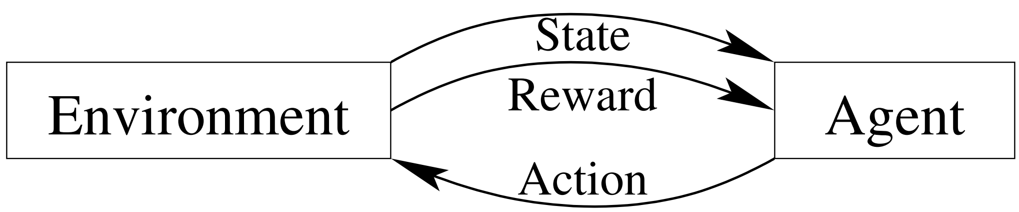
Reinforcement Learning

Many learning situations are characterized by finding a sequence of actions that maximize utility (reward), e.g.,

Actions	Reward
listen to lectures	exam scores
write programs	efficiency
make dinner	taste
going places	timeliness
write papers	tenure



Framework

Repeatedly, an *agent* senses the *state* of its environment, performs an *action*, and receives a *reward*. A *policy* maps from states to actions. Some considerations are:

- Delayed Reward. Maybe no immediate gratification.
- Exploration vs. Exploitation. Evaluate different policies or take advantage of current knowledge?
- Partially Observable States. Might not know complete state. Some actions might clarify current state.
- Changes to Environment. Change might invalidate previous learning, so learning never ends.

Markov Decision Processes (MDPs)

A process is Markov if the next state depends only on the current state and action.

S is a finite set of states, one initial and some terminal
A is a finite set of actions
At each time point t,
  The agent senses $s_t \in S$
  Performs some action $a_t \in A$
  Receives reward $r_t = r(s_t, a_t)$
  New state is $s_{t+1} = \delta(s_t, a_t)$
Task is learn policy $\pi \colon S \to A$

r (reward) and δ (state transition) fns. might be probabilistic. *episode* = path from initial to a terminal state.

Discounted Reward

Is a large reward in the future better than a smaller reward now? Consider lotto tickets, interest rates, inflation, lifetimes, errors in model.

Discounted cumulative reward is:

$$V^\pi(s_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} \ldots = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$$

$0 \leq \gamma < 1$ is the discount factor. γ closer to 0 (or 1) gives more emphasis to immediate (or future) reward.

Options: total reward, finite horizon, average reward.

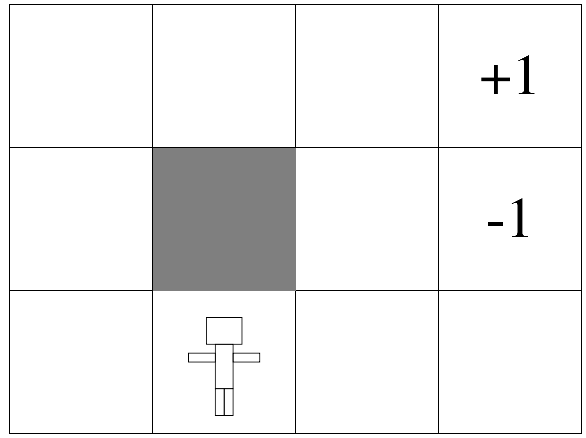
Goal: Learn the policy $\pi*$ that maximizes expected value of V^π.

Reinforcement Learning Example

Suppose a robot in this environment.

One terminal square has +1 reward (recharge station).

One terminal square has -1 reward (falling down stairs).



An action to stay put always succeeds.
An action to move to a neighbor square,
  succeeds with probability 0.8,
  stays in the same square with prob. 0.1,
  goes to another neighbor with prob. 0.1
Should the robot try moving left or right?

Policy Iteration

If the state transition function $\delta(s, a)$ and the reward function $r(s, a)$ are known, then *policy iteration* can find the optimal policy $\pi*$.

