**CS 287: Advanced Robotics**
**Fall 2009**

Lecture 13: Reinforcement Learning

Pieter Abbeel
UC Berkeley EECS

# Outline

- Model-free approaches
  - Recap TD(0)
  - Sarsa
  - Q learning
  - TD($\lambda$), sarsa($\lambda$), Q($\lambda$)
  - Function approximation and TD
  - TD Gammon

# TD(0) for estimating $V^{\pi}$

Stochastic version of the policy evaluation update:

$$V^{\pi}(s) = \sum_{s'} T(s, \pi(s), s') \left[ R(s, \pi(s), s') + \gamma V^{\pi}(s') \right]$$

Initialize $V(s)$ arbitrarily, $\pi$ to the policy to be evaluated
Repeat (for each episode):
  Initialize $s$
  Repeat (for each step of episode):
    $a \leftarrow$ action given by $\pi$ for $s$
    Take action $a$; observe reward, $r$, and next state, $s'$
    $V(s) \leftarrow V(s) + \alpha \left[ r + \gamma V(s') - V(s) \right]$
    $s \leftarrow s'$
  until $s$ is terminal

Note: this is really $V^{\pi}$

# Problems with TD Value Learning

- TD value leaning is model-free for policy evaluation
- However, if we want to turn our value estimates into a policy---as required for a policy update step---we're sunk:

$$\pi_{k+1}(s) = \arg\max_{a} Q^{\pi_k}(s, a)$$

$$Q^{\pi_k}(s, a) = \sum_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma V^{\pi_k}(s') \right]$$

- Idea: learn Q-values directly
- Makes action selection model-free too!

## Update Q values directly

- When experiencing $s_t$, $a_t$, $s_{t+1}$, $r_{t+1}$, $a_{t+1}$ perform the following "sarsa" update:

$$\begin{aligned} Q^\pi(s_t,a_t) &\leftarrow (1-\alpha)Q^\pi(s_t,a_t) + \alpha\left[r(s_t,a_t,s_{t+1}) + \gamma Q^\pi(s_{t+1},a_{t+1})\right] \\ &= Q^\pi(s_t,a_t) + \alpha\left[r(s_t,a_t,s_{t+1}) + \gamma Q^\pi(s_{t+1},a_{t+1}) - Q^\pi(s_t,a_t)\right] \end{aligned}$$

- Will find the Q values for the current policy $\pi$.
- How about Q(s,a) for action a inconsistent with the policy $\pi$ at state s?
- Converges (w.p. 1) to Q function for current policy $\pi$ for all states and actions *if* all states and actions are visited infinitely often (assuming proper step-sizing)

## Exploration aspect

- To ensure convergence for all Q(s,a) we need to visit every (s,a) infinitely often

  → The policy $\pi$ needs to include some randomness

  - Simplest: random actions ($\varepsilon$ greedy)
    - Every time step, flip a coin
    - With probability $\varepsilon$, act randomly
    - With probability 1-$\varepsilon$, act according to some current policy
  - → This results in a new policy $\pi$'
  - *We end up finding the Q values for this new policy* $\pi$'

## Does policy iteration still work when we execute epsilon greedy policies?

- Policy iteration iterates:
  - Evaluate value of current policy $V^{\pi}$
  - Improve policy by choosing the greedy policy w.r.t. $V^{\pi}$

- Answer: Using the epsilon greedy policies can be interpreted as running policy iteration w.r.t. a related MDP which differs slighty in its transition model: with probability $\epsilon$ the transition is according to a random action in the new MDP

## Need not wait till convergence with the policy improvement step

- Recall: Generalized policy iteration methods: interleave policy improvement and policy evaluation and guaranteed to converge to the optimal policy as long as value for every state updated infinitely often

- → Sarsa: continuously update the policy by choosing actions $\epsilon$ greedy w.r.t. the current Q function

# Sarsa: updates Q values directly

```
Initialize Q(s,a) arbitrarily
Repeat (for each episode):
   Initialize s
   Choose a from s using policy derived from Q (e.g., ε-greedy)
   Repeat (for each step of episode):
      Take action a, observe r, s'
      Choose a' from s' using policy derived from Q (e.g., ε-greedy)
      Q(s,a) ← Q(s,a) + α[r + γQ(s',a') − Q(s,a)]
      s ← s'; a ← a';
   until s is terminal
```

Sarsa converges w.p. 1 to an optimal policy and action-value function as long as all state-action pairs are visited an infinite number of times and the policy converges in the limit to the greedy policy (which can be arranged, e.g., by having $\epsilon$ greedy policies with $\epsilon$ = 1 / t).

