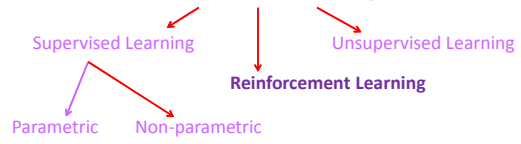
Reinforcement Learning

CSE 446 – Winter 2012

Daniel S. Weld

With slides from Mausam, Carlos Guestrin & Alan Fern

Machine Learning



2

Space of ML Problems

Type of Supervision
(eg, Experience, Feedback)

What is Being Learned?

	Labeled Examples	Reward	Nothing
Discrete Function	Classification		Clustering
Continuous Function	Regression		
Policy	Apprenticeship Learning	Reinforcement Learning	

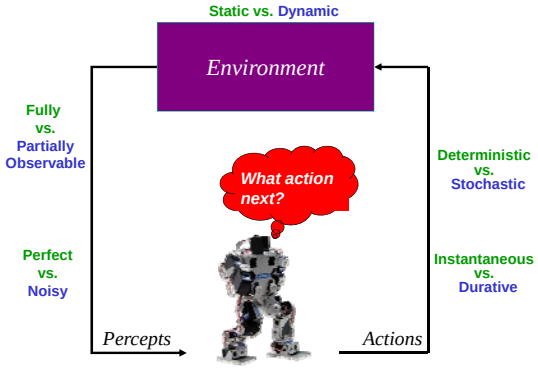
3

Outline

Fri	K-means & Agglomerative Clustering
Mon	Expectation Maximization (EM)
Wed	Principle Component Analysis (PCA)
Fri	Markov Decision Processes (MDPs)
Mon	Reinforcement Learning (RL)
Wed	Instance-Based Learning
Fri	SVMs & Summary

4

Planning Agent



Bellman Equations for MDP_2

- $\langle V, D, S, r, U, s_0, \gamma \rangle$
- Define **V*(s)** {optimal **value**} as the **maximum** expected **discounted reward** from this state.
- V* should satisfy the following equation:

$$V^*(s) = \max_{a \in Ap(s)} \sum_{s' \in \mathcal{S}} \mathcal{P}r(s'|s,a)\left[\mathcal{R}(s,a,s') + \gamma V^*(s')\right]$$

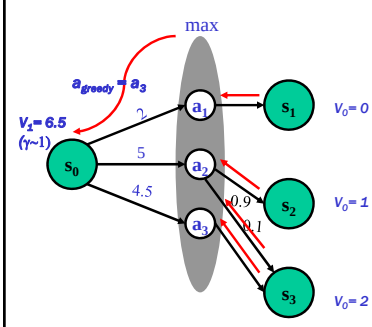
Bellman Backup (MDP_2)

- Given an estimate of V* function (say V_n)
- Backup V_n function at state s
  - calculate a new estimate (V_{n+1}) :

$$Q_{n+1}(s,a) = \sum_{s' \in \mathcal{S}} Pr(s'|s,a)\left[\mathbf{U}(s,a,s') + \gamma V_n(s')\right]$$
$$V_{n+1}(s) = \max_{a \in Ap(s)} [Q_{n+1}(s,a)]$$

- $Q_{n+1}(s,a)$: value/cost of the strategy:
  - execute action a in s, execute π_n subsequently
  - $\pi_n = \text{argmax}_{a \in Ap(s)} Q_n(s,a)$

Bellman Backup



$Q_1(s,a_1) = 2 + \gamma\, 0$
~ 2

$Q_1(s,a_2) = 5 + \gamma\, 0.9 \times 1 + \gamma\, 0.1 \times 2$
~ 6.1

$Q_1(s,a_3) = 4.5 + \gamma\, 2$
~ 6.5

Value iteration [Bellman'57]

- assign an arbitrary assignment of V_0 to each state.
- repeat
  - for all states s
    - compute $V_{n+1}(s)$ by Bellman backup at s. ← Iteration n+1
- until $\max_s |V_{n+1}(s) - V_n(s)| < \varepsilon$ ← Residual(s)

Policy Computation

Optimal policy is stationary and time-independent.