Policy-Iteration(π_0)
  $i \leftarrow 0$
  repeat
    for each state s_j, compute $V^{\pi_i}(s_j)$
    Compute π_{i+1} so $V^{\pi_{i+1}}(s_j)$ is the action a
      maximizing $r(s_j, a) + \gamma V^{\pi_i}(\delta(s_j, a))$
    $i \leftarrow i + 1$
  until $\pi_i = \pi_{i-1}$

Policy Evaluation by Dynamic Programming

Can compute $V^\pi(s_j)$ by dynamic programming:

for each state s_j, $V_0^\pi(s_j) \leftarrow 0$
$i \leftarrow 0$
repeat
  for each state s_j
    $V_{i+1}^\pi(s_j) \leftarrow r(s_j, \pi(s_j)) + \gamma V_i^\pi(\delta(s_j, \pi(s_j)))$
  $i \leftarrow i+1$
until convergence

If δ is probabilistic, then $V_i^\pi(\delta(s_j, \pi(s_j)))$ is:

for each state s_k,
  $sum \leftarrow sum + \Pr(\delta(s_j, s_k))\ V_i^\pi(\delta(s_j, s_k))$

The Q Function

Define $Q(s,a)$ to be the expected maximum discounted cumulative reward starting from state s and applying action a.

$$Q(s,a) = E[r(s,a) + \gamma V^{\pi*}(\delta(s,a))]$$

The optimal policy $\pi*$ corresonds to the action a maximizing $Q(s,a)$.

$$\pi*(s) = \operatorname*{argmax}_a \ Q(s,a)$$

$Q(s,a)$ can be redefined recursively:

$$Q(s,a) = E[r(s,a) + \gamma \max_{a'} \ Q(\delta(s,a), a')]$$

The Q Learning Algorithm

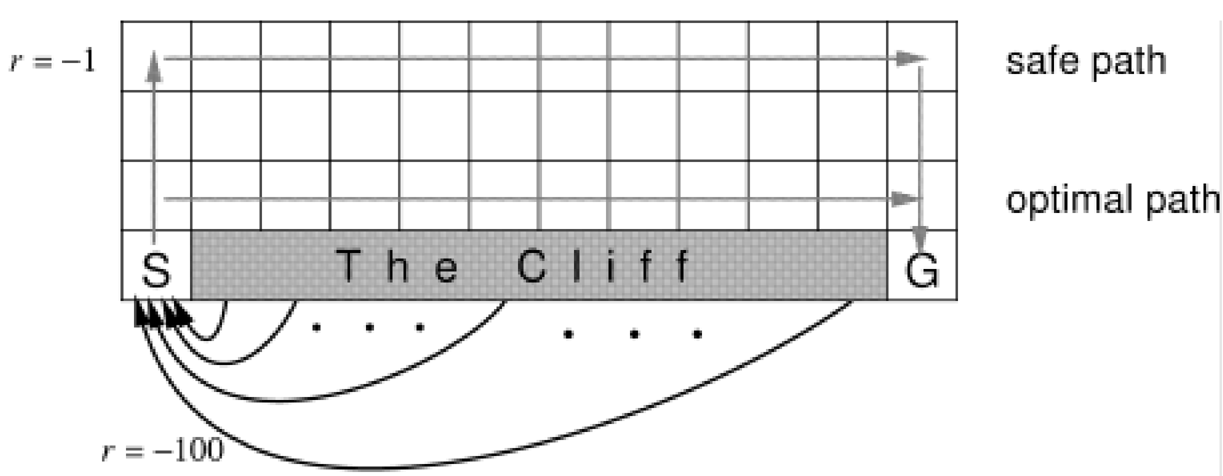
The Q learning algorithm maintains an estimate $\hat{Q}$ of the Q values. The following algorithm generalizes the one in the book nondeterministic MDPs. η is a learning rate.

Q-learning
  For all states s and actions a, $\hat{Q}(s, a) \leftarrow 0$.
  Repeat (for each episode)
    $s \leftarrow$ initial state
    While s is not terminal
      Select and execute an action a from s
      Receive reward r and observe new state s'
      $\hat{Q}(s, a) \leftarrow (1 - \eta)\hat{Q}(s, a) \; + \eta(r + \gamma \max_{a'} \hat{Q}(s', a'))$
      $s' \leftarrow s$

Exploration

How should action a be selected? The ϵ-*greedy* strategy selects the best action (based on $\hat{Q}$) with prob. $1 - \epsilon$. Otherwise, it randomly selects an action.

