

# Markov Decision Processes and Reinforcement Learning

Readings:

- Mitchell, chapter 13
- for much more, see [Reinforcement Learning, an Introduction](#) by Sutton and Barto on our class website.

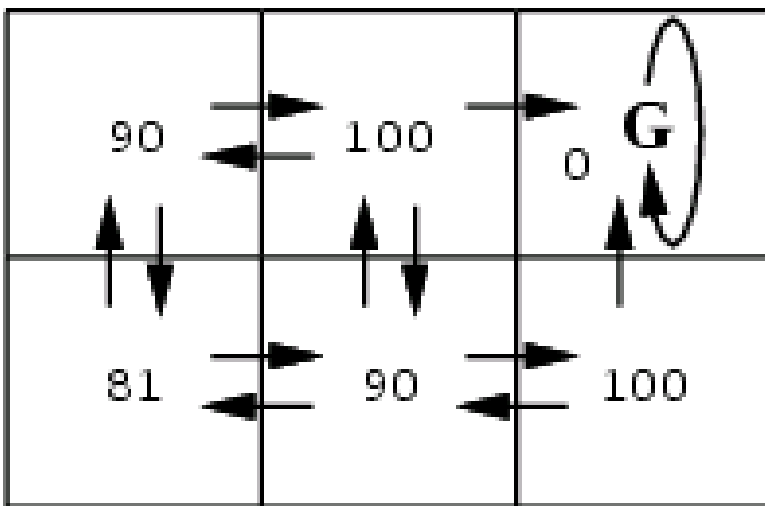
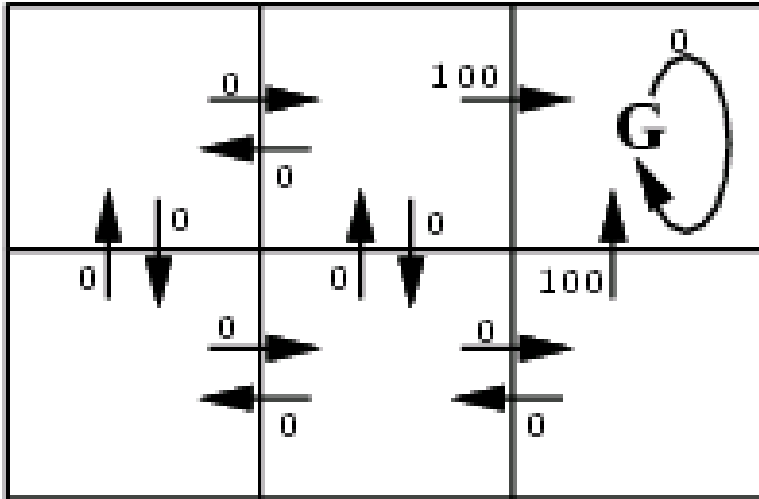
Machine Learning 10-601

April 21, 2008

Tom M. Mitchell  
Machine Learning Department  
Carnegie Mellon University

# Reinforcement Learning

[Sutton and Barto 1981; Samuel 1957; ...]

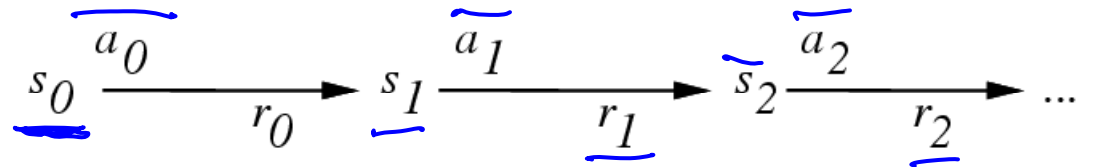
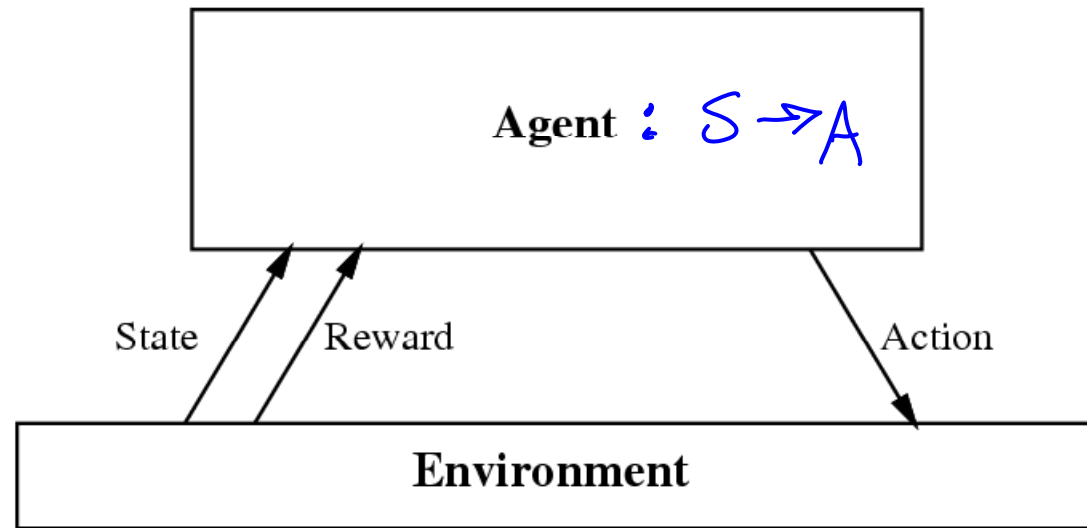


