Intelligent Systems (AI-2)

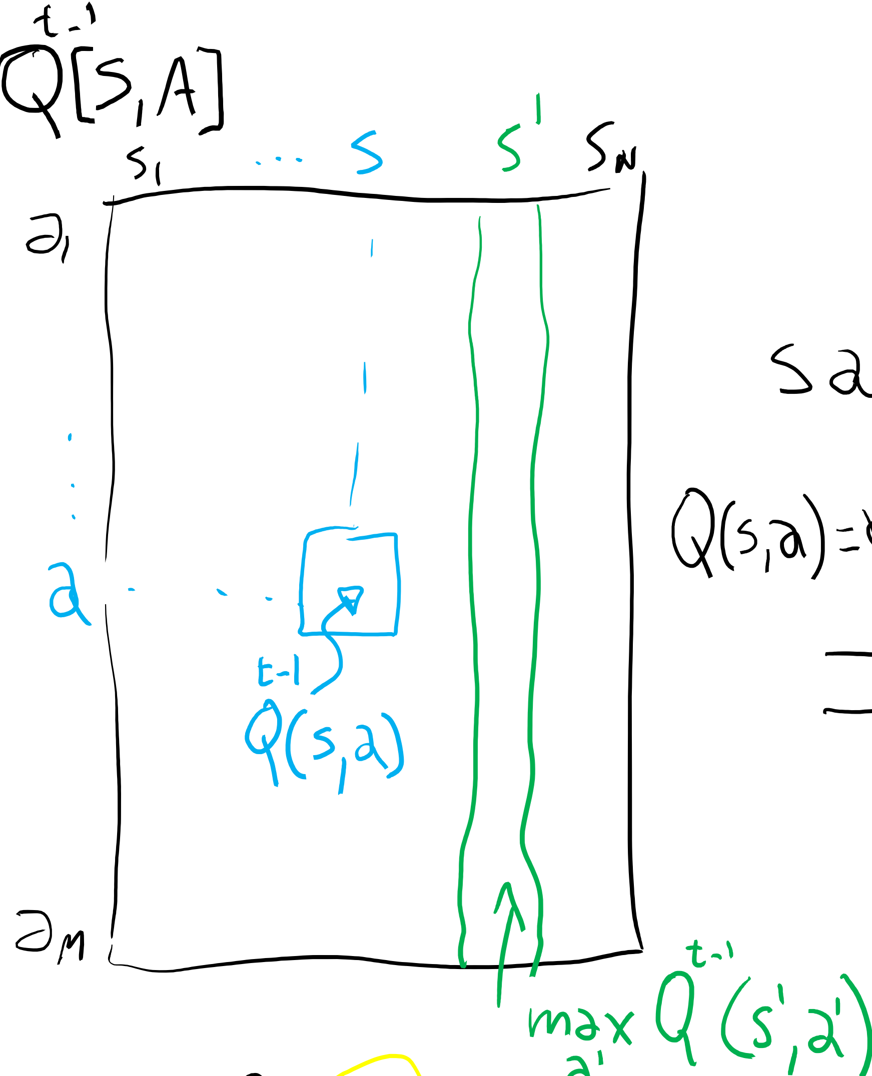
Computer Science cpsc422, Lecture 10

Sep, 30, 2016

Lecture Overview

