Reinforcement Learning

Maria-Florina Balcan
Carnegie Mellon University

11/19/2018

Today:

- Learning of control policies
- Markov Decision Processes
- Temporal difference learning
- Q learning

Readings:

- Mitchell, chapter 13
- Kaelbling, et al., *Reinforcement Learning: A Survey*

Slides courtesy: Tom Mitchell

MACHINE LEARNING DEPARTMENT

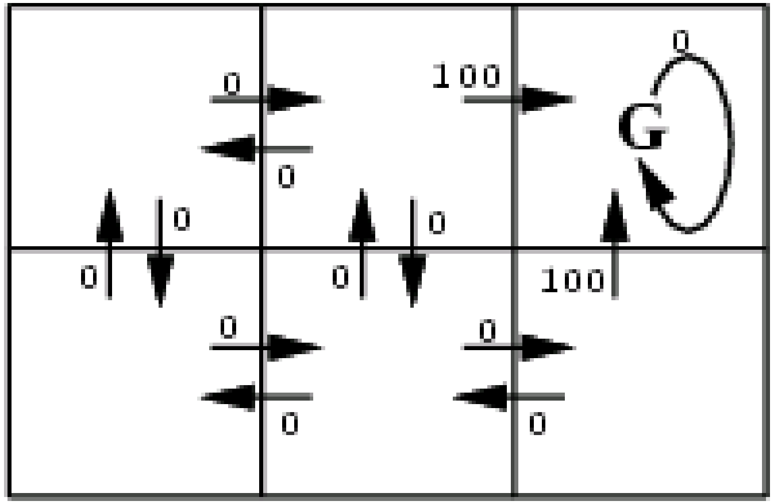
Overview

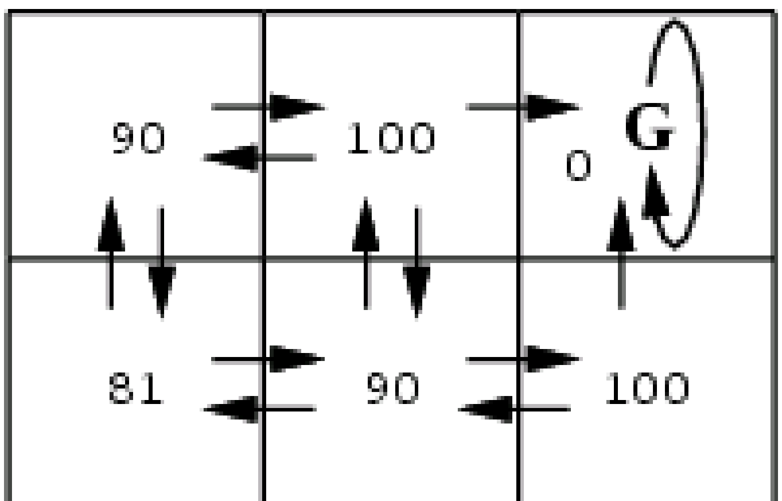
- Different from ML pbs so far:
  - Our decisions influence the next example we see. Decisions we make will be about actions to take (e.g., a robot deciding which way to move next), which will influence what we see next.
  - Goal will be not just to predict (say, whether there is a door in front of us or not) but to decide what to do.
- Model: Markov Decision Processes.

Reinforcement Learning

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

Main impact of our actions will not come right away but instead that will only come later.





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

Reinforcement Learning: Backgammon

[Tessauro, 1995]

Learning task:

- chose move at arbitrary board states

Training signal:

- final win or loss at the end of the game

Training:

- played 300,000 games against itself

Algorithm:

- reinforcement learning + neural network

Result:

- World-class Backgammon player

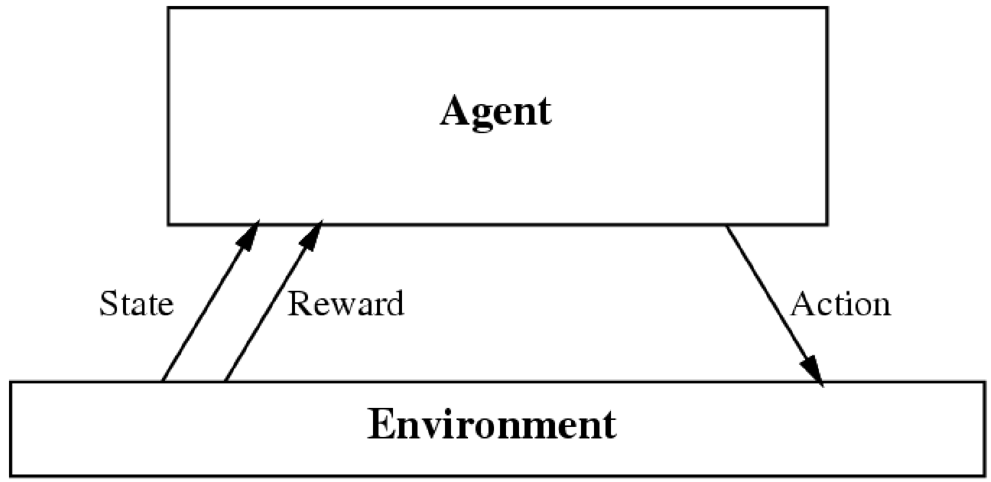
Outline

- 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

Agent lives in some environment; in some state:

- Robot: where robot is, what direction it is pointing, etc.
- Backgammon, state of the board (where all pieces are).



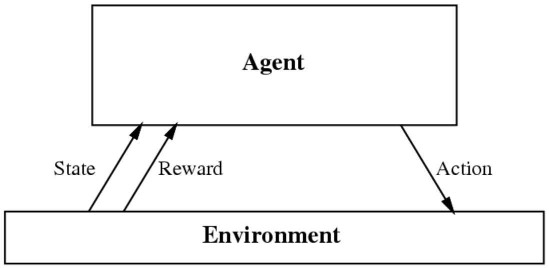
$$s_0 \xrightarrow[r_0]{a_0} s_1 \xrightarrow[r_1]{a_1} s_2 \xrightarrow[r_2]{a_2} \dots$$

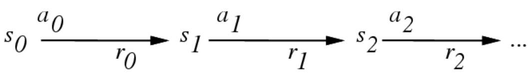
Goal: Maximize long term discounted reward. I.e.: want a lot of reward, prefer getting it earlier to getting it later.

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 = Reinforcement Learning Setting





- 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: $P(s_{t+1} \mid s_t, a_t, s_{t-1}, a_{t-1}, \ldots) = P(s_{t+1} \mid s_t, a_t)$
- Also assume reward Markov: $P(r_t \mid s_t, a_t, s_{t-1}, a_{t-1},\ldots) = P(r_t \mid s_t, a_t)$

E.g., if tell robot to move forward one meter, maybe it ends up moving forward 1.5 meters by mistake, so where the robot is at time t+1 can be a probabilistic function of where it was at time t and the action taken, but shouldn't depend on how we got to that state.

- The task: learn a policy π: S → A for choosing actions that maximizes

$$E[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots] \quad 0 < \gamma \leq 1$$

for every possible starting state s_0

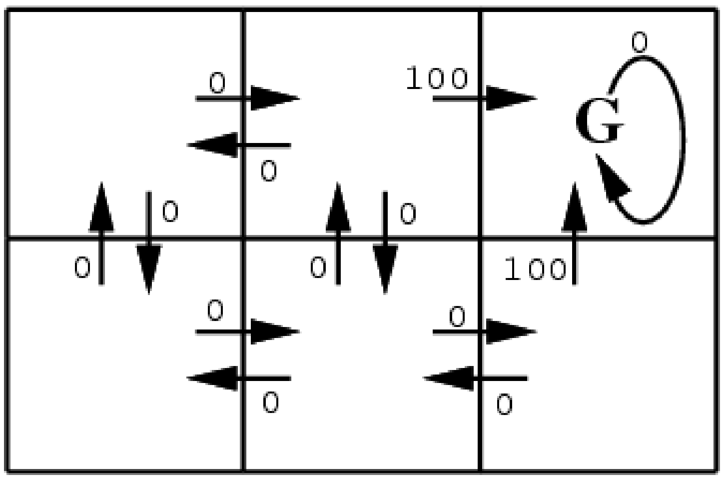
MACHINE LEARNING DEPARTMENT

Reinforcement Learning Task for Autonomous Agent

Execute actions in environment, observe results, and

- Learn control policy π: S→A that maximizes $\sum_{t=0}^{\infty} \gamma^t E[r_t]$ from every state s $\in$ S