$$\pi^*(s) = \text{argmax}_{a \in Ap(s)} Q^*(s,a)$$
$$= \text{argmax}_{a \in Ap(s)} \sum_{s' \in \mathcal{S}} \mathcal{P}r(s'|s,a)\left[\mathbf{U}(s,a,s') + \gamma V^*(s')\right]$$

Asynchronous Value Iteration

- States may be backed up in any order
  - instead of an iteration by iteration
- As long as all states backed up infinitely often
  - Asynchronous Value Iteration converges to optimal

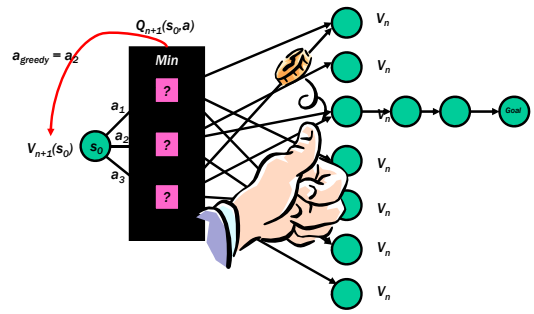
Asynch VI: Prioritized Sweeping

- Why backup a state if values of successors same?
- Prefer backing a state
  - whose successors had most change
- Priority Queue of (state, expected change in value)
- Backup in the order of priority
- After backing a state update priority queue
  - for all predecessors

Asynch VI: Real Time Dynamic Programming
[Barto, Bradtke, Singh'95]

- Trial: simulate greedy policy starting from start state; perform Bellman backup on visited states
- RTDP: repeat Trials until value function converges

RTDP Trial



Comments

- Properties
  - if all states are visited infinitely often then $V_n \rightarrow V^*$
- Advantages
  - Anytime: more probable states explored quickly
- Disadvantages
  - complete convergence can be slow!

Reinforcement Learning

- Still have an MDP
  - Still looking for policy π
- New twist: don't know S r or U
  - Don't know what actions do
  - Nor which states are good!
- Must actually try out actions to learn

Applications

- Robotic control
  - helicopter maneuvering, autonomous vehicles
  - Mars rover - path planning, oversubscription planning
  - elevator planning
- Game playing - backgammon, tetris, checkers
- Neuroscience
- Computational Finance, Sequential Auctions
- Assisting elderly in simple tasks
- Spoken dialog management
- Communication Networks – switching, routing, flow control
- War planning, evacuation planning

Formalizing the reinforcement learning problem

- Given a set of states **X** and actions **A**
- Interact with world at each time step *t*:
  - world gives state $\mathbf{x}_t$ and reward r_t
  - you give next action $\mathbf{a}_t$
- **Goal**: (quickly) learn policy that (approximately) maximizes long-term expected discounted reward

18

The "Credit Assignment" Problem

I'm in state 43,	reward = 0,	action = 2
" " " 39,	" = 0,	" = 4
" " " 22,	" = 0,	" = 1
" " " 21,	" = 0,	" = 1
" " " 21,	" = 0,	" = 1
" " " 13,	" = 0,	" = 2
" " " 54,	" = 0,	" = 2
" " " 26,	" = **100**,	

Yippee! I got to a state with a big reward!

But which of my actions along the way actually helped me get there??

This is the Credit Assignment problem.

19

Exploration-Exploitation tradeoff

- You have visited part of the state space and found a reward of 100
  - is this the best you can hope for???
- **Exploitation**: should I stick with what I know and find a good policy w.r.t. this knowledge?
  - at risk of missing out on a better reward somewhere
- **Exploration**: should I look for a region with more reward?
  - at risk of wasting time & getting some negative reward

20

Two main reinforcement learning approaches

- Model-based approaches:
  - explore environment, then learn model (P(**x**'|**x**,**a**) and R(**x**,**a**)) (almost) everywhere
  - use model to plan policy, MDP-style
  - approach leads to strongest theoretical results
  - often works well when state-space is manageable
- Model-free approach:
  - don't learn a model; learn value function or policy directly
  - weaker theoretical results
  - often works well when state space is large