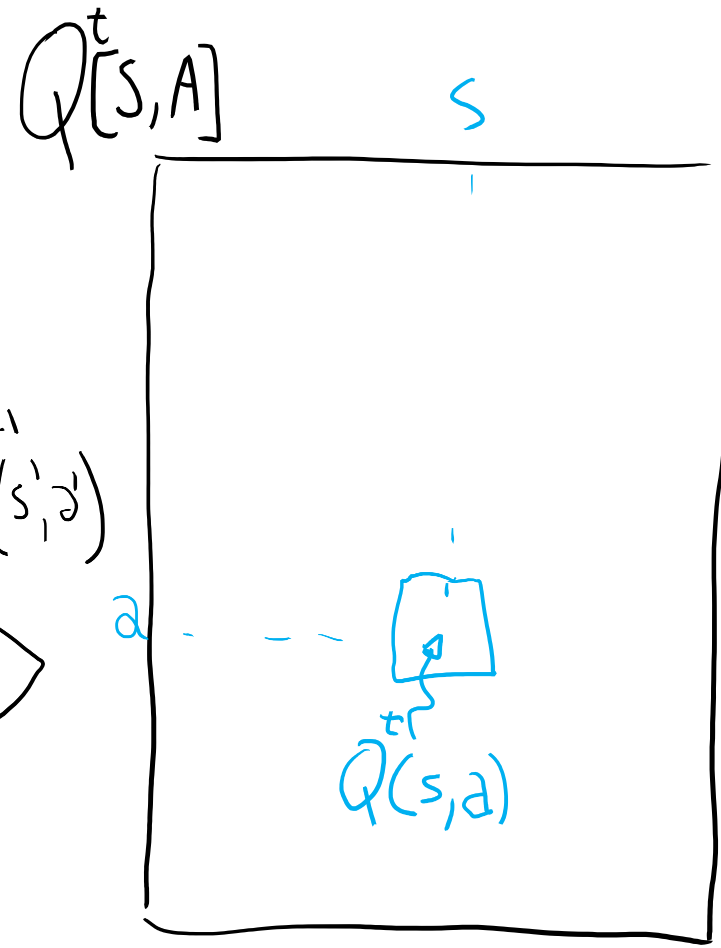
Finish Reinforcement learning

- **Exploration vs. Exploitation**
- On-policy Learning (SARSA)
- Scalability



sars'

