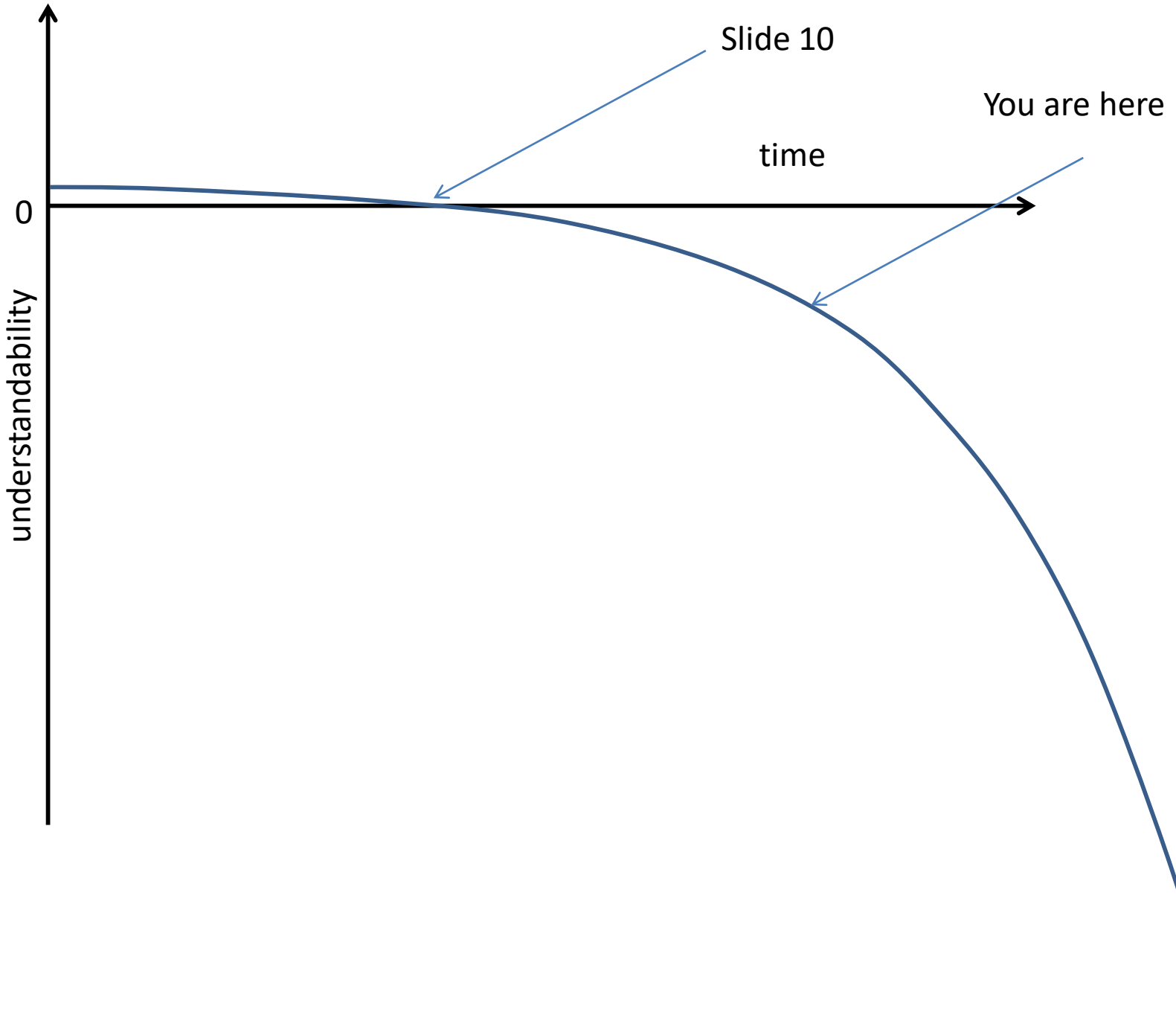
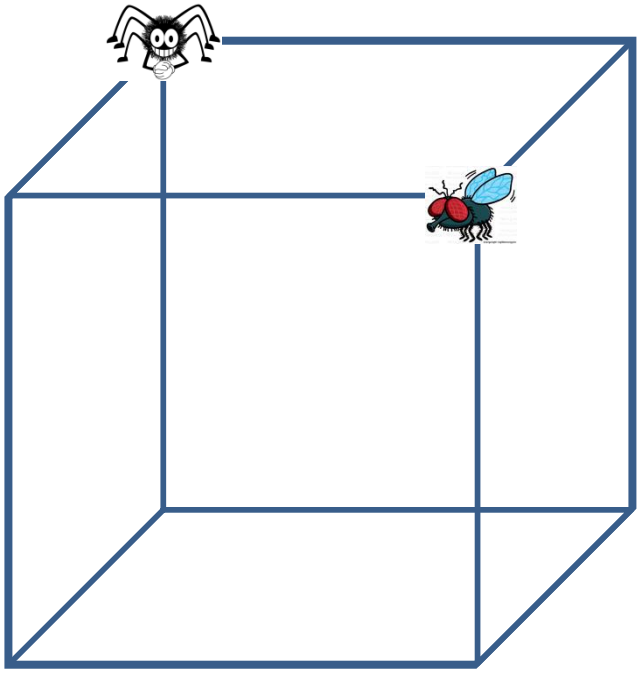
Reinforcement Learning

Summer 2017

Defining MDPs, Planning

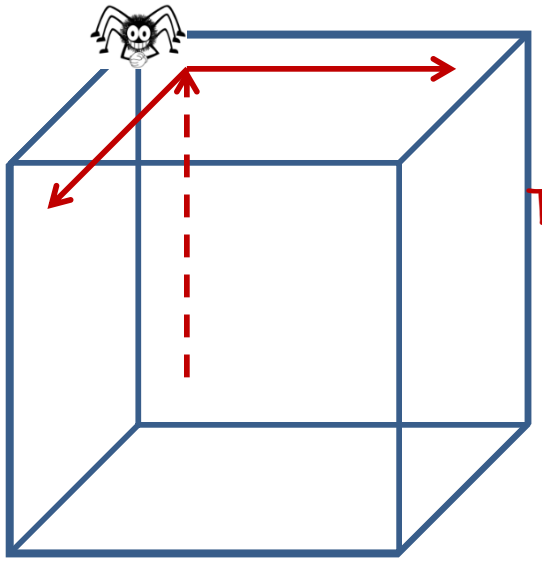

Slide 10
You are here
time
0
understandability

Markov Process



- Where you will go depends only on where you are

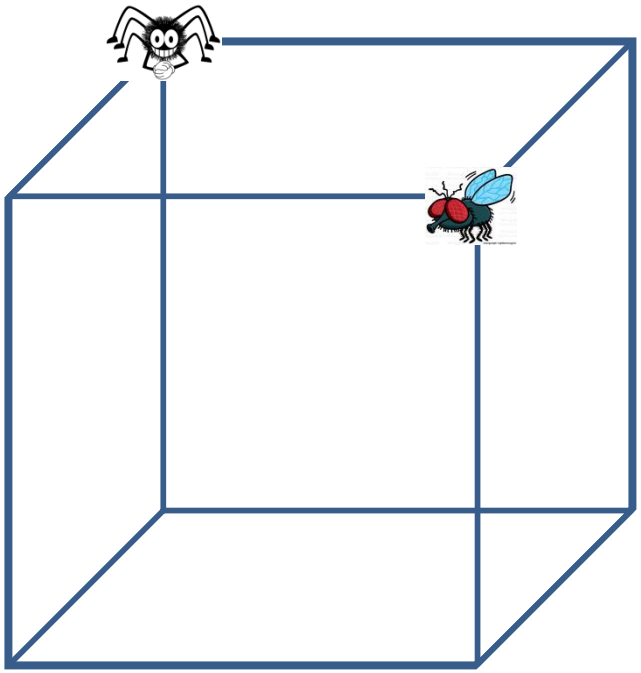
Markov Process: Information state



This spider doesn't like to turn back

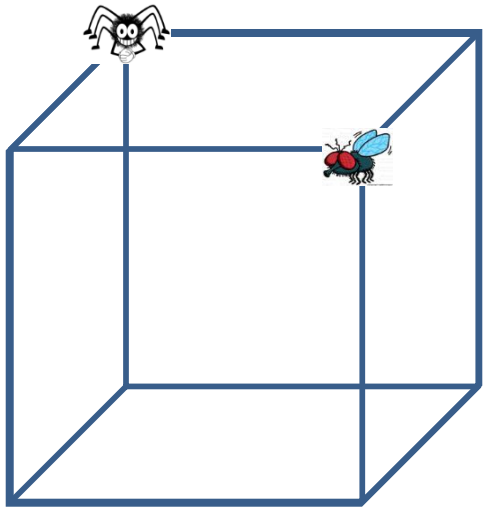
- The *information* state of a Markov process may be different from its physical state

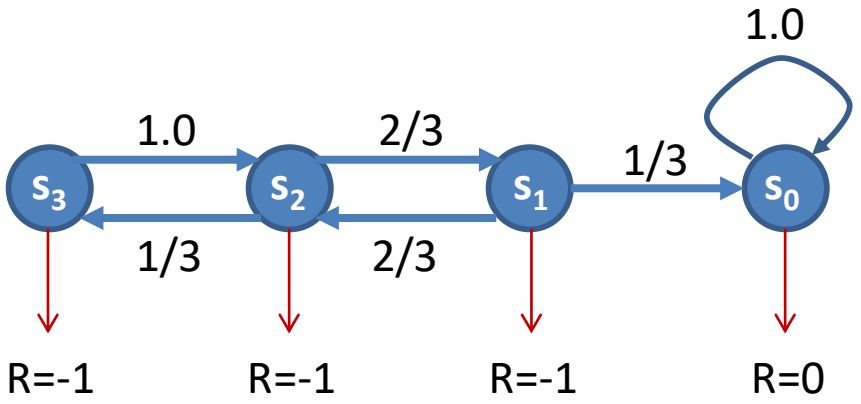
Markov Reward Process



- Random wandering through states will occasionally win you a reward

The Fly Markov Reward Process





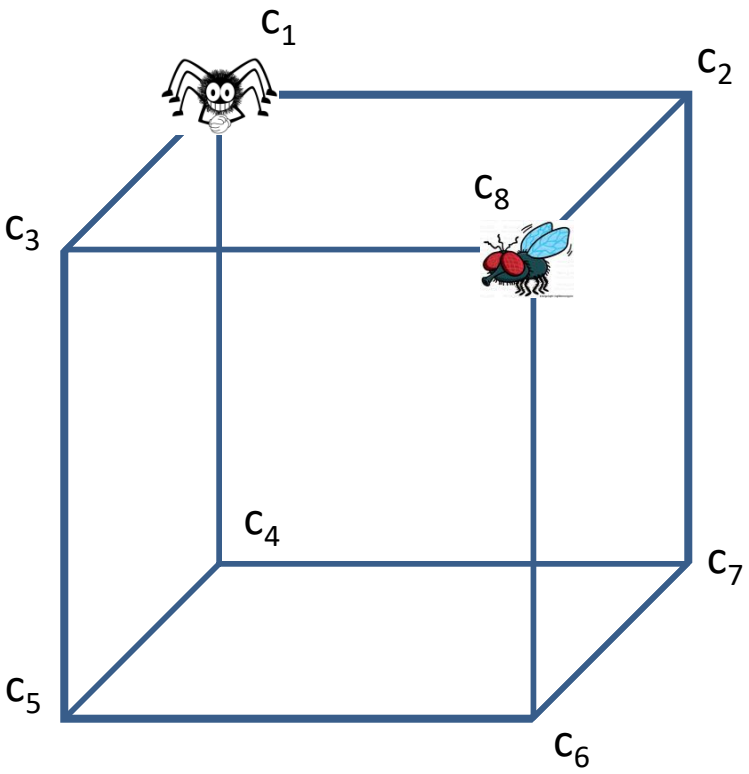
- There are, in fact, only four states, not eight
  - Manhattan distance between fly and spider = 0 (s_0)
  - Distance between fly and spider = 1 (s_1)
  - Distance between fly and spider = 2 (s_2)
  - Distance between fly and spider = 3 (s_3)
- Can, in fact, redefine the MRP entirely in terms of these 4 states

The discounted return

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$$

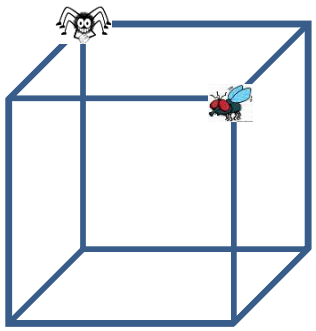
- Total *future* reward all the way to the end

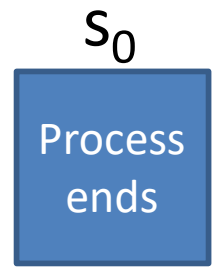
Markov Decision Process

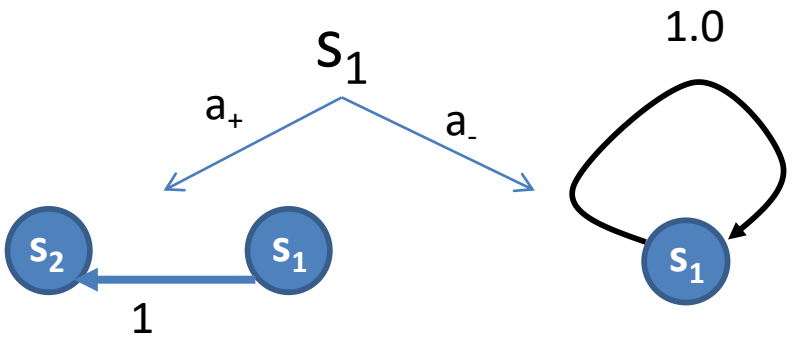


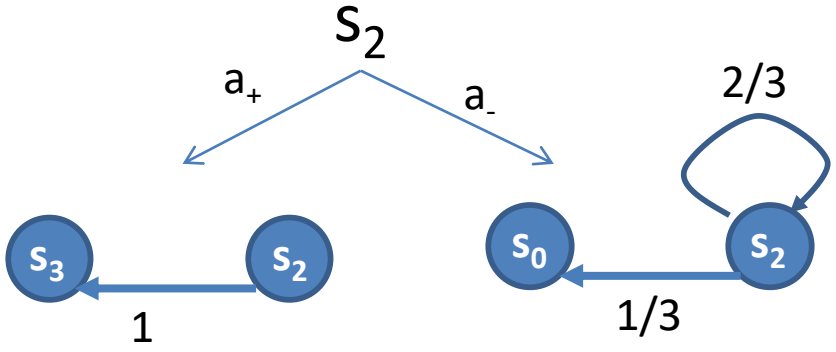
- Markov Reward Process with following change:
  - Agent has real agency
  - Agent's actions modify environment's behavior