Example: Robot grid world, deterministic reward *r(s,a)*



- Actions: move up, down, left, and right

[except when you are in the top-right you stay there, and say any action that bumps you into a wall leaves you were you were]]

- reward fns r(s,a) is deterministic with reward 100 for entering the top-right and 0 everywhere else.

$$r(s, a) \text{ (immediate reward)}$$

Reinforcement Learning Task for Autonomous Agent

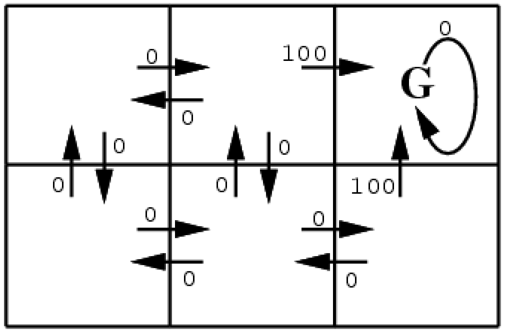
Execute actions in environment, observe results, and

- Learn control policy π: S→A that maximizes $\sum_{t=0}^{\infty} \gamma^t E[r_t]$ from every state s $\in$ S

Yikes!!

- Function to be learned is π: S→A
- But training examples are not of the form <s, a>
- They are instead of the form < <s,a>, r >

Value Function for each Policy



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

$$V^{\pi}(s) = E[\sum_{t=0}^{\infty} \gamma^t r_t]$$ assuming action sequence chosen according to π, starting at state *s*

expected discounted reward we will get starting from state s if we follow policy π.

- Goal: find the *optimal* policy π^* where

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

policy whose value function is the maximum out of all policies simultaneously for all states

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

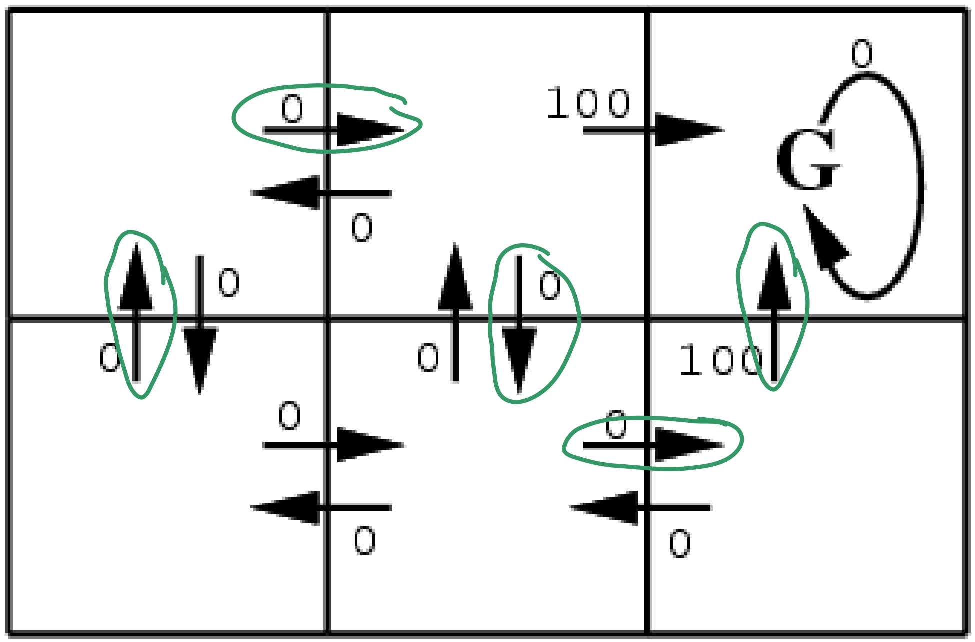
$$\pi^*(s) = argmax_a[r(s,a) + \gamma \sum_{s'} P(s'|s,a)V^*(s')]$$