$$V^*(s) = E[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots]$$

# Outline

- Learning control strategies
  - Credit assignment and delayed reward
  - Discounted rewards
- Markov Decision Processes
  - Solving a known MDP
- Online learning of control strategies
  - When next-state function is known: value function  $V^*(s)$
  - When next-state function unknown: learning  $Q^*(s,a)$
- Role in modeling reward learning in animals

# Reinforcement Learning Problem



Goal: Learn to choose actions that maximize

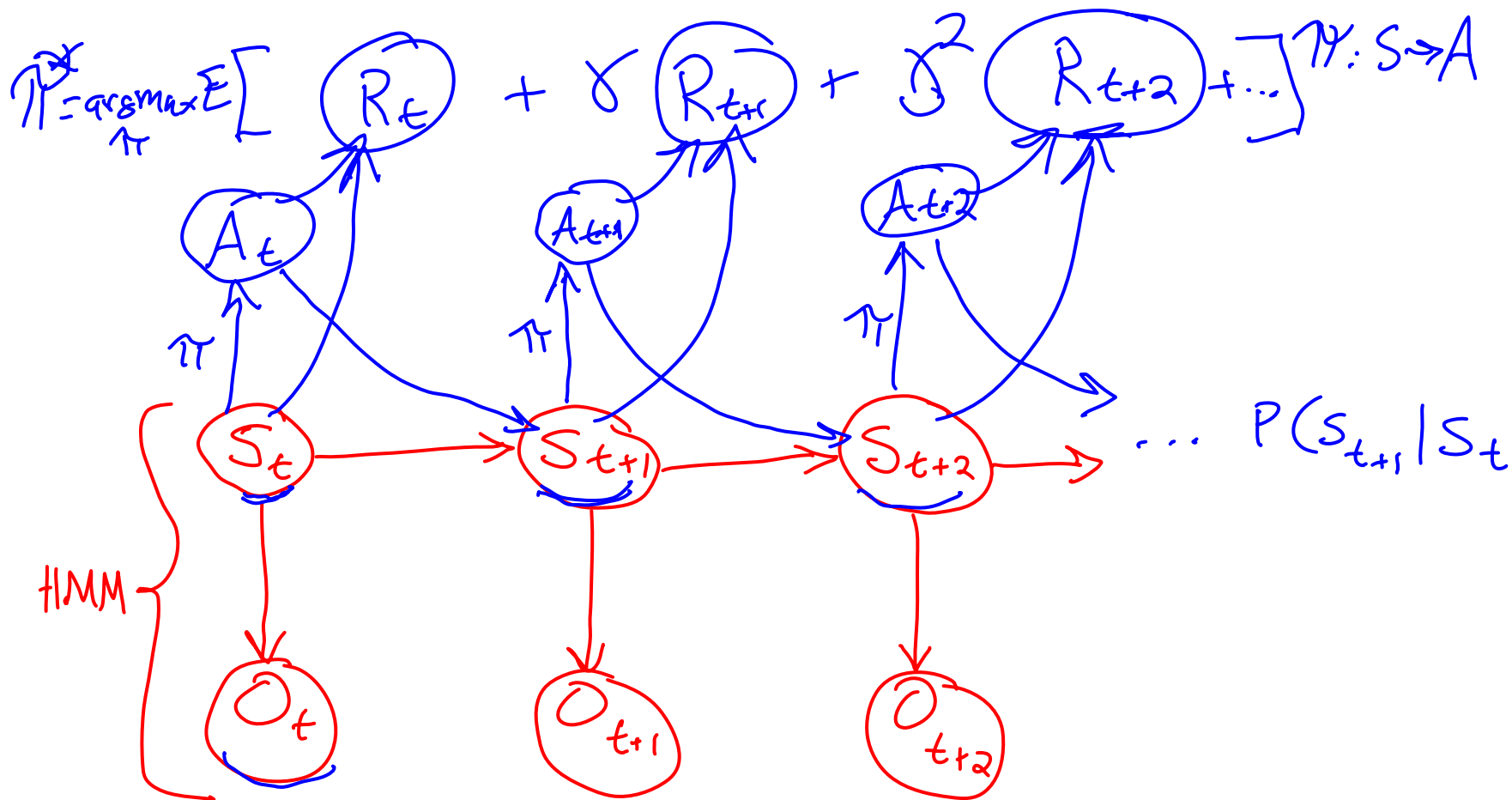
$$r_0 + \gamma r_1 + \gamma^2 r_2 + \dots, \text{ where } 0 \leq \gamma < 1$$

