Intelligent Systems (AI-2)

Computer Science cpsc422, Lecture 8

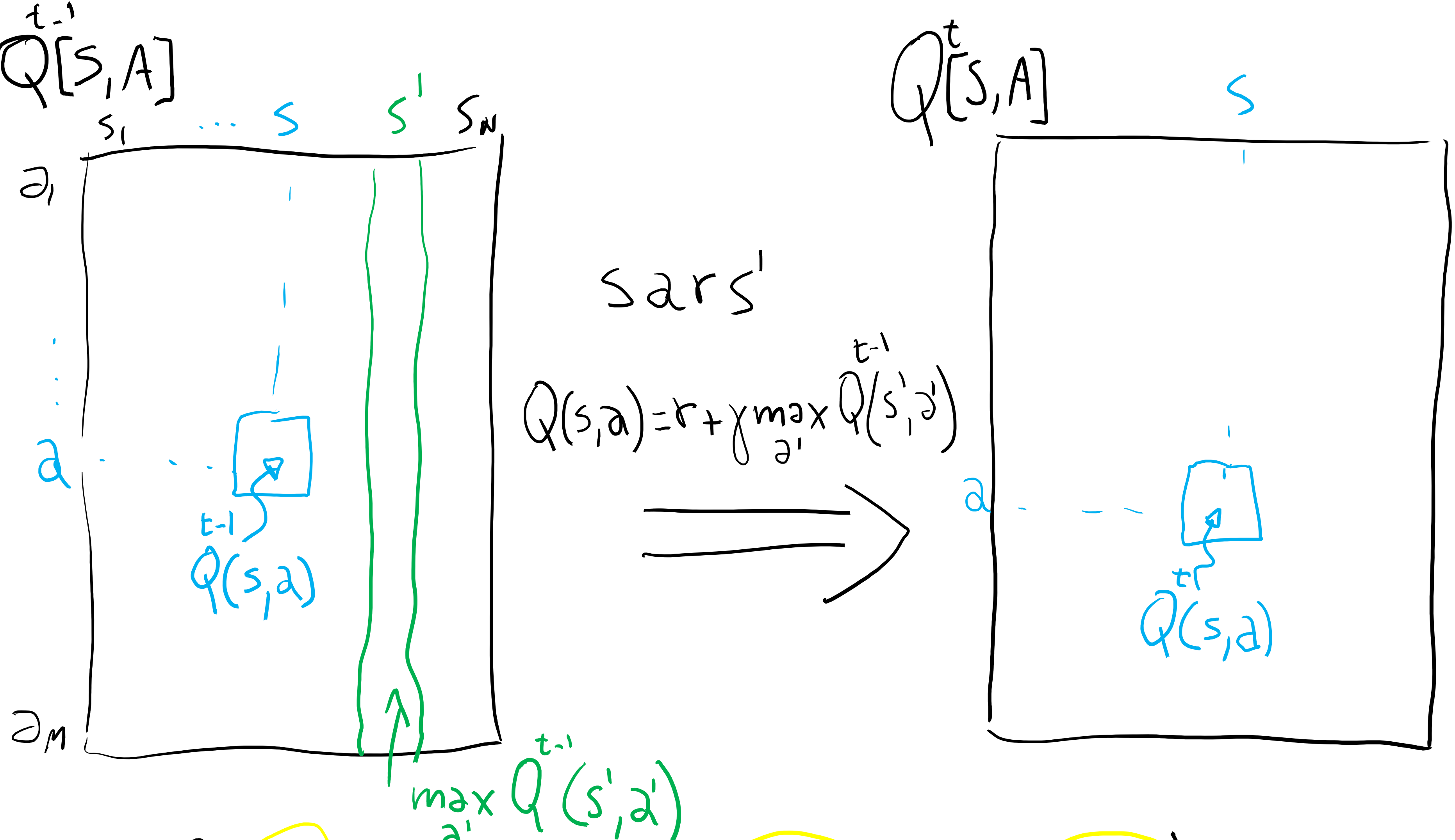
Sep, 26, 2016

Lecture Overview

Finish Q-learning

- Algorithm Summary
- Example

- Exploration vs. Exploitation



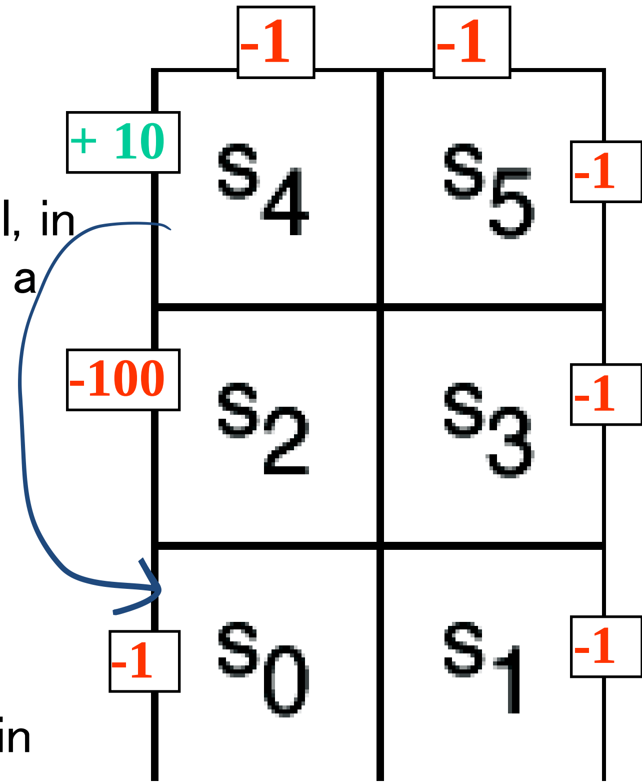
TD

$$A^{t} = A^{t-1} + \alpha_k \left(r^{t} - A^{t-1} \right)$$

$$Q^{t}(s,a) = Q^{t-1}(s,a) + \alpha_k \left(\left(r + \gamma \max_{a'} Q^{t-1}(s',a') \right) - Q^{t-1}(s,a) \right)$$

Example

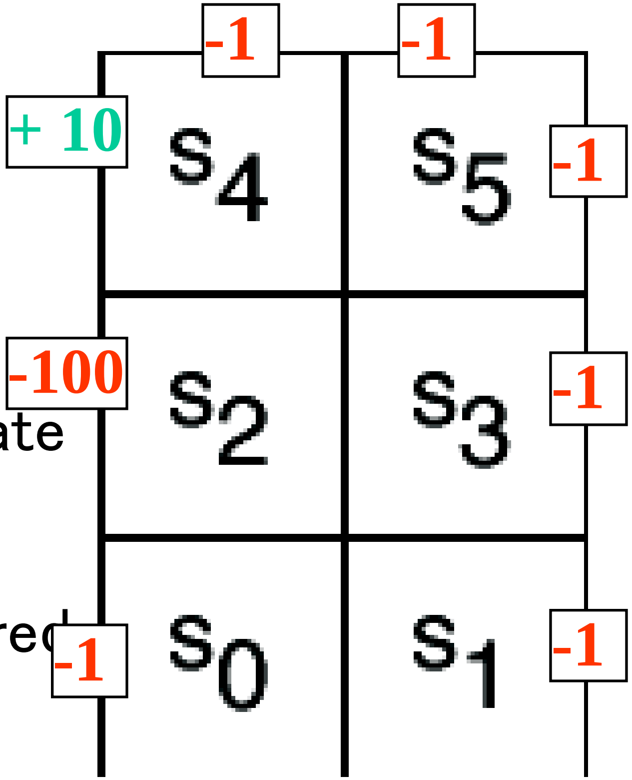
- Six possible states ⟨s_0,..,s_5⟩
- 4 actions:
  - *UpCareful:* moves one tile up unless there is wall, in which case stays in same tile. Always generates a penalty of −1
  - *Left:* moves one tile left unless there is wall, in which case
    - ✓ stays in same tile if in s_0 or s_2
    - ✓ Is sent to s_0 if in s_4
  - *Right:* moves one tile right unless there is wall, in which case stays in same tile
  - *Up:* 0.8 goes up unless there is a wall, 0.1 like *Left*, 0.1 like *Right*