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

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

Suppose π is shown by circled action from each state

Suppose $\gamma = 0.9$



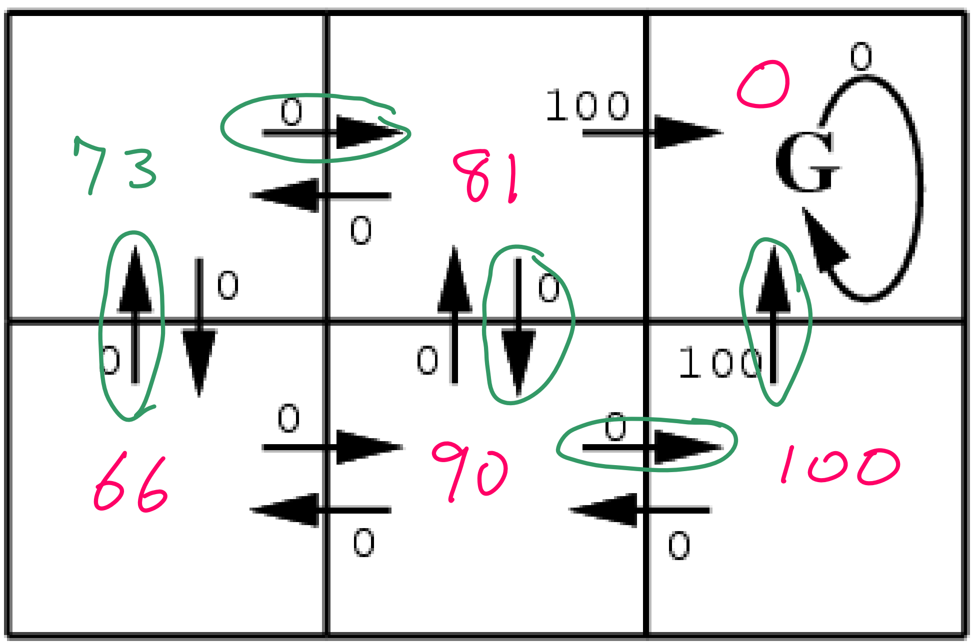
$r(s, a)$ (immediate reward)