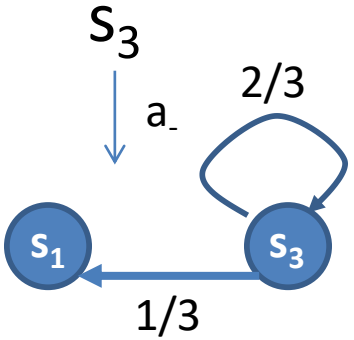
The Fly Markov Decision Process



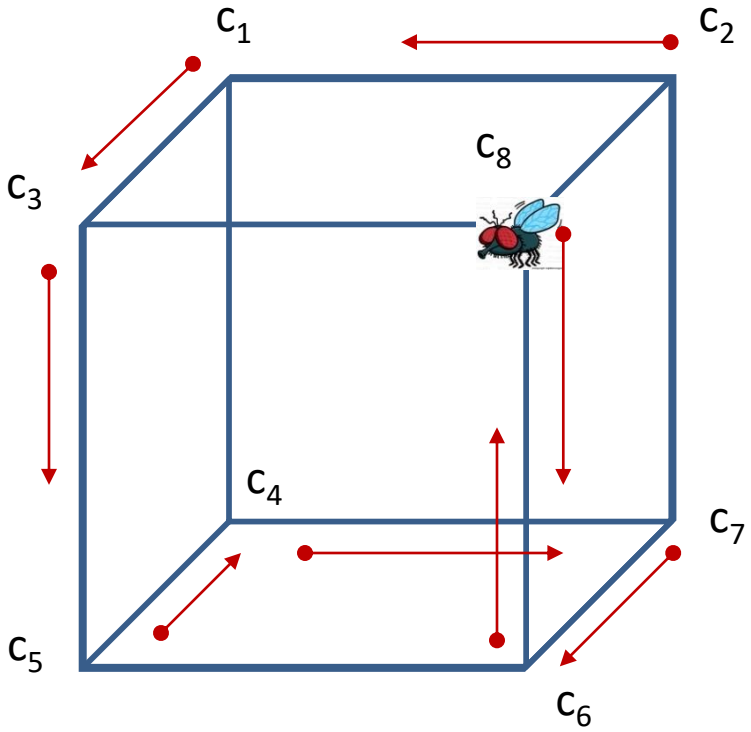








Policy



- The *policy* is the agent's choice of action in each state
  - May be stochastic

The *state value* function of an MDP

- The *expected return* from any state depends on the policy you follow
- We will index the value of any state by the policy to indicate this

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(R_s^a + \gamma \sum_{s'} P_{s,s'}^a v_\pi(s') \right)$$

Bellman Expectation Equation for State Value Functions of an MDP

Note: Although reward was not dependent on action for the fly example, more generally it will be

The *action value* function of an MDP

- The *expected return* from any state under a given policy, when you follow a specific action

$$q_\pi(s,a) = R_s^a + \gamma \sum_{s'} P_{s,s'}^a v_\pi(s')$$

Bellman Expectation Equation for Action Value Functions of an MDP

The Bellman Expectation Equations

- The Bellman expectation equation for state value function

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(R_s^a + \gamma \sum_{s\prime} P_{s,s\prime}^a v_\pi(s') \right)$$

- The Bellman expectation equation for action value function

$$q_\pi(s,a) = R_s^a + \gamma \sum_{s\prime} P_{s,s\prime}^a \sum_{a \in \mathcal{A}} \pi(a|s') q_\pi(s',a)$$

Computing value functions for the MDP

- Finding the state and/or action value functions for the MDP:
  - Given complete MDP (all transition probabilities $P^a_{s,s'}$, expected rewards R^a_s, and discount γ)
  - and a policy π
  - find all value terms $v_\pi(s)$ and/or $q_\pi(s,a)$

- The Bellman expectation equations are simultaneous equations that can be solved for the value functions
  - Although this will be computationally intractable for very large state spaces

Optimal Policies

- Different policies can result in different value functions
- What is the *optimal* policy?

- The optimal policy is the policy that will maximize the expected total discounted reward at every state: $E[G_t|S_t = s]$

$$= E\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \,|S_t = s\right]$$

The optimal policy theorem

- **Theorem**: For any MDP there exists an optimal policy π_* that is better than or equal to every other policy:

$$\pi_* \geq \pi \quad \forall \pi$$

- **Corollary**: If there are *multiple* optimal policies π_{opt1}, π_{opt2}, … all of them achieve the same value function

$$v_{\pi_{opti}}(s) = v_*(s) \quad \forall s$$

- All optimal policies achieve the same action value function

$$q_{\pi_{opti}}(s, a) = q_*(s, a) \quad \forall s, a$$

The optimal value function

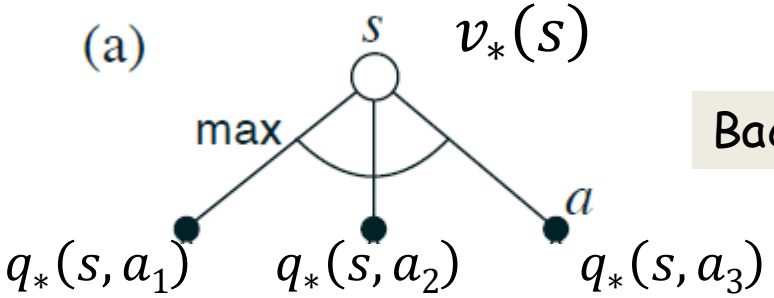
$$\pi_*(a|s) = \begin{cases} 1 \;\; for \quad \underset{a'}{\operatorname{argmax}}\, q_*(s, a') \\ 0 \quad\quad otherwise \end{cases}$$

- Which gives us

$$v_*(s) = \max_a q_*(s, a)$$

Pictorially

Figures from Sutton



Backup Diagram

$$v_*(s) = \max_a q_*(s, a)$$

- Blank circles are states, filled dots are state-action pairs

The optimal value function

$$\pi_*(a|s) = \begin{cases} 1 \;\; for \quad \underset{a'}{\operatorname{argmax}}\, q_*(s,a') \\ 0 \quad\quad otherwise \end{cases}$$

- Which gives us

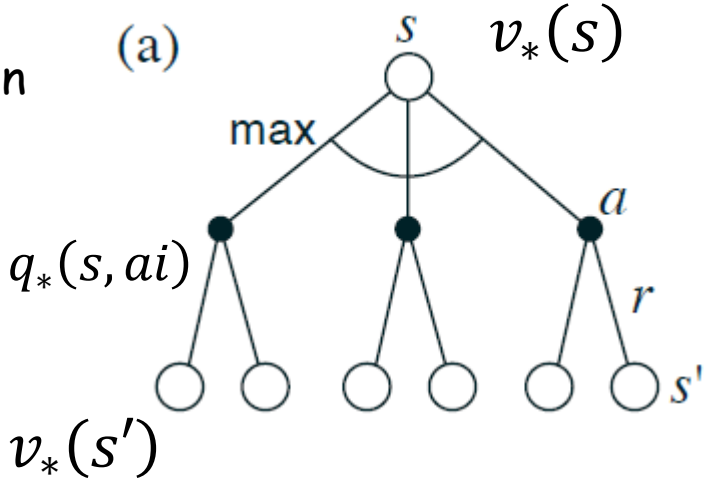
$$v_*(s) = \max_a q_*(s,a)$$

- But, for the optimal policy we also have

$$q_*(s,a) = R_s^a + \gamma \sum_{s'} P_{s,s'}^a v_*(s')$$

Backup Diagram

Figures from Sutton

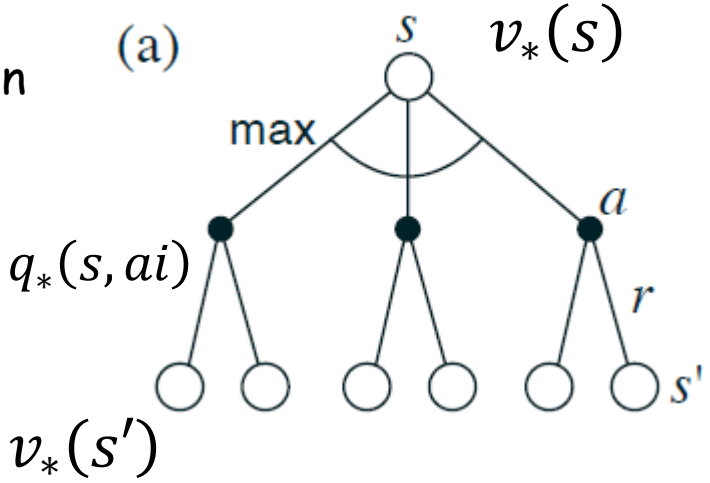


$$v_*(s) = \max_a q_*(s, a)$$

$$q_*(s, a) = R_s^a + \gamma \sum_{s'} P_{s,s'}^a v_*(s')$$

Backup Diagram

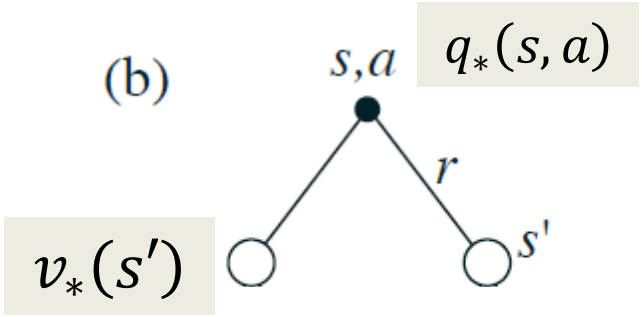
Figures from Sutton



$$v_*(s) = \max_a R_s^a + \gamma \sum_{s\prime} P_{s,s\prime}^a v_*(s')$$

Backup Diagram

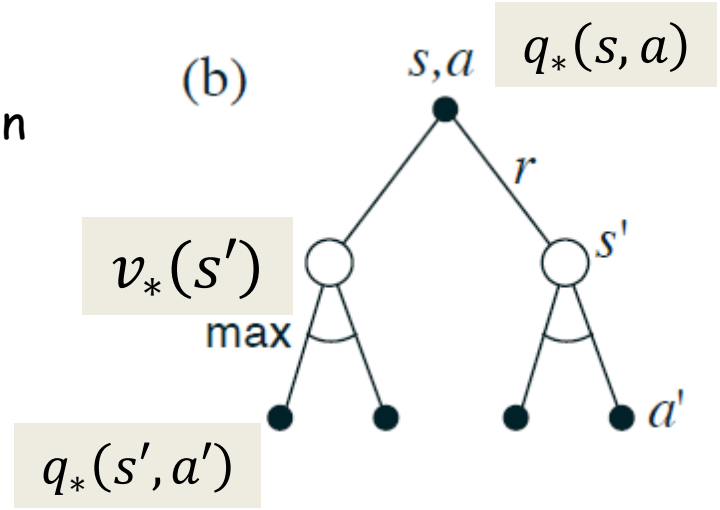
Figures from Sutton



$$q_*(s,a) = R_s^a + \gamma \sum_{s'} P_{s,s'}^a v_*(s')$$

Backup Diagram

Figures from Sutton

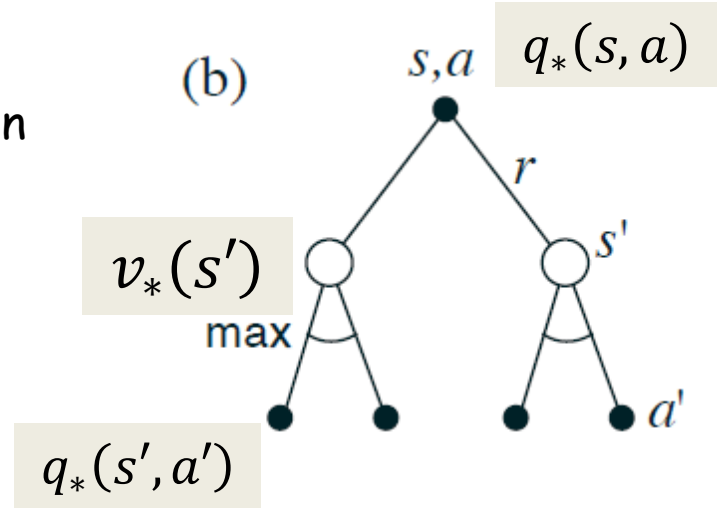


$$q_*(s,a) = R_s^a + \gamma \sum_{s'} P_{s,s'}^a v_*(s')$$

$$v_*(s') = \max_{a'} q_*(s',a')$$

Backup Diagram

Figures from Sutton



$$q_*(s,a) = R_s^a + \gamma \sum_{s'} P_{s,s'}^a \max_{a'} q_*(s',a')$$

Bellman *Optimality* Equations

- Optimal value function equation

$$v_*(s) = \max_a \; R_s^a + \gamma \sum_{s'} P_{s,s'}^a v_*(s')$$

- Optimal action value equation

$$q_*(s,a) = R_s^a + \gamma \sum_{s'} P_{s,s'}^a \max_{a'} q_*(s',a')$$

Optimality Relationships

- Given the MDP: $\langle \mathcal{S}, \mathcal{P}, \mathcal{A}, \mathcal{R}, \gamma \rangle$
- Given the optimal action value functions, the optimal value function can be found

$$v_*(s) = \max_a q_*(s,a)$$

- Given the optimal value function, the optimal action value function can be found

$$q_*(s,a) = R_s^a + \gamma \sum_{s'} P_{s,s'}^a v_*(s')$$

- Given the optimal action value function, the optimal policy can be found

$$\pi_*(a|s) = \begin{cases} 1 \;\; for \quad \underset{a'}{\operatorname{argmax}}\, q_*(s,a') \\ 0 \quad\;\; otherwise \end{cases}$$

"Solving" the MDP

- **Solving the MDP equates to finding the optimal policy** $\pi_*(a|s)$

- Which is equivalent to finding the optimal value function $v_*(s)$
- Or finding the optimal action value function $q_*(s, a)$

- Various solutions will estimate one or the other
  - Value based solutions solve for $v_*(s)$ and $q_*(s, a)$ and derive the optimal policy from them
  - Policy based solutions directly estimate $\pi_*(a|s)$

Solving the Bellman Optimality Equation

- No closed form solutions

- Solutions are iterative
- Given the MDP (Planning):
  - Value iterations
  - Policy iterations

- Not given the MDP (Reinforcement Learning):
  - Q-learning
  - SARSA..

- QUESTIONS before we dive?

