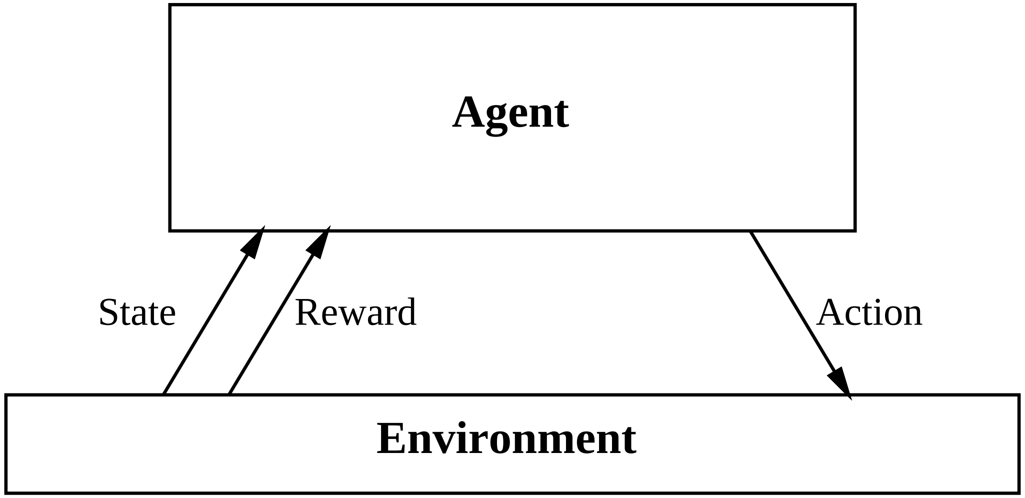
Reinforcement Learning: Value and Policy Iteration

Manuela Veloso

Carnegie Mellon University
Computer Science Department

15-381 - Fall 2001

Reinforcement Learning Problem



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

Goal: Learn to choose actions that maximize

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

Q Learning for Deterministic Worlds

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

Observe current state s

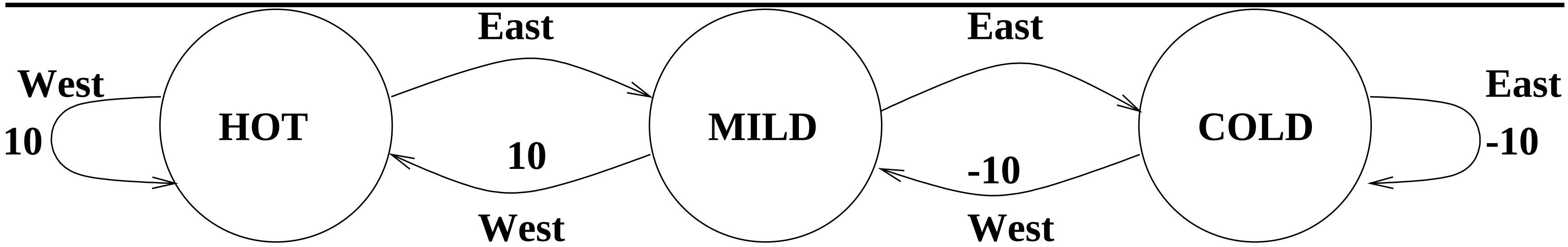
Do forever:

- 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:

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

- $s \leftarrow s'$

Example - Deterministic

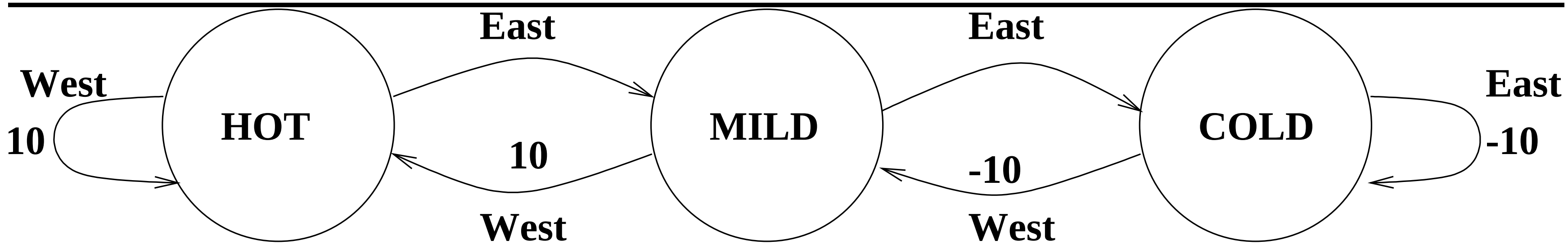


How many possible **policies** are there in this 3-state, 2-action deterministic world?

