

# Reinforcement Learning

Manuela Veloso  
Reid Simmons

PEL, Fall 2010  
October 20, 2010

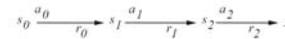
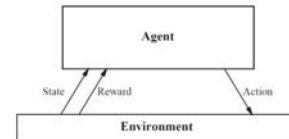
## Where We Are and Outline

- Planning
  - Deterministic state, preconditions, effects
  - Uncertainty
    - Conditional planning, conformant planning, nondeterministic
- Probabilistic modeling of systems with uncertainty and rewards
- Modeling probabilistic systems with control, i.e., action selection
- Reinforcement learning

## Reinforcement Learning

- Assume the world is a Markov Decision Process - transition and rewards unknown; states and actions known.
- Two objectives:
  - learning the *model*
  - converging to the *optimal plan*.

## Reinforcement Learning Problem



Goal: Learn to choose actions that maximize  
 $r_0 + \gamma r_1 + \gamma^2 r_2 + \dots$ , where  $0 \leq \gamma < 1$

## Reinforcement Learning

- A variety of successful algorithms
  - Mitchell's book "Machine Learning" (chapter 13)
  - Sutton and Barto's book "Reinforcement Learning"
  - Kaelbling, Moore, Littman: JAIR survey
- If we can do reinforcement learning, then:
  - outcome: for *every* state, *optimal* action is known
  - **Universal plan!**

## Learning Conditions

- Assume world can be modeled as a Markov Decision Process, with rewards as a function of state and action.
- *Markov assumption*:  
New states and rewards are a function only of the **current state and action**, i.e.,
  - $s_{t+1} = \delta(s_t, a_t)$
  - $r_t = r(s_t, a_t)$
- *Unknown and uncertain environment*:  
Functions  $\delta$  and  $r$  may be **nondeterministic** and are **not necessarily known** to learner.

## Control Learning Task

- Execute actions in world,
- Observe state of world,
- Learn action policy  $\pi : S \rightarrow A$
- Maximize expected reward

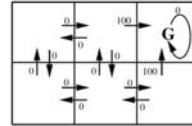
$$E[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots]$$

from any starting state in  $S$ .

–  $0 \leq \gamma < 1$ , discount factor for future rewards

## Statement of Learning Problem

- We have a target function to learn  $\pi : S \rightarrow A$
- We have **no** training examples of the form  $\langle s, a \rangle$
- We have training examples of the form  $\langle \langle s, a \rangle, r \rangle$   
(rewards can be any real number)



immediate reward values  $r(s,a)$

## Policies

Assume deterministic world

- There are many possible policies, of course not necessarily optimal, i.e., with maximum expected reward
- There can be also several OPTIMAL policies.

## Value Function

- For each possible policy  $\pi$ , define an evaluation function over states

$$V^\pi(s) \equiv r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots$$

$$\equiv \sum_{i=0}^{\infty} \gamma^i r_{t+i}$$

where  $r_t, r_{t+1}, \dots$  are generated by following policy  $\pi$  starting at state  $s$

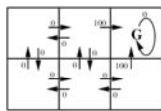
- Learning task: Learn OPTIMAL policy

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

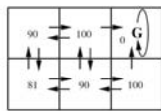
## Learn Value Function

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

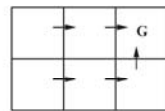
$$\pi^*(s) = \operatorname{argmax}_a [r(s,a) + \gamma V^*(\delta(s,a))]$$



rewards



$V^*(s)$  values



ONE optimal policy

## Optimal Value to Optimal Policy

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

A problem:

- This works well if agent knows  $\delta : S \times A \rightarrow S$ , and  $r : S \times A \rightarrow \mathfrak{R}$
- 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  $\delta$  or  $r$ .

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

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

## 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

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

$Q$ -learning actively generates examples. It "processes" examples by updating its  $Q$  values. While learning,  $Q$  values are approximations.

## Training Rule to Learn $Q$

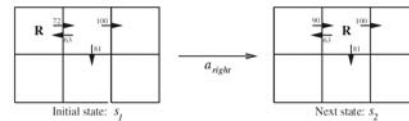
Let  $\hat{Q}$  denote current approximation to  $Q$ .