Planning with an MDP

- Problem:
  - **Given:** an MDP $\langle \mathcal{S}, \mathcal{P}, \mathcal{A}, \mathcal{R}, \gamma \rangle$
  - **Find:** Optimal policy π_*

- Can either
  - **Value-based Solution:** Find optimal value (or action value) function, and derive policy from it OR
  - **Policy-based Solution:** Find optimal policy directly

Value-based Planning

- "Value"-based solution

- **Breakdown:**
  - **Prediction:** Given *any* policy π find value function $v_\pi(s)$
  - **Control:** Find the optimal policy

Value-based Planning

- "Value"-based solution

- **Breakdown:**
  - **Prediction:** Given *any* policy π find value function $v_\pi(s)$
  - **Control:** Find the optimal policy

Preliminaries

- How do we represent the value function?
- Table:
  - Value function
    - $s \rightarrow v_\pi(s)$
    - For a process with N discrete states, must store/compute N unique values
  - Action value functions
    - $s, a \rightarrow q_\pi(s, a)$
    - For a process with N discrete states and M discrete actions, must store/compute NM unique values
- Later we will see how to represent these when the number of states/actions is too large or continuous

The Bellman Expectation Equation for the value function

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(R_s^a + \gamma \sum_{s'} P_{s,s'}^a v_\pi(s') \right)$$

- In vector form

$$\begin{bmatrix} v_\pi(s_1) \\ v_\pi(s_2) \\ \vdots \\ v_\pi(s_N) \end{bmatrix} = \begin{bmatrix} R_{s_1} \\ R_{s_2} \\ \vdots \\ R_{s_N} \end{bmatrix} + \gamma \begin{bmatrix} P_{s_1,s_1} & P_{s_2,s_1} & \cdots & P_{s_N,s_1} \\ P_{s_1,s_2} & P_{s_2,s_2} & \cdots & P_{s_N,s_2} \\ \vdots & \vdots & \ddots & \vdots \\ P_{s_1,s_N} & P_{s_2,s_N} & \cdots & P_{s_N,s_N} \end{bmatrix} \begin{bmatrix} v_\pi(s_1) \\ v_\pi(s_2) \\ \vdots \\ v_\pi(s_N) \end{bmatrix}$$

- Where
  - $R_s = \sum_{a \in \mathcal{A}} \pi(a|s) R_s^a$
  - $P_{s,s'} = \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s'} P_{s,s'}^a$

The Bellman Expectation Equation for the value function

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(R_s^a + \gamma \sum_{s'} P_{s,s'}^a v_\pi(s') \right)$$

- In vector form

$$\begin{bmatrix} v_\pi(s_1) \\ v_\pi(s_2) \\ \vdots \\ v_\pi(s_N) \end{bmatrix} = \begin{bmatrix} R_{s_1} \\ R_{s_2} \\ \vdots \end{bmatrix} + \gamma \begin{bmatrix} P_{s_1,s_1} & P_{s_2,s_1} & \cdots & P_{s_N,s_1} \\ P_{s_1,s_2} & P_{s_2,s_2} & \cdots & P_{s_N,s_2} \\ \vdots & \vdots & \ddots & \vdots \\ & & & P_{s_N,s_N} \end{bmatrix} \begin{bmatrix} v_\pi(s_1) \\ v_\pi(s_2) \\ \vdots \\ v_\pi(s_N) \end{bmatrix}$$

$$\mathcal{V}_\pi = \mathcal{R}_\pi + \gamma \mathcal{P}_\pi \mathcal{V}_\pi$$

- Where
  - $R_s = \sum_{a \in \mathcal{A}} \pi(a|s) R_s^a$
  - $P_{s,s'} = \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s'} P_{s,s'}^a$

Solving the MDP

$$\mathcal{V}_\pi = \mathcal{R}_\pi + \gamma \mathcal{P}_\pi \mathcal{V}_\pi$$

- Given the expected rewards at every state, the transition probability matrix, the discount factor and the policy:

$$\mathcal{V}_\pi = (\mathrm{I} - \gamma \mathcal{P}_\pi)^{-1} \mathcal{R}_\pi$$

- Easy for processes with a small number of states
- Matrix inversion $O(N^3)$; intractable for large state spaces

What about the action value function?

- The Bellman expectation equation for action value function

$$q_\pi(s,a) = R_s^a + \gamma \sum_{s'} P_{s,s'}^a \sum_{a \in \mathcal{A}} \pi(a|s') q_\pi(s',a)$$

$$Q_\pi = \mathcal{R}_{\pi,Q} + \gamma \mathcal{P}_{\pi,Q} Q_\pi$$

$NM \times 1$ $NM \times 1$ $NM \times NM$ $NM \times 1$

Even worse!!

So how do we solve these

- The equations are too large, how do we solve them?

- First, a little lesson – from middle school…

What they never taught you in school

- Consider the following equation:

$$ax = b$$

- Where $0 < a < 2$
- Trivial solution: $x = a^{-1}b = \frac{b}{a}$

- But my CPU does not permit division..
  - How do I solve this?

What they never taught you in school

- Must solve the following without division

$$ax = b$$

  - where $0 < a < 2$
- Rewrite as follows

$$x = (1 - a)x + b$$

- The following iteration solves the problem:

$$x^{(k+1)} = (1 - a)x^{(k)} + b$$

- Can start with any $x^{(0)}$
- Proof??

What they never taught you in school

- Must solve the following without division

$$ax = b$$

  - where $0 < a < 2$
- Rewrite as follows

$$x = (1 - a)x + b$$

- The following iteration solves the problem:

$$x^{(k+1)} = (1 - a)x^{(k)} + b$$

- Can start with any $x^{(0)}$
- Proof?? Hint: $0 < a < 2 \Rightarrow |1 - a| < 1$

What they never taught you in school

- Consider any vector equation

$$\mathbf{x} = \mathbf{A}\mathbf{x} + \mathbf{b}$$

  - Where all Eigen values $|\lambda(\mathbf{A})| \leq 1$
    - And some extra criteria…
      - The square submatrix of $(\mathbf{I} - \mathbf{A})$ corresponding to non-zero entries of $\mathbf{b}$ is full rank
      - The square submatrix of $(\mathbf{I} - \mathbf{A})$ corresponding to zero entries of $\mathbf{b}$ is an identity matrix

- The following iteration solves the problem:

$$\mathbf{x}^{(k+1)} = \mathbf{A}\mathbf{x}^{(k)} + \mathbf{b}$$

Eigen values of a probability matrix

- For any Markov transition probability matrix $\mathcal{P}$, all Eigenvalues have magnitude less than or equal to 1

$$|\lambda(\mathcal{P})| \leq 1$$

Solving for the value function

$$\mathcal{V}_\pi = \mathcal{R}_\pi + \gamma \mathcal{P}_\pi \mathcal{V}_\pi$$

- This can be solved by following iteration starting from any initial vector

$$\mathcal{V}_\pi^{(k+1)} = \mathcal{R}_\pi + \gamma \mathcal{P}_\pi \mathcal{V}_\pi^{(k)}$$

Solving for the value function

$$\mathcal{V}_\pi = \mathcal{R}_\pi + \gamma \mathcal{P}_\pi \mathcal{V}_\pi$$

- This can be solved by following iteration starting from any initial vector

$$\mathcal{V}_\pi^{(k+1)} = \mathcal{R}_\pi + \gamma \mathcal{P}_\pi \mathcal{V}_\pi^{(k)}$$

- But how did that help if we need infinite iterations to converge?

Solving for the value function

$$\mathcal{V}_\pi = \mathcal{R}_\pi + \gamma \mathcal{P}_\pi \mathcal{V}_\pi$$

- This can be solved by following iteration starting from any initial vector

$$\mathcal{V}_\pi^{(k+1)} = \mathcal{R}_\pi + \gamma \mathcal{P}_\pi \mathcal{V}_\pi^{(k)}$$

- But how did that help if we need infinite iterations to converge?
  - Solution: Stop when the changes becomes small

$$\left| \mathcal{V}_\pi^{(k+1)} - \mathcal{V}_\pi^{(k+1)} \right| < \varepsilon$$

Actual Implementation

- Initialize $v_\pi^{(0)}(s)$ for all states
- Update

$$v_\pi^{(k+1)}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(R_s^a + \gamma \sum_{s\prime} P_{s,s\prime}^a v_\pi^{(k)}(s') \right)$$

- Update may be in *batch* mode
  - Keep sweep through all states to compute $v_\pi^{(k+1)}(s)$
  - Update $k = k + 1$
- Or incremental
  - Sweep through all the states performing

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(R_s^a + \gamma \sum_{s\prime} P_{s,s\prime}^a v_\pi(s') \right)$$

Actual Implementation

- Initialize $v_\pi^{(0)}(s)$ for all states
- Update

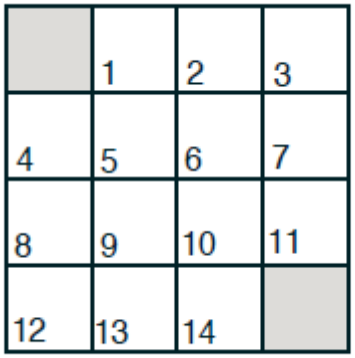
$$v_\pi^{(k+1)}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(R_s^a + \gamma \sum_{s'} P_{s,s'}^a v_\pi^{(k)}(s') \right)$$

- This is an instance of *dynamic programming:*
- **dynamic programming** (also known as **dynamic optimization**) is a method for solving a complex problem by breaking it down into a collection of simpler subproblems, solving each of those subproblems just once, and storing their solutions. The next time the same subproblem occurs, instead of recomputing its solution, one simply looks up the previously computed solution, thereby saving computation time at the expense of a (hopefully) modest expenditure in storage space. (Each of the subproblem solutions is indexed in some way, typically based on the values of its input parameters, so as to facilitate its lookup.) (from wikipedia)

An Example

Example from Sutton

actions

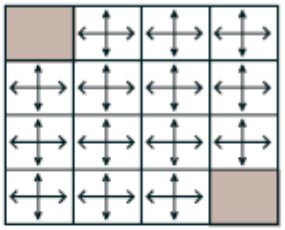
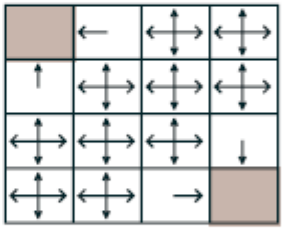
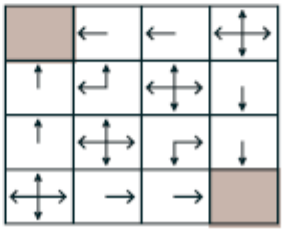


$R = -1$
on all transitions

- All squares, except shaded square have reward -1, shaded square has reward 0
- **Policy:** Random – can step in any of the four directions with equal probability
  - If you run into a wall, you just return to the square
- Find the value of being in each square

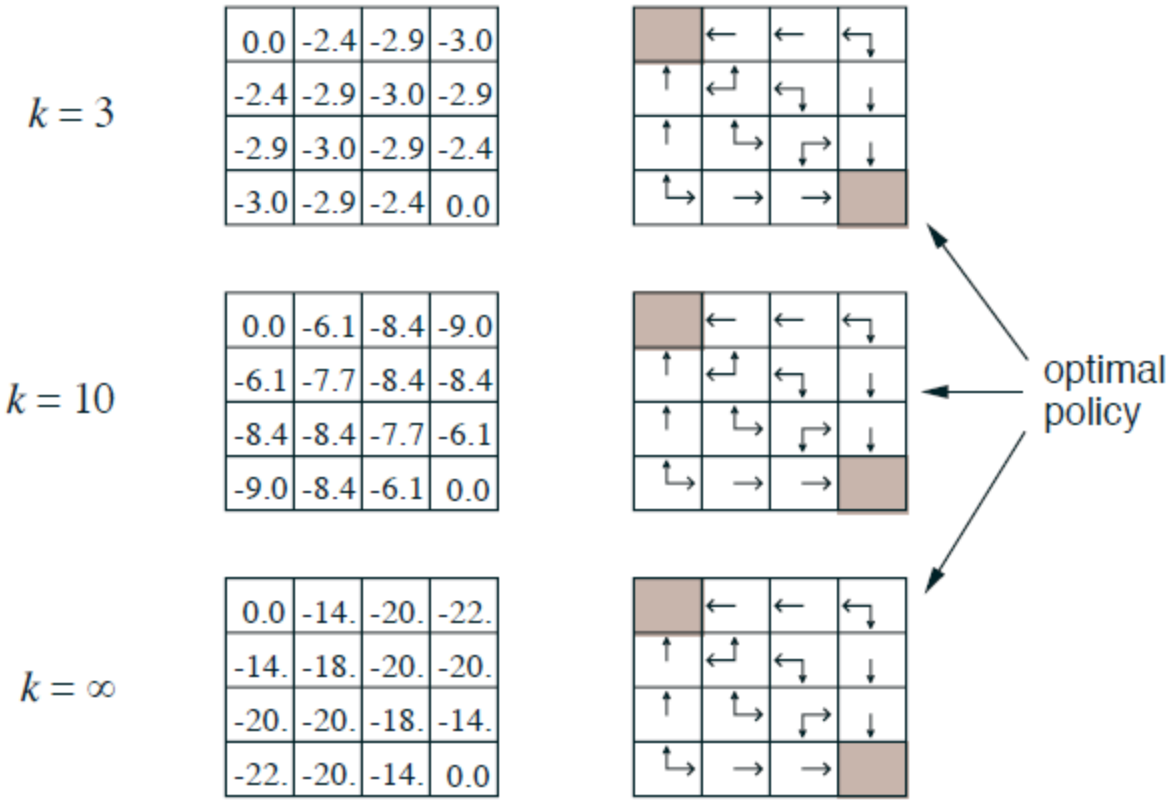
The Gridworld Example

$$v_\pi^{(k+1)}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(R_s^a + \gamma \sum_{s'} P_{s,s'}^a v_\pi^{(k)}(s') \right)$$

	v_k for the Random Policy	Greedy Policy w.r.t. v_k
$k = 0$	0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0	 ← random policy
$k = 1$	0.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 0.0	
$k = 2$	0.0 -1.7 -2.0 -2.0 -1.7 -2.0 -2.0 -2.0 -2.0 -2.0 -2.0 -1.7 -2.0 -2.0 -1.7 0.0	

- Actual iterations use random policy
- Right column shows greedy policy according to current value function

The Gridworld Example



- Iterations use random policy
- Greedy policy converges to optimal long before value function of random policy converges!

Value-based Planning

- "Value"-based solution

- **Breakdown:**
  - **Prediction:** Given *any* policy π find value function $v_\pi(s)$
  - **Control:** Find the optimal policy