The Q algorithm has problems with ϵ-greedy. In the following environment, it walks off the cliff too much.



The Sarsa Algorithm

The Sarsa algorithm is better when using ϵ-greedy.

Sarsa-learning

For all states s and actions a, $\hat{Q}(s,a) \leftarrow 0$
Repeat (for each episode)
  $s \leftarrow$ initial state
  Select action a from s (ϵ-greedy)
  While s is not terminal
    Execute action a
    Receive reward r and observe new state s'
    Select action a' from s' (ϵ-greedy)
    $\hat{Q}(s,a) \leftarrow (1-\eta)\hat{Q}(s,a) + \eta(r + \gamma\hat{Q}(s',a'))$
    $a \leftarrow a'$ and $s \leftarrow s'$

Function Approximation

In many environments, the states are not a small finite set. Consider representing games like chess or checkers.

Often, the state and action can be represented by a vector $\mathbf{x}$, and the Q function by a linear function $\mathbf{w} \cdot \mathbf{x}$. For example, the Sarsa $\hat{Q}$ update can be replaced with:

$$\mathbf{w} \leftarrow \mathbf{w} + \eta(r + \gamma(\mathbf{w} \cdot \mathbf{x}') - (\mathbf{w} \cdot \mathbf{x}))\mathbf{x}$$

where $\mathbf{x}$ represent state s and action a, and $\mathbf{x}'$ represents state s' and action a'.

If a more complicated function is desired (e.g., ANNs), the generalized delta rule can be applied.

The Temporal Difference Algorithm

Given a sequence of states $s_1, s_2, \ldots$ and rewards $r_1, r_2, \ldots$, the goal is to learn $V(s)$, the expected cumulative discounted reward.

$TD(0)$

  Initialize $V(s)$

  Repeat (for each episode)

    $s \leftarrow$ initial state of episode

    While s is not last state of the episode

      Receive reward r and observe next state s'

      $V(s) \leftarrow (1-\eta)V(s) + \eta(r + \gamma V(s'))$

      $s \leftarrow s'$

Comments on $TD(0)$

Temporal difference learning is a generalization of Q learning.

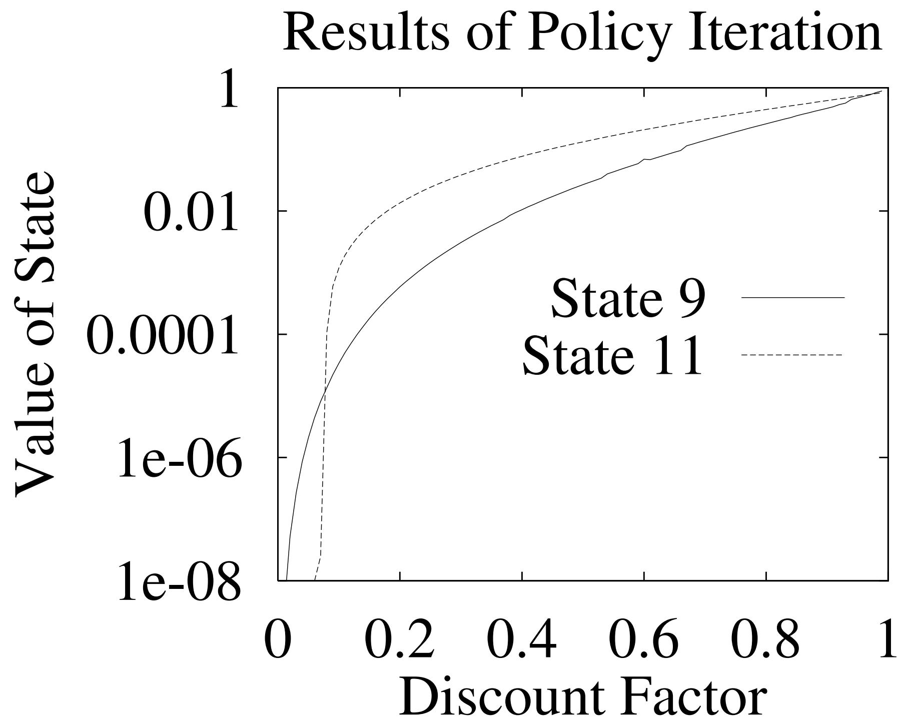
The reward is assumed to be a (possibly probabilistic) function of the current state.