# Q learning

- Directly approximate the optimal Q function Q*:

$$Q(s_t, a_t) \quad \leftarrow \quad (1-\alpha)Q(s_t, a_t) + \alpha\left[r(s_t, a_t, s_{t+1}) + \max_a \gamma Q^\pi(s_{t+1}, a)\right]$$

- Compare to sarsa:

$$Q^\pi(s_t, a_t) \quad \leftarrow \quad (1-\alpha)Q^\pi(s_t, a_t) + \alpha\left[r(s_t, a_t, s_{t+1}) + \gamma Q^\pi(s_{t+1}, a_{t+1})\right]$$

# Q learning

Initialize $Q(s,a)$ arbitrarily
Repeat (for each episode):
  Initialize $s$
  Repeat (for each step of episode):
    Choose $a$ from $s$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
    Take action $a$, observe $r$, $s'$
    $Q(s,a) \leftarrow Q(s,a) + \alpha\big[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\big]$
    $s \leftarrow s'$;
  until $s$ is terminal

# Q-Learning Properties

- Will converge to optimal Q function if
  - Every (s,a) visited infinitely often
  - $\alpha$ is chosen to decay according to standard stochastic approximation requirements

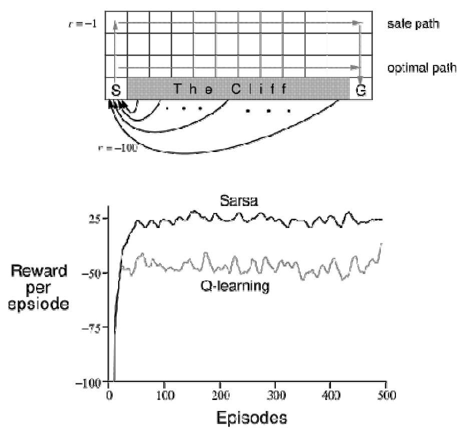
- Neat property: learns optimal Q-values regardless of policy used to collect the experience

→ "Off policy" method

- Strictly better than TD, sarsa? Some caveats.

# Behaviour of Q-learning vs. sarsa

- Reward = 0 at goal; -100 in cliff region; -1 everywhere else
- $\epsilon$ = 0.1




# Exploration / Exploitation

- Several schemes for forcing exploration
  - Simplest: random actions (ε greedy)
    - Every time step, flip a coin
    - With probability ε, act randomly
    - With probability 1-ε, act according to current policy
  - Problems with random actions?
    - You do explore the space, but keep thrashing around once learning is done
    - Takes a long time to explore certain spaces
    - One solution: lower ε over time
    - Another solution: exploration functions

# Exploration Functions

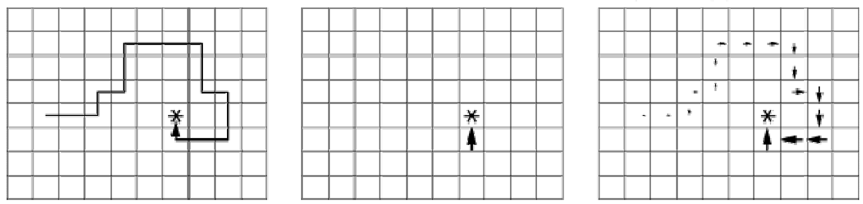
- When to explore
  - Random actions: explore a fixed amount
  - Better idea: explore areas whose badness is not (yet) established
- Exploration function
  - Takes a value estimate and a count, and returns an optimistic utility, e.g. $f(u,n) = u + k/n$ (exact form not important---for optimality guarantees: it should guarantee that every (s,a) is visited infinitely often _or_ that Q(s,a) is always optimistic)

$$Q_{i+1}(s,a) \leftarrow_\alpha R(s,a,s') + \gamma \max_{a'} Q_i(s',a')$$

$$Q_{i+1}(s,a) \leftarrow_\alpha R(s,a,s') + \gamma \max_{a'} f(Q_i(s',a'), N(s',a'))$$

# TD($\lambda$) --- motivation (grid world)

# TD($\lambda$) --- motivation



# TD($\lambda$) "backward view"

- t: $V(s_t) \leftarrow V(s_t) + \alpha[R(s_t) + \gamma V(s_{t+1}) - V(s_t)]$
- t+1: $V(s_{t+1}) \leftarrow V(s_{t+1}) + \alpha[R(s_{t+1}) + \gamma V(s_{t+2}) - V(s_{t+1})]$