Revisit the gridworld

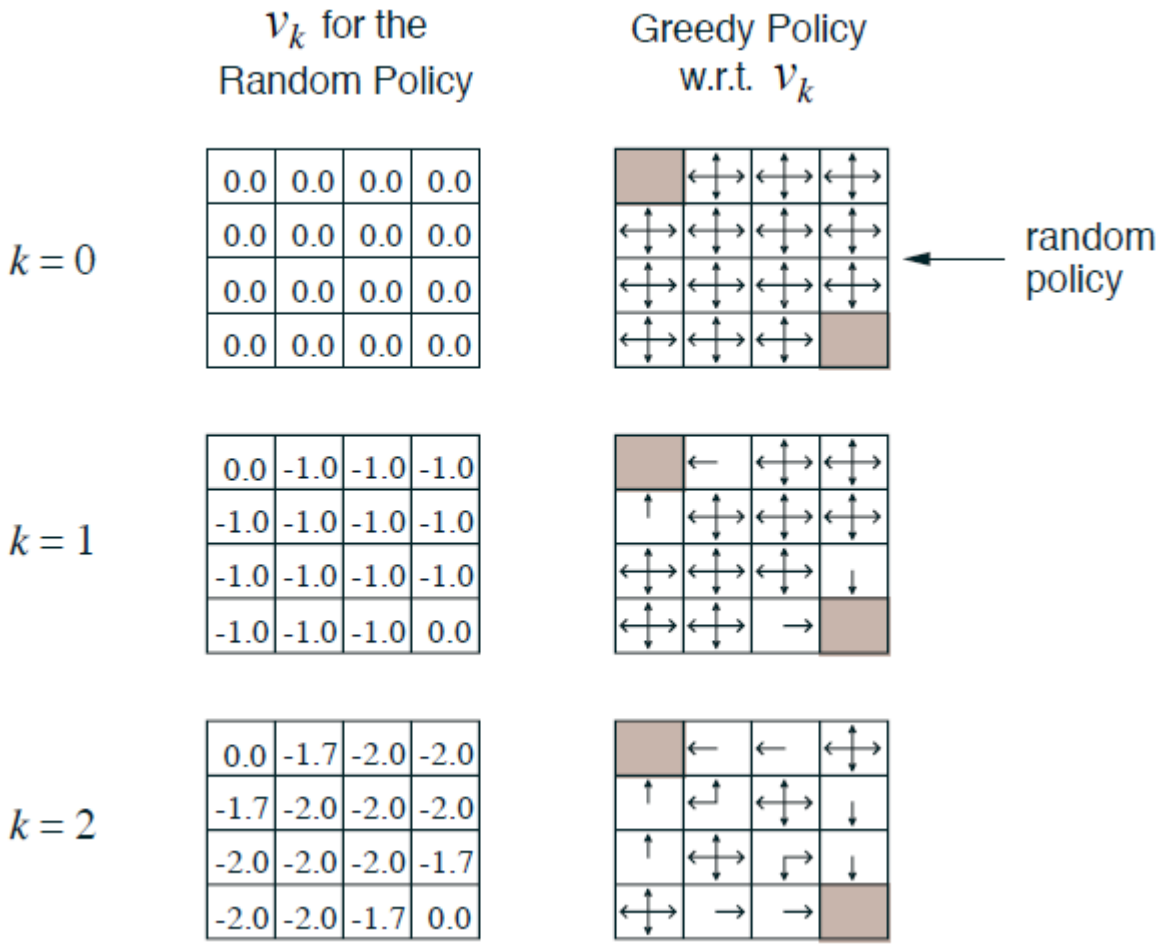
Example from Sutton

actions

	1	2	3
4	5	6	7
8	9	10	11
12	13	14	

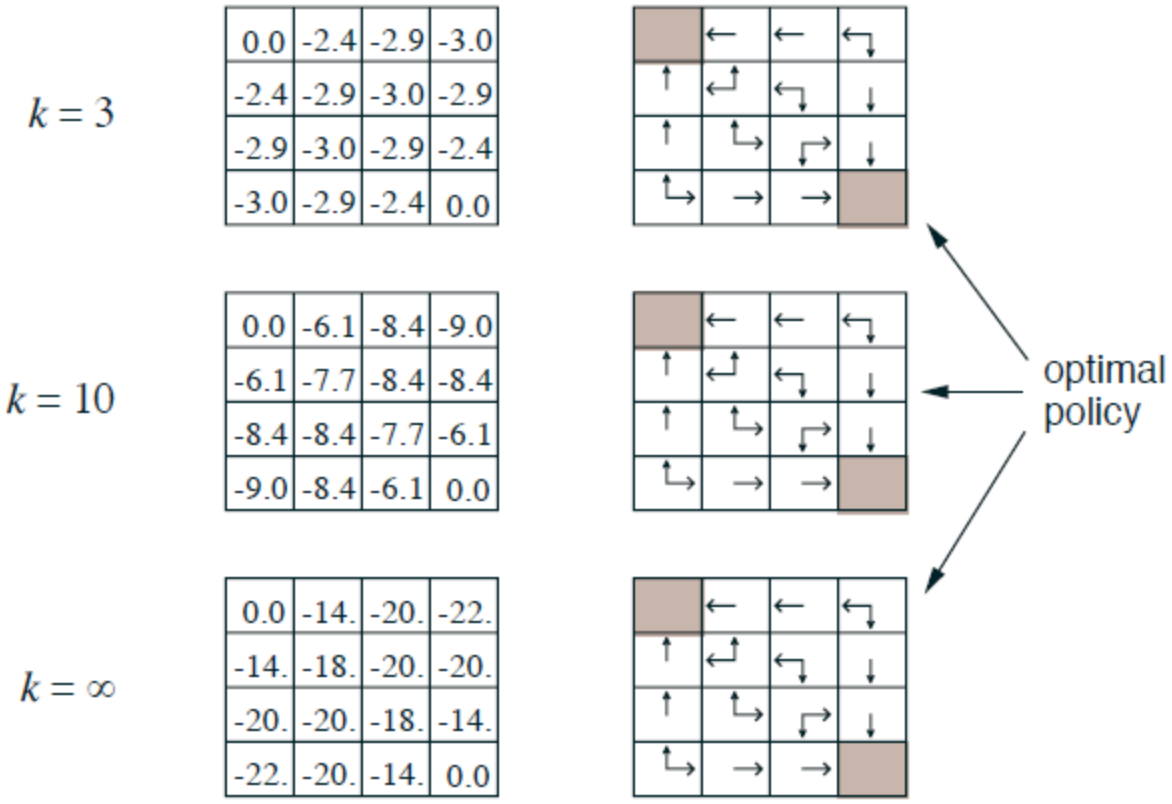
$R = -1$
on all transitions

Revisit the gridworld



- Actual iterations use random policy
- Right column shows greedy policy according to current value function

Revisit the gridworld



- Iterations use random policy
- Greedy policy converges to optimal long before value function of random policy converges!

Finding an optimal policy

- Start with any policy, e.g. random policy $\pi^{(0)}$
- Iterate ($k = 0$ … convergence):
  - Use prediction DP to find the value function $v_{\pi^{(k)}}(s)$
  - Compute action value function $\forall s, a$:

$$q_{\pi^{(k)}}(s,a) = R_s^a + \gamma \sum_{s'} P_{s,s'}^a v_{\pi^{(k)}}(s')$$

  - Find the greedy policy

$$\pi^{(k+1)}(a|s) = \begin{cases} 1 \;\; for\; a = \underset{a'}{\text{argmax}} \;\; q_{\pi^{(k)}}(s,a') \\ 0 \qquad\qquad otherwise \end{cases}$$

Finding an optimal policy: Compact

- Start with any policy $\pi^{(0)}$
- Iterate ($k = 0$ … convergence):
  - Use prediction DP to find the value function $v_{\pi^{(k)}}(s)$
  - Find the greedy policy

$$\pi^{(k+1)}(a|s) = \begin{cases} 1 \;\; for\; a = \underset{a'}{\operatorname{argmax}} \;\; R_s^{a'} + \gamma \sum_{s'} P_{s,s'}^{a'} v_{\pi^{(k)}}(s') \\ 0 \qquad\qquad otherwise \end{cases}$$

Finding an optimal policy: Shorthand

- Start with any policy $\pi^{(0)}$
- Iterate ($k = 0$ … convergence):
  - Use prediction DP to find the value function $v_{\pi^{(k)}}(s)$
  - Find the greedy policy

$$\pi^{(k+1)}(\mathrm{s}) = greedy\left(v_{\pi^{(k)}}(s)\right)$$

THIS IS KNOWN AS **POLICY ITERATION**
In each iteration, we find a policy, and then find its value

Policy Iteration

- Start with any policy $\pi^{(0)}$

- Iterate ($k = 0$ … convergence):
  - Use prediction DP to find the value function $v_{\pi^{(k)}}(s)$
  - Find the greedy policy

$$\pi^{(k+1)}(\mathrm{s}) = greedy\left(v_{\pi^{(k)}}(s)\right)$$

- This will provably converge to the optimal policy π_*
- In the Gridworld example this converged in one iteration
- More generally, it will take several iterations
  - Convergence when policy no longer changes

Generalized Policy Iteration

- Start with any policy $\pi^{(0)}$
- Iterate ($k = 0$ … convergence):
  - Use *any algorithm* to find the value function $v_{\pi^{(k)}}(s)$
  - Use *any algorithm* to find an update policy

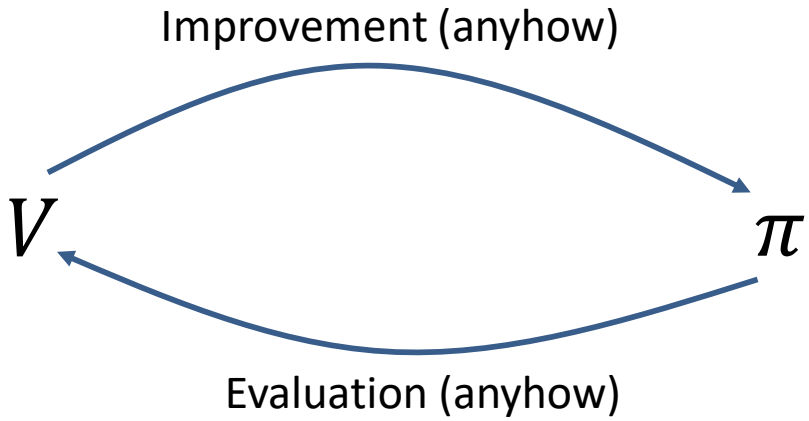
$$\pi^{(k+1)}(\mathrm{s}) = algorithm\left(v_{\pi^{(k)}}(s)\right)$$

Such that $\pi^{(k+1)}(\mathrm{s}) \geq \pi^{(k)}(\mathrm{s})$

- Guaranteed to converge to the optimal policy

Generalized Policy Iteration

- Start with any policy $\pi^{(0)}$



- Guaranteed to converge to the optimal policy

Optimality theorem

- *All* states will hit their optimal value together

- **Theorem:**

  A policy $\pi(a|s)$ has optimal value

  $$v_\pi(s) = v_*(s)$$

  in any state s if and only if for *every* state s' reachable from s,

  $$v_\pi(s') = v_*(s')$$

Policy Iteration

- Start with any policy $\pi^{(0)}$

- Iterate ($k = 0$ … convergence):
  - Use prediction DP to find the value function $v_{\pi^{(k)}}(s)$
  - Find the greedy policy

$$\pi^{(k+1)}(\mathrm{s}) = greedy\left(v_{\pi^{(k)}}(s)\right)$$

- This will provably converge to the optimal policy π_*
- In the Gridworld example this converged in one iteration
- More generally, it will take several iterations
  - Convergence when policy no longer changes

Policy Iteration

- Start with any policy $\pi^{(0)}$

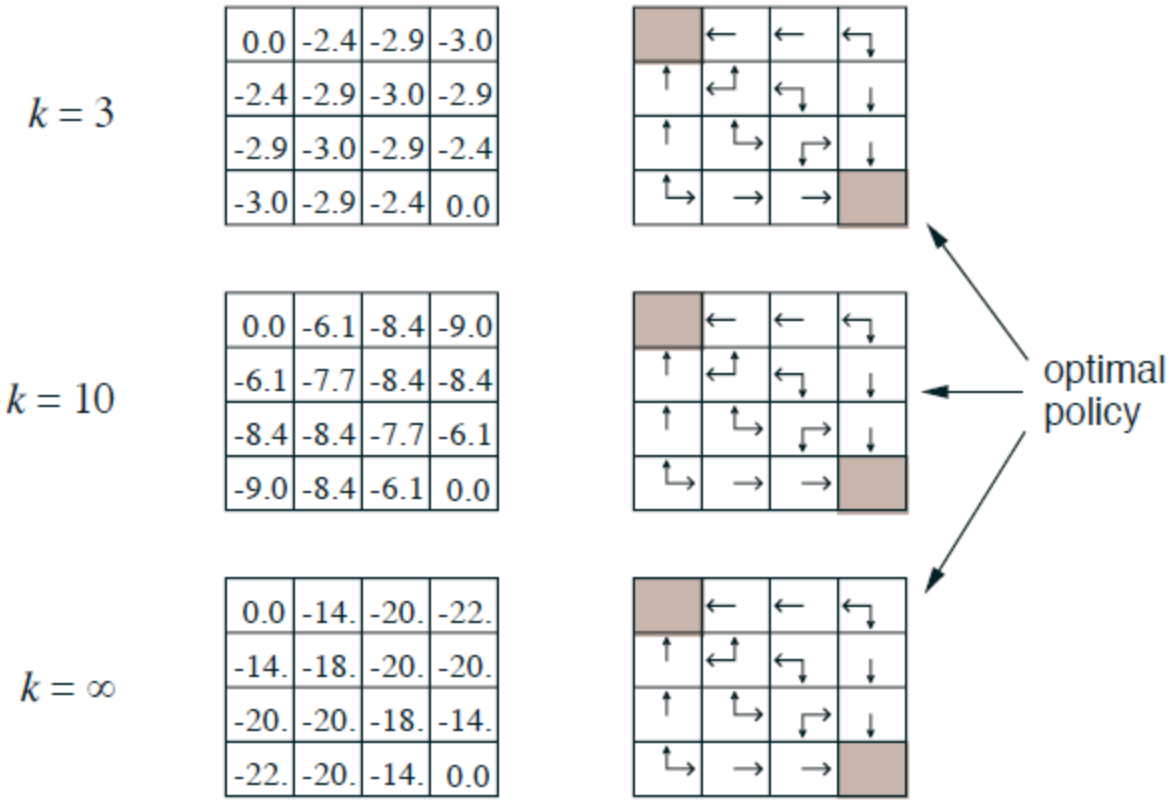
- Iterate ($k = 0$ … convergence):
  - Use prediction DP to find the value function $v_{\pi^{(k)}}(s)$
  - Find the greedy policy

In the gridworld example we didn't even need to run *this* to convergence

The optimal policy was found long before the actual value function converged even in the first upper iteration

- This will provably converge to the optimal policy π_*
- In the Gridworld example this converged in one iteration
- More generally, it will take several iterations
  - Convergence when policy no longer changes

Revisit the gridworld



- Iterations use random policy
- Greedy policy converges to optimal long before value function of random policy converges!

Policy Iteration

- Start with any policy $\pi^{(0)}$
- Iterate ($k = 0$ … convergence):
  - Use prediction DP to find the value function $v_{\pi^{(k)}}(s)$
  - Find the greedy policy

$$\pi^{(k+1)}(\mathrm{s}) = greedy\left(v_{\pi^{(k)}}(s)\right)$$

In the gridworld example we didn't even need to run *this* to convergence

The optimal policy was found long before the actual value function converged even in the first upper iteration

Do we even need the prediction DP to converge?

- More generally, it will take several iterations
  - Convergence when policy no longer changes

Optimal policy estimation

- Start with any policy $\pi^{(0)}$

- Iterate ($k = 0$ … convergence):
  - Use L iterations of prediction DP to find the value function $v_{\pi^{(k)}}(s)$
  - Find the greedy policy

$$\pi^{(k+1)}(\mathrm{s}) = greedy\left(v_{\pi^{(k)}}(s)\right)$$

- This will provably converge to the optimal policy π_*

Optimal policy estimation

- Start with any policy $\pi^{(0)}$

- Iterate ($k = 0$ … convergence):
  - Use 1 iterations of prediction DP to find the value function $v_{\pi^{(k)}}(s)$
  - Find the greedy policy

$$\pi^{(k+1)}(\mathrm{s}) = greedy\left(v_{\pi^{(k)}}(s)\right)$$

Optimal policy estimation

- Start with any policy $\pi^{(0)}$

- Iterate ($k = 0$ … convergence):
  - Use 1 iterations of prediction DP to find the value function $v_{\pi^{(k)}}(s)$

$$v_{\pi^{(k)}}(s) = \sum_{a \in \mathcal{A}} \pi^{(k)}(a|s) \left(R_s^a + \gamma \sum_{s'} P_{s,s'}^a v_{\pi^{(k)}}(s') \right)$$

  - Find the greedy policy

$$\pi^{(k+1)}(\mathrm{s}) = \operatorname*{argmax}_a \; R_s^a + \gamma \sum_{s'} P_{s,s'}^a v_{\pi^{(k)}}(s')$$

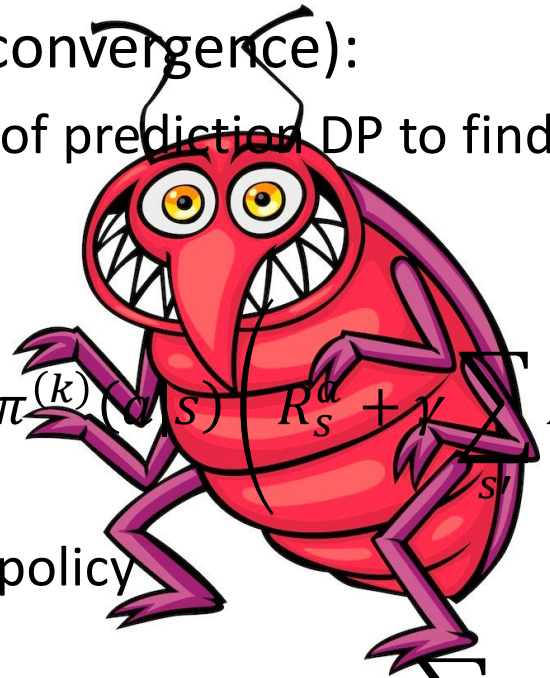
Optimal policy estimation

- Start with any policy $\pi^{(0)}$

- Iterate ($k = 0$ … convergence):
  - Use 1 iterations of prediction DP to find the value function $v_{\pi^{(k)}}(s)$

