$\text{IsEnd}(s)$: whether at end of game

$0 \leq \gamma \leq 1$: discount factor (default: 1)

Policy A mapping from states to action A policy tells u where to go/which action to take

When we run a policy, we get a path until you hit the end state (In RL it's called an episode)

What happens out of the episode: You can look at the utility: The discounted sum of rewards along the way.

Review: MDPs

- Following a **policy** π produces a path (**episode**)

$s_0; a_1, r_1, s_1; a_2, r_2, s_2; a_3, r_3, s_3; \dots; a_n, r_n, s_n$

- Value** function $V_\pi(s)$: expected utility if follow π from state s

$$V_\pi(s) = \begin{cases} 0 & \text{if IsEnd}(s) \\ Q_\pi(s, \pi(s)) & \text{otherwise.} \end{cases}$$

- Q-value** function $Q_\pi(s, a)$: expected utility if first take action a from state s and then follow π

$$Q_\pi(s, a) = \sum_{s'} T(s, a, s') [\text{Reward}(s, a, s') + \gamma V_\pi(s')]$$

In last lecture we didn't really worked with utility, because we can use iterative method to compute/approximate the expected utility(The value)

Unknown transitions and rewards



Definition: Markov decision process

States: the set of states

$s_{\text{start}} \in \text{States}$: starting state

Actions(s): possible actions from state s

$T(s, a, s')$: probability of s' if take action a in state s

Reward(s, a, s'): reward for the transition (s, a, s')

IsEnd(s): whether at end of game

$0 \leq \gamma \leq 1$: discount factor (default: 1)

Unknown transitions and rewards



Definition: Markov decision process

States: the set of states

$s_{\text{start}} \in \text{States}$: starting state

Actions(s): possible actions from state s

IsEnd(s): whether at end of game

$0 \leq \gamma \leq 1$: discount factor (default: 1)

reinforcement learning!

If you say goodbye to transition and rewards, that is called reinforcement learning

For MDP: I give you everything there, and u just need to find the optimal policy
For ML: We don't know what rewards and transition(probability) that we are gonna get

Reinforcement Learning Intro

MDP and RL

From MDPs to reinforcement learning



Markov decision process (offline)

- Have mental model of how the world works.
- Find policy to collect maximum rewards.



Reinforcement learning (online)

- Don't know how the world works.
- Perform actions in the world to find out and collect rewards.

MDP is a offline thing

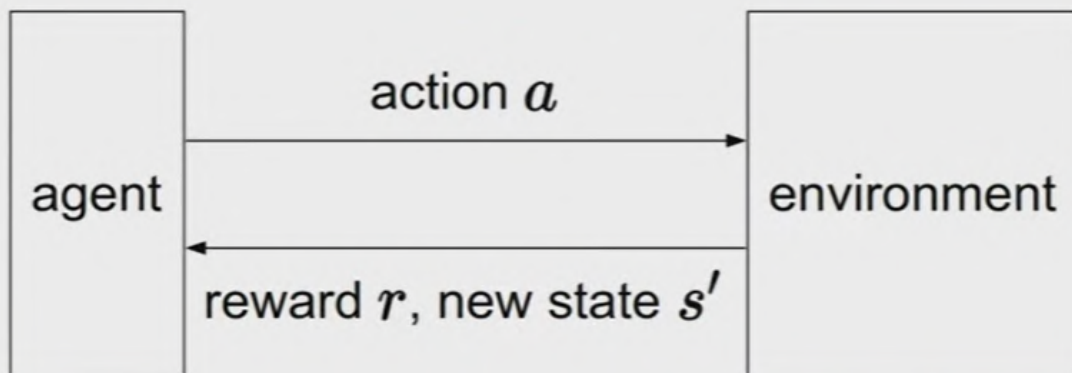
- You already have a exact model of how the "world" works
- All you need to do is just sit down and think(Compute) the best policy that collects the best rewards.

RL is a online thing

- We don't know how the "real world" works
- So we cannot just sit there and think(Compute) because it won't help at all
- We have to go out and perform action in the real worlds, by doing so, hopefully we'll learn something and we'll get some rewards.

General Framework

Reinforcement learning framework



Algorithm: reinforcement learning template

For $t = 1, 2, 3, \dots$

Choose action $a_t = \pi_{\text{act}}(s_{t-1})$ (**how?**)

Receive reward r_t and observe new state s_t

Update parameters (**how?**)

You can treat yourself as an agent You take action a to the environment, the environment returns you a new state s' and reward r Two main questions here:

- What am I gonna act(Which action should I choose)
- Upon recived reward and new state, How/what should I do to update my model of "the world".

Monte Carlo methods

Model-based Monte Carlo

Monte Carlo: means using samples

In the beginning of RL, we have nth but data Let's try to build MDP from that data (Estimate MDP)

Model-based Monte Carlo

Data: $s_0; a_1, r_1, s_1; a_2, r_2, s_2; a_3, r_3, s_3; \dots; a_n, r_n, s_n$



Key idea: model-based learning

Estimate the MDP: $T(s, a, s')$ and $\text{Reward}(s, a, s')$

Transitions:

$$\hat{T}(s, a, s') = \frac{\# \text{ times } (s, a, s') \text{ occurs}}{\# \text{ times } (s, a) \text{ occurs}}$$

Rewards:

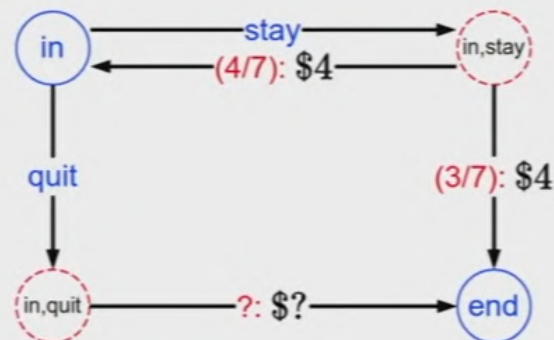
$$\widehat{\text{Reward}}(s, a, s') = r \text{ in } (s, a, r, s')$$

In simple, we just need to figure out what the transitions and rewards are.