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

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

Suppose π is shown by circled action from each state

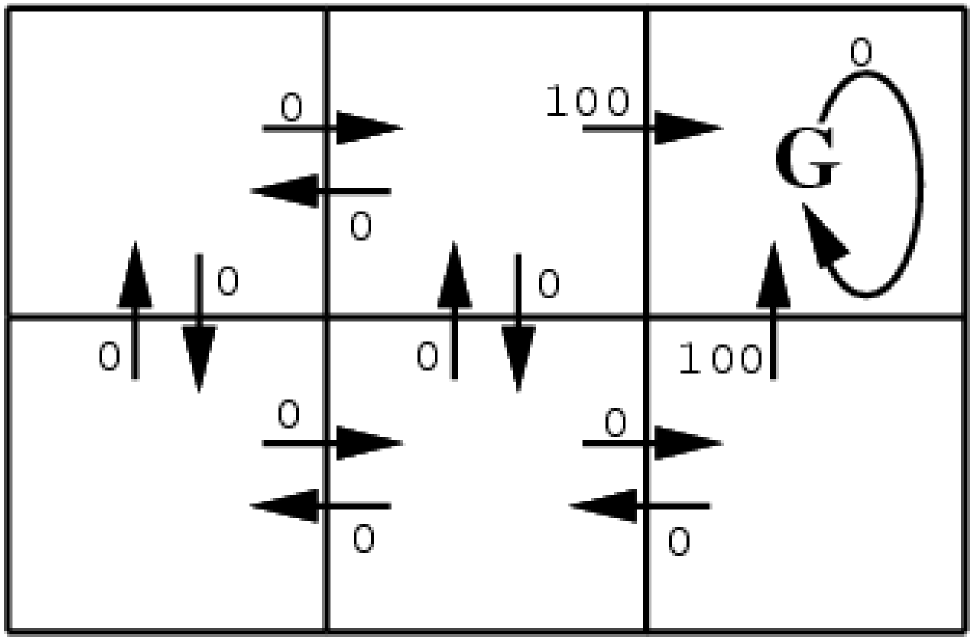
Suppose $\gamma = 0.9$



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

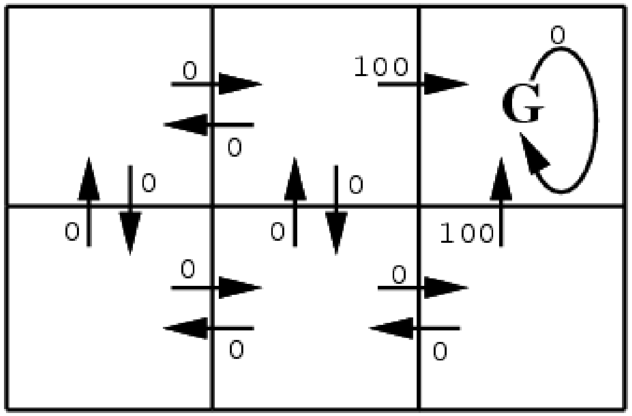
Value Function – what are the V*(s) values?

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



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

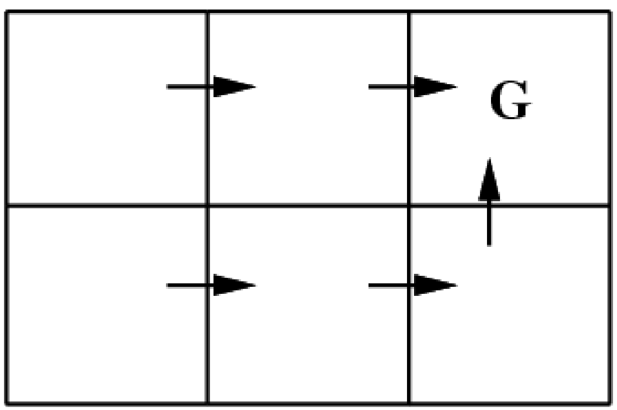
MACHINE LEARNING DEPARTMENT



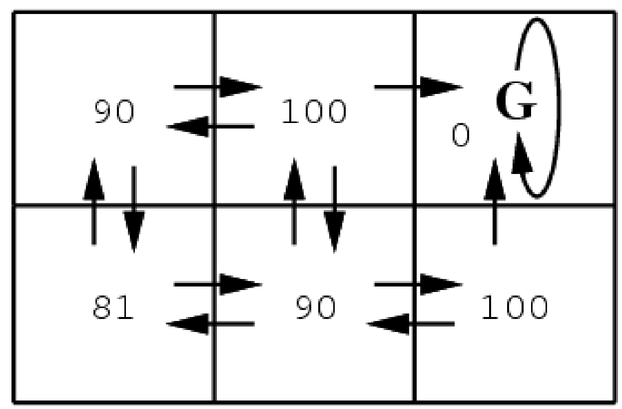
$r(s,a)$ (immediate reward) values

Immediate rewards r(s,a)

State values V*(s)



One optimal policy



$V^*(s)$ values

Recursive definition for V*(S)

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

assuming actions are chosen according to the optimal policy, π^*

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

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

Value $V^*(s_1)$ of performing optimal policy from s_1, is expected reward of the first action a_1 taken plus γ times the expected value, over states s_2 reached by performing action a_1 from s_1, of the value $V^*(s_2)$ of performing the optimal policy from then on.

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

optimal value of any state s is the expected reward of performing $\pi^*(s)$ from s plus γ times the expected value, over states s' reached by performing that action from state s, of the optimal value of s'.

