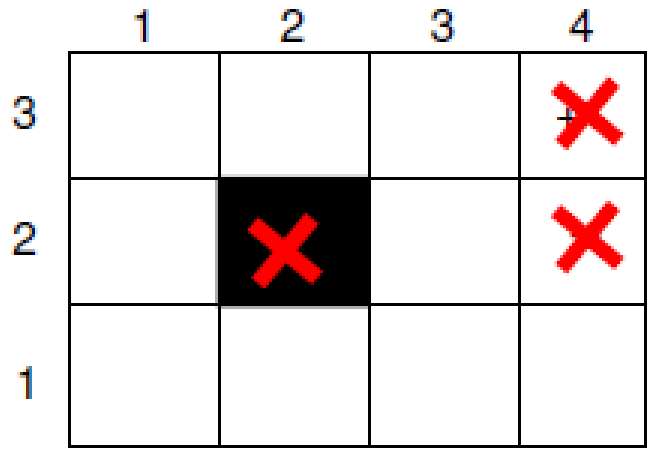
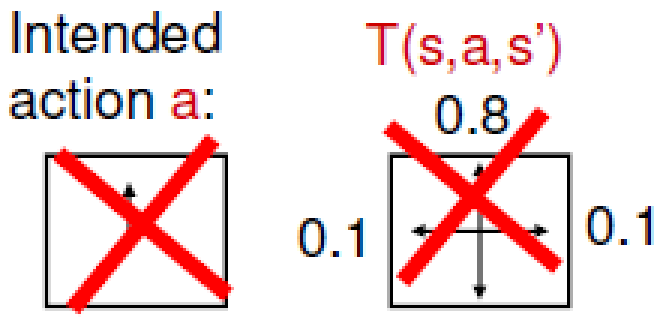
Reinforcement Learning

Different Aspects of "Machine Learning"

- Supervised learning
  - Classification - concept learning
  - Learning from labeled data
  - Function approximation
- Unsupervised learning
  - Data is not labeled
  - Data needs to be *grouped*, *clustered*
  - We need distance metric
- Control and action model learning
  - Learning to select actions efficiently
  - Feedback: goal achievement, failure, *reward*
  - Control learning, reinforcement learning

Reinforcement Learning



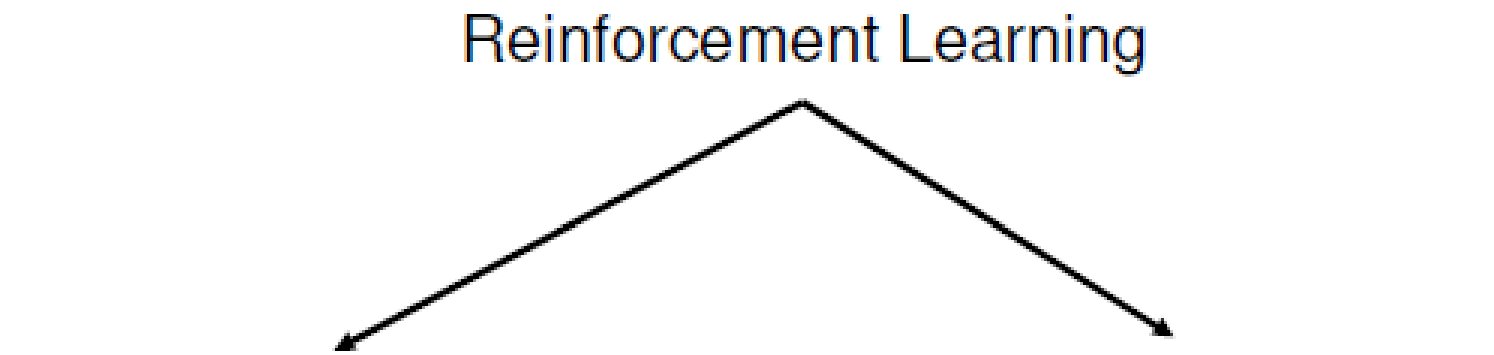


- Same (fully observable) MDP as before except:
  - We don't know the model of the environment
  - We don't know T(.,.,.)
  - We don't know R(.)
- Task is still the same:
  - Find an optimal policy

General Problem

- All we can do is try to execute actions and record the resulting rewards
  - World: You are in state **102**, you have a choice of 4 actions
  - Robot: I'll take action 2
  - World: You get a reward of 1 and you are now in state 63, you have a choice of 3 actions
  - Robot: I'll take action 3
  - World: You get a reward of -10 and you are now in state 12, you have a choice of 4 actions
  - ……..