- Transition : Collect the data and check the ratio
- Reward: Just observe the reward that the env returns.

Q/A: *Do we always know all states and actions?* For states we can just observe/collect as they come For actions we have to know because we are an agent and we have to play the game

Model-based Monte Carlo



Data (following policy $\pi(s) = \text{stay}$):

[in; stay, 4, end]

- Estimates converge to true values (under certain conditions)
- With estimated MDP $(\hat{T}, \widehat{\text{Reward}})$, compute policy using value iteration

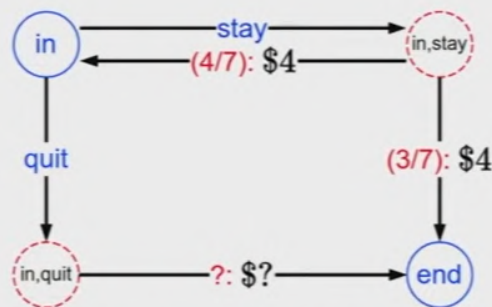
So it works like: You gather a bunch of data (Episodes), estimate them, build MDP, then compute

Potential Problem

You might not explored all the states/rewards

Unless you have a policy let u goes and covers all the states

Problem



Problem: won't even see (s, a) if $a \neq \pi(s)$ ($a = \text{quit}$)



Key idea: exploration

To do reinforcement learning, need to explore the state space.

Solution: need π to **explore** explicitly (more on this later)

So we need to figure out how to get the data, to explore the state space, which is one of the key challenge of RL Solution: We need a random policy

Model-free Monte Carlo

From model-based to model-free

$$\hat{Q}_{\text{opt}}(s, a) = \sum_{s'} \hat{T}(s, a, s') [\widehat{\text{Reward}}(s, a, s') + \gamma \hat{V}_{\text{opt}}(s')]$$

All that matters for prediction is (estimate of) $Q_{\text{opt}}(s, a)$.



Key idea: model-free learning

Try to estimate $Q_{\text{opt}}(s, a)$ directly.

For MDP, at the end of the day, all we need is $Q_{opt}(s,a)$ $Q_{opt}(s,a)$ is: The maximum possible utility I could get if I'm in chance node (s,a) and I followed the optimal policy

So if we know $Q_{opt}(s,a)$, we would just need to follow the optimal policy and it's done, we don't need to know about the rewards/transitions.

So why don't we estimate Q_{opt} directly, without building up the whole MDP model

Model-free Monte Carlo

Data (following policy π):

$$s_0; a_1, r_1, s_1; a_2, r_2, s_2; a_3, r_3, s_3; \dots; a_n, r_n, s_n$$

Recall:

$Q_{\pi}(s, a)$ is expected utility starting at s , first taking action a , and then following policy π

Utility:

$$u_t = r_t + \gamma \cdot r_{t+1} + \gamma^2 \cdot r_{t+2} + \dots$$

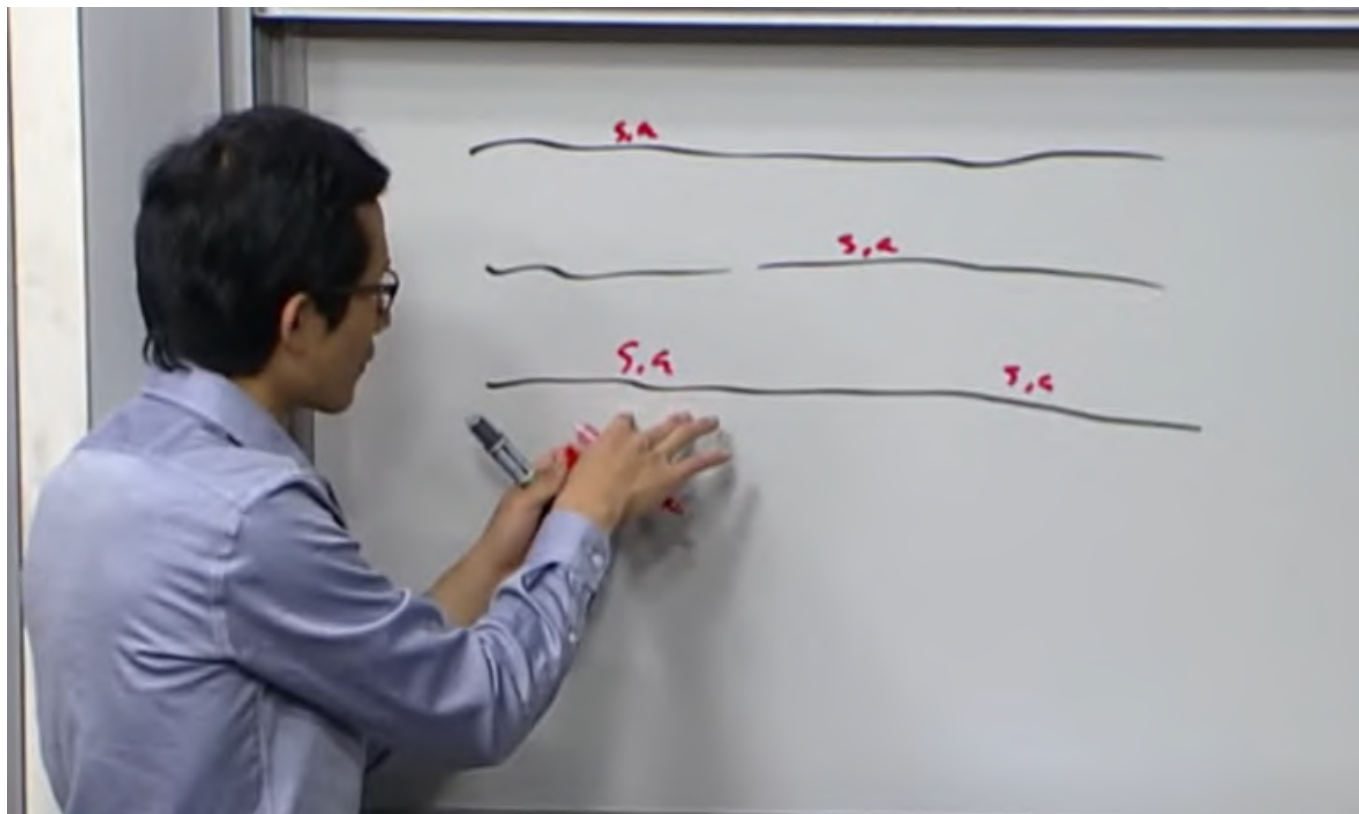
Estimate:

$$\hat{Q}_{\pi}(s, a) = \text{average of } u_t \text{ where } s_{t-1} = s, a_t = a$$

(and s, a doesn't occur in s_t, \dots)

Stanford

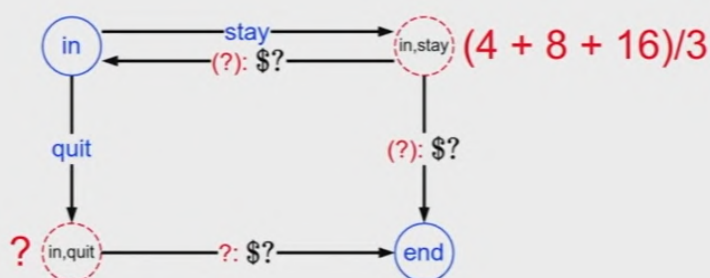
SO this means, you only Average the utility that you get, only on a timestamp (t) that i am in a particular state s and I (keep)took action a.



We just average whenever we see a utility of s,a but no double counting

The small hat $\hat{}$ on each notion means that: this comes from some quantity (estimate) and differentiate it from the true value

Model-free Monte Carlo



Data (following policy $\pi(s) = \text{stay}$):

[in; stay, 4, in; stay, 4, in; stay, 4, in; stay, 4, end]

Note: we are estimating Q_{π} now, not Q_{opt}



Definition: on-policy versus off-policy

On-policy: estimate the value of data-generating policy

Off-policy: estimate the value of another policy

Stanford

In RL we are always following some policy On-policy: you stick to one Off-policy: general, optimistic ???

Q/A: Is monte free always follow the same policy? A: It is just for this particular example, it is a on policy In other examples we can also try optimize the policy

Q/A: Which one is better, model free or model based? Interesting question. If u have a exact/correct model for the "world", model-based is kind of the way to go, because you need less data points and less computation(u already know the model!)

However in real world it is really hard to get the model 100% correct. Now with deep RL many ppl go for model free cuz u can solve difficult problem without constructing the MDP

Model-free Monte Carlo (equivalences)

Data (following policy π):

$$s_0; a_1, r_1, s_1; a_2, r_2, s_2; a_3, r_3, s_3; \dots; a_n, r_n, s_n$$

Original formulation

$$\hat{Q}_\pi(s, a) = \text{average of } u_t \text{ where } s_{t-1} = s, a_t = a$$

Equivalent formulation (convex combination)

On each (s, a, u) :