Reward Model:

- −1 for doing *UpCareful*
- Negative reward when hitting a wall, as marked on the picture

Example



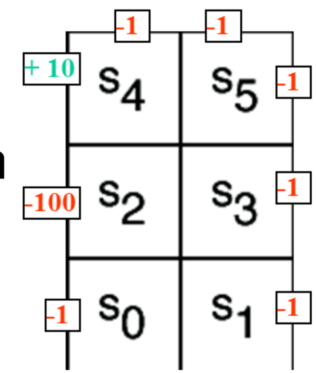
- The agent **knows** about the 6 states and 4 actions
- Can perform an action, fully observe its state and the reward it gets
- **Does not know** how the states are configured nor what the actions do
  - **no transition model, nor reward model**

Example (variable α_k)

- Suppose that in the simple world described earlier, the agent has the following sequence of experiences

  $<s_0$, *right*, 0, s_1, *upCareful*, -1, s_3, *upCareful*, -1, s_5, *left*, 0, s_4, *left*, 10, $s_0>$

- And repeats it *k* times (not a good behavior for a Q-learning agent, but good for didactic purposes)

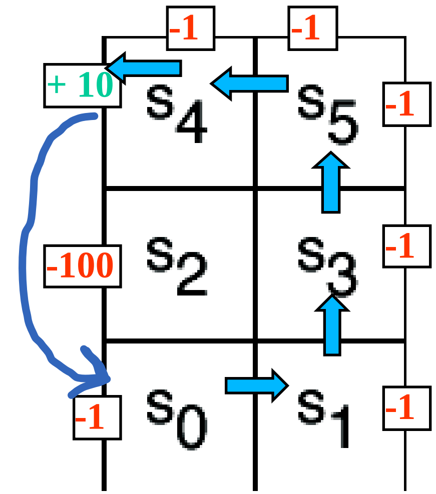


- Table shows the first 3 iterations of Q-learning when
  - *Q[s,a]* is initialized to 0 for every *a* and *s*
  - $\alpha_k = 1/k$, $\gamma = 0.9$

Iteration	$Q[s_0, right]$	$Q[s_1, upCare]$	$Q[s_3, upCare]$	$Q[s_5, left]$	$Q[s_4, left]$
1	0	-1	-1	0	10
2	0	-1	-1	4.5	10
3	0	-1	0.35	6.0	10

  - For full demo, see http://artint.info/demos/rl/tGame.html

$$\langle s_0, right, 0, s_1, upCareful, -1, s_3, upCareful, -1, s_5, left, 0, s_4, left, 10, s_0 \rangle$$



$$Q[s,a] \leftarrow Q[s,a] + \alpha((r + \gamma \max_{a'} Q[s',a']) - Q[s,a])$$

k=1

Q[s,a]	s_0	s_1	s_2	s_3	s_4	s_5
upCareful	0	0	0	0	0	0
Left	0	0	0	0	0	0
Right	0	0	0	0	0	0
Up	0	0	0	0	0	0

$$Q[s_0, right] \leftarrow Q[s_0, right] + \alpha_k((r + 0.9 \max_{a'} Q[s_1, a']) - Q[s_0, right]);$$

$$Q[s_0, right] \leftarrow$$

$$Q[s_1, upCareful] \leftarrow Q[s_1, upCareful] + \alpha_k((r + 0.9 \max_{a'} Q[s_3, a']) - Q[s_1, upCareful];$$

$$Q[s_1, upCareful] \leftarrow$$

$$Q[s_3, upCareful] \leftarrow Q[s_3, upCareful] + \alpha_k((r + 0.9 \max_{a'} Q[s_5, a']) - Q[s_3, upCareful];$$

$$Q[s_3, upCareful] \leftarrow$$

$$Q[s_5, Left] \leftarrow Q[s_5, Left] + \alpha_k((r + 0.9 \max_{a'} Q[s_4, a']) - Q[s_5, Left];$$

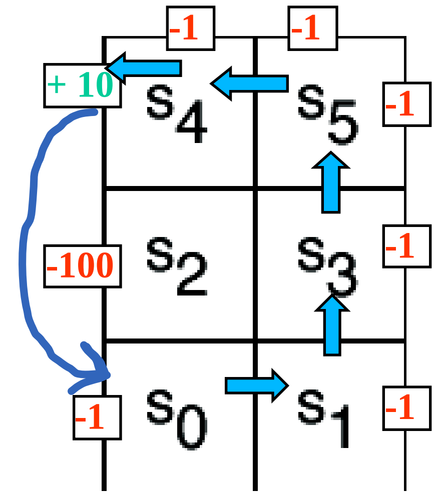
$$Q[s_5, Left] \leftarrow 0 + 1(0 + 0.9 * 0 - 0) = 0$$

Only immediate rewards are included in the update in this first pass

$$Q[s_4, Left] \leftarrow Q[s_4, Left] + \alpha_k((r + 0.9 \max_{a'} Q[s_0, a']) - Q[s_4, Left];$$

$$Q[s_4, Left] \leftarrow 0 + 1(10 + 0.9 * 0 - 0) = 10$$

$\langle s_0, right, 0, s_1, upCareful, -1, s_3, upCareful, -1, s_5, left, 0, s_4, left, 10, s_0 \rangle$



$$Q[s,a] \leftarrow Q[s,a] + \alpha((r + \gamma \max_{a'} Q[s',a']) - Q[s,a])$$

k=2

Q[s,a]	s_0	s_1	s_2	s_3	s_4	s_5
upCareful	**0**	**-1**	0	**-1**	0	0
Left	0	0	0	0	**10**	**0**
Right	0	0	0	0	0	0
Up	0	0	0	0	0	0

$$Q[s_0, right] \leftarrow Q[s_0, right] + \alpha_k((r + 0.9 \max_{a'} Q[s_1, a']) - Q[s_0, right]);$$

$$Q[s_0, right] \leftarrow 0 + 1/2(0 + 0.9 * 0 - 0) = 0$$

$$Q[s_1, upCareful] \leftarrow Q[s_1, upCareful] + \alpha_k((r + 0.9 \max_{a'} Q[s_3, a']) - Q[s_1, upCareful] =$$

$$Q[s_1, upCareful] \leftarrow -1 + 1/2(-1 + 0.9 * 0 + 1) = -1$$

$$Q[s_3, upCareful] \leftarrow Q[s_3, upCareful] + \alpha_k((r + 0.9 \max_{a'} Q[s_5, a']) - Q[s_3, upCareful] =$$