# Markov Decision Process

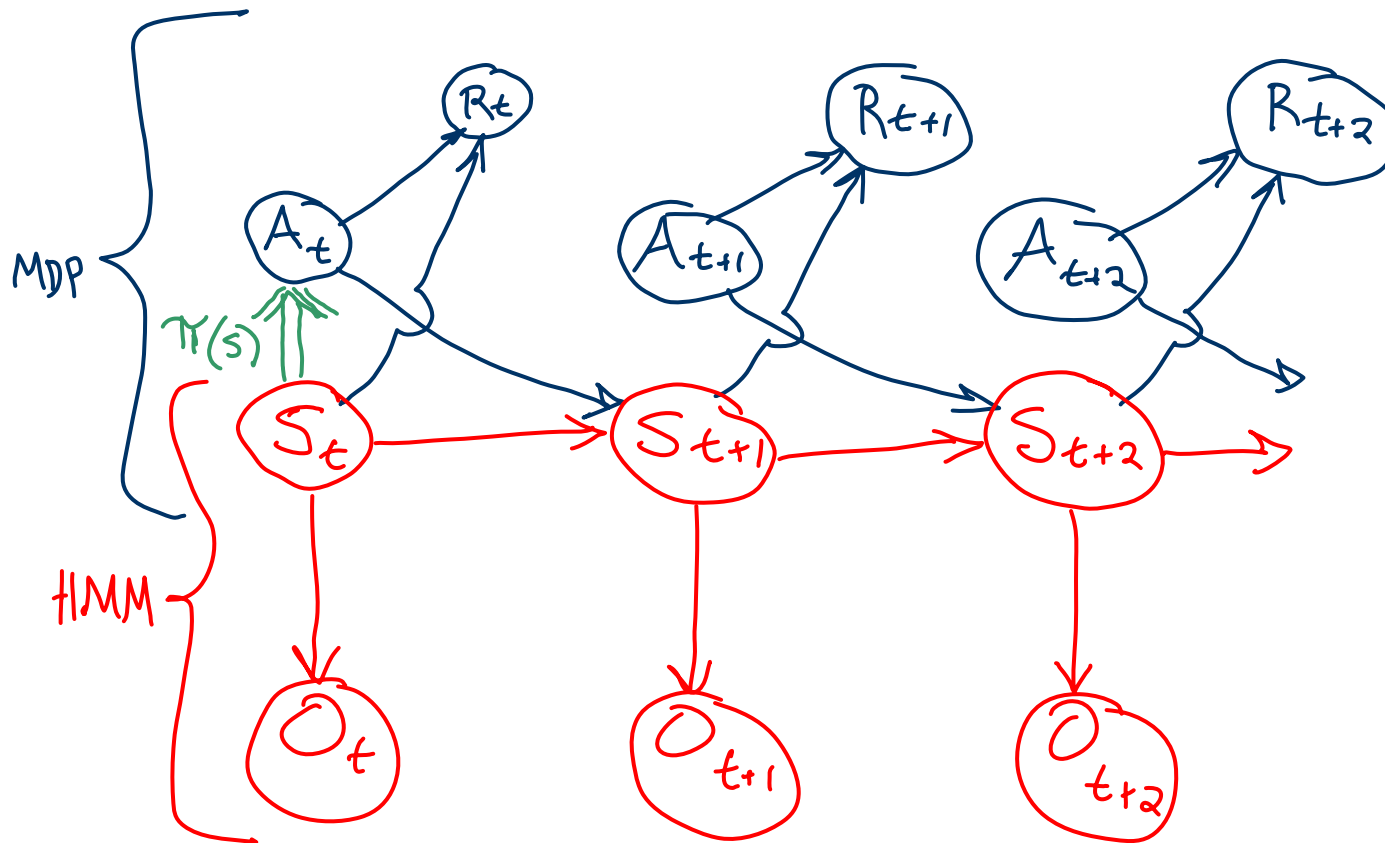
- Set of states  $S$
- Set of actions  $A$
- At each time, agent observes state  $s_t \in S$ , then chooses action  $a_t \in A$
- Then receives reward  $r_t$ , and state changes to  $s_{t+1}$
- Markov assumption:  $P(s_{t+1} | s_t, a_t, s_{t-1}, a_{t-1}, \dots) = P(s_{t+1} | s_t, a_t)$
- Also assume  $P(r_t | s_t, a_t, s_{t-1}, a_{t-1}, \dots) = P(r_t | s_t, a_t)$
- The task: learn a policy  $\pi: S \rightarrow A$  for choosing actions that maximizes  $E[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots]$

$$0 < \gamma \leq 1$$

# HMM, Markov Process, Markov Decision Process



# HMM, Markov Process, Markov Decision Process



# Reinforcement Learning Task for Autonomous Agent

Execute actions in environment, observe results, and

- Learn control policy  $\pi: S \rightarrow A$  that maximizes  $\sum_{t=0}^{\infty} \gamma^t E[r_t]$  from every state  $s \in S$
- Where  $0 < \lambda < 1$  is the discount factor for future rewards

Note:

- Function to be learned is  $\pi: S \rightarrow A$
- But training examples of the form  $\langle \langle s, a \rangle, r \rangle$
- $\rightarrow$  available training experience is not input-output pairs of the function to be learned !

