Reinforcement Learning

www.biostat.wisc.edu/~dpage/cs760/

Goals for the lecture

you should understand the following concepts

- the reinforcement learning task
- Markov decision process
- value functions
- value iteration
- Q functions
- Q learning
- exploration vs. exploitation tradeoff
- compact representations of Q functions

Reinforcement learning (RL)

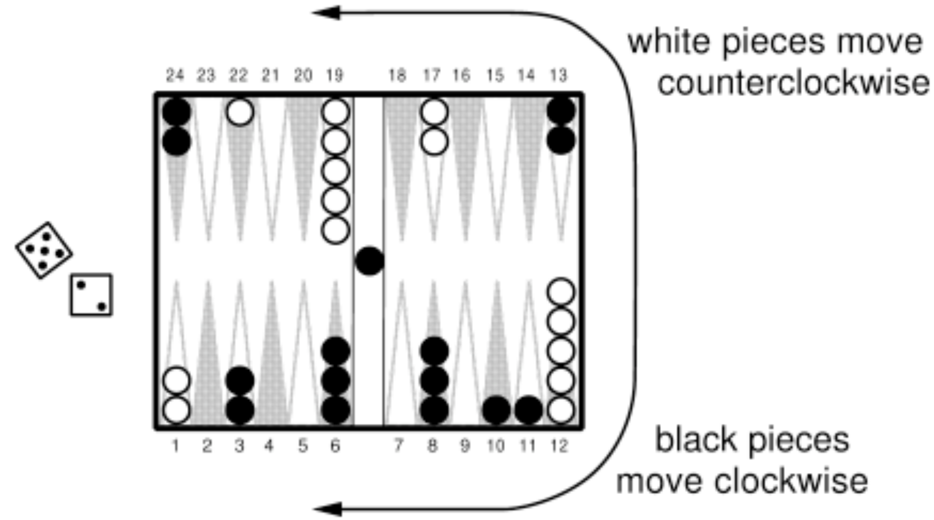
Task of an agent embedded in an environment

repeat forever

1) sense world
2) reason
3) choose an action to perform
4) get feedback (usually reward = 0)
5) learn

the environment may be the physical world or an artificial one

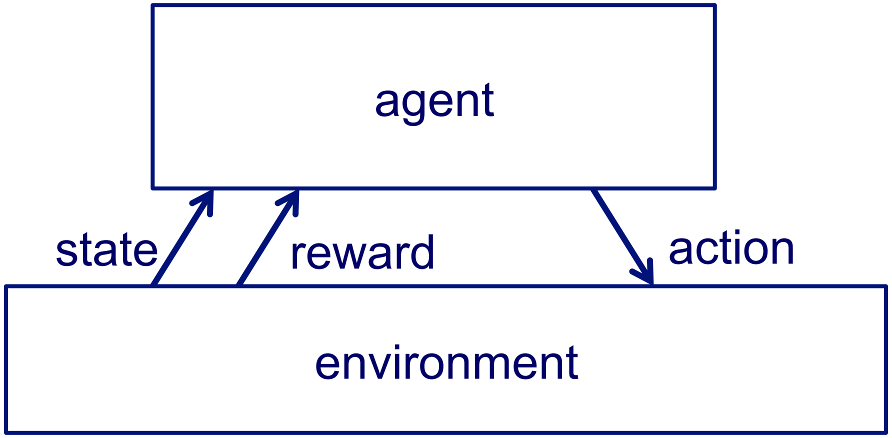
Example: RL Backgammon Player
[Tesauro, *CACM* 1995]



- world
  - 30 pieces, 24 locations
- actions
  - roll dice, e.g. 2, 5
  - move one piece 2
  - move one piece 5
- rewards
  - win, lose
- TD-Gammon 0.0
  - trained against itself (300,000 games)
  - as good as best previous BG computer program (also by Tesauro)
- TD-Gammon 2
  - beat human champion