$$Q(s,a) = r + \gamma \max_{a'} \overset{t-1}{Q}(s',a')$$

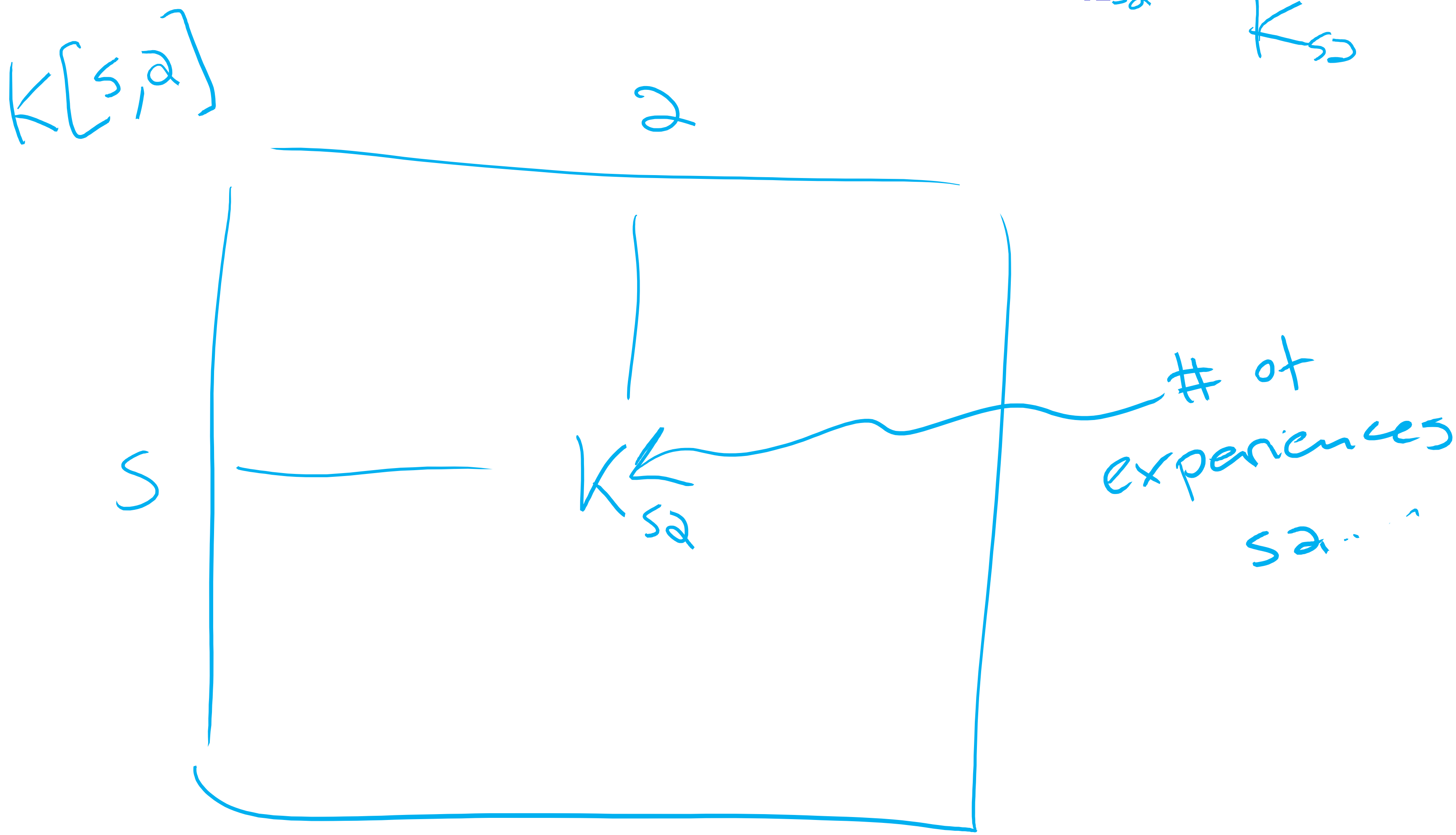


TD

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

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

Clarification on the $\alpha_{\kappa_{sa}} = \frac{1}{K_{sa}}$



What Does Q-Learning learn

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

- Given the Q-function the agent can……

…. either exploit it or explore more….

Any effective strategy should

- **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

Soft-Max

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

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

- τ (tau) in the formula above influences how randomly values should be chosen
  - if τ is high, >> Q[s,a]?