# Value Function for each Policy

- Given a policy  $\pi : S \rightarrow A$ , define

$$V^\pi(s) = E\left[\sum_{t=0}^{\infty} \gamma^t r_t\right] \quad \text{assuming actions are chosen according to } \pi$$

- Then we want the policy  $\pi^*$  where

$$\pi^* = \arg \max_{\pi} V^\pi(s), \quad (\forall s)$$

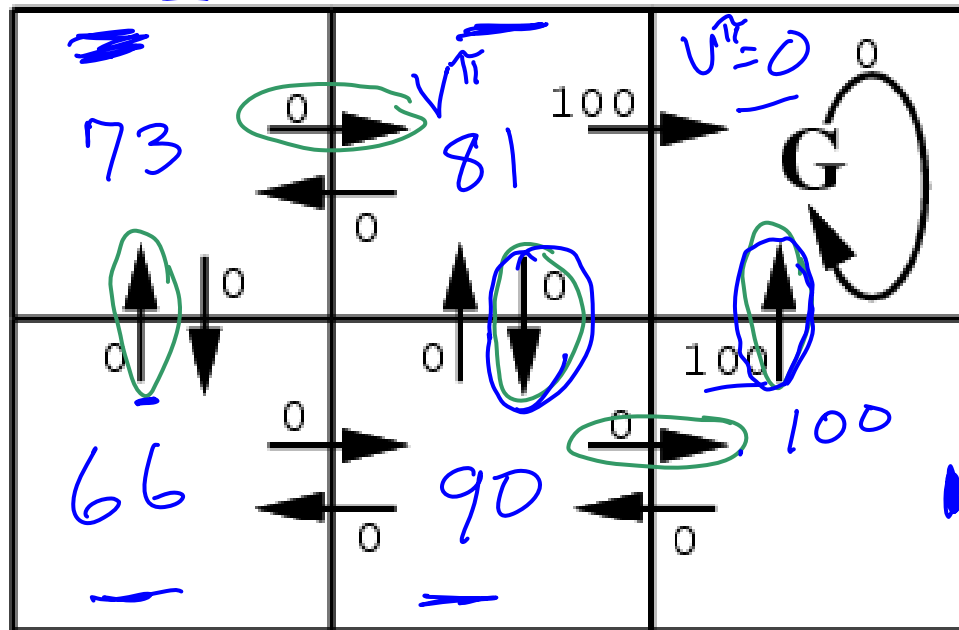
- For any MDP, such a policy exists
- We'll write  $V^*(s) = V^{\pi^*}(s)$
- Note if we have  $V^*(s)$  and  $P(s_{t+1}|s_t, a)$ , we can compute  $\pi^*(s)$

# Value Function – what are the $V^\pi(s)$ values?

$$V^\pi(s) = E\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$$

$$V^\pi_t(s) = r(s, \pi(s)) + \gamma V^\pi(\pi(s))$$

Suppose  $\pi$  is shown by circled action from each state  
 Suppose  $\gamma = 0.9$



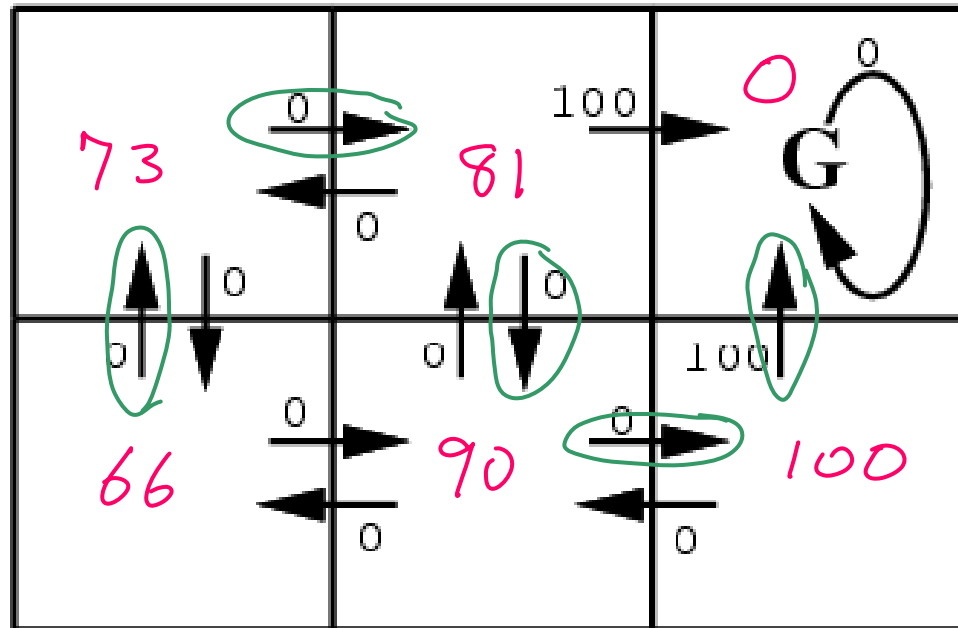
$$0 + \gamma 0 + \gamma^2 100$$

$r(s, a)$  (immediate reward)