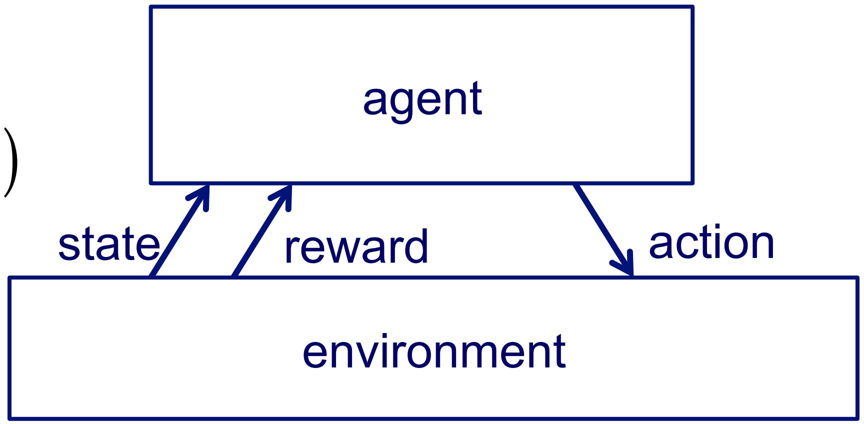
Reinforcement learning

- set of states S
- set of actions A
- at each time t, agent observes state $s_t \in S$ then chooses action $a_t \in A$
- then receives reward r_t and changes to state s_{t+1}



$$s_0 \xrightarrow[r_0]{a_0} s_1 \xrightarrow[r_1]{a_1} s_2 \xrightarrow[r_2]{a_2}$$

Reinforcement learning as a Markov decision process (MDP)



- Markov assumption

$$P\left(s_{t+1} \mid s_t,\ a_t,\ s_{t-1},\ a_{t-1},\ \ldots\right) = P\left(s_{t+1} \mid s_t,\ a_t\right)$$

- also assume reward is Markovian

$$P\left(r_t \mid s_t,\ a_t,\ s_{t-1},\ a_{t-1},\ \ldots\right) = P\left(r_t \mid s_t,\ a_t\right)$$

$$s_0 \xrightarrow[r_0]{a_0} s_1 \xrightarrow[r_1]{a_1} s_2 \xrightarrow[r_2]{a_2}$$

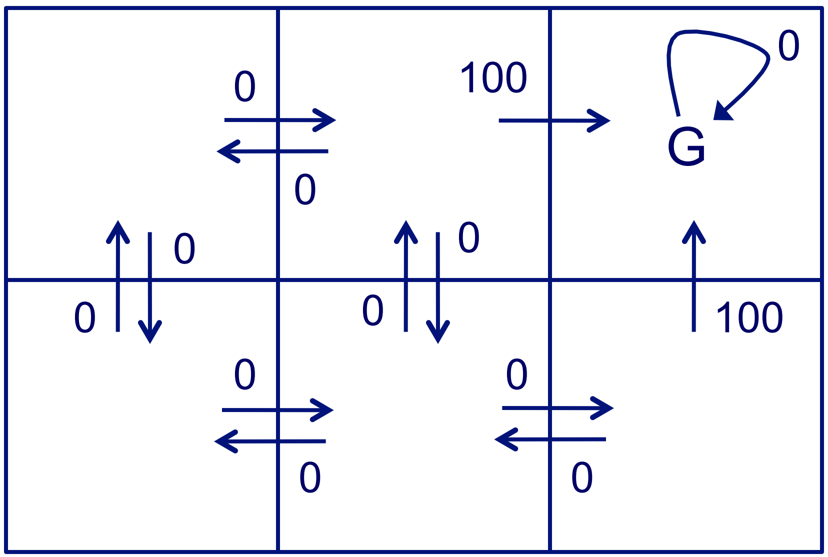
Goal: learn a policy $\pi : S \rightarrow A$ for choosing actions that maximizes

$$E\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots\right] \qquad \text{where } 0 \leq \gamma < 1$$

for every possible starting state s_0

Reinforcement learning task

- Suppose we want to learn a control policy $\pi : S \rightarrow A$ that maximizes $\sum_{t=0}^{\infty} \gamma^t E[r_t]$ from every state $s \in S$



each arrow represents an action a and the associated number represents deterministic reward $r(s, a)$