BUG

$$v_{\pi^{(k)}}(s) = \sum_{a \in \mathcal{A}} \pi^{(k)}(a|s) \left(R_s^a + \gamma \sum_{s'} P_{s,s'}^a v_{\pi^{(k)}}(s') \right)$$



  - Find the greedy policy

$$\pi^{(k+1)}(\mathrm{s}) = \operatorname*{argmax}_a \; R_s^a + \gamma \sum_{s'} P_{s,s'}^a v_{\pi^{(k)}}(s')$$

Reordering and writing carefully

- Start with any initial value function $v_{\pi^{(0)}}(s)$
- Iterate ($k = 1$ … convergence):
  - Find the greedy policy

$$\pi^{(k)}(a|s) = \begin{cases} 1 \ for \ a = \underset{a'}{\text{argmax}} \ R_s^{a'} + \gamma \sum_{s'} P_{s,s'}^{a'} v_{\pi^{(k-1)}}(s') \\ 0 \qquad otherwise \end{cases}$$

  - Use 1 iterations of prediction DP to find the value function $v_{\pi^{(k)}}(s)$

$$v_{\pi^{(k)}}(s) = \sum_{a \in \mathcal{A}} \pi^{(k)}(a|s) \left(R_s^a + \gamma \sum_{s'} P_{s,s'}^a v_{\pi^{(k-1)}}(s') \right)$$

Merging

- Start with any initial value function $v_{\pi(0)}(s)$
- Iterate ($k = 1$ … convergence):
  - Update the value function

$$v_{\pi(k)}(s) = \max_a R_s^a + \gamma \sum_{s'} P_{s,s'}^a v_{\pi(k-1)}(s')$$

- Note: no explicit policy estimation
  - Directly learns value
  - The subscript π is a misnomer

Value Iteration

- Start with any initial value function $v_*^{(0)}(s)$

- Iterate ($k = 1$ … convergence):
  - Update the value function

$$v_*^{(k)}(s) = \max_a \; R_s^a + \gamma \sum_{s'} P_{s,s'}^a v_*^{(k-1)}(s')$$

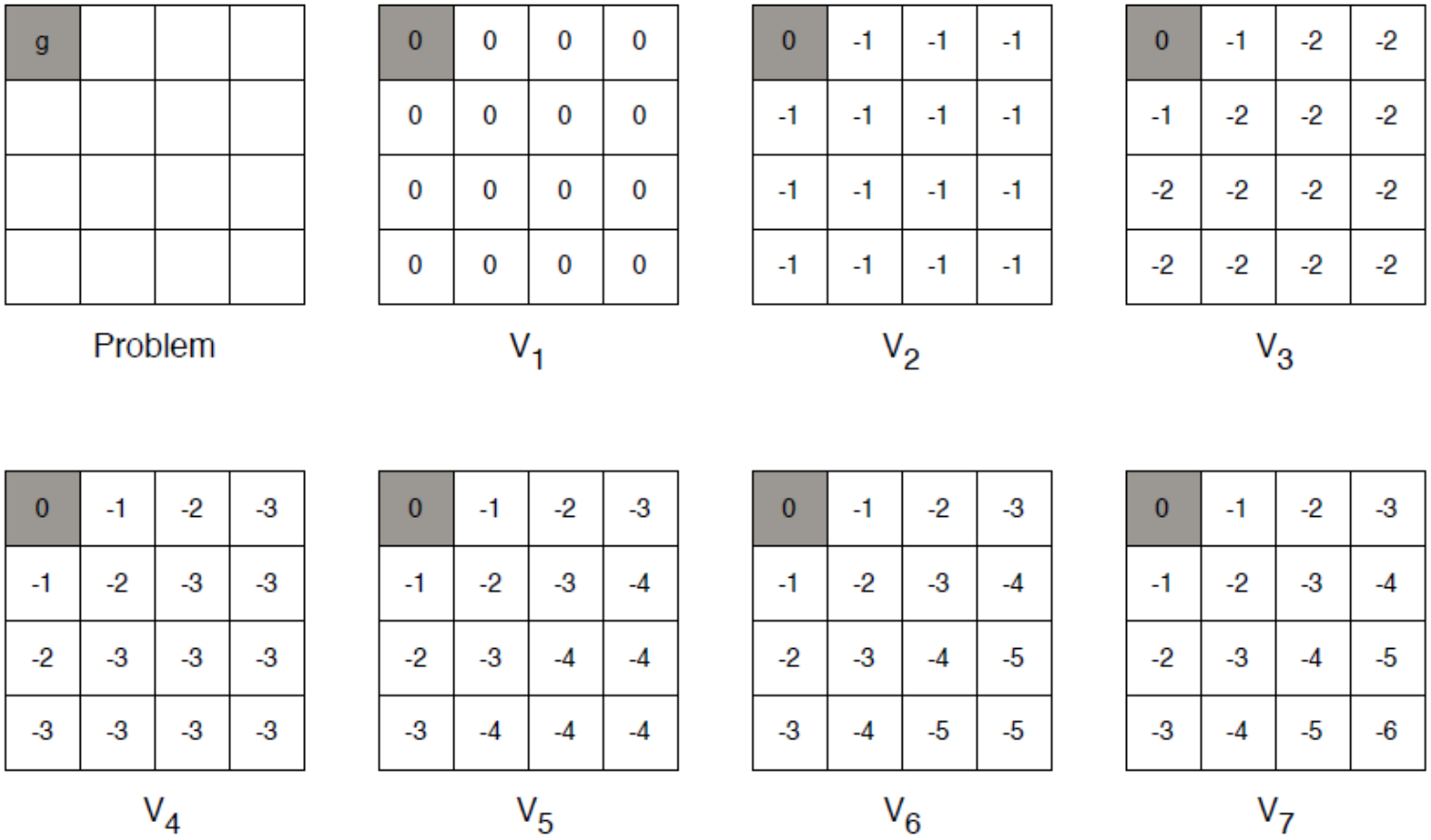
- Note: no explicit policy estimation
- Directly learning *optimal* value function
- Guaranteed to give you optimal value function at convergence
  - But intermediate value function estimates may not represent any policy

Value iteration

$$v_*^{(k)}(s) = \max_a \; R_s^a + \gamma \sum_{s'} P_{s,s'}^a v_*^{(k-1)}(s')$$

- Each state simply inherits the cost of its best neighbour state
  - Cost of neighbor is the value of the neighbour plus cost of getting there

Value Iteration Example



- Target: Find the shortest path
- Every step costs -1

Practical Issues

- Updates can be batch mode
  - Explicitly compute $v_*^{(k+1)}(s)$ from $v_*^{(k)}(s)$for all states
  - Set k = k+1

- Or asynchronous
  - Compute $v_*(s)$ in place while we sweep over states
  - $v_*(s) \leftarrow \max_a \; R_s^a + \gamma \sum_{s'} P_{s,s'}^a v_*(s)$

Recap

- Learned about *prediction*
  - Estimating value function given MDP and policy
- Learned *Policy* iteration
  - Iterate prediction and policy estimation
- Learned about *Value* iteration
  - Directly estimate optimal value function

Alternate strategy

- Worked with *Value function*
  - For N states, estimates N terms
- Could alternately work with *action-value function*
  - For M actions, must estimate MN terms
    - Much more expensive
    - But more useful in some scenarios

Next Up

- We've worked so far with planning
  - Someone gave us the MDP
- Next: Reinforcement Learning
  - MDP unknown..

Problem so far

- *Given all details of the MDP*
  - Compute optimal value function
  - Compute optimal action value function
  - *Compute optimal policy*
- This is the problem of *planning*

- **Problem:** In real life, nobody gives you the MDP
  - How do we plan???

Model-Free Methods

- AKA model-free **reinforcement learning**

- How do you find the value of a policy, without knowing the underlying MDP?
  - Model-free *planning*
- How do you find the optimal policy, without knowing the underlying MDP?
  - Model-free *control*

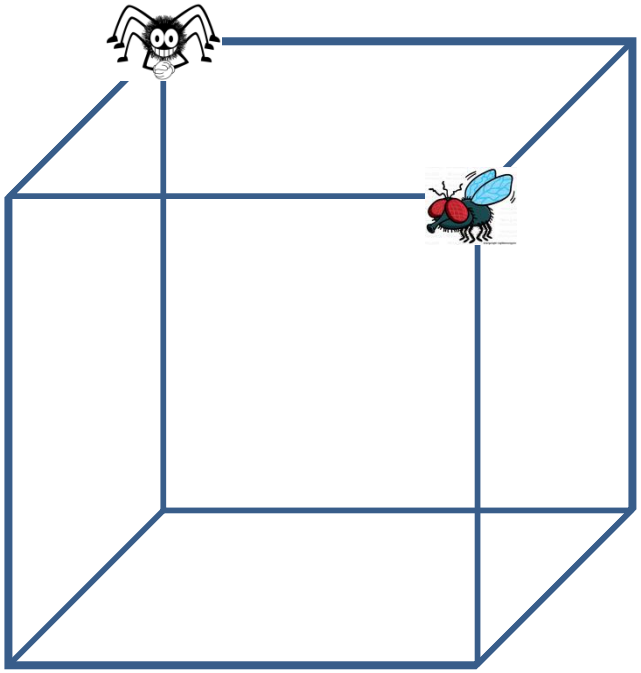
Model-Free Methods

- AKA model-free **reinforcement learning**

- How do you find the value of a policy, without knowing the underlying MDP?
  - Model-free *planning*
- How do you find the optimal policy, without knowing the underlying MDP?
  - Model-free *control*

- ***Assumption:*** We can identify the states, know the *actions,* and measure rewards, but have no knowledge of the system dynamics
  - The key knowledge required to "solve" for the best policy
  - A reasonable assumption in many discrete-state scenarios
  - Can be generalized to other scenarios with infinite or unknowable state

Model-Free Assumption



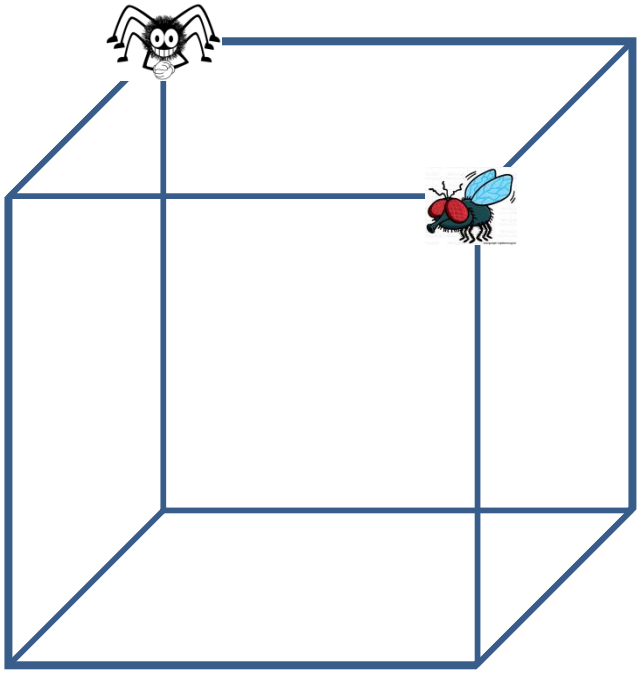
- Can see the fly
- Know the distance to the fly
- Know possible actions (get closer/farther)
- But have no idea of how the fly will respond
  - Will it move, and if so, to what corner

Model-Free Methods

- AKA model-free **reinforcement learning**

- How do you find the value of a policy, without knowing the underlying MDP?
  - Model-free *planning*
- How do you find the optimal policy, without knowing the underlying MDP?
  - Model-free *control*

Model-Free Assumption



- Can see the fly and distance to the fly
- But have no idea of how the fly will respond to actions
  - Will it move, and if so, to what corner
- *But will always try to reduce distance to fly (have a known, fixed, policy)*
- ***What is the value of being a distance D from the fly?***

Methods

- *Monte-Carlo* Learning

- *Temporal-Difference* Learning
  - *TD(1)*
  - *TD(K)*
  - *TD*(λ)

Monte-Carlo learning to learn the value of a policy π

- Just "let the system run" while following the policy π and learn the value of different states

- Procedure: Record several *episodes* of the following
  - Take actions according to policy π
  - Note states visited and rewards obtained as a result
  - Record entire sequence:
  - $S_1, A_1, R_2, S_2, A_2, R_3, \ldots, S_T$
  - **Assumption:** Each "episode" ends at some time

- Estimate value functions based on observations by counting

Monte-Carlo Value Estimation

- Objective: Estimate value function $v_\pi(s)$ for every state s, given recordings of the kind:

$$S_1, A_1, R_2, S_2, A_2, R_3, \dots, S_T$$

- Recall, the value function is the expected return:

$$v_\pi(s) = E[G_t | S_t = s]$$
$$= E\left[R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-t-1} R_T | S_t = s\right]$$

- To estimate this, we replace the *statistical* expectation $E[G_t | S_t = s]$ by the *empirical* average $avg[G_t | S_t = s]$

A bit of notation

- We actually record *many* episodes
  - $episode(1) = S_{11}, A_{11}, R_{12}, S_{12}, A_{12}, R_{13}, \ldots, S_{1T_1}$
  - $episode(2) = S_{21}, A_{21}, R_{22}, S_{22}, A_{22}, R_{23}, \ldots, S_{2T_2}$
  - …
  - Different episodes may be different lengths

Counting Returns

- For each episode, we count the returns at all times:
  - $S_{11}, A_{11}, R_{12}, S_{12}, A_{12}, R_{13}, S_{13}, A_{13}, R_{14}, \dots, S_{1T_1}$

- Return at time t
  - $G_{1,1} = R_{12} + \gamma R_{13} + \cdots + \gamma^{T_1-2} R_{1T_1}$

Counting Returns

- For each episode, we count the returns at all times:
  - $S_{11}, A_{11}, R_{12}, S_{12}, A_{12}, R_{13}, S_{13}, A_{13}, R_{14}, \dots, S_{1T_1}$
- Return at time t $G_{1,2}$
  - $G_{1,1} = R_{12} + \gamma R_{13} + \cdots + \gamma^{T_1-2} R_{1T_1}$
  - $G_{1,2} = R_{13} + \gamma R_{14} + \cdots + \gamma^{T_1-3} R_{1T_1}$

Counting Returns

- For each episode, we count the returns at all times:
  - $S_{11}, A_{11}, R_{12}, S_{12}, A_{12}, R_{13}, S_{13}, A_{13}, R_{14}, \dots, S_{1T_1}$
- Return at time t
  - $G_{1,1} = R_{12} + \gamma R_{13} + \cdots + \gamma^{T_1-2} R_{1T_1}$
  - $G_{1,2} = R_{13} + \gamma R_{14} + \cdots + \gamma^{T_1-3} R_{1T_1}$
  - …
  - $G_{1,t} = R_{1,t+1} + \gamma R_{1,t+2} + \cdots + \gamma^{T_1-t-2} R_{1T_1}$

Estimating the Value of a State

- To estimate the value of any state, identify the instances of that state in the episodes:

$$-S_{11}, A_{11}, R_{12}, S_{12}, A_{12}, R_{13}, S_{13}, A_{13}, R_{14}, \dots, S_{1T_1}$$

$$\boldsymbol{s_a} \qquad\qquad s_b \qquad\qquad \boldsymbol{s_a} \quad \dots$$

- Compute the average return from those instances

$$v_\pi(s_a) = avg\big(G_{1,1}, G_{1,3}, \dots\big)$$

Estimating the Value of a State

- For every state s
  - Initialize: Count $N(s) = 0,$ Total return $v_\pi(s) = 0$
  - For every episode e
    - For every time $t = 1 \ldots T_e$
      - Compute G_t
      - If $(S_t == s)$
        - » $N(s) = N(s) + 1$
        - » $v_\pi(s) = v_\pi(s) + G_t$
  - $v_\pi(s) = v_\pi(s)/N(s)$

- Can be done more efficiently..