# Value Function – what are the $V^\pi(s)$ values?

$$V^\pi(s) = E\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$$

Suppose  $\pi$  is shown by circled action from each state  
 Suppose  $\gamma = 0.9$

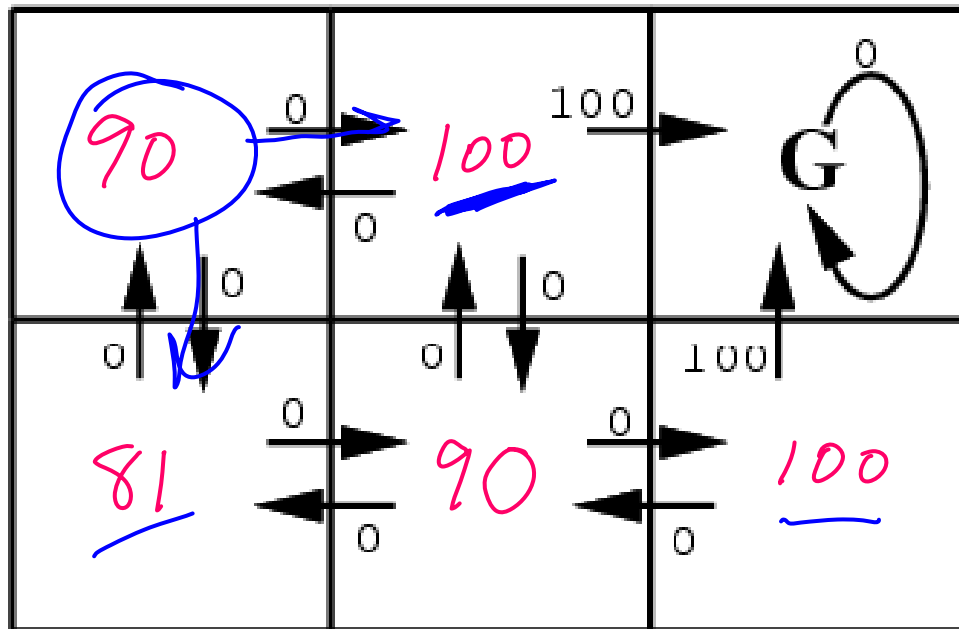


$r(s, a)$  (immediate reward)

# Value Function – what are the $V^*(s)$ values?

$$V^\pi(s) = E\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$$

$V^{\pi^*} \equiv V^*$   
 ↑  
 optimal policy

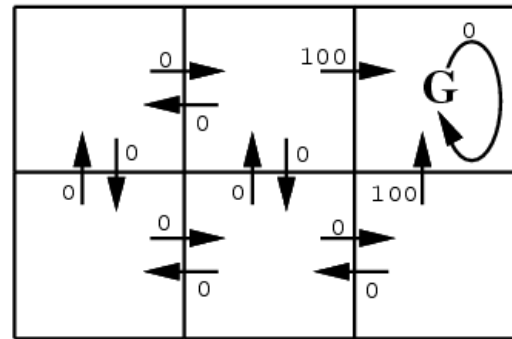


$r(s, a)$  (immediate reward)