$$Q[s_3, upCareful] \leftarrow -1 + 1/2(-1 + 0.9 * 0 + 1) = -1$$

1 step backup from previous positive reward in s4

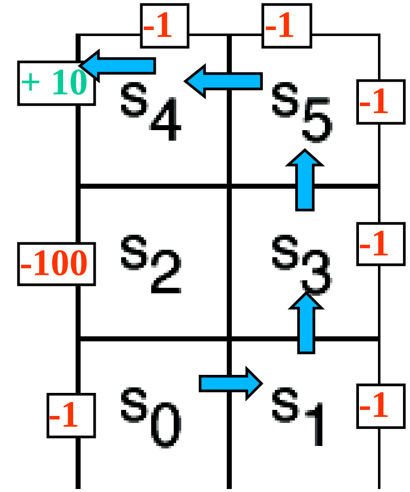
$$Q[s_5, Left] \leftarrow Q[s_5, Left] + \alpha_k((r + 0.9 \max_{a'} Q[s_4, a']) - Q[s_5, Left] =$$

$$Q[s_5, Left] \leftarrow$$

$$Q[s_4, Left] \leftarrow Q[s_4, Left] + \alpha_k((r + 0.9 \max_{a'} Q[s_0, a']) - Q[s_4, Left] =$$

$$Q[s_4, Left] \leftarrow 10 + 1(10 + 0.9 * 0 - 10) = 10$$

$\langle s_0, right, 0, s_1, upCareful, -1, s_3, upCareful, -1, s_5, left, 0, s_4, left, 10, s_0 \rangle$



$$Q[s,a] \leftarrow Q[s,a] + \alpha((r + \gamma \max_{a'} Q[s',a']) - Q[s,a])$$

k=3

Q[s,a]	s_0	s_1	s_2	s_3	s_4	s_5
upCareful	**0**	**-1**	0	**0.35**	0	0
Left	0	0	0	0	**10**	**6**
Right	0	0	0	0	0	0
Up	0	0	0	0	0	0

$$Q[s_0, right] \leftarrow Q[s_0, right] + \alpha_k((r + 0.9 \max_{a'} Q[s_1, a']) - Q[s_0, right]);$$

$$Q[s_0, right] \leftarrow 0 + 1/3(0 + 0.9 * 0 - 0) = 0$$

$$Q[s_1, upCareful] \leftarrow Q[s_1, upCareful] + \alpha_k((r + 0.9 \max_{a'} Q[s_3, a']) - Q[s_1, upCareful] =$$

$$Q[s_1, upCareful] \leftarrow -1 + 1/3(-1 + 0.9 * 0 + 1) = -1$$

$$Q[s_3, upCareful] \leftarrow Q[s_3, upCareful] + \alpha_k((r + 0.9 \max_{a'} Q[s_5, a']) - Q[s_3, upCareful] =$$

$$Q[s_3, upCareful] \leftarrow -1 + 1/3(-1 + 0.9 * 4.5 + 1) = 0.35$$

The effect of the positive reward in s4 is felt two steps earlier at the 3rd iteration

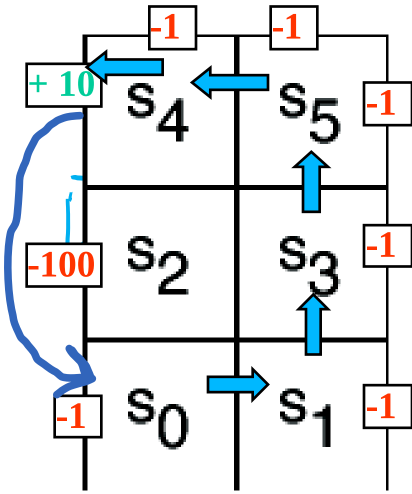
$$Q[s_5, Left] \leftarrow Q[s_5, Left] + \alpha_k((r + 0.9 \max_{a'} Q[s_4, a']) - Q[s_5, Left] =$$

$$Q[s_5, Left] \leftarrow 4.5 + 1/3(0 + 0.9 * 10 - 4.5) = 6$$

$$Q[s_4, Left] \leftarrow Q[s_4, Left] + \alpha_k((r + 0.9 \max_{a'} Q[s_0, a']) - Q[s_4, Left] =$$

$$Q[s_4, Left] \leftarrow 10 + 1/3(10 + 0.9 * 0 - 10) = 10$$

Example (variable α_k)



Iteration	$Q[s_0, right]$	$Q[s_1, upCare]$	$Q[s_3, upCare]$	$Q[s_5, left]$	$Q[s_4, left]$
1	0	-1	-1	0	10
2	0	-1	-1	4.5	10
3	0	-1	0.35	6.0	10
4	0	-0.92	1.36	6.75	10
10	0.03	0.51	4	8.1	10
100	2.54	4.12	6.82	9.5	11.34
1000	4.63	5.93	8.46	11.3	13.4
10000	6.08	7.39	9.97	12.83	14.9
100000	7.27	8.58	11.16	14.02	16.08
1000000	8.21	9.52	12.1	14.96	17.02
10000000	8.96	10.27	12.85	15.71	17.77
∞	11.85	13.16	15.74	18.6	20.66

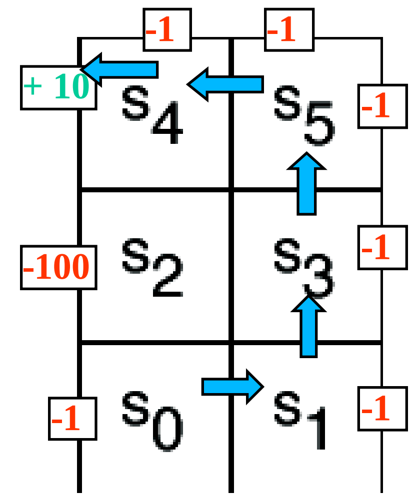
- As the number of iterations increases, the effect of the positive reward achieved by moving left in s_4 trickles further back in the sequence of steps
- $Q[s_4,left]$ starts changing only after the effect of the reward has reached s_0 (i.e. after iteration 10 in the table)

Example (Fixed α=1)

- First iteration same as before, let's look at the second

$$\langle s_0, right, 0, s_1, upCareful, -1, s_3, upCareful, -1, s_5, left, 0, s_4, left, 10, s_0 \rangle$$

$$Q[s,a] \leftarrow Q[s,a] + \alpha((r + \gamma \max_{a'} Q[s',a']) - Q[s,a])$$



k=2

Q[s,a]	s_0	s_1	s_2	s_3	s_4	s_5
upCareful	**0**	**-1**	0	**-1**	0	0
Left	0	0	0	0	**10**	**0**
Right	0	0	0	0	0	0
Up	0	0	0	0	0	0

$$Q[s_0, right] \leftarrow 0 + 1(0 + 0.9 * 0 - 0) = 0$$

$$Q[s_1, upCareful] \leftarrow -1 + 1(-1 + 0.9 * 0 + 1) = -1$$

$$Q[s_3, upCareful] \leftarrow -1 + 1(-1 + 0.9 * 0 + 1) = -1$$

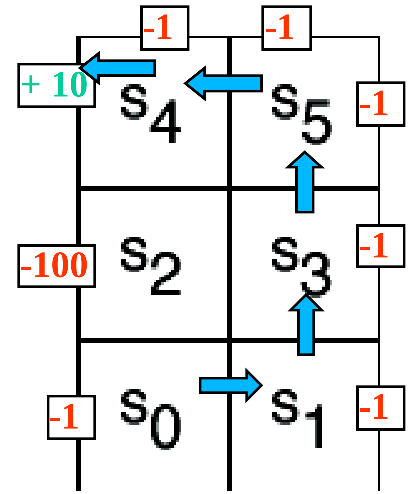
$$Q[s_5, Left] \leftarrow Q[s_5, Left] + \alpha_k((r + 0.9 \max_{a'} Q[s_4, a']) - Q[s_5, Left] =$$