Online Version

- For all s Initialize: Count $N(s) = 0,$ Total return $totv_\pi(s) = 0$
- For every episode e
  - For every time $t = 1 \ldots T_e$
    - Compute G_t
    - $N(S_t) = N(S_t) + 1$
    - tot$v_\pi(S_t) = totv_\pi(S_t) + G_t$
  - For every state s : $v_\pi(s) = totv_\pi(s)/N(s)$

- Updating values at the end of each episode
- Can be done more efficiently..

Monte Carlo estimation

- Learning from experience explicitly
- After a sufficiently large number of episodes, in which all states have been visited a sufficiently large number of times, we will obtain good estimates of the value functions of all states
- Easily extended to evaluating *action value functions*

Estimating the Action Value function

- To estimate the value of any state-action pair, identify the instances of that state-action pair in the episodes:
  - $S_1, A_1, R_2, S_2, A_2, R_3, S_3, A_3, R_4, \ldots, S_T$

    $\boldsymbol{s_a}\ \boldsymbol{a_x}$ $\quad s_b\ a_y$ $\quad \boldsymbol{s_a}\ a_y\ \ldots$

- Compute the average return from those instances

$$q_\pi(s_a, a_x) = avg(G_{1,1}, \ldots)$$

Online Version

- For all s, a Initialize: Count $N(s,a) = 0$, Total value $totq_\pi(s,a) = 0$
- For every episode e
  - For every time $t = 1 \ldots T_e$
    - Compute G_t
    - $N(S_t, A_t) = N(S_t, A_t) + 1$
    - tot$q_\pi(S_t, A_t) = totq_\pi(S_t, A_t) + G_t$
  - For every $s, a : q(s,a) = totq_\pi(s,a)/N(s,a)$
- Updating values at the end of each episode

Monte Carlo: Good and Bad

- Good:
  - Will eventually get to the right answer
  - *Unbiased* estimate
- Bad:
  - Cannot update anything until the end of an episode
    - Which may last for ever
  - High variance! Each return adds many random values
  - Slow to converge

Online methods for estimating the value of a policy: Temporal Difference Leaning (TD)

- Idea: Update your value estimates after every observation

$$S_1, A_1, R_2, S_2, A_2, R_3, S_3, A_3, R_4, \ldots, S_T$$

Update for S_1 | Update for S_2 | Update for S_3

  - Do not actually wait until the end of the episode

Incremental Update of Averages

- Given a sequence $x_1, x_2, x_3, \ldots$ a running estimate of their average can be computed as

$$\mu_k = \frac{1}{k}\sum_{i=1}^{k} x_i$$

- This can be rewritten as:

$$\mu_k = \frac{(k-1)\mu_{k-1} + x_k}{k}$$

- And further refined to

$$\mu_k = \mu_{k-1} + \frac{1}{k}(x_k - \mu_{k-1})$$

Incremental Update of Averages

- Given a sequence $x_1, x_2, x_3, \ldots$ a running estimate of their average can be computed as

$$\mu_k = \mu_{k-1} + \frac{1}{k}(x_k - \mu_{k-1})$$

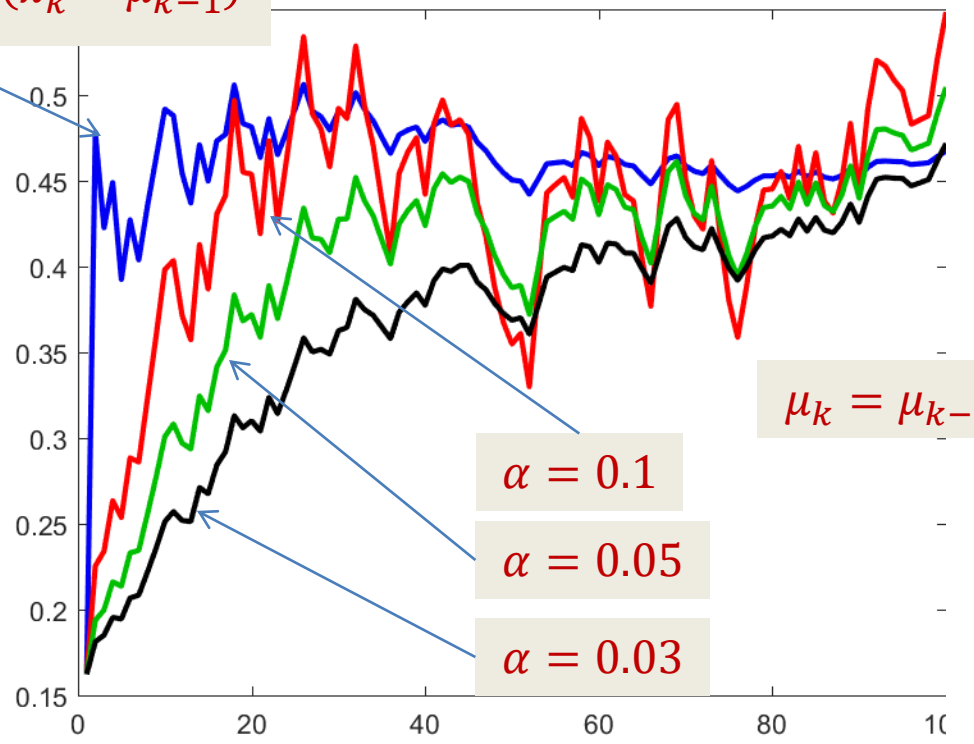
- Or more generally as

$$\mu_k = \mu_{k-1} + \alpha(x_k - \mu_{k-1})$$

- The latter is particularly useful for non-stationary environments

Incremental Updates

$$\mu_k = \mu_{k-1} + \frac{1}{k}(x_k - \mu_{k-1})$$



$$\mu_k = \mu_{k-1} + \alpha(x_k - \mu_{k-1})$$

$\alpha = 0.1$

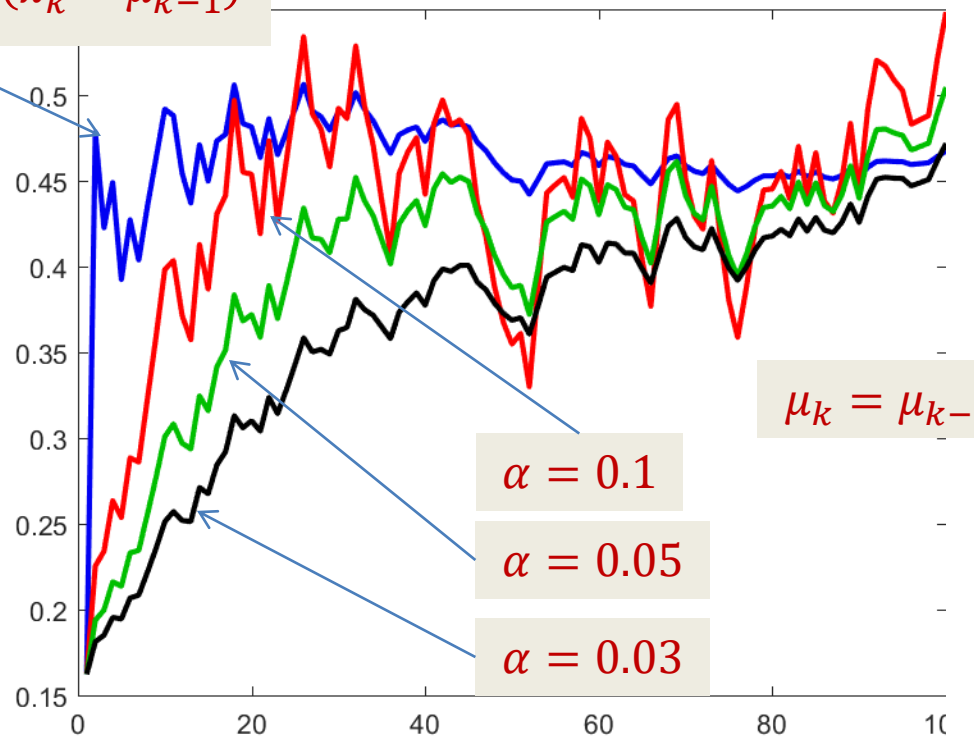
$\alpha = 0.05$

$\alpha = 0.03$

- Example of running average of a uniform random variable

Incremental Updates

$$\mu_k = \mu_{k-1} + \frac{1}{k}(x_k - \mu_{k-1})$$



$$\mu_k = \mu_{k-1} + \alpha(x_k - \mu_{k-1})$$

$\alpha = 0.1$

$\alpha = 0.05$

$\alpha = 0.03$

- Correct equation is *unbiased* and converges to true value
- Equation with α is *biased* (early estimates can be expected to be wrong) but *converges* to true value

Updating Value Function Incrementally

- Actual update

$$v_\pi(s) = \frac{1}{N(s)} \sum_{i=1}^{N(s)} G_{t(i)}$$

  - $N(s)$ is the total number of visits to state s across all episodes
  - $G_{t(i)}$ is the discounted return at the time instant of the i-th visit to state s

Online update

- Given any episode

$$S_1, A_1, R_2, S_2, A_2, R_3, S_3, A_3, R_4, \dots, S_T$$

- Update the value of each state visited

$$N(S_t) = N(S_t) + 1$$

$$v_\pi(S_t) = v_\pi(S_t) + \frac{1}{N(S_t)}\big(G_t - v_\pi(S_t)\big)$$

- Incremental version

$$v_\pi(S_t) = v_\pi(S_t) + \alpha\big(G_t - v_\pi(S_t)\big)$$

- Still an unrealistic rule
  - Requires the entire track until the end of the episode to compute Gt

Online update

- Given any episode

$$S_1, A_1, R_2, S_2, A_2, R_3, S_3, A_3, R_4, \dots, S_T$$

- Update the value of each state visited

$$N(S_t) = N(S_t) + 1$$

$$v_\pi(S_t) = v_\pi(S_t) + \frac{1}{N(S_t)}\left(G_t - v_\pi(S_t)\right)$$

Problem

- Incremental version

$$v_\pi(S_t) = v_\pi(S_t) + \alpha\left(G_t - v_\pi(S_t)\right)$$

- Still an unrealistic rule
  - Requires the entire track until the end of the episode to compute Gt

TD solution

$$v_\pi(S_t) = v_\pi(S_t) + \alpha\big(G_t - v_\pi(S_t)\big)$$

Problem

- But

$$G_t = R_{t+1} + \gamma G_{t+1}$$

- We can approximate G_{t+1} by the *expected* return at the next state S_{t+1}

Counting Returns

- For each episode, we count the returns at all times:
  - $S_1, A_1, R_2, S_2, A_2, R_3, S_3, A_3, R_4, \dots, S_T$
- Return at time t
  - $G_1 = R_2 + \gamma R_3 + \cdots + \gamma^{T-2} R_T$
  - $G_2 = R_3 + \gamma R_4 + \cdots + \gamma^{T-3} R_T$
  - …
  - $G_t = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-t-2} R_T$
- **Can rewrite as**
  - $G_1 = R_2 + \gamma G_2$
- Or
  - $G_1 = R_2 + \gamma R_3 + \gamma^2 G_3$
  - …
  - $G_t = R_{t+1} + \sum_{i=1}^{N} \gamma^i R_{t+1+i} + \gamma^{N+1} G_{t+1+N}$

TD solution

$$v_\pi(S_t) = v_\pi(S_t) + \alpha\big(G_t - v_\pi(S_t)\big)$$

Problem

- But

$$G_t = R_{t+1} + \gamma G_{t+1}$$

- We can approximate G_{t+1} by the *expected* return at the next state $S_{t+1} \approx v_\pi(S_{t+1})$

$$G_t \approx R_{t+1} + \gamma v_\pi(S_{t+1})$$

- We don't know the real value of $v_\pi(S_{t+1})$ but we can "bootstrap" it by its current estimate

TD(1) true online update

$$v_\pi(S_t) = v_\pi(S_t) + \alpha\big(G_t - v_\pi(S_t)\big)$$

- Where

$$G_t \approx R_{t+1} + \gamma v_\pi(S_{t+1})$$

- Giving us

$$- \; v_\pi(S_t) = v_\pi(S_t) + \alpha\big(R_{t+1} + \gamma v_\pi(S_{t+1}) - v_\pi(S_t)\big)$$

TD(1) true online update

$$v_\pi(S_t) = v_\pi(S_t) + \alpha\delta_t$$

- Where

$$\delta_t = R_{t+1} + \gamma v_\pi(S_{t+1}) - v_\pi(S_t)$$

- δ_t is the TD *error*
  - The error between an (estimated) *observation* of G_t and the current estimate $v_\pi(S_t)$

TD(1) true online update

- For all s Initialize: $v_\pi(s) = 0$
- For every episode e
  - For every time $t = 1 \ldots T_e$
    - $v_\pi(S_t) = v_\pi(S_t) + \alpha\big(R_{t+1} + \gamma v_\pi(S_{t+1}) - v_\pi(S_t)\big)$

- There's a "lookahead" of one state, to know which state the process arrives at at the next time
- But is otherwise online, with continuous updates

TD(1)

- Updates continuously – improve estimates as soon as you observe a state (and its successor)

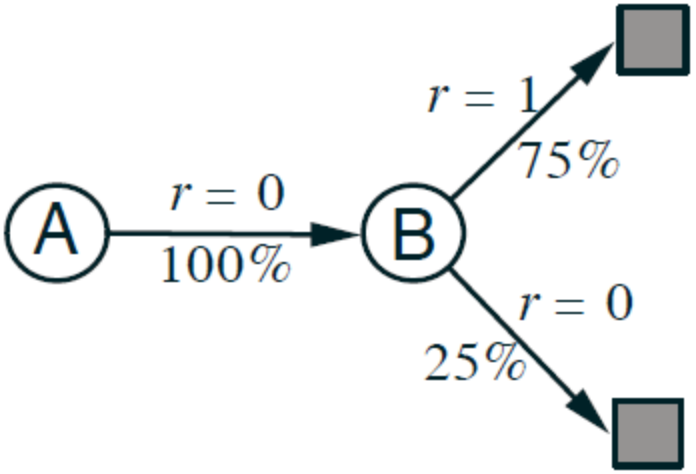
- Can work even with *infinitely long* processes that never terminate
- Guaranteed to converge to the true values eventually
  - Although initial values will be biased as seen before
  - Is actually lower variance than MC!!
    - Only incorporates one RV at any time

- TD can give correct answers when MC goes wrong
  - Particularly when TD is allowed to *loop* over all learning episodes

TD vs MC

A, 0, B, 0
B, 1
B, 1
B, 1

B, 1
B, 1
B, 1
B, 0



- What are V(A) and V(B)
  - Using MC
  - Using TD(1), where you are allowed to repeatedly go over the data

- NOT PRESENTED : THE RANDOM WALK EXAMPLE

TD – look ahead further?