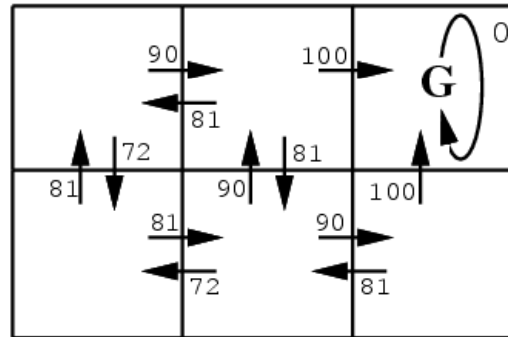
Immediate rewards  $r(s,a)$

State values  $V^*(s)$

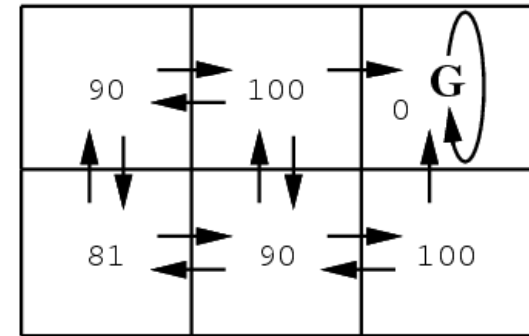
State-action values  $Q^*(s,a)$



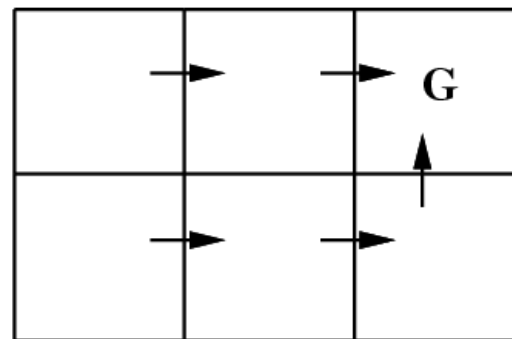
$r(s, a)$  (immediate reward) values



$Q(s, a)$  values



$V^*(s)$  values



One optimal policy

## Recursive definition for $V^*(S)$

$$V^*(s) = E\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$$

assuming actions are  
chosen according to the  
optimal policy,  $\pi^*$

$$V^*(s_1) = E[r(s_1, a_1)] + E[\gamma r(s_2, a_2)] + E[\gamma^2 r(s_3, a_3)] + \dots]$$

$$V^*(s_1) = E[r(s_1, a_1)] + \gamma E_{s_2|s_1, a_1}[V^*(s_2)]$$

$$V^*(s) = E[r(s, \pi^*(s))] + \gamma E_{P(s'|s, \pi^*(s))}[V^*(s')]$$

# Value Iteration for learning $V^*$ : assumes $P(S_{t+1}|S_t, A)$ known

Initialize  $V(s)$  arbitrarily *← to zeros*

Loop until policy good enough

Loop for  $s$  in  $S$

Loop for  $a$  in  $A$

$$Q(s, a) \leftarrow r(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) V(s')$$

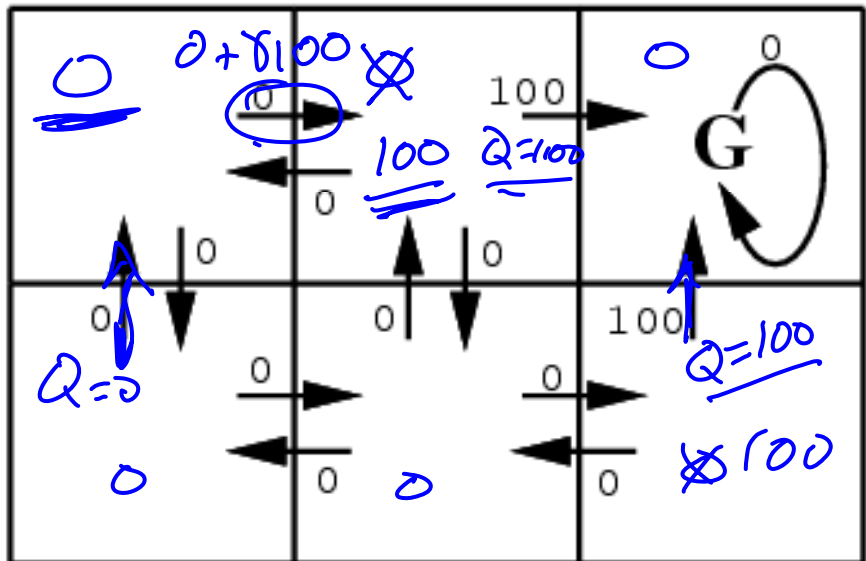
$$V(s) \leftarrow \max_a Q(s, a)$$

End loop

End loop

$V(s)$  converges to  $V^*(s)$ .

Same alg works if we randomly traverse the environment, as long as visit every transition repeatedly




# Value Iteration

Interestingly, value iteration works even if we randomly traverse the environment instead of looping through each state and action methodically

- but we must still visit each state infinitely often on an infinite run
- For details: [Bertsekas 1989]
- Implications: online learning as agent randomly roams

If max (over states) difference between two successive value function estimates is less than  $\epsilon$ , then the value of the greedy policy differs from the optimal policy by no more than

$$2\epsilon\gamma/(1 - \gamma)$$



So far: learning optimal policy when we  
know  $P(s_t | s_{t-1}, a_{t-1})$

What if we don't?