Classes of Techniques



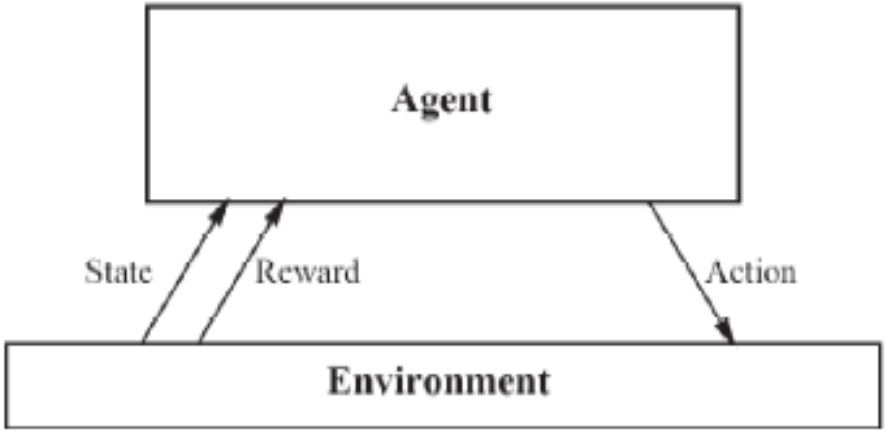
Model-Based

- Try to learn an explicit model of T(.,.,.) and R(.)

Model-Free

- Recover an optimal policy without ever estimating a model

Reinforcement Learning Problem



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

Goal: Learn to choose actions that maximize

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

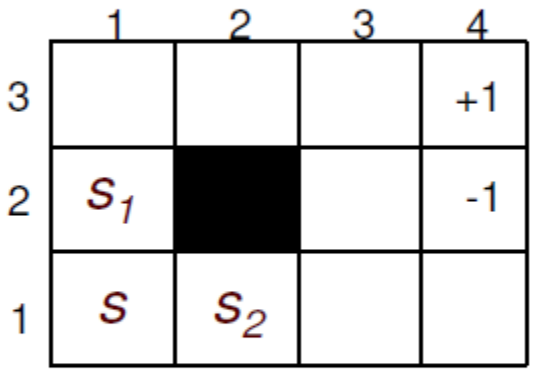
Model-Based

- If we knew a good estimate $T^{est}(.,.,.)$ of $T(.,.,.)$ and $R(.)$, we could evaluate the optimal policy by solving the fundamental MDP relations:

$$U^{est}(s) = R(s) + \gamma \max_a \left(\sum_{s'} T^{est}(s,a,s') \, U^{est}(s')\right)$$

$$\pi^*(s) = \text{argmax}_a \left(\sum_{s'} T^{est}(s,a,s') \, U^{est}(s')\right)$$

Model Estimation



I observed a trajectory during which, when I moved Up from s = (1,1), I ended up in

s_1 = (1,2) 10 times

s_2 = (2,1) 2 times

T(s, Up, s_1) ~ 10/(10+2) = 0.83

T(s, Up, s_2) ~ 2/(10+2) = 0.17

Model-Based

- Move through the environment by executing a sequence of actions
- Evaluate T and R:
  - R(s) = Reward received when visiting state s
  - $T^{est}(s, a, s')$ ~ (# times we moved from s to s' on action a)/(# times we applied action a from s)
- This gives us an estimated model of the Markov system