$r + \gamma V(s')$ is treated as an estimate of $V(s)$.

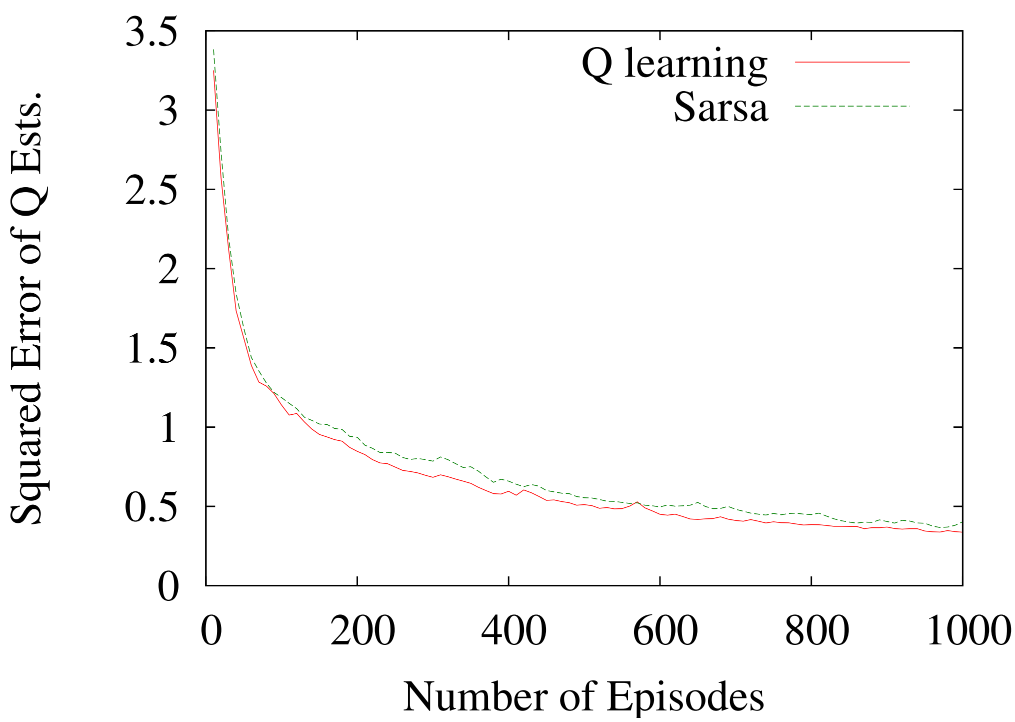
Over a finite number of states, $TD(0)$ converges to an locally optimal estimate of $V(s)$.

For more information, see R. S. Sutton and A. G. Barto (1998). *Reinforcement Learning: An Introduction*, MIT Press. Part of the notes come from this online book.

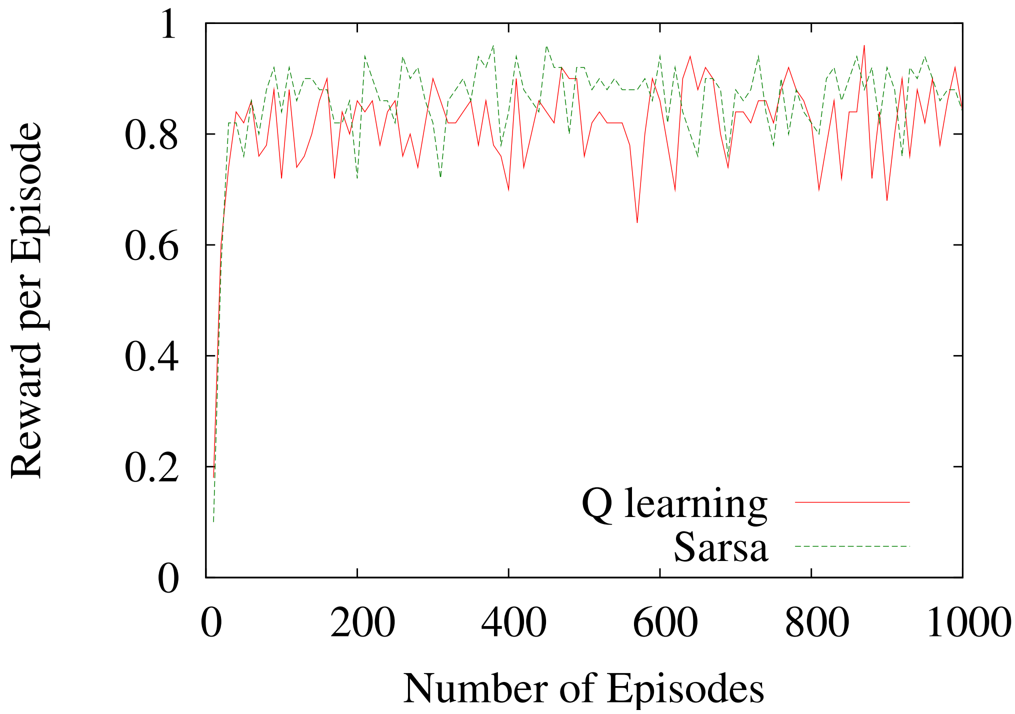
Which State is Better in 11-State Robot Environment