iclicker

A. It will mainly exploit

B. It will mainly explore

C. It will do both with equal probability

Soft-Max

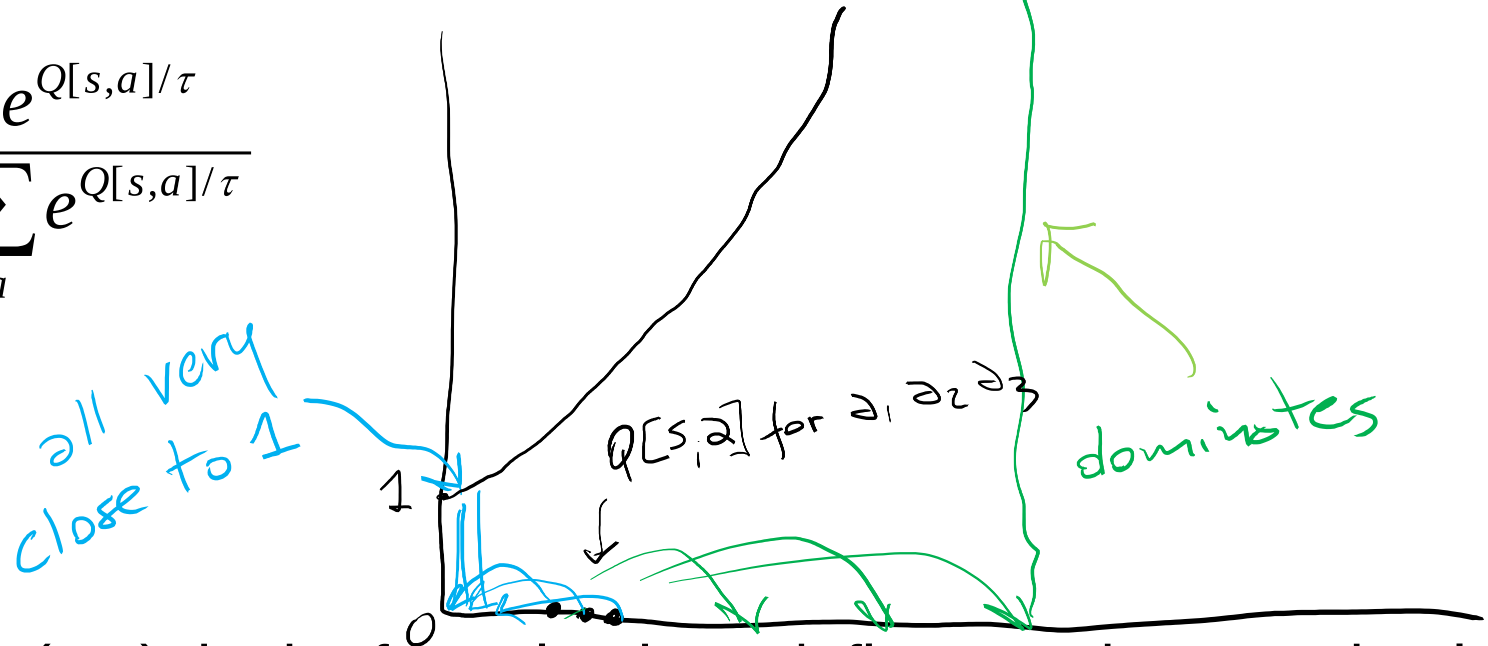
- Takes into account improvement in estimates of expected reward function Q[s,a]
  - Choose action *a* in state *s* with a probability proportional to current estimate of Q[s,a]

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

- τ (tau) in the formula above influences how randomly values 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 $\tau \rightarrow 0$, the exponential with the highest Q[s,a] dominates, and the current best action is always chosen (exploration or exploitation?)

Soft-Max

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



- τ (tau) in the formula above influences how randomly values 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 $\tau \rightarrow 0$, the exponential with the highest Q[s,a] dominates, and the current best action is always chosen (exploration or exploitation?)

Lecture Overview

<u>Finish Reinforcement learning</u>

- Exploration vs. Exploitation
- **On-policy Learning (SARSA)**
- **RL scalability**

Learning before vs. during deployment

- Our learning agent can:
  A. act in the environment to learn how it works (before deployment)
  B. Learn as you go (after 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

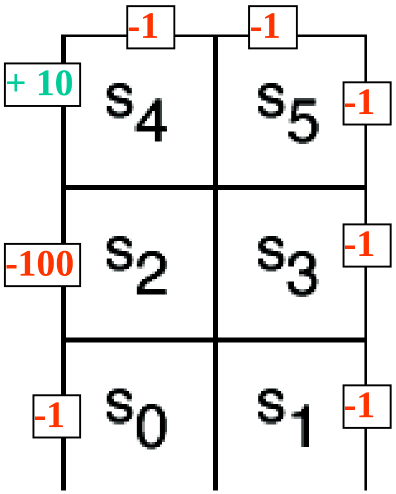
Comment Sept 2015

Best way to present this

- If agent is not deployed it should do random all the time (ε=1) and Q-learning
  - When Q values have converged then deploy
- If the agent is deployed it should apply one of the explore/exploit strategies (e.g., ε=.5) and do sarsa