$$\eta = \frac{1}{1 + (\# \text{ updates to } (s, a))}$$

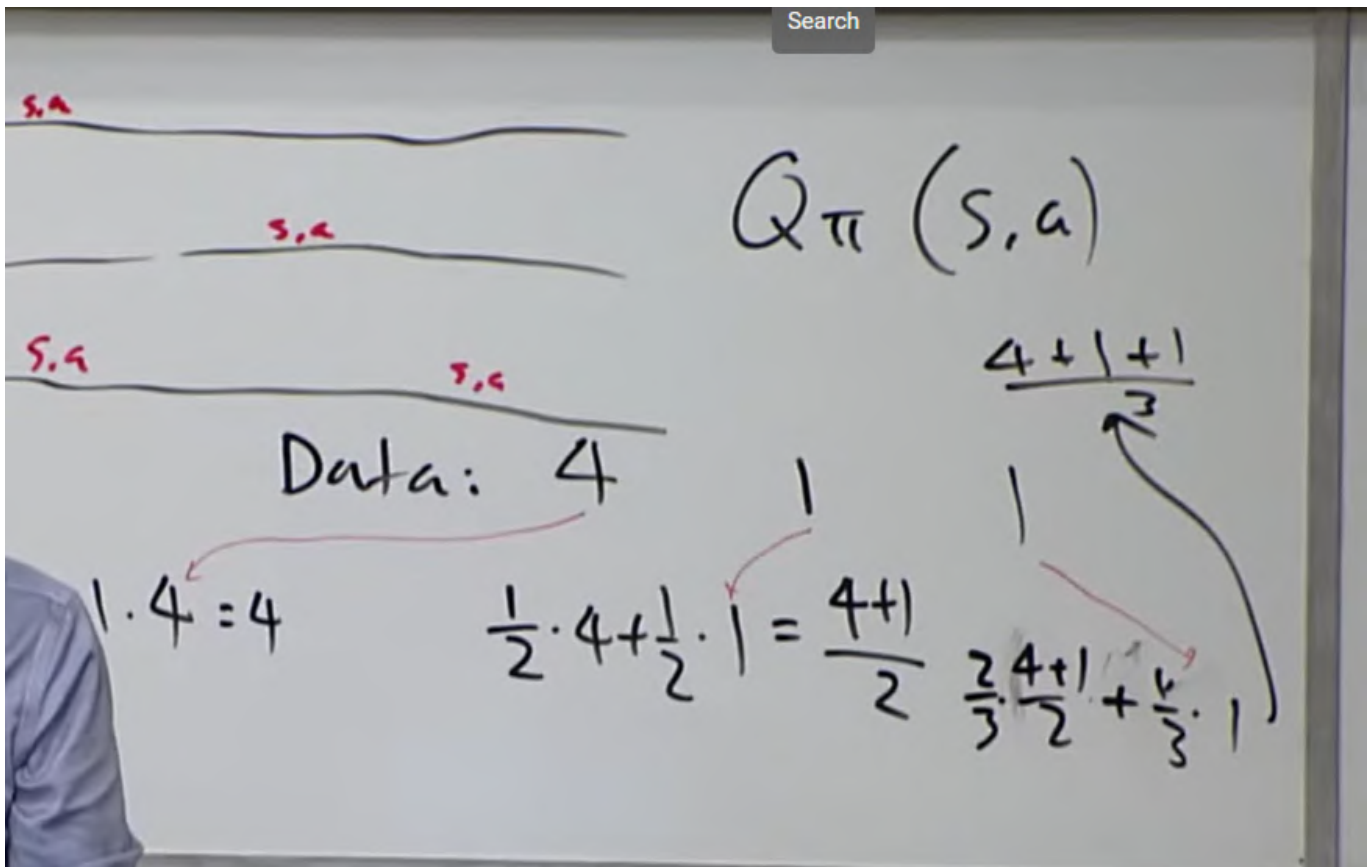
$$\hat{Q}_\pi(s, a) \leftarrow (1 - \eta)\hat{Q}_\pi(s, a) + \eta u$$

Interpretations

Average

We've talked about it

Convex Combination



Theme variations N : number of occurrences It is balancing between the old utilities/values that I had and the new utility that I saw

The old is with weight $1-n$, the new one is weight n $n = 1/\text{number of occurrences}$

Stochastic Gradient Decent

Model-free Monte Carlo (equivalences)

Equivalent formulation (convex combination)

On each (s, a, u) :

$$\hat{Q}_\pi(s, a) \leftarrow (1 - \eta)\hat{Q}_\pi(s, a) + \eta u$$

Equivalent formulation (stochastic gradient)

On each (s, a, u) :

$$\hat{Q}_\pi(s, a) \leftarrow \hat{Q}_\pi(s, a) - \eta \left[\underbrace{\hat{Q}_\pi(s, a)}_{\text{prediction}} - \underbrace{u}_{\text{target}} \right]$$

Implied objective: least squares regression

$$(\hat{Q}_\pi(s, a) - u)^2$$

u : the new data u've got All the updates are like:

Residual = Prediction - Target

This is implicitly doing SGD on the square loss of the Residual, so u want them to close to each other

An Exapmle

Volcanic model-free Monte Carlo

```

// Model
moveReward = 0 // For every action you take
passReward = 20 // If get to far green
volcanoReward = -50 // If fall into volcano
slipProb = 0 // If slip, go in random direction
discount = 1 // How much to value the future

// Algorithms
numEpisodes = 1000 // # simulations
eta = 0.5 // step size
epsilon = 1 // exploration probability
rl = "monte-carlo" // "monte-carlo", "sarsa", "q"
                    
```

Run

 (or press ctrl-enter)

Average utility: -18.5

a	r	s
E	0	(2,1)
S	0	(3,2)
N	0	(2,2)
S	0	(3,2)
E	0	(3,3)
E	0	(3,4)
E	0	(3,4)
S	0	(3,4)
N	0	(2,4)
S	0	(3,4)
S	0	(3,4)
E	0	(3,4)
W	0	(3,3)
N	-50	(2,3)

A state (square) divided into 4 pieces, which corresponding to 4 different actions(North south east west), and the number there is the Qpi value if take that action from that state The policy is : move completely

Bootstrapping

Using the utility

Data (following policy $\pi(s) = \text{stay}$):

[in; stay, 4, end]	$u = 4$
[in; stay, 4, in; stay, 4, end]	$u = 8$
[in; stay, 4, in; stay, 4, in; stay, 4, end]	$u = 12$
[in; stay, 4, in; stay, 4, in; stay, 4, in; stay, 4, end]	$u = 16$



Algorithm: model-free Monte Carlo

On each (s, a, u) :

$$\hat{Q}_{\pi}(s, a) \leftarrow (1 - \eta)\hat{Q}_{\pi}(s, a) + \eta \underbrace{u}_{\text{data}}$$

Each episode has a utility associated with it

The idea was to use average, with large amount of data to compute, u can get close to true value But with limited data the variance is big

The key idea is ?