- TD(1) has a look ahead of 1 time step

$$G_t \approx R_{t+1} + \gamma v_\pi(S_{t+1})$$

- But we can look ahead further out
  - $G_t(2) = R_{t+1} + \gamma R_{t+2} + \gamma^2 v_\pi(S_{t+2})$
  - …
  - $G_t(N) = R_{t+1} \sum_{i=1}^{N} \gamma^i R_{t+1+i} + \gamma^{N+1} v_\pi(S_{t+N})$

TD(N) with lookahead

$$v_\pi(S_t) = v_\pi(S_t) + \alpha\delta_t(N)$$

- Where

$$\delta_t(N) = R_{t+1} + \sum_{i=1}^{N} \gamma^i \, R_{t+1+i} + \gamma^{N+1} v_\pi(S_{t+N}) - v_\pi(S_t)$$

- $\delta_t(N)$ is the TD *error* with *N* step lookahead

-

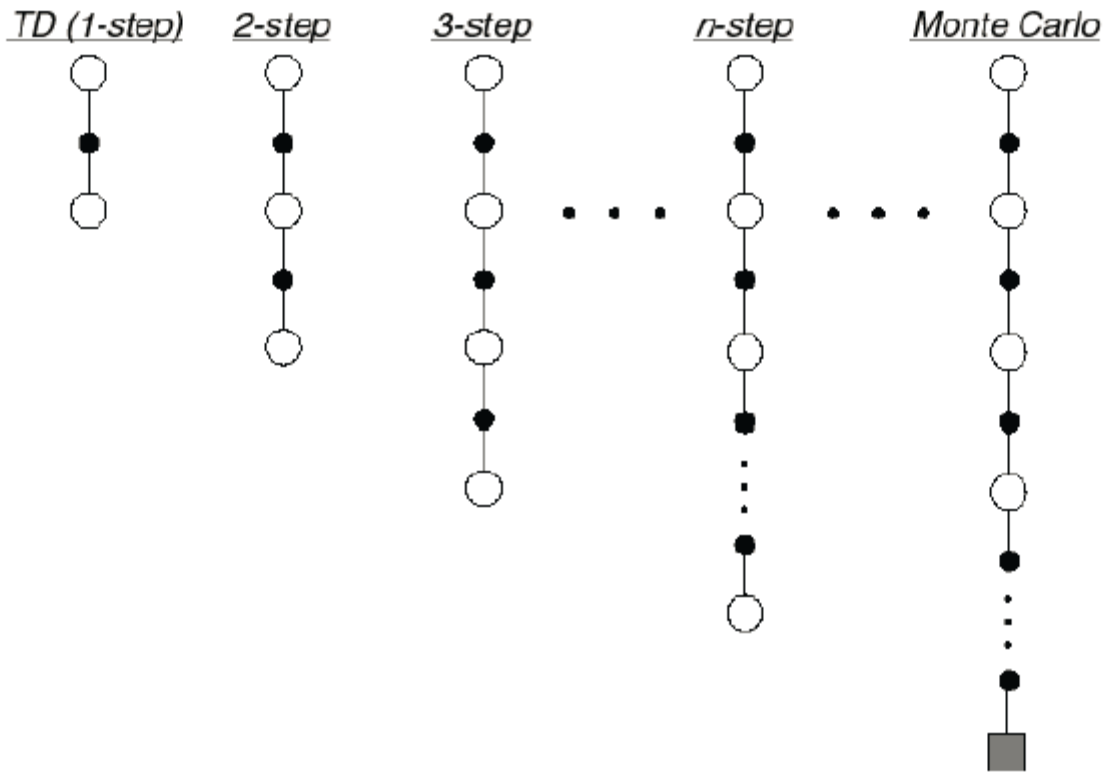
- RANDOM WALK EXAMPLE: LOOKAHEAD IS GOOD PLOTS

Lookahead is good

- Good: The further you look ahead, the better your estimates get
- Problems:
  - But you also get more variance
  - At infinite lookahead, you're back at MC
- Also, you have to wait to update your estimates
  - A lag between observation and estimate
- So how much lookahead must you use

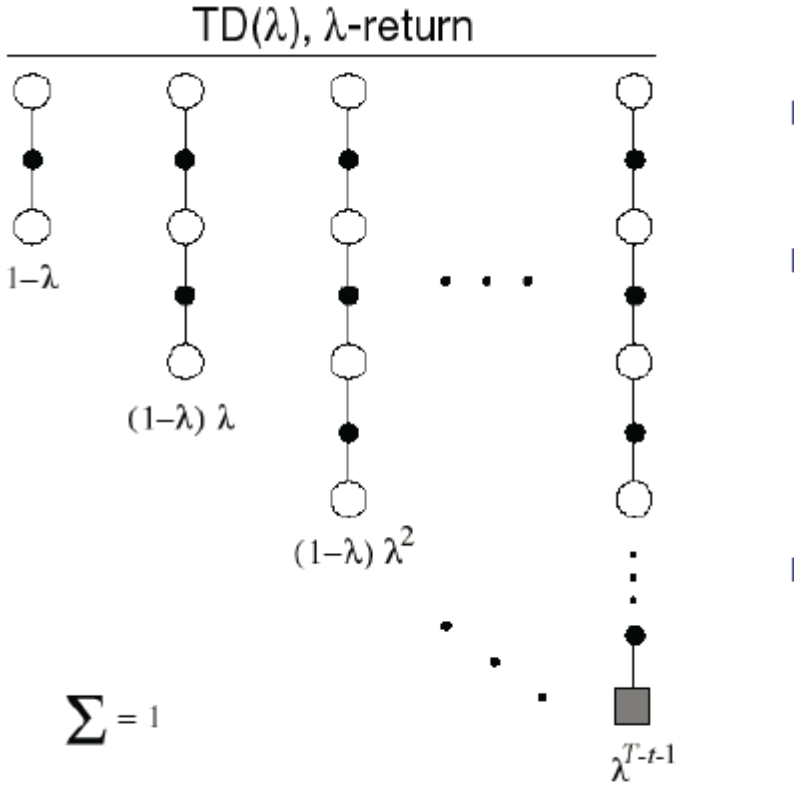
Looking Into The Future

- Let TD target look *n* steps into the future



- How much various TDs look into the future
- Which do we use?

Solution: Why choose?



- Each lookahead provides an estimate of G_t
- Why not just combine the lot with discounting?

TD(λ)

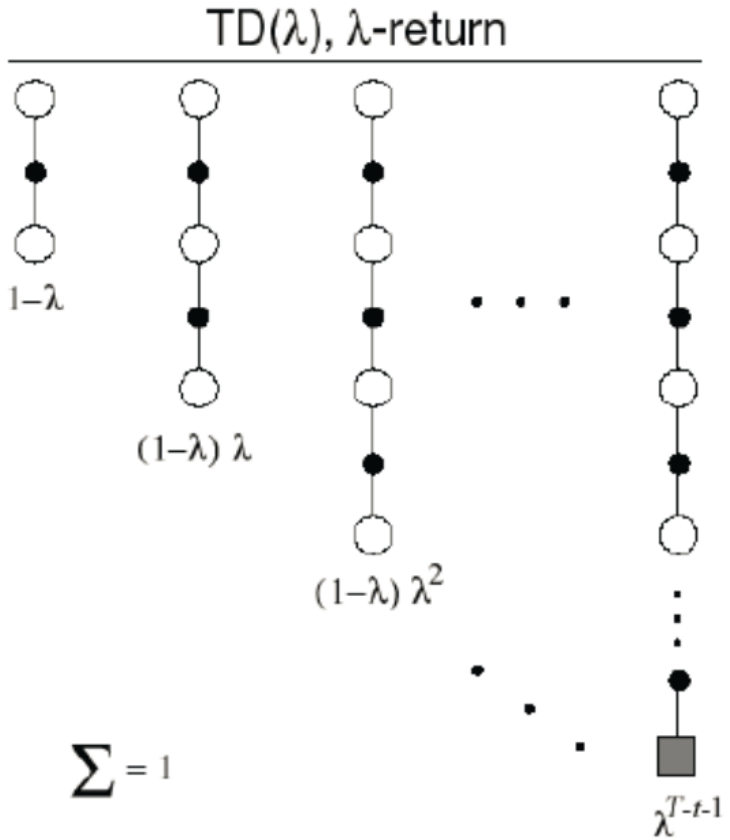
$$G_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)}$$

- Combine the predictions from all lookaheads with an exponentially falling weight
  - Weights sum to 1.0

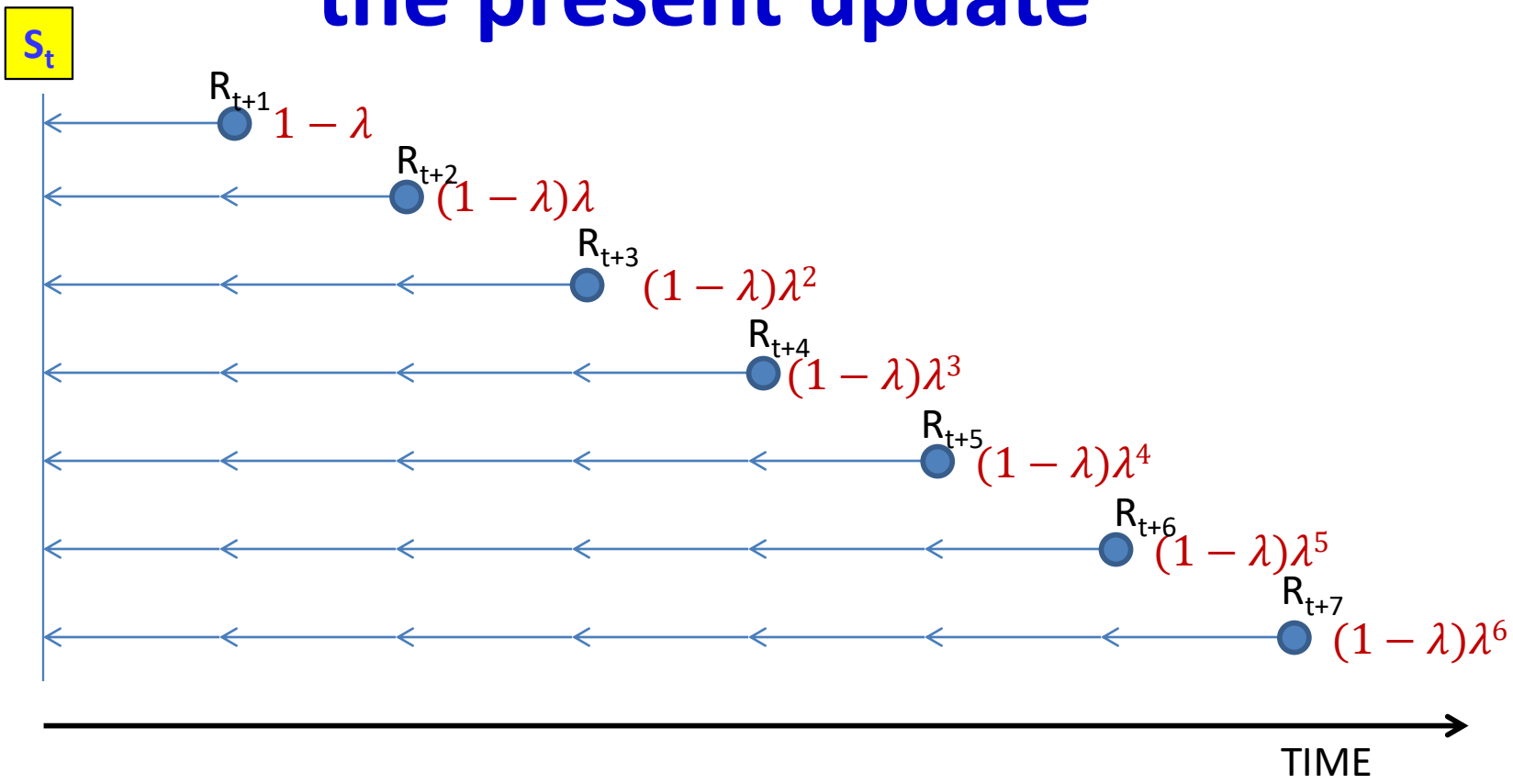
$$V(S_t) \leftarrow V(S_t) + \alpha\left(G_t^\lambda - V(S_t)\right)$$

Something magical just happened

- TD(λ) looks into the infinite future
  - I.e. we must have all the rewards of the future to compute our updates
  - How does that help?