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

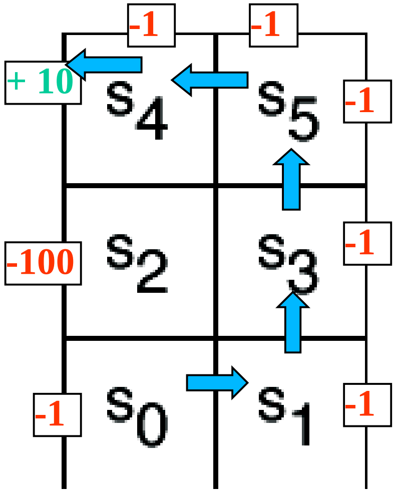
$$Q[s,a] \leftarrow Q[s,a] + \alpha((r + \gamma Q[s',a']) - Q[s,a])$$

- 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 current policy

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



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

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

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

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

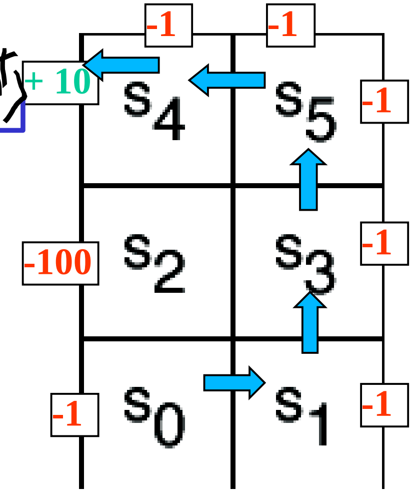
$$Q[s_3, upCarfull] \leftarrow Q[s_3, upCarfull] + \alpha_k (r + 0.9 Q[s_5, Left] - Q[s_3, upCarfull]);$$
$$Q[s_3, upCarfull] \leftarrow 0 + 1(-1 + 0.9 * 0 - 0) = -1$$

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

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

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

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



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

k=2

Q[s,a]	***s₀***	***s₁***	***s₂***	***s₃***	***s₄***	***s₅***
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

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

$$Q[s_3, upCarfull] \leftarrow Q[s_3, upCarfull] + \alpha_k(r + 0.9Q[s_5, Left] - Q[s_3, upCarfull]);$$
$$Q[s_3, upCarfull] \leftarrow -1 + 1/2(-1 + 0.9*0 + 1) = -1$$

$$Q[s_5, Left] \leftarrow Q[s_5, Left] + \alpha_k(r + 0.9Q[s_4, left] - Q[s_5, Left]);$$
$$Q[s_5, Left] \leftarrow 0 + 1/2(0 + 0.9*10 - 0) = 4.5$$

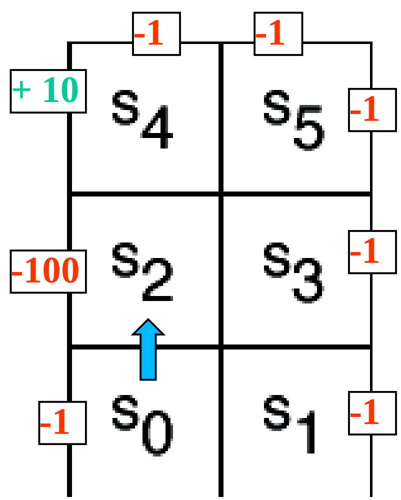
$$Q[s_4, Left] \leftarrow Q[s_4, Left] + \alpha_k(r + 0.9Q[s_0, Right] - Q[s_4, Left]);$$
$$Q[s_4, Left] \leftarrow 10 + 1/2(10 + 0.9*0 - 10) = 10$$

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

Iterations	Q[s_0,Up]	Q[s_1,Up]	Q[s_2,UpC]	Q[s_3,Up]	Q[s_4,Left]	Q[s_5,Left]
40000000	19.1	17.5	22.7	20.4	26.8	23.7