Model-Based

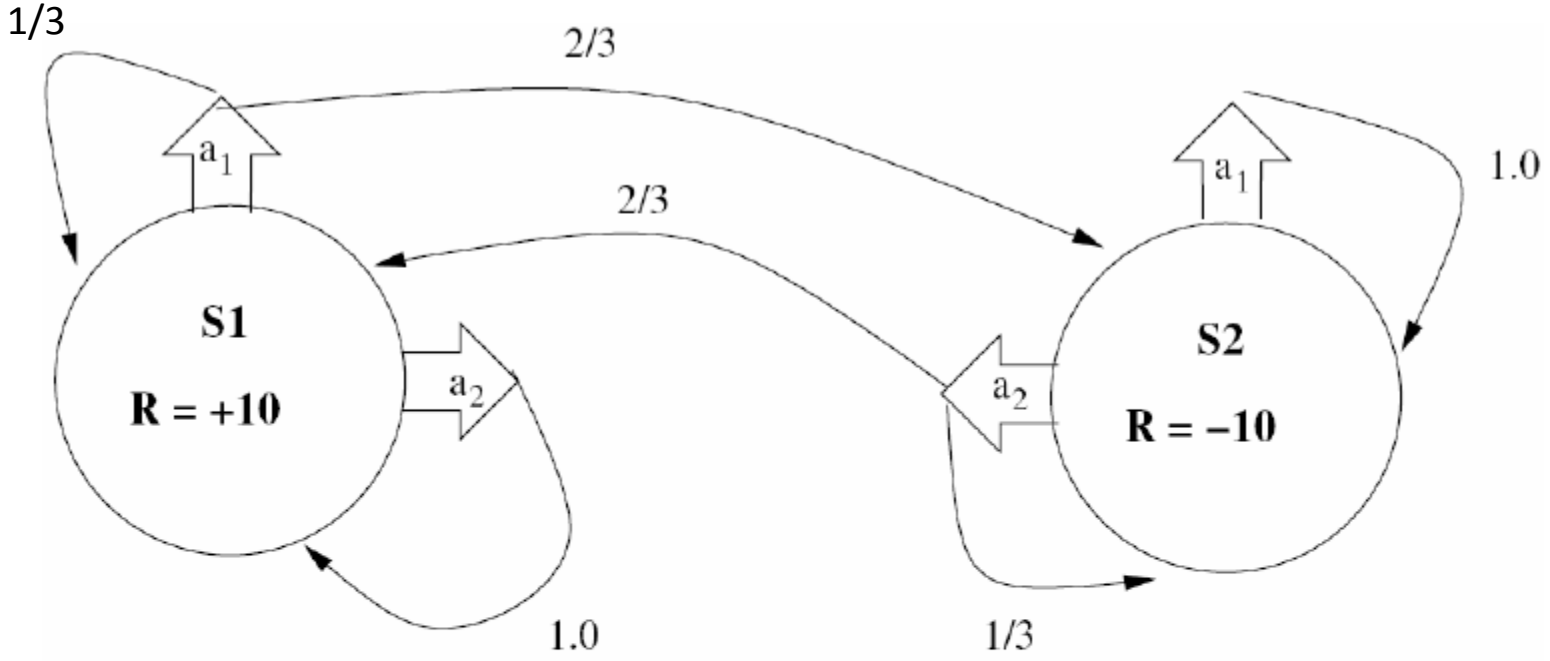
- Given T^{est} and R, we can now estimate the value at each state:

$$U^{est}(s) = R(s) + \gamma \max_a \left(\sum_{s'} T^{est}(s,a,s') \, U^{est}(s') \right)$$

  - Value iteration
  - Policy iteration
- This can be expensive if we do that at each step
- May require matrix inversion (size = number of states) or
- Many iterations of value iteration

Best Policy?

(Start = S_1, Action = a_1, Reward = 10, End = S_2)
(Start = S_2, Action = a_2, Reward = -10, End = S_1)
(Start = S_1, Action = a_2, Reward = 10, End = S_1)
(Start = S_1, Action = a_1, Reward = 10, End = S_1)
(Start = S_1, Action = a_2, Reward = 10, End = S_1)
(Start = S_1, Action = a_1, Reward = 10, End = S_2)
(Start = S_2, Action = a_1, Reward = -10, End = S_2)
(Start = S_2, Action = a_2, Reward = -10, End = S_2)
(Start = S_2, Action = a_2, Reward = -10, End = S_1)



$$\pi^*(S_1) = a_2$$
$$\pi^*(S_2) = a_2$$

Model-Free

- We are not interested in T(.,.,.), we are only interested in the resulting values and policies
- Can we compute something without an explicit model of T(.,.,.) ?