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 $\hat{Q}$



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

## Q Learning for Deterministic Worlds

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

Observe current state  $s$

Do forever:

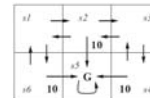
- Select an action  $a$  and execute it
- Receive immediate reward  $r$
- Observe the new state  $s'$
- Update the table entry for  $\hat{Q}(s,a)$  as follows:

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

- $s \leftarrow s'$

## Q Learning Iterations

Starts at top 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	$r + \gamma \max\{Q(s5,Loop)\} = 10 + 0.8 \cdot 0 = 10$
0	0	$r + \gamma \max\{Q(s4,W), Q(s4,N)\} = 0 + 0.8 \max\{10, 0\} = 8$	10
0	$r + \gamma \max\{Q(s3,W), Q(s3,S)\} = 0 + 0.8 \max\{0, 8\} = 6.4$	8	10

## Problem - Deterministic

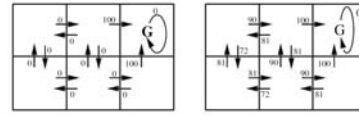


How many possible **policies** are there in this 3-state, 2-action deterministic world?

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  $\gamma = 0.5$

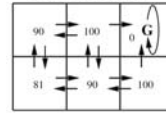
	Initial State: MILD		Action: West New State: HOT		Action: East New State: MILD		Action: East New State: COLD		Action: West New State: MILD	
	East	West	East	West	East	West	East	West	East	West
HOT	0	0	0	0	5	0	5	0	5	0
MILD	0	0	0	10	0	10	0	10	0	10
COLD	0	0	0	0	0	0	0	0	0	-5

## Another Deterministic Example

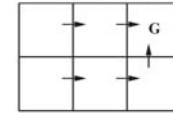


$r(s, a)$  values

$Q(s, a)$  values



$V^*(s)$  values



One optimal policy

## Nondeterministic Case

What if reward and next state are non-deterministic?

We redefine  $V, Q$  by taking expected values

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

$$\equiv E\left[\sum_{i=0}^{\infty} \gamma^i r_{t+i}\right]$$

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

## Nondeterministic Case

$Q$  learning generalizes to nondeterministic worlds

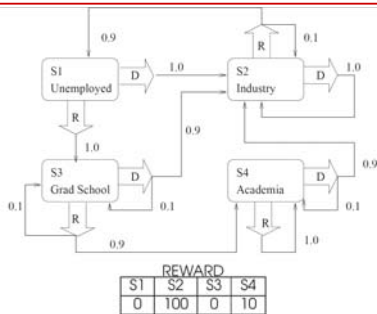
Alter training rule to

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

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

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

## Nondeterministic Example



## Nondeterministic Example

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

$$V^*(S2) = r(S2, D) + 0.9 (1.0 V^*(S2))$$

$$V^*(S2) = 100 + 0.9 V^*(S2)$$

$$V^*(S2) = 1000.$$

$$V^*(S1) = r(S1, D) + 0.9 (1.0 V^*(S2))$$

$$V^*(S1) = 0 + 0.9 \times 1000$$

$$V^*(S1) = 900.$$

$$V^*(S3) = r(S3, D) + 0.9 (0.9 V^*(S2) + 0.1 V^*(S3))$$

$$V^*(S3) = 0 + 0.9 (0.9 \times 1000 + 0.1 V^*(S3))$$

$$V^*(S3) = 81000/91.$$

$$V^*(S4) = r(S4, D) + 0.9 (0.9 V^*(S2) + 0.1 V^*(S4))$$

$$V^*(S4) = 40 + 0.9 (0.9 \times 1000 + 0.1 V^*(S4))$$

$$V^*(S4) = 85000/91.$$

## Nondeterministic Example

---

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

$$Q(S2,R) = r(S2,R) + 0.9 (0.9 V^*(S1) + 0.1 V^*(S2))$$

$$Q(S2,R) = 100 + 0.9 (0.9 \times 900 + 0.1 \times 1000)$$

$$Q(S2,R) = 100 + 0.9 (810 + 100)$$

$$Q(S2,R) = 100 + 0.9 \times 910$$

$$Q(S2,R) = 919.$$

## Discussion

---

- How should the learning agent use the *intermediate Q* values?
  - Exploration
  - Exploitation
- Scaling up in the size of the state space
  - Function approximator (neural net instead of table)
  - Generalization
  - Reuse, use of macros
  - Abstraction, learning substructure

## Summary

- Markov Models with Reward
- Value iteration
- Markov Decision Process
- Value Iteration
- Policy Iteration
- Reinforcement Learning