Value function for a policy

- given a policy $\pi : S \rightarrow A$ define

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

assuming action sequence chosen according to π starting at state s

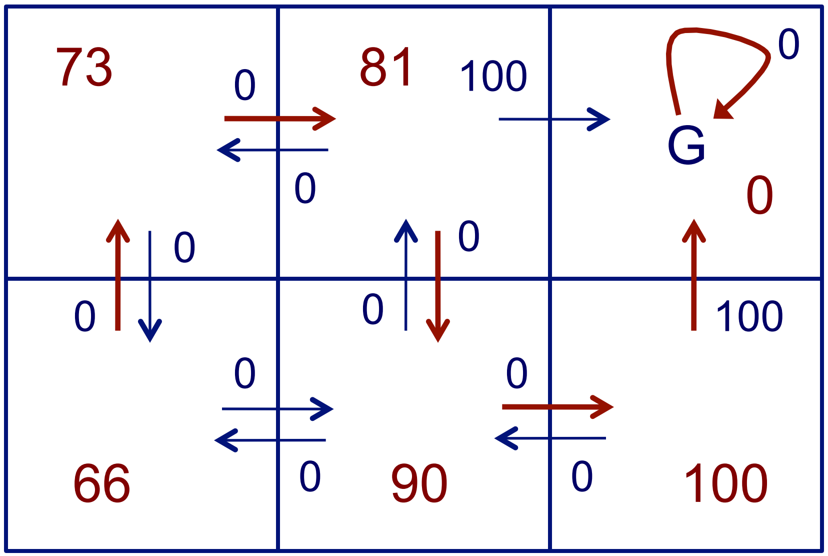
- we want the optimal policy π^* where

$$\pi^* = \arg\max_{\pi} V^{\pi}(s) \quad \text{for all } s$$

we'll denote the value function for this optimal policy as $V^*(s)$

Value function for a policy π

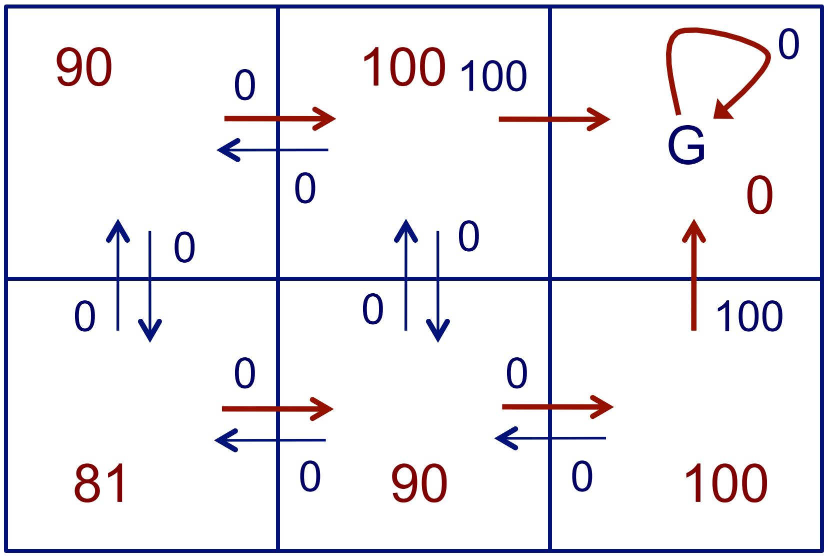
- Suppose π is shown by red arrows, $\gamma = 0.9$



$V^{\pi}(s)$ values are shown in red

Value function for an optimal policy π^*

- Suppose π^* is shown by red arrows, $\gamma = 0.9$



$V^*(s)$ values are shown in red

Using a value function

If we know $V^*(s)$, $r(s_t, a)$, and $P(s_t \mid s_{t-1}, a_{t-1})$
we can compute $\pi^*(s)$

$$\pi^*(s_t) = \arg\max_{a \in A} \left[r(s_t, a) + \gamma \sum_{s \in S} P(s_{t+1} = s \mid s_t, a) V^*(s) \right]$$