$$Q[s_5, Left] \leftarrow 0 + 1(0 + 0.9 * 10 - 0) = 9$$

New evidence is given much more weight than original estimate

$$Q[s_4, Left] \leftarrow 10 + 1(10 + 0.9 * 0 - 10) = 10$$

$\langle s_0, right, 0, s_1, upCareful, -1, s_3, upCareful, -1, s_5, left, 0, s_4, left, 10, s_0 \rangle$



$$Q[s,a] \leftarrow Q[s,a] + \alpha((r + \gamma \max_{a'} Q[s',a']) - Q[s,a])$$

k=3

Q[s,a]	s_0	s_1	s_2	s_3	s_4	s_5
upCareful	**0**	**-1**	0	**-1**	0	0
Left	0	0	0	0	**10**	**9**
Right	0	0	0	0	0	0
Up	0	0	0	0	0	0

$$Q[s_0, right] \leftarrow 0 + 1(0 + 0.9 * 0 - 0) = 0$$

$$Q[s_1, upCareful] \leftarrow -1 + 1(-1 + 0.9 * 0 + 1) = -1$$

$$Q[s_3, upCareful] \leftarrow Q[s_3, upCareful] + \alpha_k((r + 0.9 \max_{a'} Q[s_5, a']) - Q[s_3, upCareful] =$$

Same here

$$Q[s_3, upCareful] \leftarrow -1 + 1(-1 + 0.9 * 9 + 1) = 7.1$$

$$Q[s_5, Left] \leftarrow 9 + 1(0 + 0.9 * 10 - 9) = 9$$

$$Q[s_4, Left] \leftarrow 10 + 1(10 + 0.9 * 0 - 10) = 10$$

No change from previous iteration, as all the reward from the step ahead was included there

Comparing fixed α and …

Iteration	$Q[s_0, right]$	$Q[s_1, upCare]$	$Q[s_3, upCare]$	$Q[s_5, left]$	$Q[s_4, left]$
1	0	-1	-1	0	10
2	0	-1	-1	9	10
3	0	-1	7.1	9	10
4	0	5.39	7.1	9	10
5	4.85	5.39	7.1	9	14.37
6	4.85	5.39	7.1	12.93	14.37
10	7.72	8.57	10.64	15.25	16.94
20	10.41	12.22	14.69	17.43	19.37
30	11.55	12.83	15.37	18.35	20.39
40	11.74	13.09	15.66	18.51	20.57
∞	11.85	13.16	15.74	18.6	20.66

variable α

Iteration	$Q[s_0, right]$	$Q[s_1, upCare]$	$Q[s_3, upCare]$	$Q[s_5, left]$	$Q[s_4, left]$
1	0	-1	-1	0	10
2	0	-1	-1	4.5	10
3	0	-1	0.35	6.0	10
4	0	-0.92	1.36	6.75	10
10	0.03	0.51	4	8.1	10
100	2.54	4.12	6.82	9.5	11.34
1000	4.63	5.93	8.46	11.3	13.4
10000	6.08	7.39	9.97	12.83	14.9
100000	7.27	8.58	11.16	14.02	16.08
1000000	8.21	9.52	12.1	14.96	17.02
10000000	8.96	10.27	12.85	15.71	17.77
∞	11.85	13.16	15.74	18.6	20.66

Fixed α generates faster update:

all states see some effect of the positive reward from <s4, left> by the 5th iteration

Each update is much larger

Gets very close to final numbers by iteration 40, while with variable α still not there by iteration 10^7

However:

Q-learning with fixed α is not guaranteed to converge

On the approximation…

$$Q(s,a) = R(s) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q(s',a')$$

True relation between Q(s.a) and Q(s'a')

$$Q[s,a] \leftarrow Q[s,a] + \alpha((r + \gamma \max_{a'} Q[s',a']) - Q[s,a])$$

Q-learning approximation based on each individual experience $<s, a, r, s'>$

- For the approximation to work…..

A. There is positive reward in most states

B. Q-learning tries each action an unbounded number of times

C. The transition model is not sparse

Why approximations work…

$$Q(s,a) = R(s) + \gamma \sum_{s'} \underbrace{P(s' \mid s,a)}_{} \max_{a'} Q(s',a')$$

True relation between Q(s.a) and Q(s'a')

$$Q[s,a] \leftarrow Q[s,a] + \alpha(\underbrace{(r + \gamma \max_{a'} Q[s',a'])}_{} - Q[s,a])$$

Q-learning approximation based on each individual experience *<s, a, s'>*

- Way to get around the **missing transition model and reward model**
- Aren't we in danger of using data coming from unlikely transition to make incorrect adjustments?
- No, as long as Q-learning tries each action an unbounded number of times
    - Frequency of updates reflects transition model, *P(s'|a,s)*

Lecture Overview

Finish Q-learning

- Algorithm
- Example

- Exploration vs. Exploitation

What Does Q-Learning learn

- Does Q-learning gives the agent an optimal policy?

Q values

	s_0	s_1	…	s_k
a_0	$Q[s_0,a_0]$	$Q[s_1,a_0]$	….	$Q[s_k,a_0]$
a_1	$Q[s_0,a_1]$	$Q[s_1,a_1]$	…	$Q[s_k,a_1]$
…	…	…	….	…
a_n	$Q[s_0,a_n]$	$Q[s_1,a_n]$	….	$Q[s_k,a_n]$

what to do in S_1

$\arg\max_a Q[s_1, a]$

Exploration vs. Exploitation

- Q-learning does not explicitly tell the agent what to do

- just computes a Q-function Q[s,a] that allows the agent to see, for every state, which is the action with the highest expected reward

- Given a Q-function the agent can :

  - **Exploit** the knowledge accumulated so far, and chose the action that maximizes Q[s,a] in a given state (***greedy behavior***)

  - **Explore** new actions, hoping to improve its estimate of the optimal Q-function, i.e. *do not chose* the action suggested by the current Q[s,a]

Exploration vs. Exploitation

- When to explore and when the exploit?
  1. Never exploring may lead to being stuck in a suboptimal course of actions
  2. Exploring too much is a waste of the knowledge accumulated via experience

A. Only (1) is true

B. Only (2) is true

C. Both are true

D. Both are false

iclicker

Exploration vs. Exploitation

- When to explore and when the exploit?
  - Never exploring may lead to being stuck in a suboptimal course of actions
  - Exploring too much is a waste of the knowledge accumulated via experience
- Must find the right compromise

Exploration Strategies

- Hard to come up with an optimal exploration policy (problem is widely studied in *statistical decision theory*)

- But intuitively, any such strategy should be *greedy in the limit of infinite exploration* (**GLIE**), i.e.
  - **Choose the predicted best action in the limit**
  - **Try each action an unbounded number of times**

- We will look at two exploration strategies
  - ε-greedy
  - soft-max

ε-greedy

- Choose a **random action with probability ε** and choose **best action with probability 1- ε**

$P(\text{random action}) = \varepsilon$

$P(\text{best action}) = 1 - \varepsilon$

- First GLIE condition (try every action an unbounded number of times) is satisfied via the ε random selection

- What about second condition?
  - Select predicted best action in the limit.