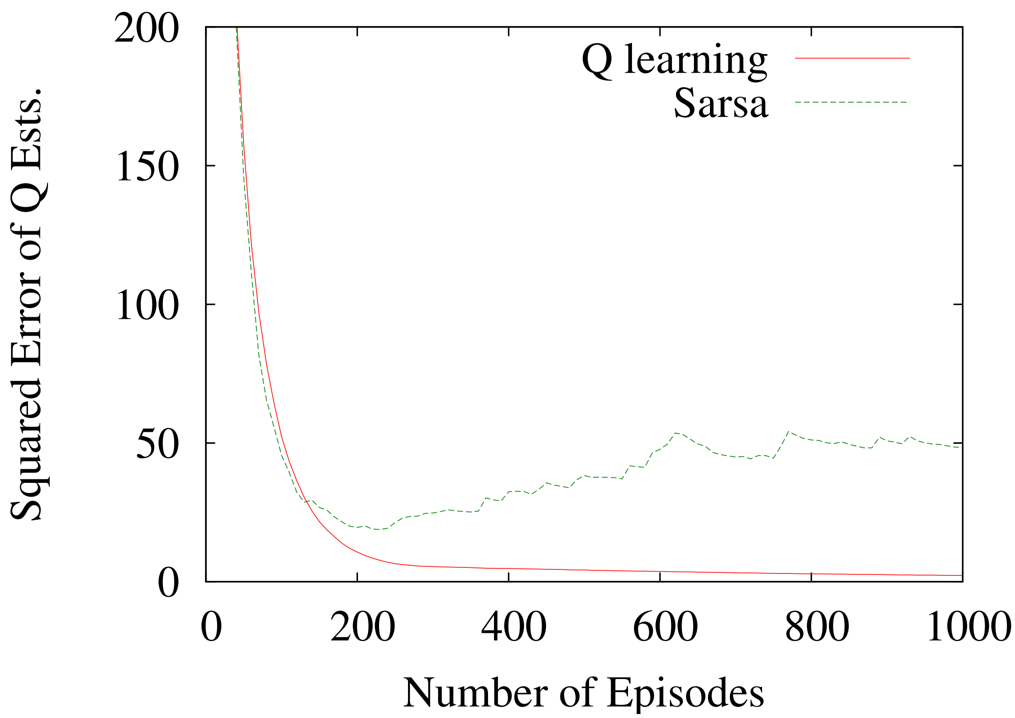
Convergence to Q Values in 11-State Robot Environment
($\gamma = 0.9$, $\eta = 0.1$, $\epsilon = 0.1$, 10 runs)



Rewards in 11-State Robot Environment ($\gamma = 0.9$, $\eta = 0.1$, $\epsilon = 0.1$, 10 runs)



Convergence to Q Values in Cliff Environment ($\gamma = 0.9$, $\eta = 0.1$, $\epsilon = 0.1$, 10 runs)



Rewards in Cliff Environment ($\gamma = 0.9$, $\eta = 0.1$, $\epsilon = 0.1$, 10 runs)