- First, let's fix a policy and compute the resulting values

Recall Value Function

- For each possible policy π, define an *evaluation function over states*

$$V^{\pi}(s) \equiv r_t + \gamma r_{t+1} + \gamma^2 r_{t+1} + \ldots \equiv \sum_{i=0}^{\infty} \gamma^i r_{t+i}$$

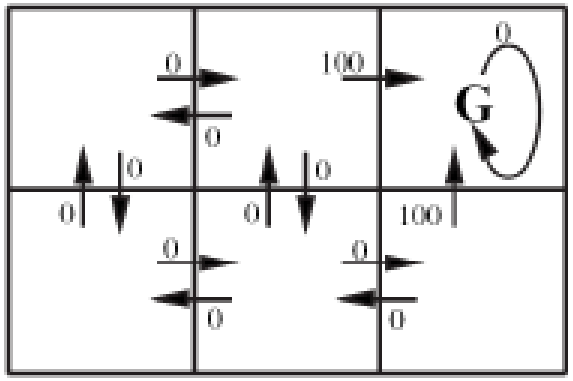
  where r_t, r_{t+1},... are generated by following policy π starting at state s

- Learning task: Learn OPTIMAL policy

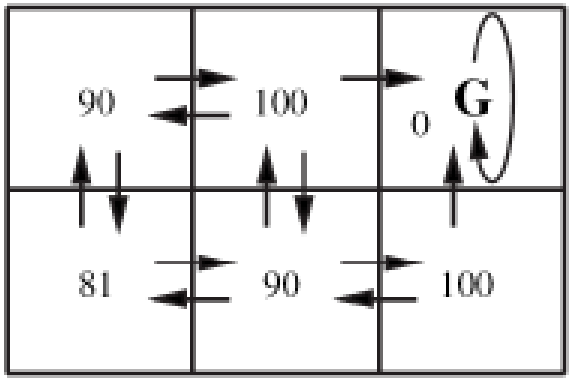
$$\pi^* \equiv \operatorname{argmax}_{\pi} V^{\pi}(s), (\forall s)$$

- Learn the evaluation function $V^{\pi *}$ - V^*.
- Select the optimal action from any state s, i.e., have an optimal policy, by using V^* with one step lookahead:

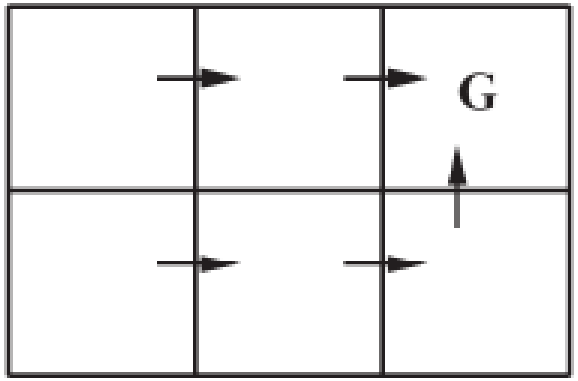
$$\pi^*(s) = \arg\max_a \left[r(s,a) + \gamma V^*(\delta(s,a)) \right]$$



rewards



$V^*(s)$ values



ONE optimal policy

Optimal Value to Optimal Policy

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

A problem:

- This works well if agent knows $\delta : S \times A \rightarrow S$, and $r : S \times A \rightarrow \Re$
- When it doesn't, it can't choose actions this way

Q Function

- Define new function very similar to V^*

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

Learn Q function – Q-learning

- If agent learns Q, it can choose optimal action even without knowing δ or r.

$$\pi^*(s) = \arg\max_a \left[r(s,a) + \gamma V^*(\delta(s,a)) \right]$$

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

Q-Learning

Note that Q and V^* are closely related:

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

Which allows us to write Q recursively as

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

Training Rule to Learn Q

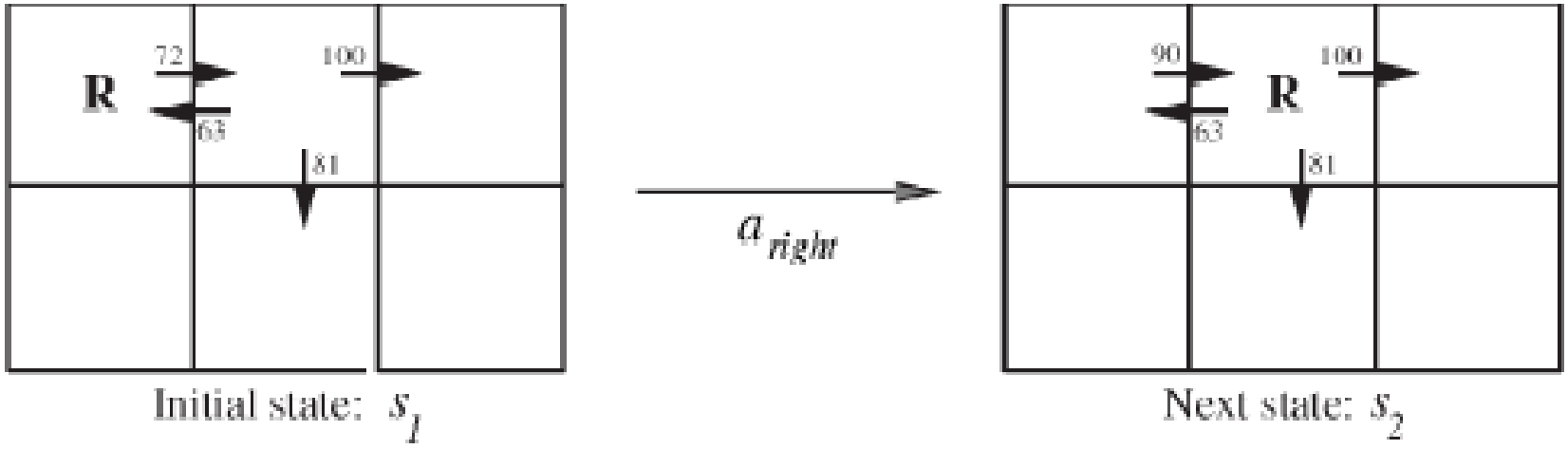
Let $\hat{Q}$ denote current approximation to Q.

Then Q-learning uses the following **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,
and r is the reward that is returned.

Example - Updating $\hat{Q}$



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

$$\leftarrow 0 + 0.9 \max\{63, 81, 100\}$$

$$\leftarrow 90$$

Q Learning for Deterministic Worlds

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

Observe current state s

Do forever:

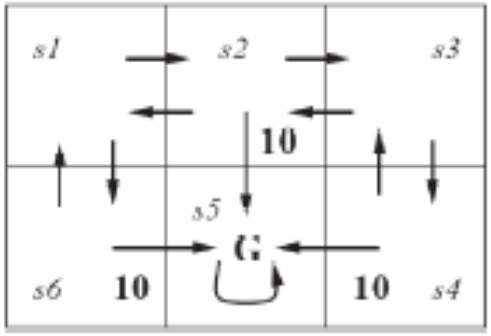
- Select an action a and execute it
- Receive immediate reward r
- Observe the new state s'
- Update the table entry for $\hat{Q}(s,a)$ as follows:

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

- $s \leftarrow s'$

Q Learning Iterations

Starts at bottom left corner – moves clockwise around perimeter;
Initially $Q(s,a) = 0$; $\gamma = 0.8$