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

Initialize V(s) to 0 [optimal value can get in zero steps]

For t=1, 2, … [Loop until policy good enough]

Loop for s in S

Loop for a in A

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

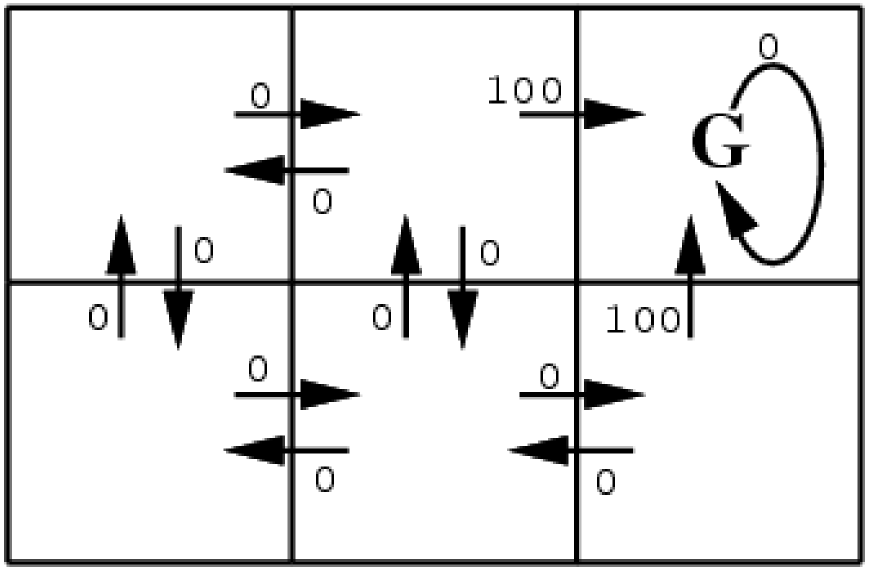
$V(s) \leftarrow \max_a Q(s,a)$

End loop

End loop

Inductively, if V is optimal discounted reward can get in t-1 steps, Q(s,a) is value of performing action a from state s and then being optimal from then on for the next t-1 steps.

Optimal expected discounted reward can get by taking an action and then being optimal for t-1 steps= optimal expected discounted reward can get in t steps.



V(s) converges to V*(s)

Dynamic programming

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

Initialize V(s) to 0 [optimal value can get in zero steps]

For t=1, 2, … [Loop until policy good enough]

Loop for s in S

each round we are computing the value of performing the optimal t-step policy starting from t=0, then t=1, t=2, etc, and since γ^t goes to 0, once t is large enough this will be close to the optimal value V^* for the infinite-horizon case.

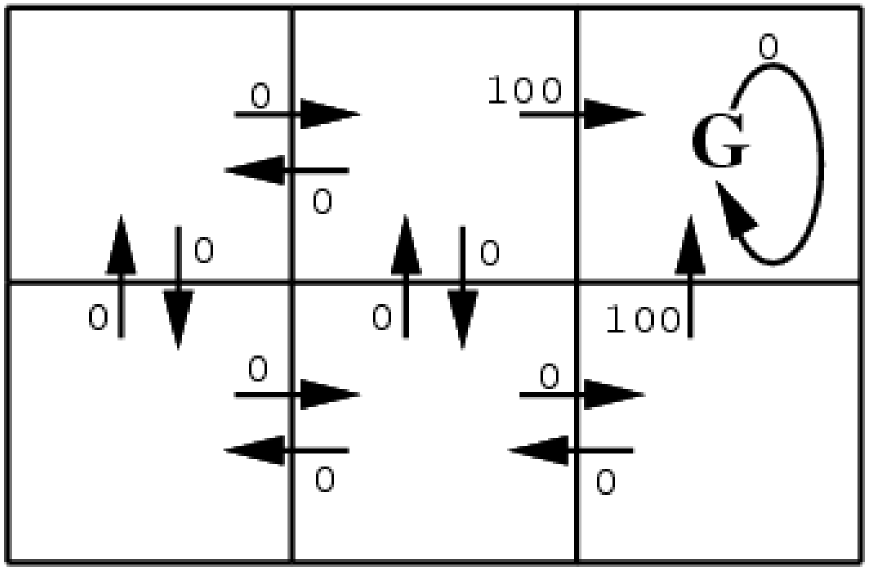
Loop for a in A

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

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

End loop

End loop



V(s) converges to V*(s)

Dynamic programming

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

Initialize V(s) to 0 [optimal value can get in zero steps]

For t=1, 2, … [Loop until policy good enough]