SARSA

Using the reward + Q-value

Current estimate: $\hat{Q}_{\pi}(s, \text{stay}) = 11$

Data (following policy $\pi(s) = \text{stay}$):

[in; stay, 4, end]	$4 + 0$
[in; stay, 4, in; stay, 4, end]	$4 + 11$
[in; stay, 4, in; stay, 4, in; stay, 4, end]	$4 + 11$
[in; stay, 4, in; stay, 4, in; stay, 4, in; stay, 4, end]	$4 + 11$



Algorithm: SARSA

On each (s, a, r, s', a') :

$$\hat{Q}_{\pi}(s, a) \leftarrow (1 - \eta)\hat{Q}_{\pi}(s, a) + \eta \left[\underbrace{r}_{\text{data}} + \gamma \underbrace{\hat{Q}_{\pi}(s', a')}_{\text{estimate}} \right]$$

Why called SARSA: You are in state s , you took action a , you got a reward r , and you end up in state s' , and

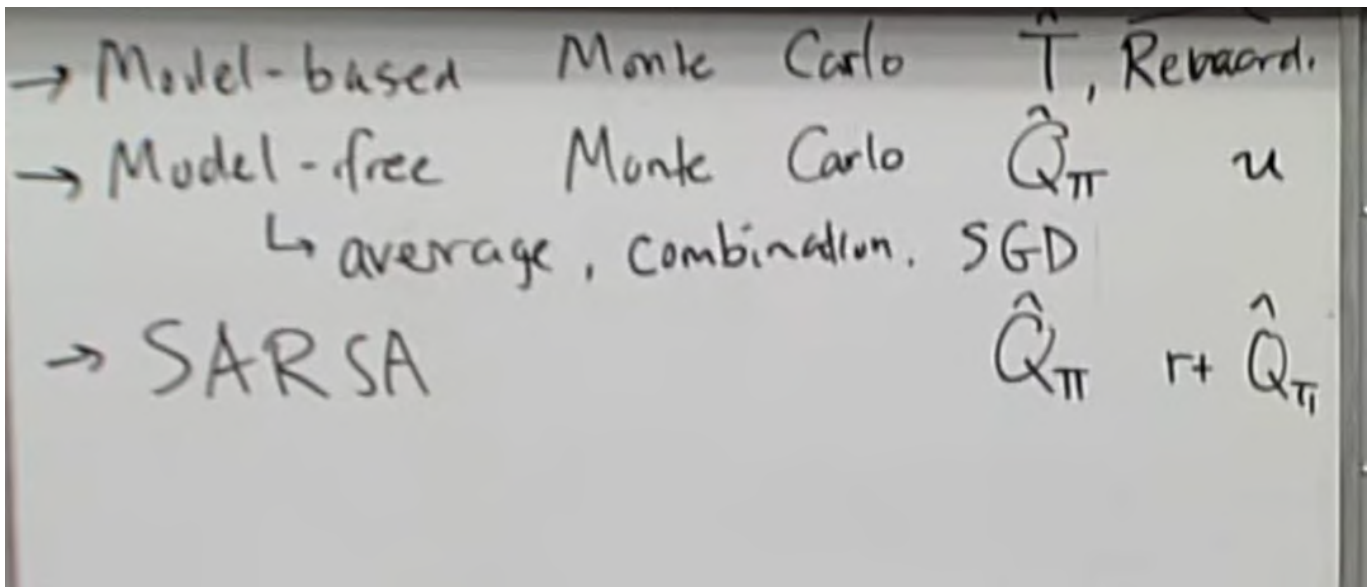
you took another action a' .

So for every quintuple that you see, you are gonna perform an update. And this is the convex combination, where you take part of the old value and try to merge them with the new value.

The new value is: the **immediat**e reward + the discount of your estimate.

Remember what is the estimate: It is trying to be the expectation of rewards that you will get in the future.

So if Q_{π} estimate could be close to the true value then this will be a lot better cuz it will reduce the variance.



SO Model-free monte carlo is updating based on u SARSA is updating based on **immediat**e reward r + the discount of your estimate.

Model-free Monte Carlo versus SARSA



Key idea: bootstrapping

SARSA uses estimate $\hat{Q}_{\pi}(s, a)$ instead of just raw data u .

u

based on one path

unbiased

large variance

wait until end to update

$r + \hat{Q}_{\pi}(s', a')$

based on estimate


biased

small variance

can update immediately

SARSA is biased because it's based on its previous experiences Monte Carlo - wait until end to update: Cuz you

need to reach the end state of your model so that u will know the utility But for SARSA you can immediately update because all u need to see is just a local windows of (s,a,r,s',a')

 cs221.stanford.edu/q

Question

Which of the following algorithms allows you to estimate $Q_{\text{opt}}(s, a)$ (select all that apply)?

model-based Monte Carlo

model-free Monte Carlo

SARSA

activate

deactivate

reset

report

A. With the exact model u can compute everything B & C. Model-free and SARSA are estimating the Q_{π} (the value of a policy) but not Q_{opt} , which is a problem, because all we want is the Q_{opt} That's when we need **Q-learning** which allows you to get Q_{opt}

Q-learning

Problem: model-free Monte Carlo and SARSA only estimate Q_π , but want Q_{opt} to act optimally

Output	MDP	reinforcement learning
Q_π	policy evaluation	model-free Monte Carlo, SARSA
Q_{opt}	value iteration	Q-learning

→ Model-based Monte Carlo $\hat{T}, \text{Revard.}$
 → Model-free Monte Carlo \hat{Q}_π u
 ↳ average, combination. SGD
 → SARSA \hat{Q}_π $r + \hat{Q}_\pi$
 → Q-learning \hat{Q}_{opt} $r + \hat{Q}_{\text{opt}}$