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

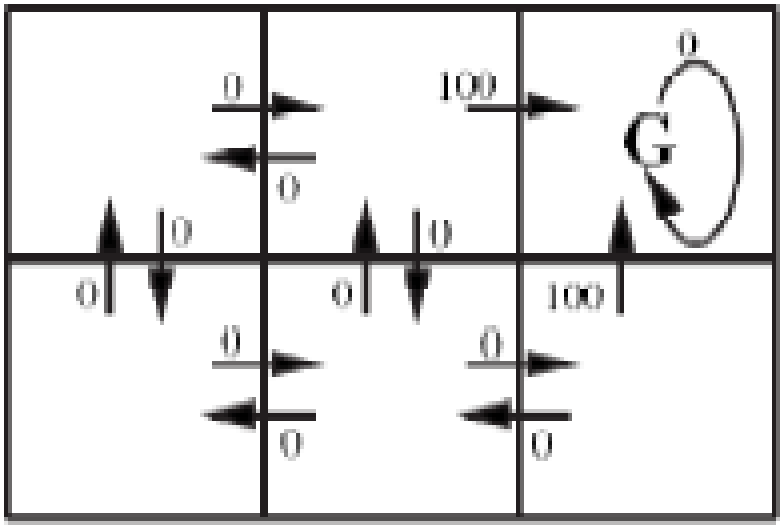
Q(s1,E)	Q(s2,E)	Q(s3,S)	Q(s4,W)
0	0	0	r + γ max{Q(s5,loop)} = 10 + 0.8 . 0 = 10
0	0	r + γ max{Q(s4,W),Q(s4,N)} = 0 + 0.8 max{10,0} = 8	10
0	r + γ max{Q(s3,W),Q(s3,S)} = 0 + 0.8 max{0,8} = 6.4	8	10

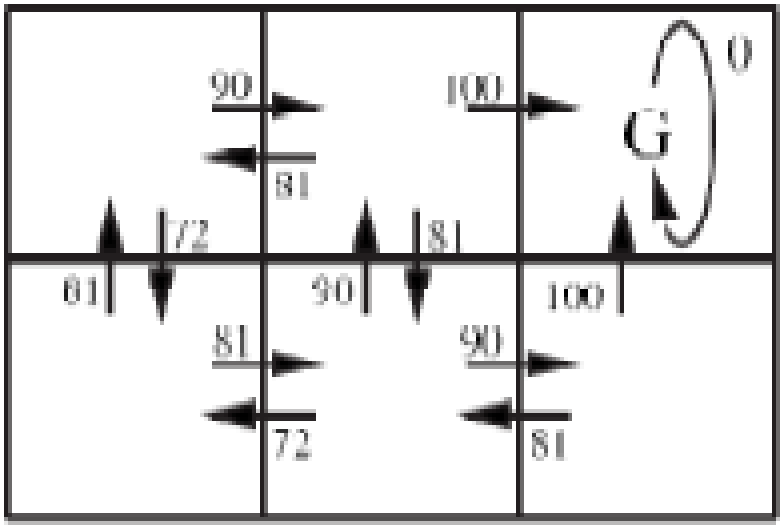
Problem - Deterministic

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 $\gamma = 0.5$

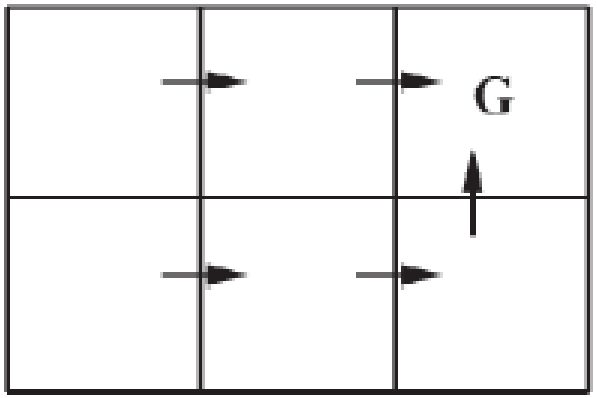
	Initial State: MILD		Action: West New State: HOT		Action: East New State: MILD		Action: East New State: COLD		Action: West New State: MILD	
	East	West	East	West	East	West	East	West	East	West
HOT	0	0	0	0	**5**	0	5	0	5	0
MILD	0	0	0	**10**	0	10	**0**	10	0	10
COLD	0	0	0	0	0	0	0	0	0	**-5**

Another Deterministic Example





$r(s,a)$ values $Q(s,a)$ values



One optimal policy