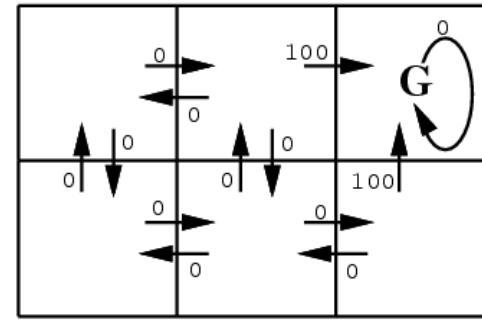
Immediate rewards  $r(s,a)$

State values  $V^*(s)$

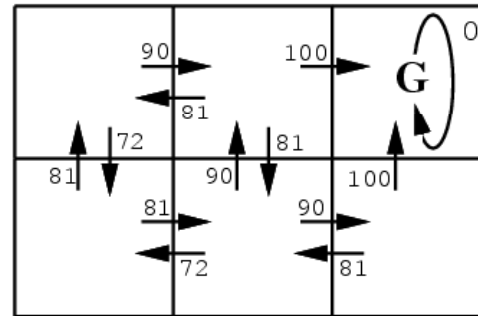
State-action values  $Q^*(s,a)$

$$V^*(s) = E[r(s, \pi^*(s))] + \gamma E_{s'|s, \pi^*(s)}[V^*(s')]$$

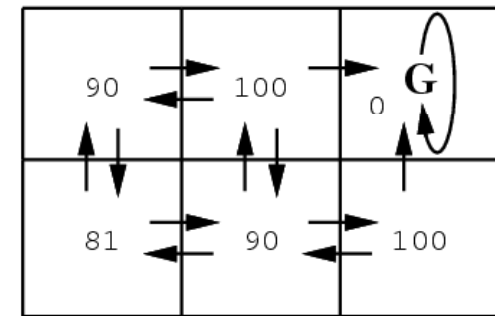
Bellman equation.



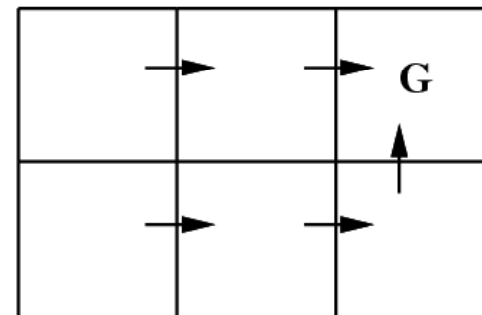
$r(s, a)$  (immediate reward) values



$Q(s, a)$  values



$V^*(s)$  values



One optimal policy

# Q learning

Define new function, closely related to  $V^*$

$$V^*(s) = E[r(s, \pi^*(s))] + \gamma E_{s'|s, \pi^*(s)}[V^*(s')]$$

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

If agent knows  $Q(s, a)$ , it can choose optimal action without knowing  $P(s_{t+1}|s_t, a)$  !

$$\pi^*(s) = \arg \max_a Q(s, a) \quad V^*(s) = \max_a Q(s, a)$$

And, it can learn  $Q$  without knowing  $P(s_{t+1}|s_t, a)$

## Q Function

Consider first the deterministic case.  $P(s'|s,a)$  deterministic, denoted  $\delta(s,a)$

Define new function very similar to  $V^*$

$$Q(s, a) \equiv r(s, a) + \gamma V^*(\delta(s, a))$$

If agent learns  $Q$ , it can choose optimal action even without knowing  $\delta$ !

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

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

$Q$  is the evaluation function the agent will learn

# Training Rule to Learn $Q$

---

Note  $Q$  and  $V^*$  closely related:

$$V^*(s) = \max_{a'} Q(s, a')$$

Which allows us to write  $Q$  recursively as

$$\begin{aligned} Q(s_t, a_t) &= r(s_t, a_t) + \gamma V^*(\delta(s_t, a_t)) \\ &= r(s_t, a_t) + \gamma \max_{a'} Q(s_{t+1}, a') \end{aligned}$$

Nice! Let  $\hat{Q}$  denote learner's current approximation to  $Q$ . Consider training rule

$$\hat{Q}(s, a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s', a')$$

where  $s'$  is the state resulting from applying action  $a$  in state  $s$

# Q Learning for Deterministic Worlds

---

For each  $s, a$  initialize table entry  $\hat{Q}(s, a) \leftarrow 0$

Observe current state  $s$

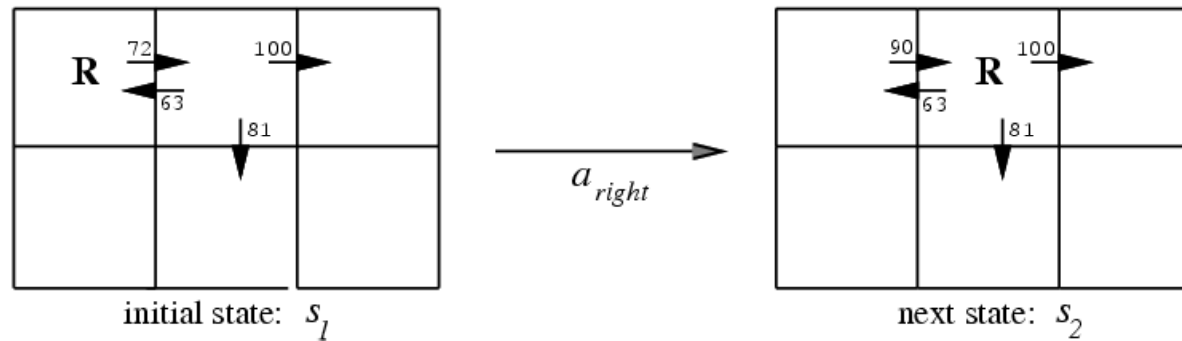
Do forever:

- Select an action  $a$  and execute it
- Receive immediate reward  $r$
- Observe the new state  $s'$
- Update the table entry for  $\hat{Q}(s, a)$  as follows:

$$\hat{Q}(s, a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s', a')$$

- $s \leftarrow s'$

# Updating $\hat{Q}$



$$\begin{aligned}\hat{Q}(s_1, a_{right}) &\leftarrow r + \gamma \max_{a'} \hat{Q}(s_2, a') \\ &\leftarrow 0 + 0.9 \max\{63, 81, 100\} \\ &\leftarrow 90\end{aligned}$$

notice if rewards non-negative, then

$$(\forall s, a, n) \quad \hat{Q}_{n+1}(s, a) \geq \hat{Q}_n(s, a)$$

and

$$(\forall s, a, n) \quad 0 \leq \hat{Q}_n(s, a) \leq Q(s, a)$$

$\hat{Q}$  converges to  $Q$ . Consider case of deterministic world where see each  $\langle s, a \rangle$  visited infinitely often.

*Proof:* Define a full interval to be an interval during which each  $\langle s, a \rangle$  is visited. During each full interval the largest error in  $\hat{Q}$  table is reduced by factor of  $\gamma$

Let  $\hat{Q}_n$  be table after  $n$  updates, and  $\Delta_n$  be the maximum error in  $\hat{Q}_n$ ; that is

$$\Delta_n = \max_{s,a} |\hat{Q}_n(s, a) - Q(s, a)|$$

For any table entry  $\hat{Q}_n(s, a)$  updated on iteration  $n + 1$ , the error in the revised estimate  $\hat{Q}_{n+1}(s, a)$  is

$$\begin{aligned} |\hat{Q}_{n+1}(s, a) - Q(s, a)| &= |(r + \gamma \max_{a'} \hat{Q}_n(s', a')) \\ &\quad - (r + \gamma \max_{a'} Q(s', a'))| \\ &= \gamma |\max_{a'} \hat{Q}_n(s', a') - \max_{a'} Q(s', a')| \\ &\leq \gamma \max_{a'} |\hat{Q}_n(s', a') - Q(s', a')| \\ &\leq \gamma \max_{s'', a'} |\hat{Q}_n(s'', a') - Q(s'', a')| \end{aligned}$$

$$|\hat{Q}_{n+1}(s, a) - Q(s, a)| \leq \gamma \Delta_n$$

Use general fact:

$$\begin{aligned} |\max_a f_1(a) - \max_a f_2(a)| &\leq \\ &\max_a |f_1(a) - f_2(a)| \end{aligned}$$



# Nondeterministic Case

---

$Q$  learning generalizes to nondeterministic worlds

Alter training rule to

$$\hat{Q}_n(s, a) \leftarrow (1 - \alpha_n)\hat{Q}_{n-1}(s, a) + \alpha_n[r + \max_{a'} \hat{Q}_{n-1}(s', a')]$$

where

$$\alpha_n = \frac{1}{1 + \text{visits}_n(s, a)}$$

Can still prove convergence of  $\hat{Q}$  to  $Q$  [Watkins and Dayan, 1992]

# Temporal Difference Learning

---

$Q$  learning: reduce discrepancy between successive  $Q$  estimates

One step time difference:

$$Q^{(1)}(s_t, a_t) \equiv r_t + \gamma \max_a \hat{Q}(s_{t+1}, a)$$

Why not two steps?

$$Q^{(2)}(s_t, a_t) \equiv r_t + \gamma r_{t+1} + \gamma^2 \max_a \hat{Q}(s_{t+2}, a)$$

Or  $n$ ?

$$Q^{(n)}(s_t, a_t) \equiv r_t + \gamma r_{t+1} + \dots + \gamma^{(n-1)} r_{t+n-1} + \gamma^n \max_a \hat{Q}(s_{t+n}, a)$$

Blend all of these:

$$Q^\lambda(s_t, a_t) \equiv (1-\lambda) [Q^{(1)}(s_t, a_t) + \lambda Q^{(2)}(s_t, a_t) + \lambda^2 Q^{(3)}(s_t, a_t) + \dots]$$

# Temporal Difference Learning

---

$$Q^\lambda(s_t, a_t) \equiv (1-\lambda) [Q^{(1)}(s_t, a_t) + \lambda Q^{(2)}(s_t, a_t) + \lambda^2 Q^{(3)}(s_t, a_t) + \dots]$$

Equivalent expression:

$$Q^\lambda(s_t, a_t) = r_t + \gamma [ (1 - \lambda) \max_a \hat{Q}(s_t, a_t) + \lambda Q^\lambda(s_{t+1}, a_{t+1}) ]$$

TD( $\lambda$ ) algorithm uses above training rule

- Sometimes converges faster than  $Q$  learning
- converges for learning  $V^*$  for any  $0 \leq \lambda \leq 1$  (Dayan, 1992)
- Tesauro's TD-Gammon uses this algorithm

# Subtleties and Ongoing Research

---

- Replace  $\hat{Q}$  table with neural net or other generalizer
- Handle case where state only partially observable
- Design optimal exploration strategies
- Extend to continuous action, state
- Learn and use  $\hat{\delta} : S \times A \rightarrow S$
- Relationship to dynamic programming