21

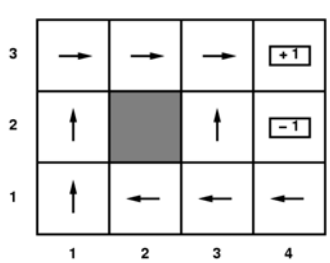
Passive vs. Active learning

- Passive learning
  - The agent has a fixed policy and tries to learn the utilities of states by observing the world go by
  - Analogous to policy evaluation
  - Often serves as a component of active learning algorithms
  - Often inspires active learning algorithms
- Active learning
  - The agent attempts to find an optimal (or at least good) policy by acting in the world
  - Analogous to solving the underlying MDP, but without first being given the MDP model

22

Example: Passive RL

- Suppose given a stationary policy (shown by arrows)
  - Actions can stochastically lead to unintended grid cell
- Want to determine how good it is



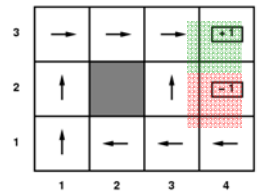
23

Objective: Value Function

3	0.812	0.868	0.918	+1
2	0.762		0.660	−1
1	0.705	0.655	0.611	0.388
	1	2	3	4

24

Passive RL



- Estimate $V^{\pi}(s)$
- Not given
  - transition matrix, nor
  - reward function!
- Follow the policy for many epocns giving training sequences:

(1,1)→(1,2)→(1,3)→(1,2)→(1,3)→(2,3)→(3,3) →(3,4) +1
(1,1)→(1,2)→(1,3)→(2,3)→(3,3)→(3,2)→(3,3)→(3,4) +1
(1,1)→(2,1)→(3,1)→(3,2)→(4,2) -1

- Assume that after entering +1 or -1 state the agent enters zero reward terminal state
  - So we don't bother showing those transitions

25

Approach 1: Direct Estimation
(Model Free)

- Direct estimation (also called Monte Carlo)
  - Estimate $V^{\pi}(s)$ as average total reward of epochs containing s (calculating from s to end of epoch)
- ***Reward to go*** of a state s = the sum of the (discounted) rewards from that state until a terminal state is reached
- Key: use observed ***reward to go*** of the state as the direct evidence of the actual expected utility of that state
- Averaging the reward-to-go samples will converge to true value at state

26

Direct Estimation

- Converges very slowly to correct utilities values (requires a lot of epoch sequences)
- Doesn't exploit Bellman constraints on policy values

$$V^{\pi}(s) = R(s) + \beta \sum_{s'} T(s,a,s')V^{\pi}(s')$$

- Will consider value estimates that badly violate property.

How can we incorporate the Bellman constraints?

27

Approach 2: Adaptive Dynamic Programming (ADP)

- ADP is a model based approach
  - Follow the policy for awhile
  - Estimate transition model based on observations
  - Learn reward function
  - Use estimated model to compute utility of policy

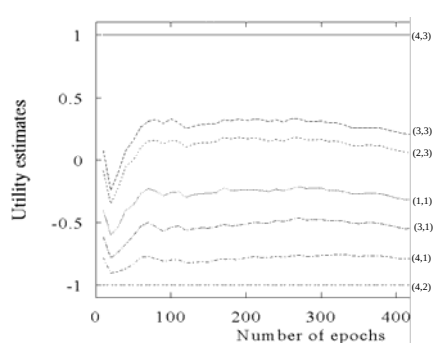
$$V^{\pi}(s) = R(s) + \beta \sum_{s'} T(s,a,s')V^{\pi}(s')$$

learned

- How can we estimate transition model T(s,a,s')?
  - Simply the fraction of times we see s' after taking a in state s.
  - NOTE: Can bound error with Chernoff bounds if we want

28

ADP learning curves



29

Active Reinforcement Learning

- So far, we've assumed agent ***has*** a policy
  - We just learned how good it is
- Now, suppose agent must learn a good policy (ideally optimal)
  - While acting in uncertain world

30

Naïve Model-Based Approach

1. Act Randomly for a (long) time
   - Or systematically explore all possible actions
2. Learn
   - Transition function
   - Reward function
3. Use value iteration, policy iteration, …
4. Follow resulting policy thereafter.