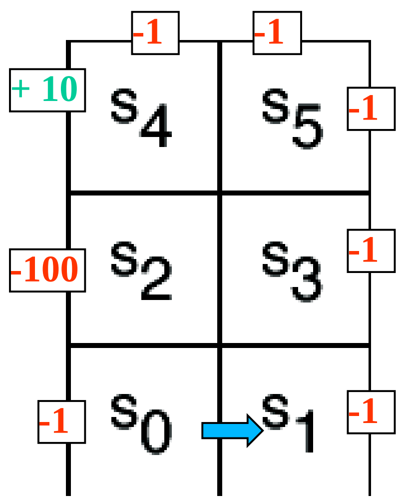


- Verify running full demo, see http://www.cs.ubc.ca/~poole/aibook/demos/rl/tGame.html

Comparing SARSA and Q-learning

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

Iterations	Q[s_0,Right]	Q[s_1,Up]	Q[s_2,UpC]	Q[s_3,Up]	Q[s_4,Left]	Q[s_5,Left]
40000000	6.8	8.1	12.3	10.4	15.6	13.2



CPSC 422, Lecture 10

- Verify running full demo, see http://www.cs.ubc.ca/~poole/aibook/demos/rl/tGame.html

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

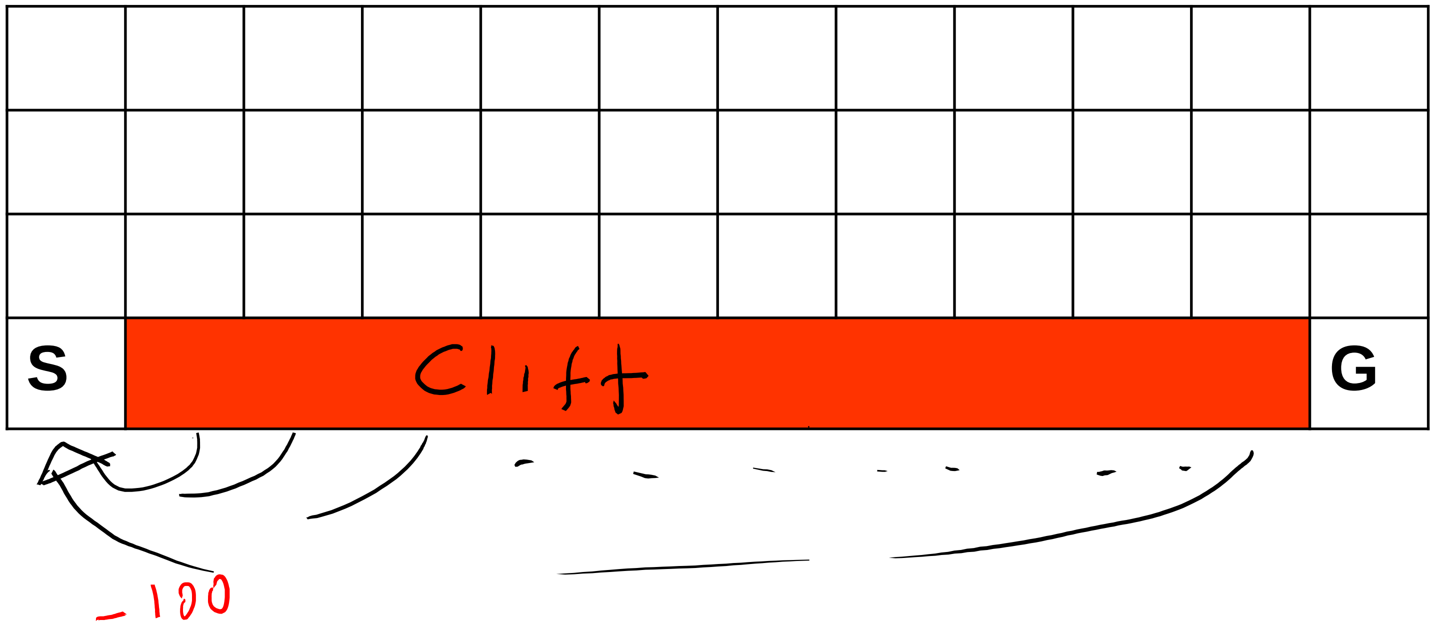
This could be, for instance any ε-greedy strategy:
-Choose random ε times, and max the rest