+also perform: $V(s_t) \leftarrow V(s_t) + \alpha\gamma\lambda\delta_{t+1}$

- t+2: $V(s_{t+2}) \leftarrow V(s_{t+2}) + \alpha[R(s_{t+2}) + \gamma V(s_{t+3}) - V(s_{t+2})]$

+also: $V(s_{t+1}) \leftarrow V(s_{t+1}) + \alpha\gamma\lambda\delta_{t+2}$

$V(s_t) \leftarrow V(s_t) + \alpha\gamma^2\lambda^2\delta_{t+2}$

# TD($\lambda$) --- backward view wordy

$$V(s_t) \leftarrow V(s_t) + \alpha \underbrace{[R(s_t) + \gamma V(s_{t+1}) - V(s_t)]}_{\delta_t}$$

Similarly, the update at the next time step is

$$V(s_{t+1}) \leftarrow V(s_{t+1}) + \alpha \underbrace{(R(s_{t+1}) + \gamma V(s_{t+2}) - V(s_{t+1})}_{\delta_{t+1}}$$

Note that at the next time step we update $V(s_{t+1})$. This (crudely speaking) results in having a better estimate of the value function for state $s_{t+1}$. TD($\lambda$) takes advantage of the availability of this better estimate to improve the update we performed for $V(s_t)$ in the previous step of the algorithm. Concretely, TD($\lambda$) performs another update on $V(s_t)$ to account for our improved estimate of $V(s_{t+1})$ as follows:

$$V(s_t) \leftarrow V(s_t) + \alpha\gamma\lambda\delta_{t+1}$$

where $\lambda$ is a fudge factor that determines how heavily we weight changes in the value function for $s_{t+1}$.

Similarly, at time $t+2$ we perform the following set of updates:

$$V(s_{t+2}) \leftarrow V(s_{t+2}) + \alpha \underbrace{[R(s_{t+2}) + \gamma V(s_{t+2}) - V(s_{t+3})]}_{\delta_{t+2}} \ldots$$

$$V(s_{t+1}) \leftarrow V(s_{t+1}) + \alpha\gamma\lambda\delta_{t+2}$$

$$V(s_t) \leftarrow V(s_t) + \alpha \underbrace{\gamma^2\lambda^2}_{e(s_t)} \delta_{t+2}$$

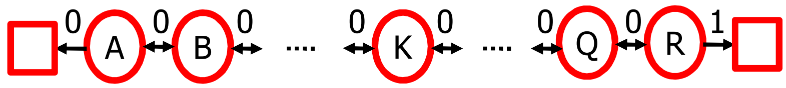
The term $e(s_t)$ is called the *eligibility vector*.

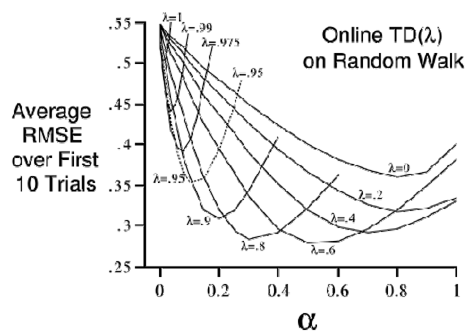
# TD($\lambda$)

```
Initialize V(s) arbitrarily and e(s) = 0, for all s ∈ S
Repeat (for each episode):
    Initialize s
    Repeat (for each step of episode):
        a ← action given by π for s
        Take action a, observe reward, r, and next state, s'
        δ ← r + γV(s') − V(s)
        e(s) ← e(s) + 1
        For all s:
            V(s) ← V(s) + αδe(s)
            e(s) ← γλe(s)
        s ← s'
    until s is terminal
```

## TD($\lambda$) --- example

Random walk over 19 states. Left and rightmost states are sinks. Rewards always zero, except when entering right sink.




## TD($\lambda$) --- "forward view"

- TD: $V(s_t) \leftarrow (1-\alpha)V(s_t) + \alpha \ \ \text{sample}$
- Sample = $R(s_t) + \gamma V(s_{t+1})$