- reduce ε overtime!

Soft-Max

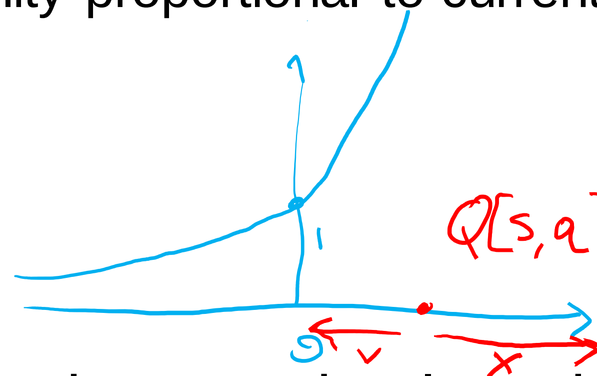
if Q[s,a] close to 0 each action selected with prob $\frac{1}{\# \text{of actions}}$ UNIFORM DISTRIB.

- Takes into account improvement in estimates of expected reward function Q[s,a]
  - Choose action ***a*** in state ***s*** with a probability proportional to current estimate of **Q[s,a]**

$$\frac{e^{Q[s,a]}}{\sum_a e^{Q[s,a]}}$$

or controlled by τ parameter

$$\frac{e^{Q[s,a]/\tau}}{\sum_a e^{Q[s,a]/\tau}}$$



- τ (tau) in the formula above influences how randomly actions should be chosen
  - if τ is high, the exponentials approach 1, the fraction approaches 1/(number of actions), and each action has approximately the same probability of being chosen (exploration or exploitation?)
  - as $\tau \to 0$, the exponential with the highest Q[s,a] dominates, and the current best action is always chosen (exploration or exploitation?)

Learning Goals for today's class

➢**You can:**

- Explain, trace and implement Q-learning
- Describe and compare techniques to combine exploration with exploitation

TODO for Wed

- **Carefully read : A Markov decision process approach to multi-category patient scheduling in a diagnostic facility,** Artificial Intelligence in Medicine Journal, **2011**

- **Follow instructions on course WebPage** <Readings>

- Keep working on assignment-1 (due next Mon)

Overview (NOT FOR 422)

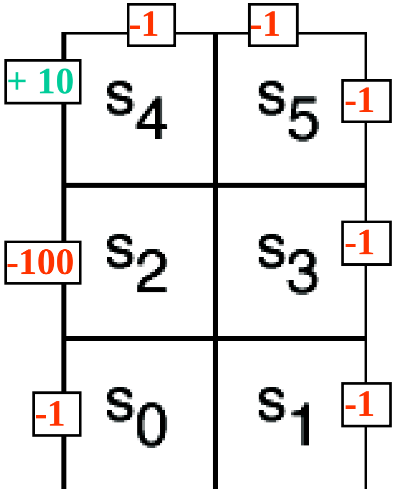
- Introduction
- Q-learning
- Exploration vs. Exploitation
- Evaluating RL algorithms
- On-Policy Learning: SARSA
- Model-based Q-learning

Learning before vs. during deployment

- As we saw earlier, there are two possible modus operandi for our learning agents
  - act in the environment to learn how it works:
    - ✓ first learn an optimal policy, *then* use this policy to act (there is a learning phase *before* deployment)
  - Learn as you go:
    - ✓ start operating in the environment right away and learn from actions (learning happens *during* deployment)
- If there is time to learn before deployment, the agent should try to do its best to learn as much as possible about the environment
  - even engage in locally suboptimal behaviors, because this will guarantee reaching an optimal policy in the long run
- If learning while "at work", suboptimal behaviors could be costly

Example

- Consider, for instance, our sample grid game:
  - the optimal policy is to go *up* in S_0
  - But if the agent includes some exploration in its policy (e.g. selects 20% of its actions randomly), exploring in S_2 could be dangerous because it may cause hitting the -100 wall
  - No big deal if the agent is not deployed yet, but not ideal otherwise



- Q-learning would not detect this problem
  - It does *off-policy learning*, i.e., it focuses on the optimal policy
- *On-policy* learning addresses this problem

On-policy learning: SARSA

- On-policy learning learns the value of the policy being followed.
  - e.g., act greedily 80% of the time and act randomly 20% of the time
  - Better to be aware of the consequences of exploration has it happens, and avoid outcomes that are too costly while acting, rather than looking for the true optimal policy
- SARSA
  - So called because it uses <*state*, *action*, *reward*, *state*, *action*> experiences rather than the <*state*, *action*, *reward*, *state*> used by Q-learning
  - Instead of looking for the best action at every step, it evaluates the actions suggested by the current policy
  - Uses this info to revise it

On-policy learning: SARSA

- Given an experience *<s,a,r,s',a'>*, *SARSA* updates Q[s,a] as follows

$$Q[s,a] \leftarrow Q[s,a] + \alpha((r + \gamma Q[s',a']) - Q[s,a])$$

What's different from Q-learning?

On-policy learning: SARSA

- Given an experience *<s ,a, r, s', a'>, SARSA* updates Q[s,a] as follows

$$Q[s,a] \leftarrow Q[s,a] + \alpha((r + \gamma Q[s',a']) - Q[s,a])$$

- While Q-learning was using

$$Q[s,a] \leftarrow Q[s,a] + \alpha((r + \gamma \max_{a'} Q[s',a']) - Q[s,a])$$

- There is no more MAX operator in the equation, there is instead the Q-value of the action suggested by the policy

On-policy learning: SARSA

- Does SARSA remind you of any other algorithm we have seen before?

Policy Iteration

- Algorithm
  - π ← an arbitrary initial policy, U ← A vector of utility values, initially 0
  - 2. Repeat until no change in π

(a) Compute new utilities given π and current U (***policy evaluation***)

$$U(s) = R(s) + \gamma \sum_{s'} T(s, \pi_i(s), s') U(s')$$

(b) Update π as if utilities were correct (***policy improvement***)

Expected value of following another action in s

Expected value of following current πi from *s*

For ∀s

Policy Improvement step

$$If \max_a \sum_{s'} T(s, a, s') U(s') > \sum_{s'} T(s, \pi_i(s), s') U(s')$$

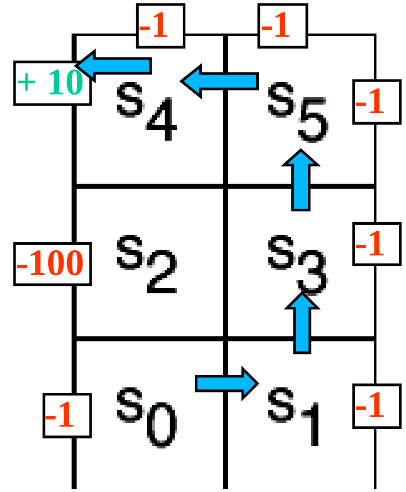
$$\text{then } \pi_i(s) \leftarrow \arg\max_a \sum_{s'} T(s, a, s') U^{\pi_i}(s')$$