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** (terminal state with reward > 0)
  - **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**



➢ With an **ε-greedy strategy (e.g., ε =0.1)**

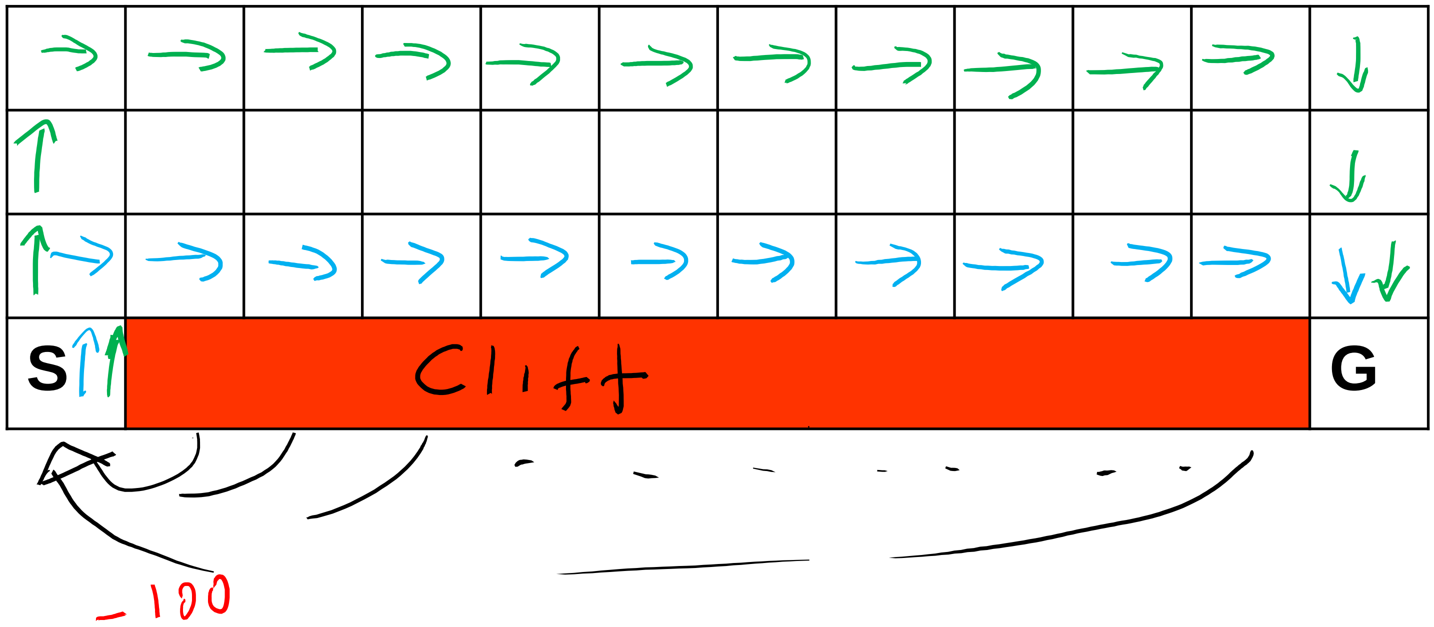
iclicker

A. SARSA will learn policy p1 while Q-learning will learn p2

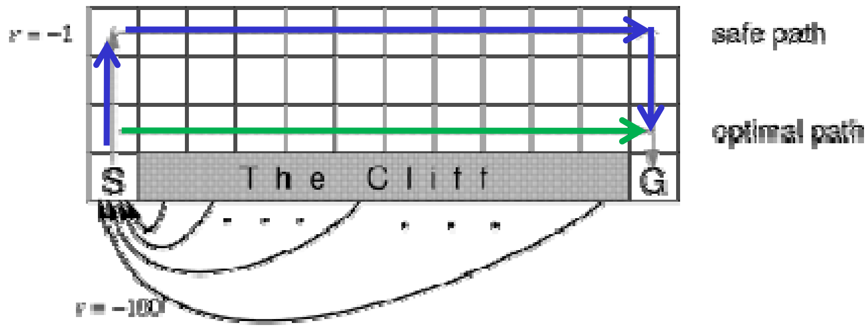
B. Q-learning will learn policy p1 while SARSA will learn p2