Value iteration for learning $V^*(s)$

initialize $V(s)$ arbitrarily

loop until policy good enough

{

    loop for $s \in S$

    {

        loop for $a \in A$

        {

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

        }

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

    }

}

Value iteration for learning $V^*(s)$

- $V(s)$ converges to $V^*(s)$
- works even if we randomly traverse environment instead of looping through each state and action methodically
  - but we must visit each state infinitely often
- implication: we can do online learning as an agent roams around its environment
- assumes we have a model of the world: i.e. know $P(\mathrm{s}_\mathrm{t} \mid s_{t-1}, a_{t-1})$
- What if we don't?

Q learning

define a new function, closely related to V^*

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

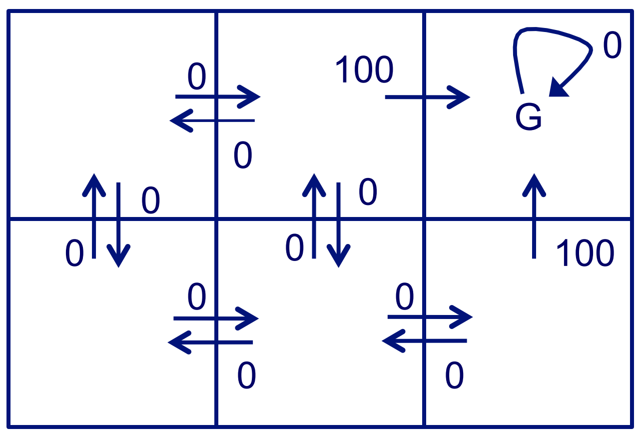
$$Q(s,a) = E\left[r(s,\ a)\right] + \gamma\, E_{s'\,|\,s,\,a}\left[V^*(s')\right]$$

if agent knows $Q(s, a)$, it can choose optimal action without knowing $P(s' \mid s, a)$

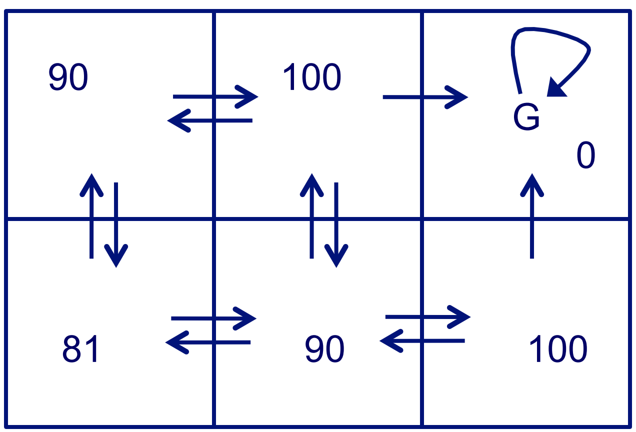
$$\pi^*(s) = \arg\max_a Q(s,a) \qquad\qquad V^*(s) = \max_a Q(s,a)$$