Q-learning

Q-learning

MDP recurrence:

$$Q_{\text{opt}}(s, a) = \sum_{s'} T(s, a, s') [\text{Reward}(s, a, s') + \gamma V_{\text{opt}}(s')]$$



Algorithm: Q-learning [Watkins/Dayan, 1992]

On each (s, a, r, s') :

$$\hat{Q}_{\text{opt}}(s, a) \leftarrow (1 - \eta) \underbrace{\hat{Q}_{\text{opt}}(s, a)}_{\text{prediction}} + \eta \underbrace{(r + \gamma \hat{V}_{\text{opt}}(s'))}_{\text{target}}$$

Recall: $\hat{V}_{\text{opt}}(s') = \max_{a' \in \text{Actions}(s')} \hat{Q}_{\text{opt}}(s', a')$

SARSA versus Q-learning



Algorithm: SARSA

On each (s, a, r, s', a') :

$$\hat{Q}_{\pi}(s, a) \leftarrow (1 - \eta) \hat{Q}_{\pi}(s, a) + \eta(r + \gamma \hat{Q}_{\pi}(s', a'))$$



Algorithm: Q-learning [Watkins/Dayan, 1992]

On each (s, a, r, s') :

$$\hat{Q}_{\text{opt}}(s, a) \leftarrow (1 - \eta) \hat{Q}_{\text{opt}}(s, a) + \eta(r + \gamma \max_{a' \in \text{Actions}(s')} \hat{Q}_{\text{opt}}(s', a'))]$$

For Q-learning it doesn't matter what action a' you gonna took for the next step, cuz we'll just take the one that maximizes(optimal one).

This is why SARSA is estimating the value of a policy

- Cuz what a' shows up there is a function of policy

and Q-learning is estimating the optimal policy cuz i don't care the next action, i'll just take the maximum(optimal) one

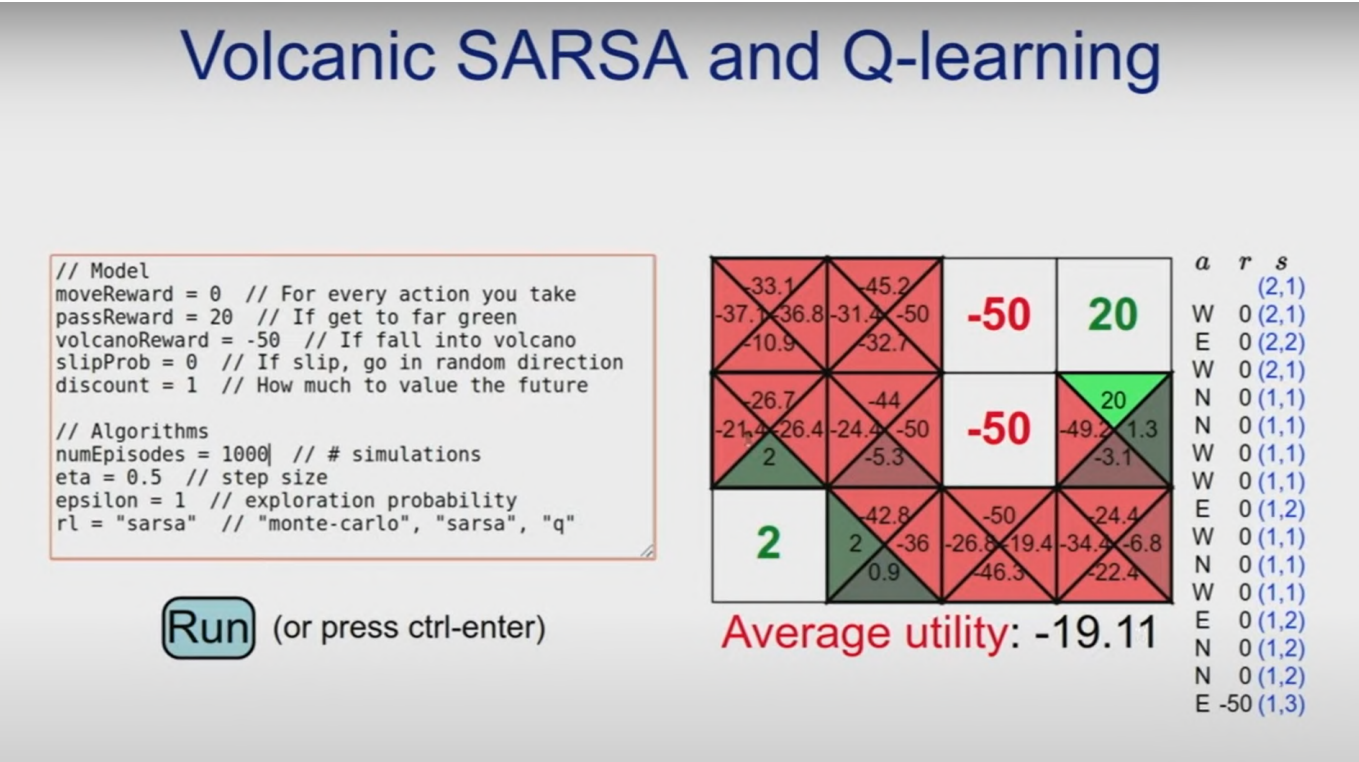
This is same ituation of policy evaluaiton and value iteration.