A robot starts in the state Mild. It moves for 4 steps choosing actions **West, East, East, West**. The initial values of its Q-table are 0 and the discount factor is $\gamma = 0.5$.

	Initial State: MILD		Action: West New State: HOT		Action: East New State: MILD		Action: East New State: COLD		Action: We New State: M	
	East	West	East	West	East	West	East	West	East	Wes
HOT	0	0	0	0	**5**	0	5	0	5	0
MILD	0	0	0	**10**	0	10	**0**	10	0	10
COLD	0	0	0	0	0	0	0	0	0	**-5**

Problem - Deterministic



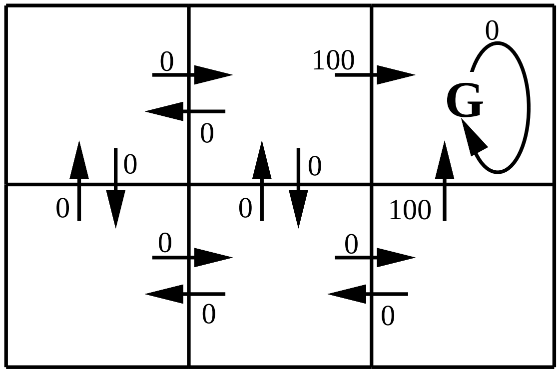
Why is the policy $\pi(s) =$ West, for all states, *better than* the policy $\pi(s) =$ East, for all states?

- $\pi_1(s) =$ West, for all states, $\gamma = 0.5$

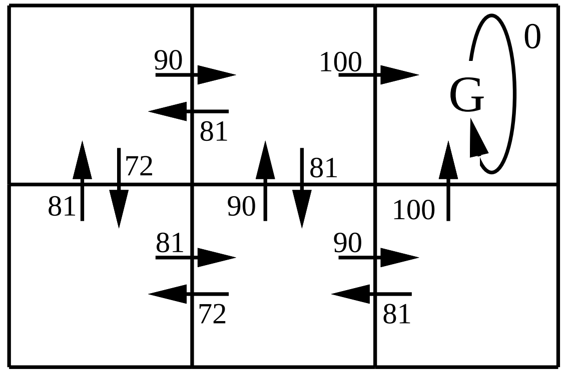
$$V^{\pi_1}(\text{HOT}) = 10 + \gamma V^{\pi_1}(\text{HOT}) = 20.$$

- $\pi_2(s) =$ East, for all states, $\gamma = 0.5$
  - $V^{\pi_2}(\text{COLD}) = -10 + \gamma V^{\pi_2}(\text{COLD}) = -20,$
  - $V^{\pi_2}(\text{MILD}) = 0 + \gamma V^{\pi_2}(\text{COLD}) = -10,$
  - $V^{\pi_2}(\text{HOT}) = 0 + \gamma V^{\pi_2}(\text{MILD}) = -5.$