$$R(s_t) + \gamma R(s_{t+1}) + \gamma^2 V(s_{t+2})$$

$$R(s_t) + \gamma R(s_{t+1}) + \gamma^2 R(s_{t+2}) + \gamma^3 V(s_{t+3})$$

$$\ldots$$

$$R(s_t) + \gamma R(s_{t+1}) + \gamma^2 R(s_{t+2}) + \ldots + \gamma^T R(s_T)$$

- $\lambda \in$ [0,1]
- Forward view equivalent to backward view

## Sarsa($\lambda$)

Initialize $Q(s,a)$ arbitrarily and $e(s,a) = 0$, for all $s, a$
Repeat (for each episode):
  Initialize $s$, $a$
  Repeat (for each step of episode):
    Take action $a$, observe $r$, $s'$
    Choose $a'$ from $s'$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
    $\delta \leftarrow r + \gamma Q(s', a') - Q(s,a)$
    $e(s,a) \leftarrow e(s,a) + 1$
    For all $s, a$:
      $Q(s,a) \leftarrow Q(s,a) + \alpha \delta e(s,a)$
      $e(s,a) \leftarrow \gamma \lambda e(s,a)$
    $s \leftarrow s'$; $a \leftarrow a'$
  until $s$ is terminal

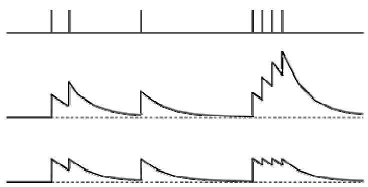
## Watkins Q($\lambda$)

Initialize $Q(s,a)$ arbitrarily and $e(s,a) = 0$, for all $s, a$
Repeat (for each episode):
  Initialize $s$, $a$
  Repeat (for each step of episode):
    Take action $a$, observe $r$, $s'$
    Choose $a'$ from $s'$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
    $a^* \leftarrow \arg\max_b Q(s', b)$ (if $a'$ ties for the max, then $a^* \leftarrow a'$)
    $\delta \leftarrow r + \gamma Q(s', a^*) - Q(s,a)$
    $e(s,a) \leftarrow e(s,a) + 1$
    For all $s, a$:
      $Q(s,a) \leftarrow Q(s,a) + \alpha \delta e(s,a)$
      If $a' = a^*$, then $e(s,a) \leftarrow \gamma \lambda e(s,a)$
        else $e(s,a) \leftarrow 0$
    $s \leftarrow s'$; $a \leftarrow a'$
  until $s$ is terminal

# Replacing traces

- What if a state is visited at two different times $t_1$ and $t_2$ ?
- Recall TD($\lambda$)

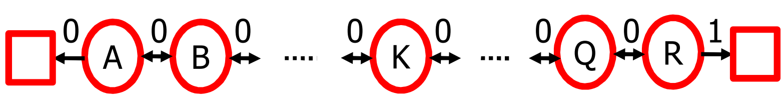
Initialize $V(s)$ arbitrarily and $e(s) = 0$, for all $s \in \mathcal{S}$
Repeat (for each episode):
  Initialize $s$
  Repeat (for each step of episode):
    $a \leftarrow$ action given by $\pi$ for $s$
    Take action $a$, observe reward, $r$, and next state, $s'$
    $\delta \leftarrow r + \gamma V(s') - V(s)$
    $e(s) \leftarrow e(s) + 1$
    For all $s$:
      $V(s) \leftarrow V(s) + \alpha \delta e(s)$
      $e(s) \leftarrow \gamma \lambda e(s)$
    $s \leftarrow s'$
  until $s$ is terminal

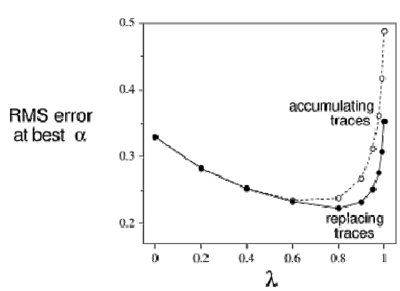



$$e_t(s) = \begin{cases} \gamma \lambda e_{t-1}(s) & \text{if } s \neq s_t; \\ 1 & \text{if } s = s_t. \end{cases}$$

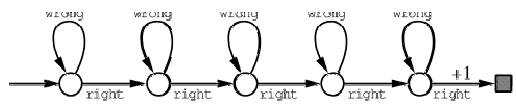
# Replacing traces: example 1

Random walk over 19 states. Left and rightmost states are sinks. Rewards always zero, except when entering right sink.



# Replacing traces: example 2




# Recap RL so far

- When model is available:
  - VI, PI, GPI, LP
- When model is not available:
  - Model-based RL: collect data, estimate model, run one of the above methods for estimated model
  - Model-free RL: learn V, Q directly from experience:
    - TD($\lambda$), sarsa($\lambda$), Q($\lambda$)

- What about large MDPs for which we cannot represent all states in memory or cannot collect experience from all states?

  → Function Approximation