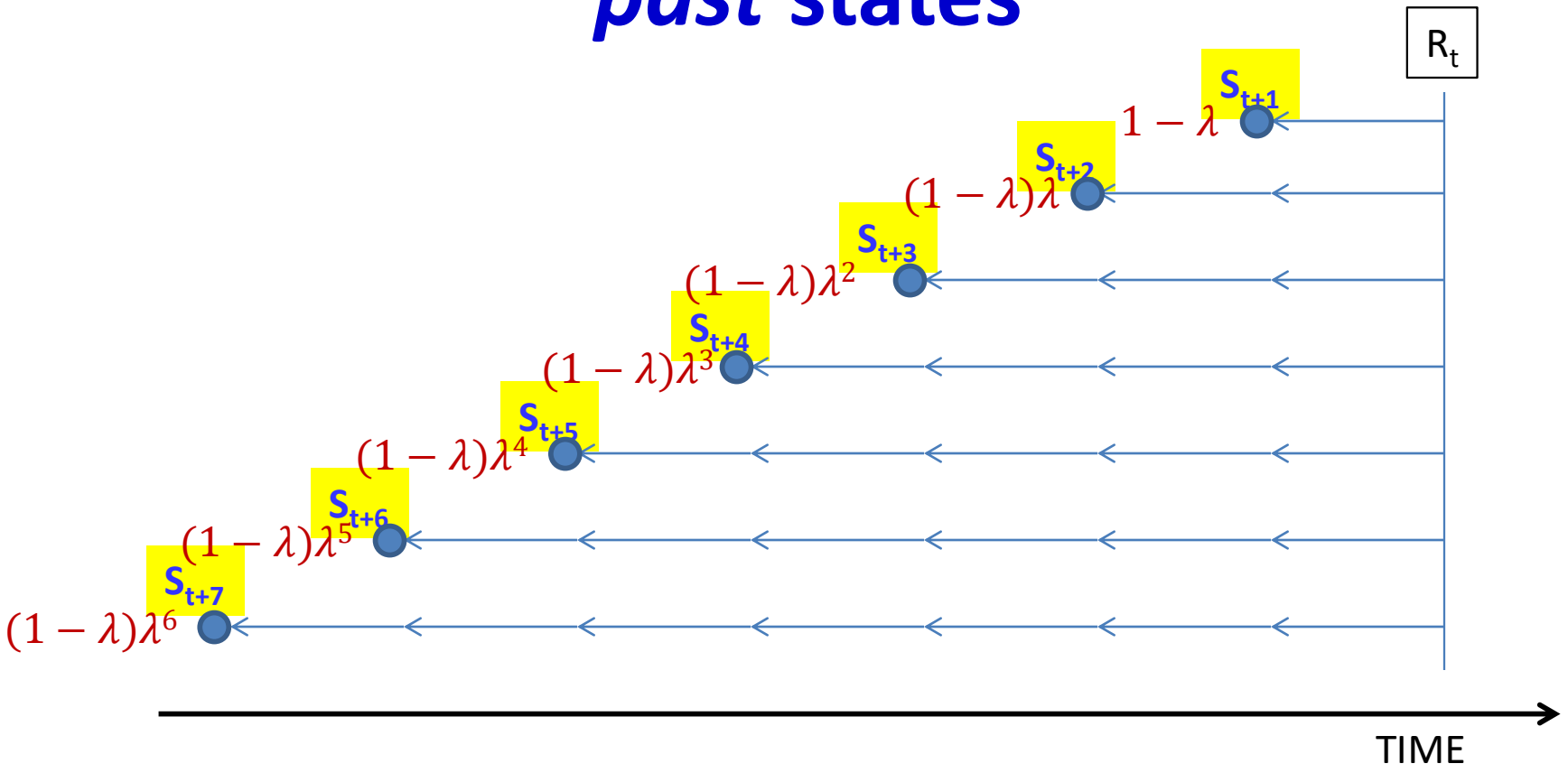


The contribution of future rewards to the present update



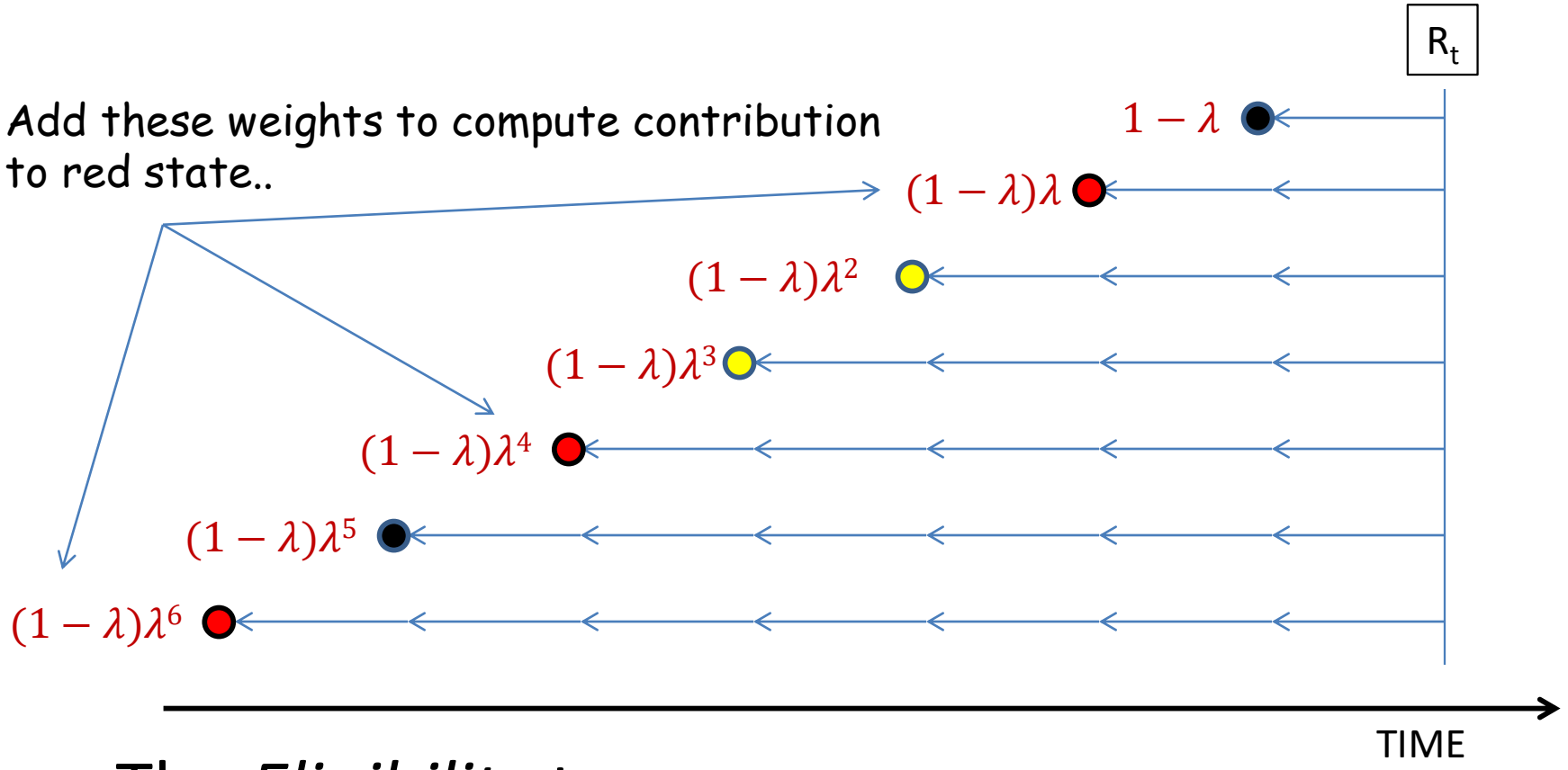
- All future rewards contribute to the update of the value of the current state

The contribution of current reward to *past* states



- All current reward contributes to the update of the value of all past states!

TD(λ) backward view



- The *Eligibility* trace:
  - Keeps track of *total* weight for any state
    - Which may have occurred at multiple times in the past

TD(λ)

- Maintain an eligibility trace for *every* state

$$E_0(s) = 0$$
$$E_t(s) = \gamma\lambda E_{t-1}(s) + \mathbf{1}(S_t = s)$$

- Computes total weight for the state until the present time

TD(λ)

- At every time, update the value of *every state* according to its eligibility trace

$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$$

$$V(s) \leftarrow V(s) + \alpha \delta_t E_t(s)$$

- Any state that was visited will be updated
  - Those that were not will not be, though

The magic of TD(λ)

- Managed to get the effect of inifinite lookahead, by performing infinite *lookbehind*
  - Or at least look behind to the beginning

- Every reward updates the value of *all states* leading to the reward!
  - E.g., in a chess game, if we win, we want to increase the value of all game states we visited, not just the final move
  - But early states/moves must gain much less than later moves

- When $\lambda = 1$ this is exactly equivalent to MC

Story so far

- Want to compute the *values* of all states, given a policy, but no knowledge of dynamics

- Have seen monte-carlo and temporal difference solutions
  - TD is quicker to update, and in many situations the better soluton
  - TD(λ) actually emulates an infinite lookahead
    - But we must choose good values of α and λ

Optimal Policy: Control

- We learned how to estimate the state value functions for an MDP whose transition probabilities are unknown ***for a given policy***

- *How do we find the optimal policy?*

Value vs. Action Value

- The solution we saw so far only computes the *value functions* of states

- Not sufficient – to compute the optimal policy from value functions alone, we will need extra information, namely transition probabilities
  - Which we do not have

- Instead, we can use the same method to compute *action value* functions
  - Optimal policy in any state : Choose the action that has the largest *optimal* action value

Value vs. Action value

- Given only value functions, the optimal policy must be estimated as:

$$\pi'(s) = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, \mathcal{R}_s^a + \mathcal{P}_{ss'}^a V(s')$$

  - Needs knowledge of transition probabilities
- Given action value functions, we can find it as:

$$\pi'(s) = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, Q(s, a)$$

- This is *model free* (no need for knowledge of model parameters)

Problem of optimal control

- From a series of episodes of the kind:

$$S_1, A_1, R_2, S_2, A_2, R_3, S_3, A_3, R_4, \ldots, S_T$$

- Find the optimal action value function $q_*(s, a)$
  - The optimal policy can be found from it

- Ideally do this *online*
  - So that we can continuously improve our policy from *ongoing experience*

Exploration vs. Exploitation

- Optimal policy search happens while gathering experience *while following a policy*
- For fastest learning, we will follow an estimate of the optimal policy
- Risk: We run the risk of positive feedback
  - Only learn to evaluate our current policy
  - Will never learn about alternate policies that may turn out to be better
- Solution: We will follow our current optimal policy $1 - \epsilon$ of the time
  - But choose a random action ϵ of the time
  - The “epsilon-greedy” policy

GLIE Monte Carlo

- Greedy in the limit with infinite exploration
- Start with some random initial policy π
- Start the process at the initial state, and follow an action according to initial policy π
- Produce the episode

$$S_1, A_1, R_2, S_2, A_2, R_3, S_3, A_3, R_4, \dots, S_T$$

- Process the episode using the following online update rules:

$$N(S_t, A_t) \leftarrow N(S_t, A_t) + 1$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{N(S_t, A_t)} (G_t - Q(S_t, A_t))$$

- Compute the ϵ-greedy policy for each state

$$\pi(a|s) = \begin{cases} 1 - \epsilon & for\ a = arg\max_{a'} Q(s, a') \\ \dfrac{\epsilon}{N_a - 1} & otherwise \end{cases}$$

- Repeat

GLIE Monte Carlo

- Greedy in the limit with infinite exploration
- Start with some random initial policy π
- Start the process at the initial state, and follow an action according to initial policy π
- Produce the episode

$$S_1, A_1, R_2, S_2, A_2, R_3, S_3, A_3, R_4, \dots, S_T$$

- Process the episode using the following online update rules:

$$N(S_t, A_t) \leftarrow N(S_t, A_t) + 1$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{N(S_t, A_t)} (G_t - Q(S_t, A_t))$$

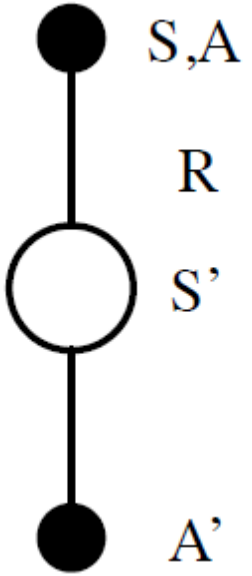
- Compute the ϵ-greedy policy for each state

$$\pi(a|s) = \begin{cases} 1 - \epsilon & for\ a = arg\max_{a'} Q(s, a') \\ \dfrac{\epsilon}{N_a - 1} & otherwise \end{cases}$$

- Repeat

On-line version of GLIE: SARSA

- Replace G_t with an estimate
- TD(1) or TD(λ)
  - Just as in the prediction problem



- **TD(1) → SARSA**

$$Q(S,A) \leftarrow Q(S,A) + \alpha\left(R + \gamma Q(S',A') - Q(S,A)\right)$$

SARSA

- Initialize $Q(s, a)$ for all s, a
- Start at initial state S_1
- Select an initial action A_1
- For t = 1.. Terminate
  - Get reward R_t
  - Let system transition to new state S_{t+1}
  - Draw A_{t+1} according to ϵ -greedy policy

$$\pi(a|s) = \begin{cases} 1 - \epsilon & for\ a = arg\max_{a'} Q(s, a') \\ \dfrac{\epsilon}{N_a - 1} & otherwise \end{cases}$$

  - Update
  - $Q(S_t, A_t) = Q(S_t, A_t) + \alpha\big(R_t + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\big)$

SARSA(λ)

- Again, the TD(1) estimate can be replaced by a TD(λ) estimate
- Maintain an eligibility trace for every state-action pair:

$$E_0(s,a) = 0$$
$$E_t(s,a) = \gamma\lambda E_{t-1}(s,a) + \mathbf{1}(S_t = s, A_t = a)$$

- Update every state-action pair visited so far

$$\delta_t = R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)$$
$$Q(s,a) \leftarrow Q(s,a) + \alpha\delta_t E_t(s,a)$$

-

SARSA(λ)

- For all s, a initialize $\mathrm{Q}(s, a)$
- For each episode e
  - For all s, a initialize $E(s, a) = 0$
  - Initialize S_1, A_1
  - For $t = 1 \ldots Termination$
    - Observe R_{t+1}, S_{t+1}
    - Choose action A_{t+1} using policy obtained from Q
    - $\delta = R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)$
    - $E(S_t, A_t) \mathrel{+}= 1$
    - For all s, a
      - $Q(s, a) = Q(s, a) + \alpha \delta E(s, a)$
      - $E(s, a) = \gamma \lambda E(s, a)$

On-policy vs. Off-policy

- SARSA assumes you're following the same policy that you're learning
- Its possible to follow one policy, while learning from others
  - E.g. learning by observation
- The policy for learning is the whatif policy

$$S_1, A_1, R_2, S_2, A_2, R_3, S_3, A_3, R_4, \ldots, S_T$$
$$\hat{A}_2 \qquad \hat{A}_3$$

- Modifies learning rule

hypothetical

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha\big(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\big)$$

- to

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha\left(R_{t+1} + \gamma Q\big(S_{t+1}, \hat{A}_{t+1}\big) - Q(S_t, A_t)\right)$$

- Q will actually represent the action value function of the *hypothetical policy*

SARSA: Suboptimality

- SARSA: From any state-action (S,A), accept reward (R), transition to next state (S'), choose next action(A')

- Use TD rules to update:

$$\delta = R + \gamma Q(S', A') - Q(S', A')$$

$$\delta$$

- Problem: which policy do we use to choose A'

SARSA: Suboptimality

- SARSA: From any state-action (S,A), accept reward (R), transition to next state (S'), choose next action(A')

- Problem: which policy do we use to choose A'
- If we choose the *current judgment of the best action* at S' we will become too greedy
  - Never explore
- If we choose a *sub-optimal* policy to follow, we will never find the best policy

Solution: Off-policy learning

- The policy for learning is the whatif policy

$$S_1, A_1, R_2, S_2, A_2, R_3, S_3, A_3, R_4, \ldots, S_T$$

$$\hat{A}_2 \qquad \hat{A}_3 \quad \text{hypothetical}$$

- Use the *best* action for S_{t+1} as your hypothetical off-policy action
- But actually follow an *epsilon-greedy* action
  - The hypothetical action is guaranteed to be better than the one you actually took
  - But you still explore (non-greedy)

Q-Learning

- From any state-action pair S,A
  - Accept reward R
  - Transition to S'
  - Find the *best action* A' for S'
  - Use it to update Q(S,A)
  - But then *actually perform an epsilon-greedy action A''* from S'

Q-Learning (TD(1) version)

- For all s, a initialize $\mathrm{Q}(s, a)$
- For each episode e
  - Initialize S_1, A_1
  - For $t = 1 \ldots Termination$
    - Observe R_{t+1}, S_{t+1}
    - Choose action A_{t+1} at S_{t+1} using epsilon-greedy policy obtained from Q
    - Choose action $\hat{A}_{t+1}$ at S_{t+1} as $\hat{A}_{t+1} = \underset{a}{arg\max}\, Q(S_{t+1}, a)$
    - $\delta = R_{t+1} + \gamma Q\left(S_{t+1}, \hat{A}_{t+1}\right) - Q(S_t, A_t)$
    - $Q(S_t, A_t) = Q(S_t, A_t) + \alpha\delta$

Q-Learning (TD(λ) version)

- For all s, a initialize $\mathrm{Q}(s, a)$
- For each episode e
  - For all s, a initialize $E(s, a) = 0$
  - Initialize S_1, A_1
  - For $t = 1 \ldots Termination$
    - Observe R_{t+1}, S_{t+1}
    - Choose action A_{t+1} at S_{t+1} using epsilon-greedy policy obtained from Q
    - Choose action $\hat{A}_{t+1}$ at S_{t+1} as $\hat{A}_{t+1} = arg\max_a Q(S_{t+1}, a)$
    - $\delta = R_{t+1} + \gamma Q\left(S_{t+1}, \hat{A}_{t+1}\right) - Q(S_t, A_t)$
    - $E(S_t, A_t) \mathrel{+}= 1$
    - For all s, a
      - $Q(s, a) = Q(s, a) + \alpha\delta E(s, a)$
      - $E(s, a) = \gamma\lambda E(s, a)$

What about the actual policy?

- Optimal greedy policy:

$$\pi(a|s) = \begin{cases} 1 \quad for\ a = \underset{a'}{argmax}\, Q(s,a') \\ 0\ otherwise \end{cases}$$

- Exploration policy

$$\pi(a|s) = \begin{cases} 1-\epsilon \quad for\ a = \underset{a'}{argmax}\, Q(s,a') \\ \dfrac{\epsilon}{N_a - 1}\ otherwise \end{cases}$$

- Ideally ϵ should decrease with time

Q-Learning

- Currently most-popular RL algorithm
- Topics not covered:
  - Value function approximation
  - Continuous state spaces
  - Deep-Q learning
  - Action replay
  - Application to real problem..