C. They will both learn p1

D. They will both learn p2



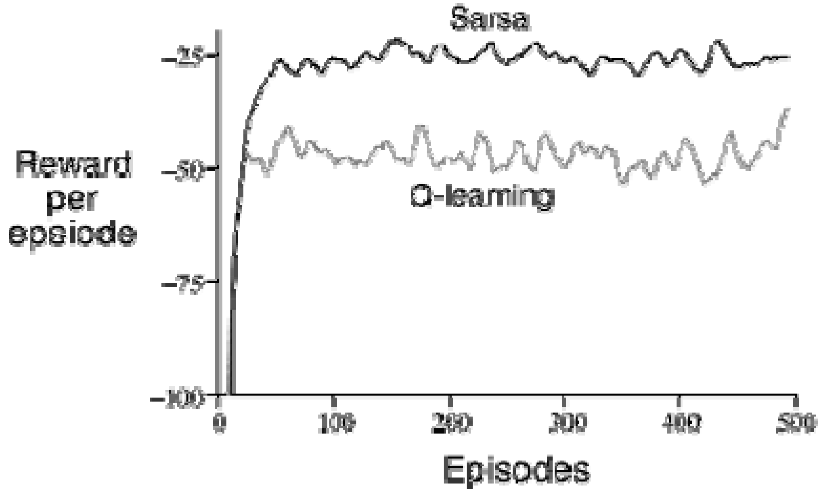
Cliff Example



- 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 $\varepsilon \rightarrow 0$, SARSA and Q-learning would asymptotically converge to the optimal policy

422 big picture: Where are we?

Hybrid: Det +Sto

Prob CFG
Prob Relational Models
Markov Logics

	Deterministic	Stochastic
Query	*Logics* *First Order Logics* *Ontologies* *Temporal rep.* • Full Resolution • SAT	*Belief Nets* Approx. : Gibbs *Markov Chains and HMMs* Forward, Viterbi…. Approx. : Particle Filtering *Undirected Graphical Models* *Conditional Random Fields*
Planning		*Markov Decision Processes and* *Partially Observable MDP* • Value Iteration • Approx. Inference *Reinforcement Learning*

Applications of AI

Representation

Reasoning Technique

Learning Goals for today's class

➢You can:

- Describe and compare techniques to combine exploration with exploitation
- On-policy Learning (SARSA)
- Discuss trade-offs in RL scalability (not required)

TODO for Mon

- Read textbook 6.4.2
- Next research paper will be next Fri
- Practice Ex 11.B

- Assignment 1 due on Mon

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
  $T[s, a, s'] \leftarrow T[s, a, s'] + 1$
  $R[s, a, s'] \leftarrow R[s, a, s'] + \dfrac{r - R[s, a, s']}{T[s, a, s']}$
  $s \leftarrow s'$
  repeat
    select state s_1, action a_1
    let $P = \sum_{s_2} (T[s_1, a_1, s_2] + c)$
    $Q[s_1, a_1] \leftarrow \sum_{s_2} \dfrac{T[s_1, a_1, s_2] + c}{P} \left(R[s_1, a_1, s_2] + \gamma \max_{a_2} Q[s_2, a_2] \right)$
  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?

Why is the reward inside the summation?

Frequency of transition from s_1 to s_2 via a_1

Discussion

- Which Q values should asynchronous 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