$\langle s_0, right, 0, s_1, upCareful, -1, s_3, upCareful, -1, s_5, left, 0, s_4, left, 10, s_0 \rangle$

$$Q[s,a] \leftarrow Q[s,a] + \alpha(r + \gamma Q[s',a'] - Q[s,a])$$

k=1

Q[s,a]	s_0	s_1	s_2	s_3	s_4	s_5
upCareful	0	0	0	0	0	0
Left	0	0	0	0	0	0
Right	0	0	0	0	0	0
Up	0	0	0	0	0	0



$$Q[s_0, right] \leftarrow Q[s_0, right] + \alpha_k(r + 0.9Q[s_1, UpCareful] - Q[s_0, right]);$$
$$Q[s_0, right] \leftarrow$$

$$Q[s_1, upCarfull] \leftarrow Q[s_1, upCarfull] + \alpha_k(r + 0.9Q[s_3, UpCareful] - Q[s_1, upCarfull]);$$
$$Q[s_1, upCarfull] \leftarrow$$

$$Q[s_3, upCarfull] \leftarrow Q[s_3, upCarfull] + \alpha_k(r + 0.9Q[s_5, Left] - Q[s_3, upCarfull]);$$
$$Q[s_3, upCarfull] \leftarrow 0 + 1(-1 + 0.9 * 0 - 0) = -1$$

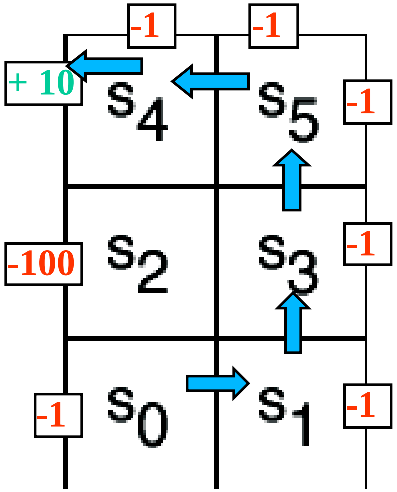
$$Q[s_5, Left] \leftarrow Q[s_5, Left] + \alpha_k(r + 0.9Q[s_4, left] - Q[s_5, Left]);$$
$$Q[s_5, Left] \leftarrow 0 + 1(0 + 0.9 * 0 - 0) = 0$$

Only immediate rewards are included in the update, as with Q-learning

$$Q[s_4, Left] \leftarrow Q[s_4, Left] + \alpha_k(r + 0.9Q[s_0, Right] - Q[s_4, Left]);$$
$$Q[s_4, Left] \leftarrow 0 + 1(10 + 0.9 * 0 - 0) = 10$$

$\langle s_0, right, 0, s_1, upCareful, -1, s_3, upCareful, -1, s_5, left, 0, s_4, left, 10, s_0 \rangle$

$$Q[s,a] \leftarrow Q[s,a] + \alpha(r + \gamma Q[s',a'] - Q[s,a])$$



k=2

Q[s,a]	s_0	s_1	s_2	s_3	s_4	s_5
upCareful	**0**	**-1**	0	**-1**	0	0
Left	0	0	0	0	**10**	**0**
Right	0	0	0	0	0	0
Up	0	0	0	0	0	0

$Q[s_0, right] \leftarrow Q[s_0, right] + \alpha_k(r + 0.9Q[s_1, UpCareful] - Q[s_0, right]);$

$Q[s_0, right] \leftarrow$

SARSA backs up the expected reward of the next action, rather than the max expected reward

$Q[s_1, upCarfull] \leftarrow Q[s_1, upCarfull] + \alpha_k(r + 0.9Q[s_3, UpCareful] - Q[s_1, upCarfull]);$

$Q[s_1, upCarfull] \leftarrow$

$Q[s_3, upCarfull] \leftarrow Q[s_3, upCarfull] + \alpha_k(r + 0.9Q[s_5, Left] - Q[s_3, upCarfull]);$

$Q[s_3, upCarfull] \leftarrow -1 + 1/2(-1 + 0.9*0 + 1) = -1$

$Q[s_5, Left] \leftarrow Q[s_5, Left] + \alpha_k(r + 0.9Q[s_4, left] - Q[s_5, Left]);$

$Q[s_5, Left] \leftarrow 0 + 1/2(0 + 0.9*10 - 0) = 4.5$

$Q[s_4, Left] \leftarrow Q[s_4, Left] + \alpha_k(r + 0.9Q[s_0, Right] - Q[s_4, Left]);$

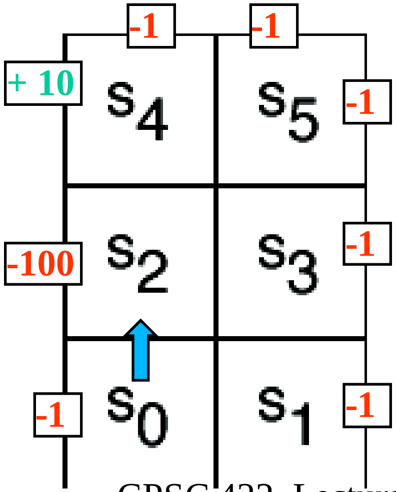
$Q[s_4, Left] \leftarrow 10 + 1/2(10 + 0.9*0 - 10) = 10$

Comparing SARSA and Q-learning

- For the little 6-states world

- Policy learned by Q-learning 80% greedy is to go *up* in s_0 to reach s_4 quickly and get the big +10 reward

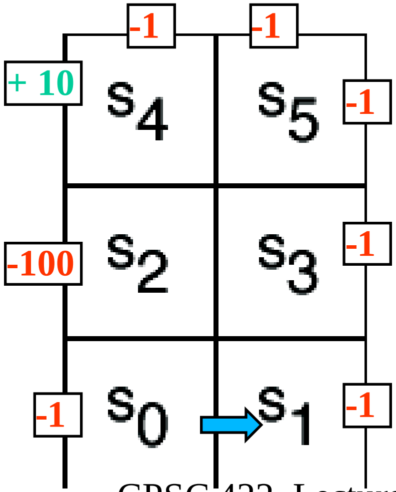
Iteration	$Q[s_0, right]$	$Q[s_1, upC]$	$Q[s_3, upC]$	$Q[s_5, left]$	$Q[s_4, left]$
∞	19.5	21.14	24.08	27.87	30.97



Comparing SARSA and Q-learning

- Policy learned by SARSA 80% greedy is to go *left* in s_0
- Safer because avoid the chance of getting the -100 reward in s_2
- but non-optimal => lower q-values

Iteration	$Q[s_0, right]$	$Q[s_1, upC]$	$Q[s_3, upC]$	$Q[s_5, left]$	$Q[s_4, left]$
∞	9.2	10.1	12.7	15.7	18.0



SARSA Algorithm

begin
initialize $Q[S, A]$ arbitrarily
observe current state s
select action a using a policy based on Q
repeat forever:
carry out an action a
observe reward r and state s'
select action a' using a policy based on Q
$Q[s, a] \leftarrow Q[s, a] + \alpha \left(r + \gamma Q[s', a'] - Q[s, a]\right)$
$s \leftarrow s'$;
$a \leftarrow a'$;
end-repeat
end

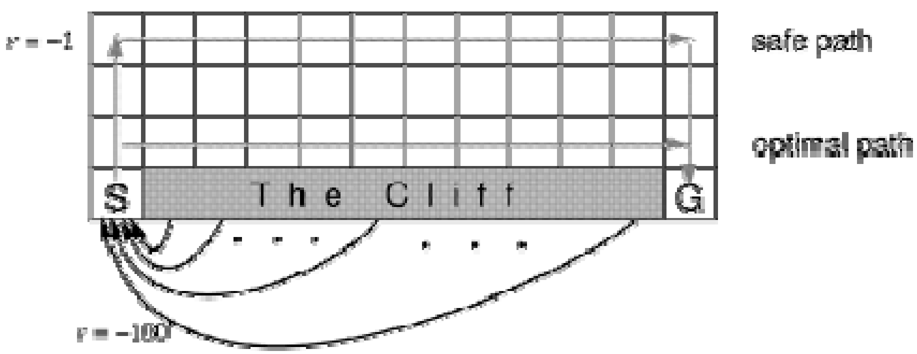
This could be, for instance any ε-greedy strategy:
-Choose random ε times, and max the rest

If the random step is chosen here, and has a bad negative reward, this will affect the value of Q[s,a].

