CSC411 Fall 2014
Machine Learning & Data Mining

Reinforcement Learning I

Slides from Rich Zemel

Reinforcement Learning

Learning classes differ in information available to learner

- Supervised: correct outputs
- Unsupervised: no feedback, must construct measure of good output
- Reinforcement learning

More realistic learning scenario:

- Continuous stream of input information, and actions
- Effects of action depend on state of the world
- Obtain reward that depends on world state and actions – not correct response, just some feedback

Formulating Reinforcement Learning

World described by a discrete, finite set of states and actions

At every time step t, we are in a state s_t, and we:

- Take an action a_t (possibly null action)
- Receive some reward r_{t+1}
- Move into a new state s_{t+1}

Decisions can be described by a policy – a selection of which action to take, based on the current state

Aim is to maximize the total reward we receive over time

Sometimes a future reward is discounted by γ^{k-1}, where k is the number of time-steps in the future when it is received

Tic-Tac-Toe

Make this concrete by considering specific example

Consider the game tic-tac-toe:

- reward: win/lose/tie the game (+1/-1/0) [only at final move in given game]
- state: positions of Xs and Os on the board
- policy: mapping from states to actions – based on rules of game: choice of one open position
- value function: prediction of reward in future, based on current state

In tic-tac-toe, since state space is tractable, can use a table to represent value function

RL & Tic-Tac-Toe

Each board position (taking into account symmetry) has associated probability

Simple learning process:

- start with all values = 0.5
- policy: choose move with highest probability of winning given current legal moves from current state
- update entries in table based on outcome of each game
- After many games value function will represent true probability of winning from each state

State	Probability of a win (Computer plays "o")
	0.5
	0.5
	1.0
	0.0
	0.5
etc	

Can try alternative policy: sometimes select moves randomly (exploration)

Acting Under Uncertainty

The world and the actor may not be deterministic, or our model of the world may be incomplete

We assume the Markov property: the future depends on the past only through the current state

We describe the environment by a distribution over rewards and state transitions:

$$P(s_{t+1} = s', r_{t+1} = r' \mid s_t = s, a_t = a)$$

The policy can also be non-deterministic:

$$P(a_t = a \mid s_t = s)$$

Policy is not a fixed sequence of actions, but instead a conditional plan

Basic Problems

Markov Decision Problem (MDP): tuple <S,A,P,γ>
where P is

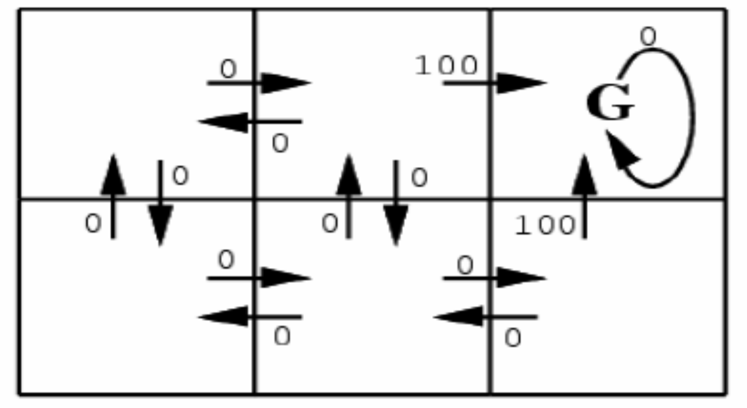
$$P(s_{t+1} = s', r_{t+1} = r' | s_t = s, a_t = a)$$

Standard MDP problems:

1. Planning: given complete Markov decision problem as input, compute policy with optimal expected return

2. Learning: Only have access to experience in the MDP, learn a near-optimal strategy

Example of Standard MDP Problem



1. Planning: given complete Markov decision problem as input, compute policy with optimal expected return

2. Learning: Only have access to experience in the MDP, learn a near-optimal strategy

We will focus on learning, but discuss planning along the way

Exploration vs. Exploitation

If we knew how world works (embodied in P), then the policy should be deterministic – just select optimal action in each state.

But if we do not have complete knowledge of the world, taking what appears to be the optimal action may prevent us from finding better states/actions

Interesting trade-off:

immediate reward (exploitation) vs. gaining knowledge that might enable higher future reward (exploration)

Bellman Equation

Decision theory: maximize expected utility (related to rewards)

Define the value function *V(s)*: measures accumulated future rewards (value) from state *s*

The relationship between a current state and its successor state is defined by the Bellman equation:

$$V(s_t) = r_t + \gamma V(s_{t+1})$$

Discount factor γ: controls whether care only about immediate reward, or can appreciate delayed gratification

Can show that if value functions updated via Bellman equation, and $\gamma < 1$, V() will converge to optimal (estimate of expected reward given best policy)

Expected value of a policy

Key recursive relationship between value function at successive states

If we fix some policy, π (defines the distribution over actions for each state), then the value of a state is the expected discounted reward for following that policy from that state on:

$$V^{\pi}(s_0) = E[\sum_{t=1}^{\infty} \gamma^{t-1} r_t | s_0]$$

This value function will satisfy the following consistency equation (generalized Bellman equation):

$$V^{\pi}(s) = \sum_{a} P^{\pi}(a_t = a \mid s_t = s) \times \sum_{s',r} P(s_{t+1} = s', r_{t+1} = r \mid s_t = s, a_t = a)(r + \gamma V^{\pi}(s'))$$