and it can learn $Q(s, a)$ without knowing $P(s' \mid s, a)$

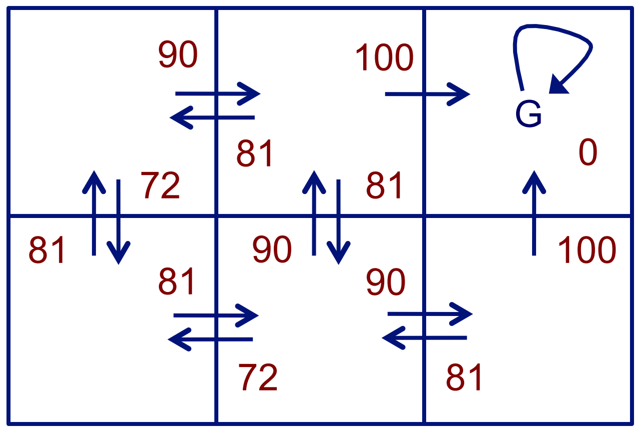
Q values



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



$V^*(s)$ values



$Q(s, a)$ values

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

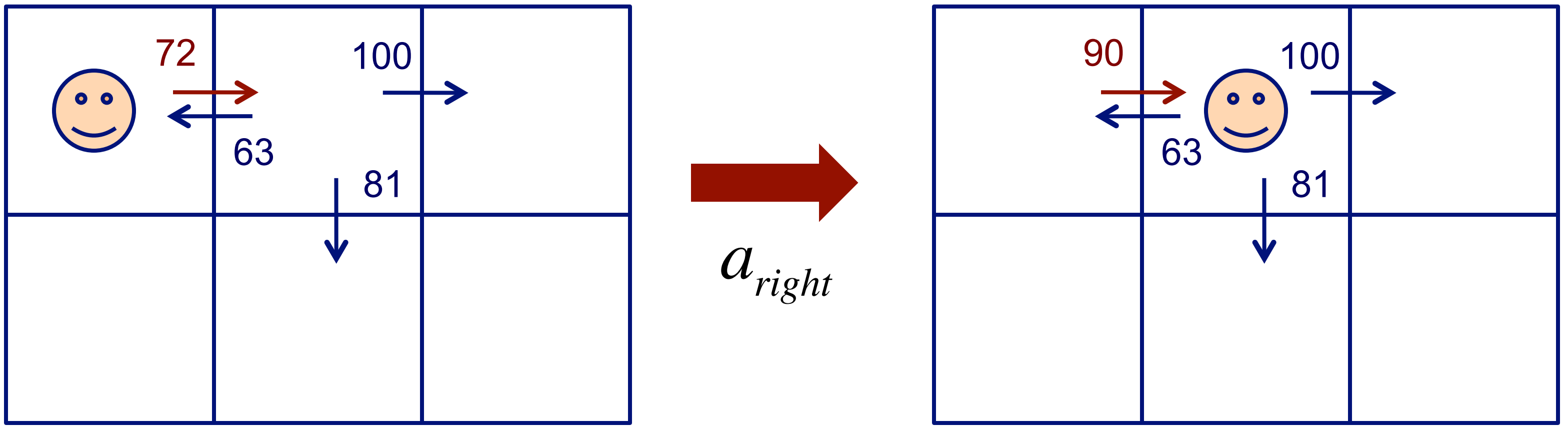
observe the new state s'

update table entry

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

$s \leftarrow s'$

Updating Q



$$\hat{Q}(s_1, a_{right}) \leftarrow r + \gamma \max_{a'} \hat{Q}(s_2, a')$$

$$\leftarrow 0 + 0.9 \max_{a'} \{63,\ 81,\ 100\}$$

$$\leftarrow 90$$

Q learning for *nondeterministic* 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 table entry

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

$s \leftarrow s'$

where α_n is a parameter dependent on the number of visits to the given (s, a) pair

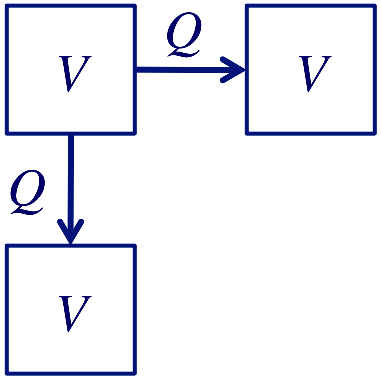
$$\alpha_n = \frac{1}{1 + \text{visits}_n(s,a)}$$

Convergence of Q learning

- Q learning will converge to the correct Q function
  - in the deterministic case
  - in the nondeterministic case (using the update rule just presented)

- in practice it is likely to take many, many iterations

Q's vs. V's



- Which action do we choose when we're in a given state?
- V's (model-based)
  - need to have a 'next state' function to generate all possible states
  - choose next state with highest V value.
- Q's (model-free)
  - need only know which actions are legal
  - generally choose next state with highest Q value.