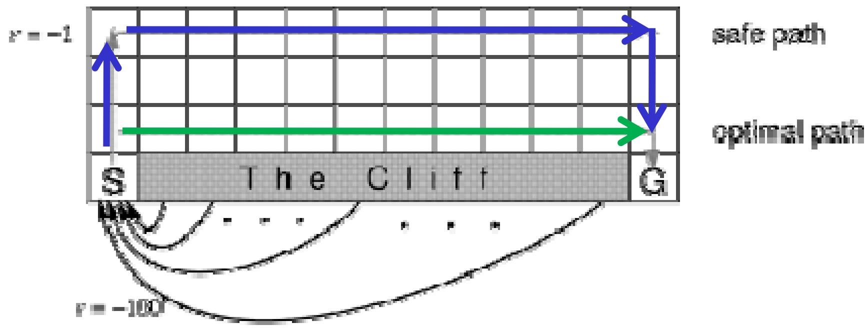
Next time in *s*, *a'* may no longer be the action selected because of its lowered Q value

Another Example

- Gridworld with:
  - Deterministic actions *up*, *down*, *left*, *right*
  - Start from S and arrive at G
  - Reward is -1 for all transitions, except those into the region marked "Cliff"
    - ✓ Falling into the cliff causes the agent to be sent back to start: **r = -100**



Another Example

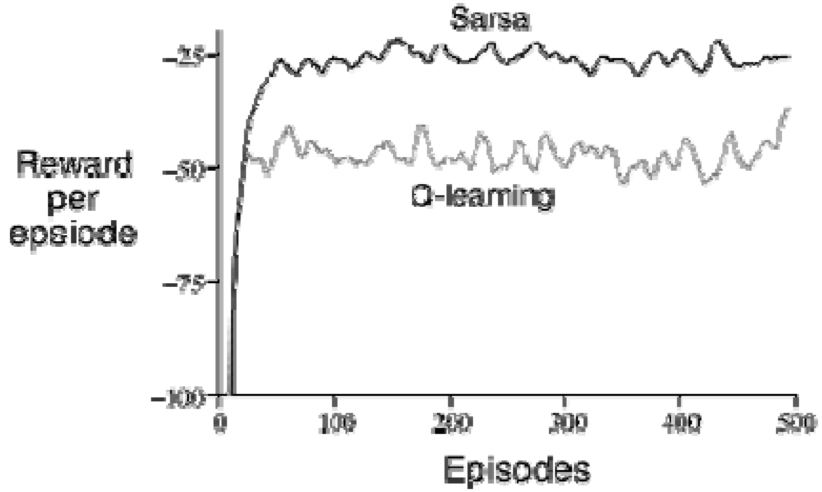


➢ Because of negative reward for every step taken, the optimal policy over the four standard actions is to take the shortest path along the cliff

➢ But if the agents adopt an ε-greedy action selection strategy with ε=0.1, walking along the cliff is dangerous

- The optimal path that considers exploration is to go around as far as possible from the cliff

Q-learning vs. SARSA



- Q-learning learns the optimal policy, but because it does so without taking exploration into account, it does not do so well while the agent is exploring
  - It occasionally falls into the cliff, so its reward per episode is not that great
- SARSA has better on-line performance (reward per episode), because it learns to stay away from the cliff while exploring
  - But note that if ε→0, SARSA and Q-learning would asymptotically converge to the optimal policy

Problem with Model-free methods

- Q-learning and SARSA are model-free methods

What does this mean?

Problems With Model-free Methods

- Q-learning and SARSA are model-free methods
  - They do not need to learn the transition and/or reward model, they are implicitly taken into account via experiences
- Sounds handy, but there is a main disadvantage:
  - How often does the agent get to update its Q-estimates?

Problems with Model-free Methods

- Q-learning and SARSA are model-free methods
  - They do not need to learn the transition and/or reward model, they are implicitly taken into account via experiences
- Sounds handy, but there is a main disadvantage:
  - How often does the agent get to update its Q-estimates?
  - Only after a new experience comes in
  - Great if the agent acts very frequently, not so great if actions are sparse, because it wastes computation time

Model-based methods

- Idea
  - learn the MDP and interleave acting and planning.
- After each experience,
  - update probabilities and the reward,
  - do some steps of value iteration (asynchronous) to get better estimates of state utilities U(s) given the current model and reward function
  - Remember that there is the following link between Q values and utility values

$$U(s) = \max_a Q(a,s) \qquad (1)$$

$$Q(s,a) \;=\; R(s) + \gamma \sum_{s'} P(s'|s,a) U(s') \qquad (2)$$

$$Q(s,a) \;=\; R(s) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q(s',a')$$

VI algorithm

function VALUE-ITERATION(mdp, ϵ) **returns** a utility function
  inputs: mdp, an MDP with states S, actions $A(s)$, transition model $P(s' \mid s, a)$,
      rewards $R(s)$, discount γ
    ϵ, the maximum error allowed in the utility of any state
  local variables: U, U', vectors of utilities for states in S, initially zero
      δ, the maximum change in the utility of any state in an iteration

  repeat
    $U \leftarrow U'$; $\delta \leftarrow 0$
    for each state s **in** S **do**
      $U'[s] \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\, U[s']$
      if $|U'[s] - U[s]| > \delta$ **then** $\delta \leftarrow |U'[s] - U[s]|$
  until $\delta < \epsilon(1-\gamma)/\gamma$
  return U

Asynchronous Value Iteration

- The "basic" version of value iteration applies the Bellman update to all states at every iteration
- This is in fact not necessary
  - On each iteration we can apply the update only to a chosen subset of states
  - Given certain conditions on the value function used to initialize the process, asynchronous value iteration converges to an optimal policy
- Main advantage
  - one can design heuristics that allow the algorithm to concentrate on states that are likely to belong to the optimal policy
  - Much faster convergence

Asynchronous VI algorithm

function VALUE-ITERATION(mdp, ϵ) **returns** a utility function
  inputs: mdp, an MDP with states S, transition model T, reward function R, discount γ
    ϵ, the maximum error allowed in the utility of any state
  local variables: U, U', vectors of utilities for states in S, initially zero
    δ, the maximum change in the utility of any state in an iteration

  repeat
    $U \leftarrow U'; \delta \leftarrow 0$
    for some state s **in** S **do**
      $U'[s] \leftarrow R[s] + \gamma \max_a \sum_{s'} T(s, a, s')\, U[s']$
      if $|U'[s] - U[s]| > \delta$ **then** $\delta \leftarrow |U'[s] - U[s]|$
  until $\delta < \epsilon(1-\gamma)/\gamma$
  return U