RL: Some Examples

Many natural problems have structure required for RL:

1. Game playing: know win/lose but not specific moves (TD-gammon)
2. Control: for traffic lights, can measure delay of cars, but not how to decrease it
3. Robot juggling
4. Robot path planning: can tell distance traveled, but not how to minimize

MDP formulation

Goal: find policy π that maximizes expected accumulated future rewards $V^{\pi}(s_t)$, obtained by following π from state s_t:

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

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

Game show example:

- assume series of questions, increasingly difficult, but increasing payoff
- choice: accept accumulated earnings and quit; or continue and risk losing everything

What to Learn

We might try to learn the value function V (which we write as V*)

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

We could then do a lookahead search to choose best action from any state s:

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

where

$$P(s_{t+1} = s', r_{t+1} = r' \mid s_t = s, a_t = a) =$$
$$P(s_{t+1} = s' \mid s_t = s, a_t = a) P(r_{t+1} = r' \mid s_t = s, a_t = a) =$$
$$\delta(s,a) r(s,a)$$

But there's a problem:

- This works well if we know δ() and r()
- But when we don't, we cannot choose actions this way

What to Learn

Let us first assume that δ() and r() are deterministic:

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

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

Remember:

At every time step t, we are in a state s_t, and we:

- Take an action a_t (possibly null action)
- Receive some reward r_{t+1}
- Move into a new state s_{t+1}

Reward function

$$r : (s,a) \rightarrow r$$

$$\delta : (s,a) \rightarrow s$$

Transition function

How can we do learning?

Q Learning

Define a new function very similar to V*

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

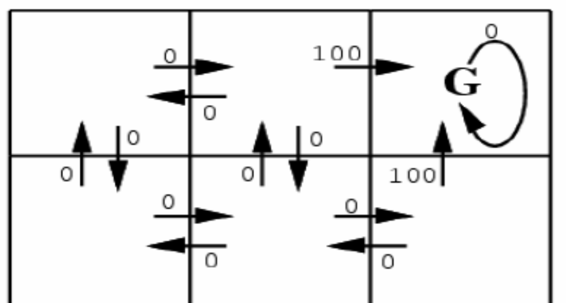
If we learn Q, we can choose the optimal action even without knowing δ!

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

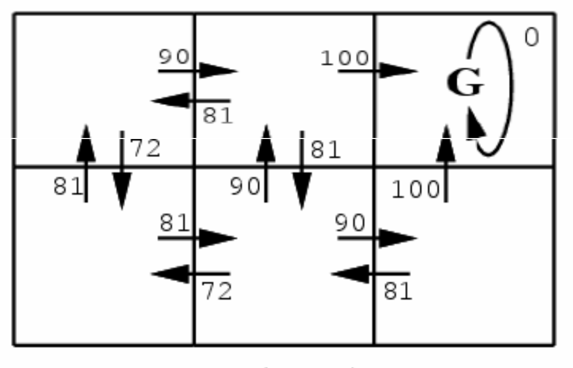
$$\pi^*(s) = \arg\max_a Q(s,a)$$

Q is then the evaluation function we will learn

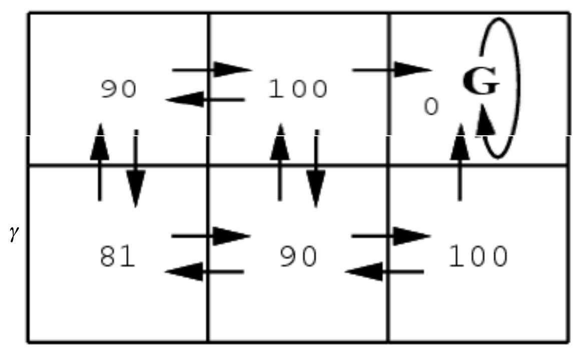
$\gamma = 0.9$



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

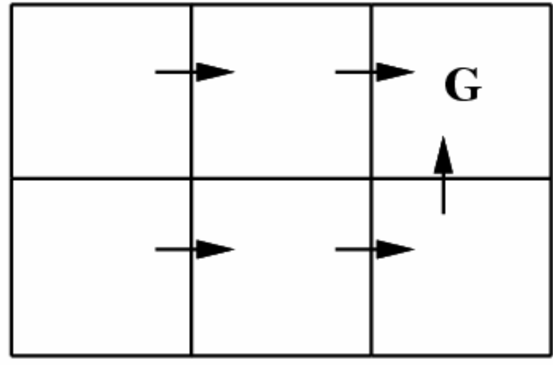


$Q(s,a)$ values



$V^*(s)$ values

$$V^*(s_5) = 0 + \gamma 100 + \gamma^2 0 + \ldots = 90$$



One optimal policy

Training Rule to Learn Q

Q and V* are closely related:

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

So we can write Q recursively:

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

Let Q^ denote the learner's current approximation to Q

Consider training rule

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

where s' is state resulting from applying action a in state s

Q Learning for Deterministic World

For each *s,a* initialize table entry Q^*(s,a)* ← 0

Start in some initial 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 *Q^(s,a)* using Q learning rule:

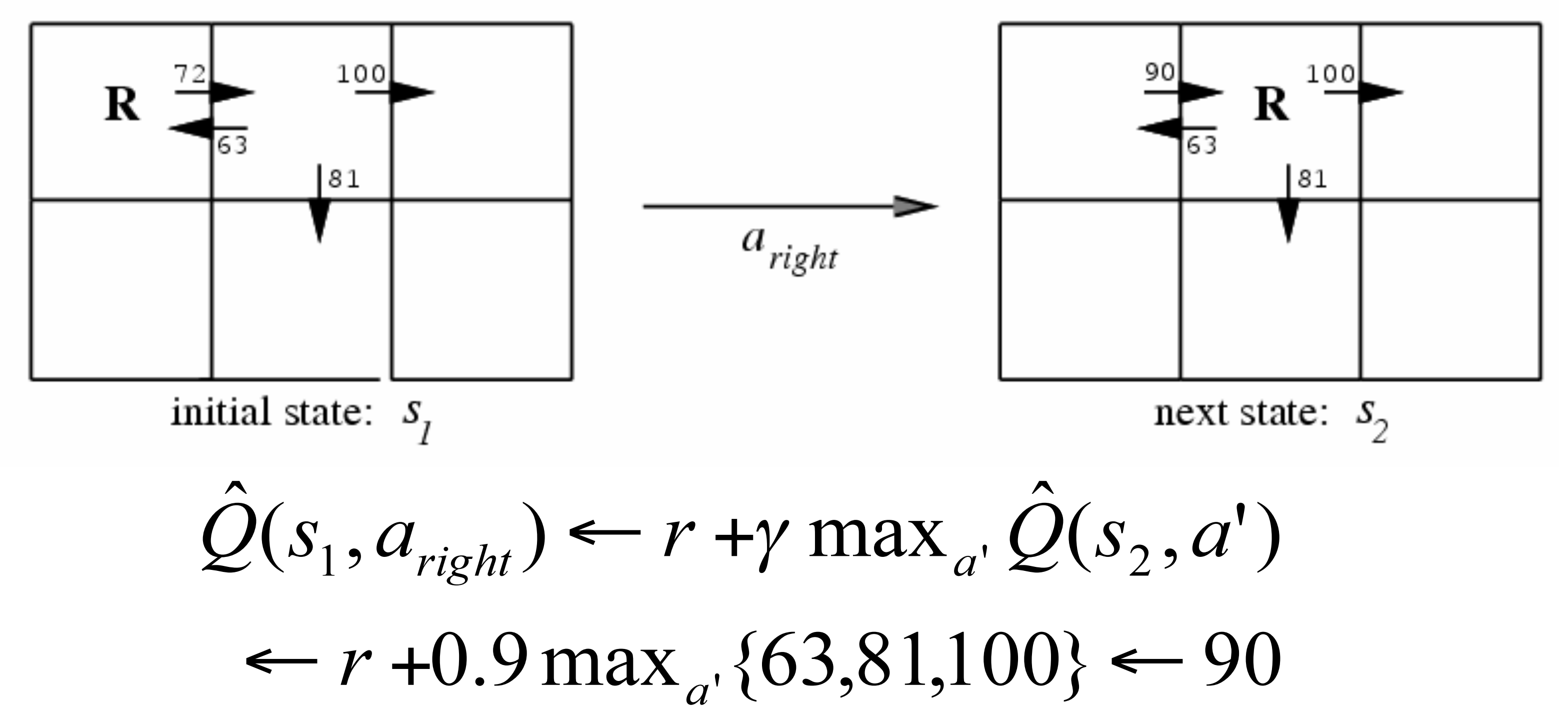
$$\hat{Q}(s,a) \leftarrow r(s,a) + \gamma \max_{a'} \hat{Q}(s',a')$$

- *s* ← *s*'

If get to absorbing state, restart to initial state, and run thru "Do forever" loop until reach absorbing state

Updating Estimated Q

Assume Robot is in state s_1; some of its current estimates of Q are as shown; executes rightward move



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

$$\leftarrow r + 0.9 \max_{a'} \{63, 81, 100\} \leftarrow 90$$

Notice that if rewards are non-negative, then Q^ values only increase from 0, approach true Q

Q Learning: Summary

training set consists of series of intervals (episodes): sequence of (state, action, reward) triples, end at absorbing state

Each executed action a results in transition from state s_i to s_j; algorithm updates Q^(s_i,a) using the learning rule

Intuition for simple grid world, reward only upon entering goal state → Q estimates improve from goal state back

1. All Q^(s,a) start at 0
2. First episode – only update Q^(s,a) for transition leading to goal state
3. Next episode – if go thru this next-to-last transition, will update Q^(s,a) another step back
4. Eventually propagate information from transitions with non-zero reward throughout state-action space