Exploration vs. Exploitation

- in order to learn about better alternatives, we shouldn't always follow the current policy (exploitation)
- sometimes, we should select random actions (exploration)
- one way to do this: select actions probabilistically according to:

$$P(a_i \mid s) = \frac{c^{\hat{Q}(s, a_i)}}{\sum_j c^{\hat{Q}(s, a_j)}}$$

where $c > 0$ is a constant that determines how strongly selection favors actions with higher Q values

Q learning with a table

As described so far, Q learning entails filling in a huge table

states

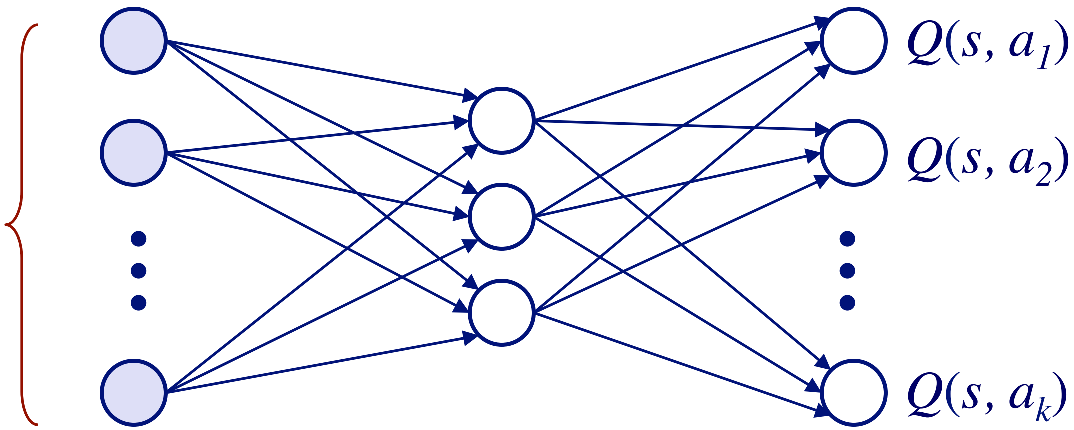
		s_0	s_1	s_2	. . .	s_n
	a_1			.		
	a_2			.		
actions	a_3	. . .		$Q(s_2, a_3)$		
	.					
	.					
	.					
	a_k					

A table is a very <u>verbose way to</u> represent a function

Representing Q functions more compactly

We can use some other function representation (e.g. a neural net) to compactly encode a substitute for the big table

encoding of the state (s)



each input unit encodes a property of the state (e.g., a sensor value)

or could have one net for each possible action

Why use a compact Q function?

1. Full Q table may not fit in memory for realistic problems
2. Can generalize across states, thereby speeding up convergence

   i.e. one instance 'fills' <u>many</u> cells in the Q table

<u>Notes</u>

1. When generalizing across states, cannot use $\alpha=1$
2. Convergence proofs only apply to Q <u>tables</u>
3. Some work on bounding errors caused by using compact representations (e.g. Singh & Yee, *Machine Learning* 1994)

Q tables vs. Q nets

Given: 100 Boolean-valued features
10 possible actions

Size of Q table

10×2^{100} entries

Size of Q net (assume 100 hidden units)

$$\underbrace{100 \times 100}_{\text{weights between inputs and HU's}} + \underbrace{100 \times 10}_{\text{weights between HU's and outputs}} = 11{,}000 \text{ weights}$$

Representing Q functions more compactly

- we can use other regression methods to represent Q functions

  k-NN

  regression trees

  support vector regression

  etc.

Q learning with function approximation

1. measure sensors, sense state s_0
2. predict $\hat{Q}_n(s_0, a)$ for each action a
3. select action a to take (with randomization to ensure exploration)
4. apply action a in the real world
5. sense new state s_1 and immediate reward r
6. calculate action a' that maximizes $\hat{Q}_n(s_1, a')$
7. train with new instance

$$\boldsymbol{x} = s_0$$

$$y = (1-\alpha)\hat{Q}(s_0, a) + \alpha\left[r + \gamma \max_{a'} \hat{Q}(s_1, a')\right]$$