Loop for s in S

Loop for a in A

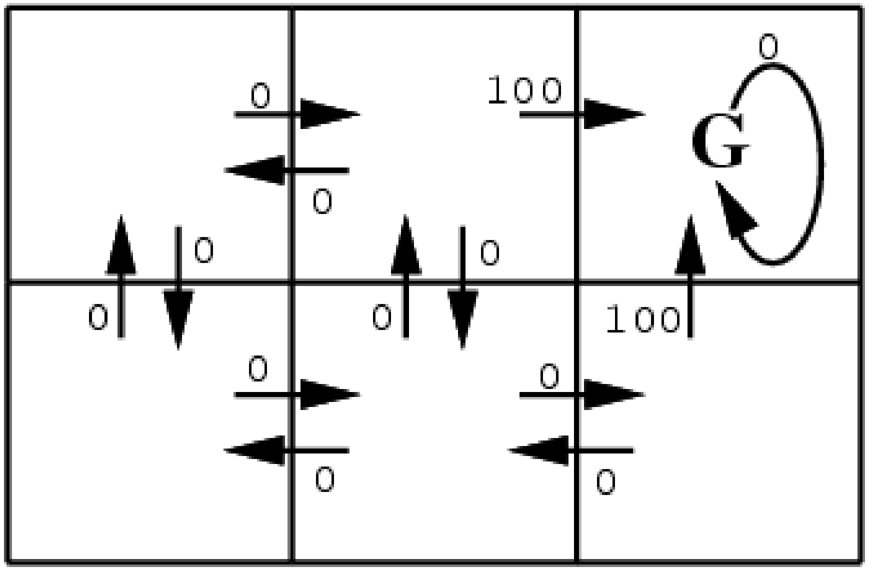
- $$Q(s,a) \leftarrow r(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V(s')$$

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

End loop

End loop

- Round t=0 we have V(s)=0 for all s.
- After round t=1, a top-row of 0, 100, 0 and a bottom-row of 0, 0, 100.
- After the next round (t=2), a top row of 90, 100, 0 and a bottom row of 0, 90, 100.
- After the next round (t=3) we will have a top-row of 90, 100, 0 and a bottom row of 81, 90, 100, and it will then stay there forever



Value Iteration

So far, in our DP, each round we cycled through each state exactly once.

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 for our DP, max (over states) difference between two successive value function estimates is less than ε, then the value of the greedy policy differs from the optimal policy by no more than

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

MACHINE LEARNING DEPARTMENT

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

What if we don't?

Q learning

Define new function, closely related to V*

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

V*(s) is the expected discounted reward of following the optimal policy from time 0 onward.

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

Q(s,a) is the expected discounted reward of first doing action a and then following the optimal policy from the next step onward.

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

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

Just chose the action that maximizes the Q value

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

using something very much like the dynamic programming algorithm we used to compute V*.

Immediate rewards r(s,a)

State values V*(s)

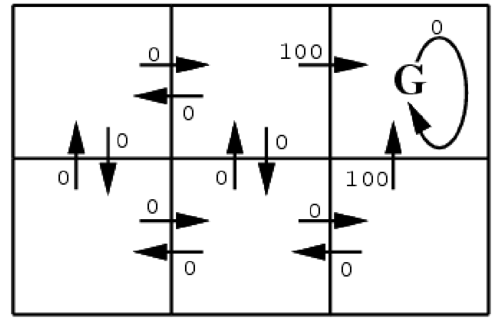
State-action values Q*(s,a)

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

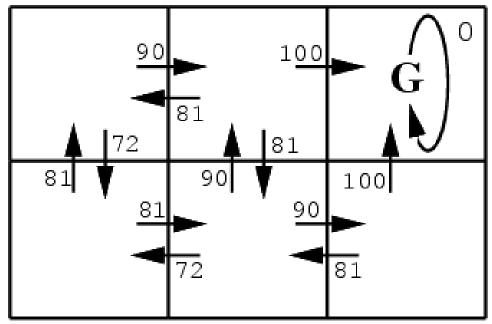
Bellman equation.