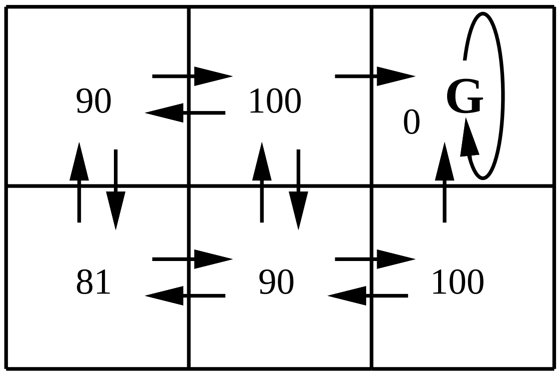
Another Deterministic Example



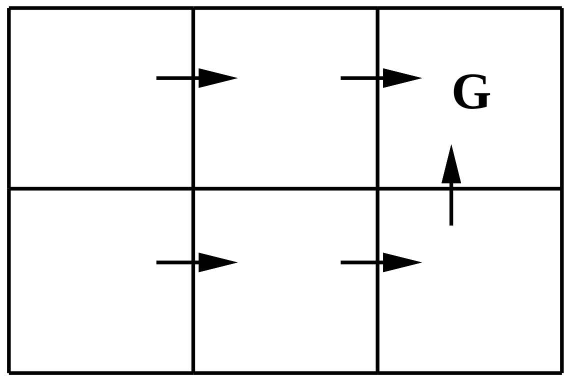
$r(s, a)$ values



$Q(s, a)$ values



$V^*(s)$ values



One optimal policy

Nondeterministic Case

What if reward and next state are non-deterministic?

We redefine V, Q by taking expected values

$$\begin{aligned} V^{\pi}(s) &\equiv E[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots] \\ &\equiv E[\sum_{i=0}^{\infty} \gamma^i r_{t+i}] \end{aligned}$$