Model-based RL algorithm

Model Based Reinfortcement Learner
inputs:
S is a set of states, A is a set of actions, γ the discount, c is a prior count
internal state:
real array *Q[S,A], R[S,A, S']*
integer array *T[S,A, S']*
previous state *s*
previous action *a*

initialize $Q[S, A]$ arbitrarily
initialize $R[S, A, S]$ arbitrarily
initialize $T[S, A, S]$ to zero
observe current state s
select and carry out action a
repeat forever:
    observe reward r and state s'
    select and carry out action a
    $T[s, a, s'] \leftarrow T[s, a, s'] + 1$
    $R[s, a, s'] \leftarrow R[s, a, s'] + \frac{r - R[s, a, s']}{T[s, a, s']}$
    $s \leftarrow s'$
    repeat
        select state s_1, action a_1
        let $P = \sum_{s_2} (T[s_1, a_1, s_2] + c)$
        $Q[s_1, a_1] \leftarrow \sum_{s_2} \frac{T[s_1, a_1, s_2] + c}{P} \left(R[s_1, a_1, s_2] + \gamma \max_{a_2} Q[s_2, a_2] \right)$
    until an observation arrives

Counts of events when action a performed in s generated s’

TD-based estimate of R(s,a,s’)

Asynchronous value iteration steps

What is this c for?

Frequency of transition from s_1 to s_2 via a_1

Why is the reward inside the summation?

Discussion

- Which Q values should asyncronous VI update?
  - At least *s* in which the action was generated
  - Then either select states randomly, or
  - States that are likely to get their Q-values changed because they can reach states with Q-values that have changed the most
- How many steps of asynchronous value-iteration to perform?

Discussion

- Which states to update?
  - At least *s* in which the action was generated
  - Then either select states randomly, or
  - States that are likely to get their Q-values changed because they can reach states with Q-values that have changed the most
- How many steps of asynchronous value-iteration to perform?
  - As many as can be done before having to act again

Q-learning vs. Model-based

- Is it better to learn a model and a utility function or an action value function with no model?
  - Still an open-question
- Model-based approaches require less data to learn well, but they can be computationally more expensive (time per iteration)
- Q-learning takes longer because it does not enforce consistency among Q-values via the model
  - Especially true when the environment becomes more complex
  - In games such as chess and backgammon, model-based approaches have been more successful that q-learning methods
- Cost/ease of acting needs to be factored in

Reinforcement Learning

Overview

- Introduction
- Q-learning
- Exploration Exploitation
- Evaluating RL algorithms
- On-Policy learning: SARSA
- Model-based Q-learning

Overview

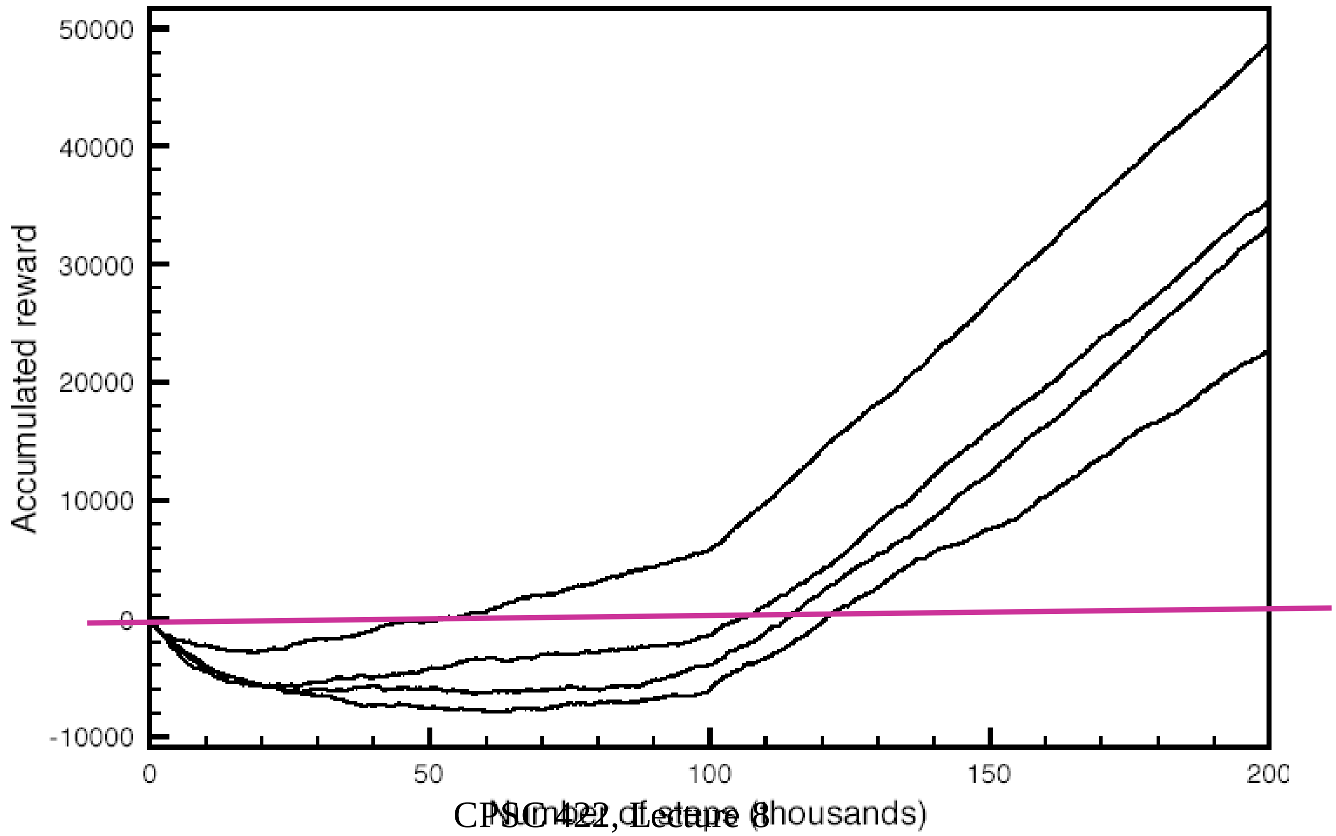
- Introduction
- Q-learning
- Exploration vs. Exploitation
- Evaluating RL algorithms
- On-Policy Learning: SARSA
- Model-based Q-learning

Evaluating RL Algorithms

- Two possible measures
  - Quality of the optimal policy
  - Reward received *while* looking for the policy
- If there is a lot of time for learning before the agent is deployed, then quality of the learned policy is the measure to consider
- If the agent has to learn while being deployed, it may not get to the optimal policy for a along time
  - Reward received while learning is the measure to look at, e.g, plot cumulative reward as a function of number of steps
  - One algorithm dominates another if its plot is consistently above

Evaluating RL Algorithms

- Plots for example 11.8 in textbook (p. 464), with
  - Either fixed or variable α
  - Different initial values for Q[s,a]



Evaluating RL Algorithms

- Lots of variability in each algorithm for different runs
  - for fair comparison, run each algorithm several times and report average behavior
- Relevant statistics of the plot
  - ***Asymptotic slopes***: how good the policy is after the algorithm stabilizes
  - ***Plot minimum***: how much reward must be sacrificed before starting to gain (cost of learning)
  - ***zero-crossing***: how long it takes for the algorithm to recuperate its cost of learning