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

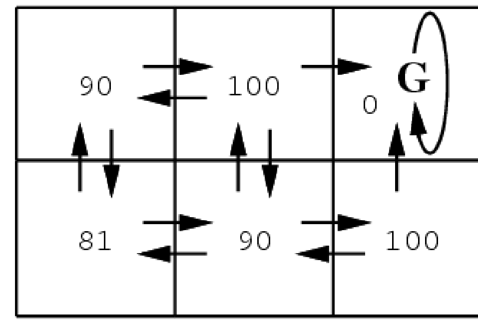
Consider first the case where P(s’| s,a) is deterministic



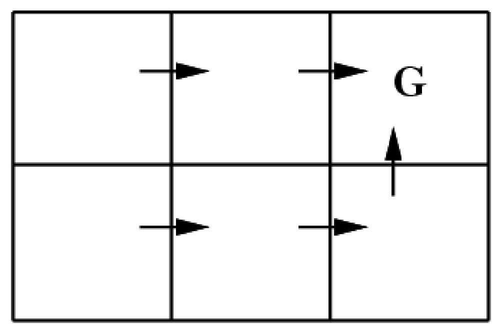
$r(s,a)$ (immediate reward) values



$Q(s,a)$ values



$V^*(s)$ values



One optimal policy

Training Rule to Learn Q

[simplicity assume the transitions and rewards are deterministic.]

Note Q and V^* closely related:

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

Optimal value of a state s is the maximum, over actions a' of 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

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

Given current approx $\hat{Q}$ to Q, if we are in state s and perform action a and get to state s', update our estimate $\hat{Q}(s,a)$ to the reward r we got plus gamma times the maximum over a' of $\hat{Q}(s',a')$

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

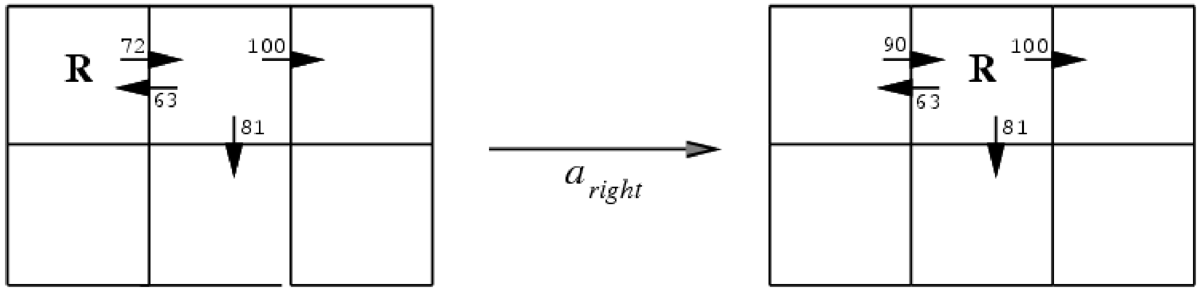
Do forever:

- 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 $\hat{Q}$



initial state: s_1 next state: s_2

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

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 γ

Let $\hat{Q}_n$ be table after n updates, and Δ_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

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

Use general fact:

$$|\max_a f_1(a) - \max_a f_2(a)| \leq \max_a |f_1(a) - f_2(a)|$$

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

$$\alpha_n = \frac{1}{1 + visits_n(s,a)}$$

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

Rather than replacing the old estimate with the new estimate, you want to compute a weighted average of them: $(1-\alpha_n)$ times your old estimate plus α_n times your new estimate. This way you average out the probabilistic fluctuations, and one can show that this still converges.

MDP's and RL: What You Should Know

- Learning to choose optimal actions A
- From *delayed reward*
- By learning evaluation functions like V(S), Q(S,A)

Key ideas:

- If next state function $S_t \times A_t \rightarrow S_{t+1}$ is known
  - can use dynamic programming to learn V(S)
  - once learned, choose action A_t that maximizes $V(S_{t+1})$
- If next state function $S_t \times A_t \rightarrow S_{t+1}$ **un**known
  - learn $Q(S_t,A_t) = E[V(S_{t+1})]$
  - to learn, sample $S_t \times A_t \rightarrow S_{t+1}$ in actual world
  - once learned, choose action A_t that maximizes $Q(S_t,A_t)$