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

Nondeterministic Case

Q learning generalizes to nondeterministic worlds

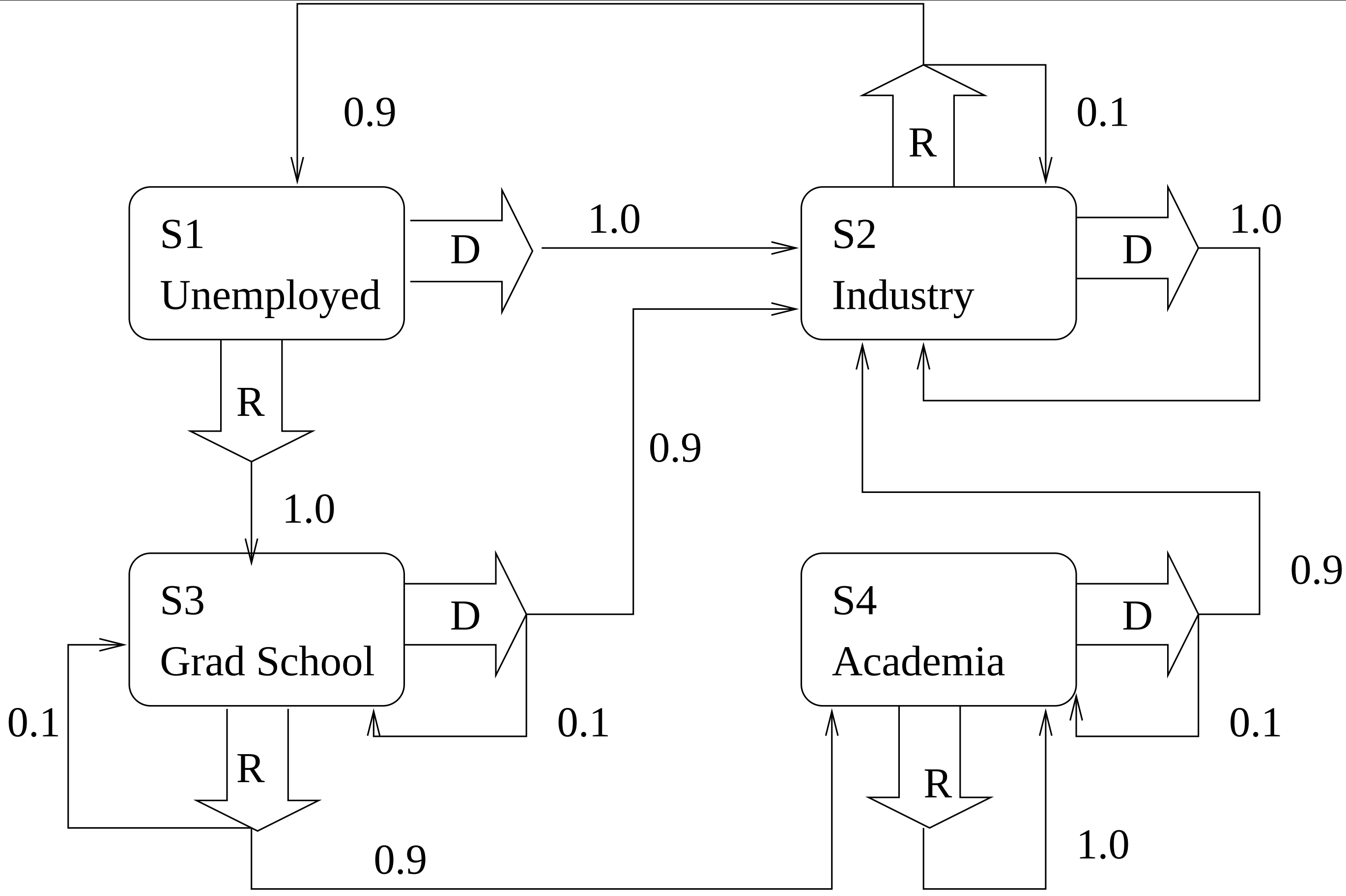
Alter training rule to

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

where $\alpha_n = \frac{1}{1+visits_n(s,a)}$, and $s' = \delta(s,a)$.

$\hat{Q}$ still converges to Q^* (Watkins and Dayan, 1992)

Nondeterministic Example



REWARD

S1	S2	S3	S4
0	100	0	10

Nondeterministic Example

π^*(s) = D, for any s= S1, S2, S3, and S4, $\gamma = 0.9$.

```
--------------------------------------------------
V*(S2) = r(S2,D) + 0.9 (1.0 V*(S2))
V*(S2) = 100 + 0.9 V*(S2)
V*(S2) = 1000.

V*(S1) = r(S1,D) + 0.9 (1.0 V*(S2))
V*(S1) = 0 + 0.9 x 1000
V*(S1) = 900.

V*(S3) = r(S3,D) + 0.9 (0.9 V*(S2) + 0.1 V*(S3))
V*(S3) = 0 + 0.9 (0.9 x 1000 + 0.1 V*(S3))
V*(S3) = 81000/91.

V*(S4) = r(S4,D) + 0.9 (0.9 V*(S2) + 0.1 V*(S4))
V*(S4) = 40 + 0.9 (0.9 x 1000 + 0.1 V*(S4))
V*(S4) = 85000/91.
```

Nondeterministic Example

What is the Q-value, Q(S2,R)?

```
Q(S2,R) = r(S2,R) + 0.9 (0.9 V*(S1) + 0.1 V*(S2))
Q(S2,R) = 100 + 0.9 (0.9 x 900 + 0.1 x 1000)
Q(S2,R) = 100 + 0.9 (810 + 100)
Q(S2,R) = 100 + 0.9 x 910
Q(S2,R) = 919.
```

Markov Decision Processes

- Finite set of states, $s_1, \ldots, s_n$
- Finite set of actions, $a_1, \ldots, a_m$
- Probabilistic state,action transitions:

  $p_{ij}^k =$ prob (next = s_j | current = s_i and take action a_k)
- Reward for each state and action.
- Process:
  - Start in state s_i
  - Choose action $a_k \in A$
  - Receive immediate reward $r_i(s_i, a_k)$
  - Change to state s_j with probability p_{ij}^k.
  - Discount future rewards

Solving an MDP

- A policy is a mapping from states to actions.
- Optimal policy - for every state, there is no other action that gets a higher sum of discounted future rewards.
- For every MDP there exists an optimal policy.
- Solving an MDP is finding an optimal policy.
- A specific policy converts an MDP into a plain Markov system with rewards.

Policy Iteration

- Start with some policy $\pi_0(s_i)$.
- Such policy transforms the MDP into a plain Markov system with rewards.
- Compute the values of the states according to current policy.
- Update policy:

$$\pi_1(s_i) = \text{argmax}_a \{r_i + \gamma \sum_j p_{ij}^a V^{\pi_0}(s_j)\}$$

- Keep computing
- Stop when $\pi_{k+1} = \pi_k$.

Value Iteration

- $V^*(s_i)$ = expected discounted future rewards, if we start from s_i and we follow the optimal policy.
- Compute V^* with value iteration:
  - $V^k(s_i)$ = maximum possible future sum of rewards starting from state s_i for k steps.
- Bellman's Equation:

$$V^{n+1}(s_i) = \text{max}_k\{r_i + \gamma \sum_{j=1}^{N} p_{ij}^k V^n(s_j)\}$$

- Dynamic programming

Summary

- Q-learning
- Markov decision processes
- Value, policy iteration