Q/A: Is q-learning on-policy or off-policy A: off-policy, cuz i am following whatever policy i can follow and estimate the value of the optimal policy, which is probably not the original policy at the beginning

While SARSA is a on-policy because i am estimating the value of Qpi

Volcano example

Notice that the slip prob is 0 here SARSA



The value here is Qpi, which is the value of the policy i'm following(the random policy)

Q-learning

Volcanic SARSA and Q-learning

```
// Model
moveReward = 0 // For every action you take
passReward = 20 // If get to far green
volcanoReward = -50 // If fall into volcano
slipProb = 0 // If slip, go in random direction
discount = 1 // How much to value the future
```

```
// Algorithms
numEpisodes = 1000 // # simulations
eta = 0.5 // step size
epsilon = 1 // exploration probability
rl = "q" // "monte-carlo", "sarsa", "q"
```

Run (or press ctrl-enter)



Average utility: -19.16

a	r	s
N	0	(2,1)
S	0	(2,1)
W	0	(2,1)
W	0	(2,1)
E	0	(2,2)
S	0	(3,2)
N	0	(2,2)
N	0	(1,2)
S	0	(2,2)
W	0	(2,1)
W	0	(2,1)
E	0	(2,2)
N	0	(1,2)
W	0	(1,1)
E	0	(1,2)
N	0	(1,2)
S	0	(2,2)
E	-50	(2,3)

The value of optimal policy Learning how to behave optimally

Encountering the Unknown

So we've just mentioned the algorithms. If I hand u some data and told u there's a fixed policy, you can estimate all those values



Exploration



Algorithm: reinforcement learning template

For $t = 1, 2, 3, \dots$

Choose action $a_t = \pi_{\text{act}}(s_{t-1})$ (**how?**)

Receive reward r_t and observe new state s_t

Update parameters (**how?**)

$s_0; a_1, r_1, s_1; a_2, r_2, s_2; a_3, r_3, s_3; \dots; a_n, r_n, s_n$

Which **exploration policy** π_{act} to use?

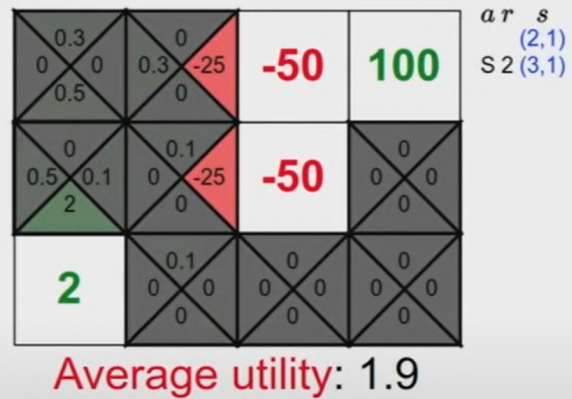
However there's a question of exploration: We don't even see all the states, how do u possibly act optimally

No exploration, all exploitation

Attempt 1: Set $\pi_{\text{act}}(s) = \arg \max_{a \in \text{Actions}(s)} \hat{Q}_{\text{opt}}(s, a)$

```
passReward = 100
numEpisodes = 1000 // # simulations
epsilon = 0 // all exploitation
rl = "q" // Q-learning
```

Run (or press ctrl-enter)



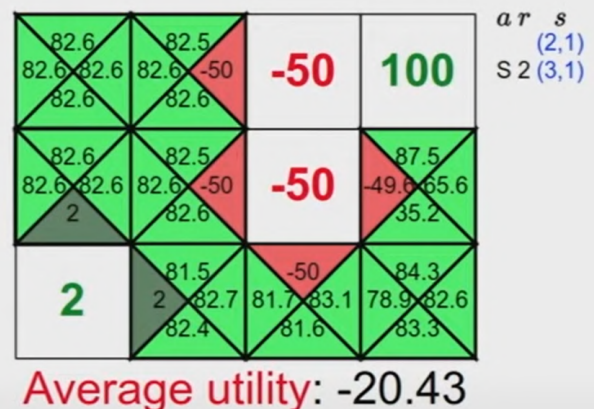
In this case, once we found a local optimum, we'll just stick to it lol

No exploitation, all exploration

Attempt 2: Set $\pi_{\text{act}}(s) = \text{random from Actions}(s)$

```
passReward = 100
numEpisodes = 1000 // # simulations
epsilon = 1 // all exploration
rl = "q" // Q-learning
```

Run (or press ctrl-enter)



Problem: average utility is low because exploration is **not guided**

I just try differently way and learn from it, however the average utility is still bad cuz the policy was just random policy The exploration is **not guided**

Exploration/exploitation tradeoff



Key idea: balance

Need to balance **exploration** and **exploitation**.



Examples from life: restaurants, routes, research

RL kind of capture the life You are aiming for a best reward, you have to learn how the world works and improves ur policy

Epsilon-greedy

Balancing the exploration and exploitation This assumes that ur doing Q-learning

Epsilon is the ramdon probility u specifiy We can do sth random once in a while

Epsilon-greedy



Algorithm: epsilon-greedy policy

$$\pi_{\text{act}}(s) = \begin{cases} \arg \max_{a \in \text{Actions}} \hat{Q}_{\text{opt}}(s, a) & \text{probability } 1 - \epsilon, \\ \text{random from Actions}(s) & \text{probability } \epsilon. \end{cases}$$

```
passReward = 100
numEpisodes = 1000 // # simulations
epsilon = [0, 0.5, 0] // balance
rl = "q" // Q-learning
```

Run (or press ctrl-enter)

	a	r	s
W	99.8	100	(2,1)
N	100	100	0 (2,1)
S	100	100	0 (1,1)
W	100	100	0 (2,1)
W	100	100	0 (2,1)
E	100	100	0 (2,2)
S	100	100	0 (3,2)
E	100	100	0 (3,3)
E	100	100	0 (3,4)
N	100	100	0 (2,4)
N	100	100	100 (1,4)

Average utility: 30.71

Epsilon When u turns older, you start to explore less and exploit more lol

Generalization

Generalization

Problem: large state spaces, hard to explore

actions
S0 (3,1)
S1 (4,1)
S2 (5,1)

Average utility: 0.44

Large state space

Q-learning

Stochastic gradient update:

$$\hat{Q}_{\text{opt}}(s, a) \leftarrow \hat{Q}_{\text{opt}}(s, a) - \eta \left[\underbrace{\hat{Q}_{\text{opt}}(s, a)}_{\text{prediction}} - \underbrace{(r + \gamma \hat{V}_{\text{opt}}(s'))}_{\text{target}} \right]$$

This is **rote learning**: every $\hat{Q}_{\text{opt}}(s, a)$ has a different value

Problem: doesn't generalize to unseen states/actions

I can't generalize between states and actions. Um, okay.

CS221 / Autumn 2018 / David

36

If we follow the current way, we cannot generalize between states and actions

Function approximation

Tell you new states that you haven't see before

Function approximation



Key idea: linear regression model

Define **features** $\phi(s, a)$ and **weights** \mathbf{w} :

$$\hat{Q}_{\text{opt}}(s, a; \mathbf{w}) = \mathbf{w} \cdot \phi(s, a)$$



Example: features for volcano crossing

$$\phi_1(s, a) = \mathbf{1}[a = \text{W}] \quad \phi_7(s, a) = \mathbf{1}[s = (5, *)]$$

$$\phi_2(s, a) = \mathbf{1}[a = \text{E}] \quad \phi_8(s, a) = \mathbf{1}[s = (*, 6)]$$

...

...

Features: eg: it's better to go to the West/East It's better to be at the 5th row/ 6th col

So we have a smaller set of features that we can try to use it, to generalize different states you might see.

Function approximation



Algorithm: Q-learning with function approximation

On each (s, a, r, s') :

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \left[\underbrace{\hat{Q}_{\text{opt}}(s, a; \mathbf{w})}_{\text{prediction}} - \underbrace{(r + \gamma \hat{V}_{\text{opt}}(s'))}_{\text{target}} \right] \phi(s, a)$$

Implied objective function:

$$\left(\underbrace{\hat{Q}_{\text{opt}}(s, a; \mathbf{w})}_{\text{prediction}} - \underbrace{(r + \gamma \hat{V}_{\text{opt}}(s'))}_{\text{target}} \right)^2$$

You take ur weight vector, take an update of residual * the feature vector. eta: like step size

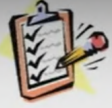
Covering the unknown



Epsilon-greedy: balance the exploration/exploitation tradeoff

Function approximation: can generalize to unseen states

Summary



Summary so far

- Online setting: learn and take actions in the real world!
- Exploration/exploitation tradeoff
- Monte Carlo: estimate transitions, rewards, Q-values from data
- Bootstrapping: update towards target that depends on estimate rather than just raw data

Challenges in reinforcement learning

Binary classification (sentiment classification, SVMs):

- Stateless, full feedback

Reinforcement learning (flying helicopters, Q-learning):

- Stateful, partial feedback



Key idea: partial feedback

Only learn about actions you take.



Key idea: state

Rewards depend on previous actions \Rightarrow can have delayed rewards.

Stanford

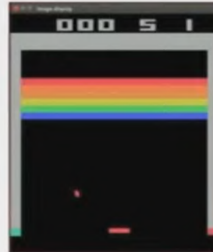
States and information

	stateless	state
full feedback	supervised learning (binary classification)	supervised learning (structured prediction)
partial feedback	multi-armed bandits	reinforcement learning

Deep reinforcement learning

just use a neural network for $\hat{Q}_{\text{opt}}(s, a)$

Playing Atari [Google DeepMind, 2013]:



- last 4 frames (images) \Rightarrow 3-layer NN \Rightarrow keystroke
- ϵ -greedy, train over 10M frames with 1M replay memory
- Human-level performance on some games (breakout), less good on others (space invaders)

Reinforcement learning is generally really hard, because of enormous states and delayed feedback.

Applications



Autonomous helicopters: control helicopter to do maneuvers in the air



Backgammon: TD-Gammon plays 1-2 million games against itself, human-level performance

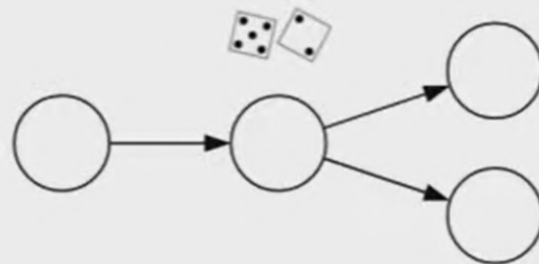


Elevator scheduling; send which elevators to which floors to maximize throughput of building



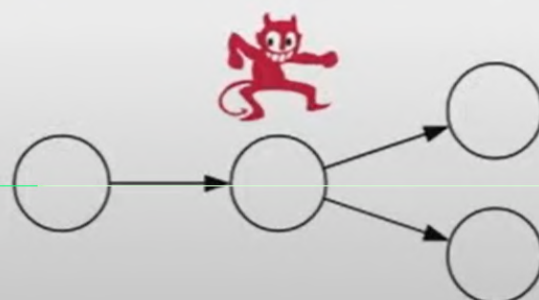
Managing datacenters; actions: bring up and shut down machine to minimize time/cost

Markov decision processes: against nature (e.g.,



Next time...

Adversarial games: against opponent (e.g., cl



For RL we are playing against nature Next time we'll play against an opponent who are trying to get us