Will this work? Yes (if we do step 1 long enough and there are no "dead-ends")

Any problems? We will act randomly for a long time before exploiting what we know.

31

Active Reinforcement Learning

- So far, we've assumed agent ***has*** a policy
  - We just learned how good it is
- Now, suppose agent must learn a good policy (ideally optimal)
  - While acting in uncertain world

32

Naïve Model-Based Approach

1. Act Randomly for a (long) time
   - Or systematically explore all possible actions
2. Learn
   - Transition function
   - Reward function
3. Use value iteration, policy iteration, …
4. Follow resulting policy thereafter.

Will this work? Yes (if we do step 1 long enough and there are no "dead-ends")

Any problems? We will act randomly for a long time before exploiting what we know.

33

Revision of Naïve Approach

1. Start with initial (uninformed) model
2. Solve for optimal policy given current model (using value or policy iteration)
3. Execute action suggested by policy in current state
4. Update estimated model based on observed transition
5. Goto 2

This is just ADP but we follow the greedy policy suggested by current value estimate

Will this work? No. Can get stuck in local minima. What can be done?

34

Exploration versus Exploitation

- Two reasons to take an action in RL
  - **Exploitation**: To try to get reward. We exploit our current knowledge to get a payoff.
  - **Exploration**: Get more information about the world. How do we know if there is not a pot of gold around the corner.
- To explore we typically need to take actions that do not seem best according to our current model.
- Managing the trade-off between exploration and exploitation is a critical issue in RL
- Basic intuition behind most approaches:
  - Explore more when knowledge is weak
  - Exploit more as we gain knowledge

35

ADP-based (model-based) RL

1. Start with initial model
2. Solve for optimal policy given current model (using value or policy iteration)
3. Take action according to an explore/exploit policy (explores more early on and gradually uses policy from 2)
4. Update estimated model based on observed transition
5. Goto 2

This is just ADP but we follow the explore/exploit policy

Will this work? Depends on the explore/exploit policy. Any ideas?

36

Explore/Exploit Policies

- Greedy action is action maximizing estimated Q-value

$$Q(s,a) = R(s) + \beta \sum_{s'} T(s,a,s')V(s')$$

  - where V is current optimal value function estimate (based on current model), and R, T are current estimates of model
  - Q(s,a) is the expected value of taking action a in state s and then getting the estimated value V(s') of the next state s'
- Want an exploration policy that is greedy in the limit of infinite exploration (GLIE)
  - Guarantees convergence
- GLIE Policy 1
  - On time step t select random action with probability p(t) and greedy action with probability 1-p(t)
  - p(t) = 1/t will lead to convergence, but is slow

37

Explore/Exploit Policies

- GLIE Policy 1
  - On time step t select random action with probability p(t) and greedy action with probability 1-p(t)
  - p(t) = 1/t will lead to convergence, but is slow
- In practice it is common to simply set p(t) to a small constant ε (e.g. ε=0.1 or ε=0.01)
  - Called ε-greedy exploration

38

Alternative Model-Based Approach: Optimistic Exploration

1. Start with initial model
2. Solve for "optimistic policy"
   (uses optimistic variant of value iteration)
   (inflates value of actions leading to unexplored regions)
3. Take greedy action according to optimistic policy
4. Update estimated model
5. Goto 2

Basically act as if all "unexplored" state-action pairs are maximally rewarding.

39

Optimistic Exploration

- Recall that value iteration iteratively performs the following update at all states:

$$V(s) \leftarrow R(s) + \beta \max_a \sum_{s'} T(s,a,s')V(s')$$

  - Optimistic variant adjusts update to make actions that lead to unexplored regions look good
- **Optimistic VI:** assigns highest possible value V^{max} to any state-action pair that has not been explored enough
  - Maximum value is when we get maximum reward forever

$$V^{\max} = \sum_{t=0}^{\infty} \beta^t R^{\max} = \frac{R^{\max}}{1-\beta}$$

- What do we mean by "explored enough"?
  - N(s,a) > N_e, where N(s,a) is number of times action a has been tried in state s and N_e is a user selected parameter

40

Optimistic Value Iteration

$$V(s) \leftarrow R(s) + \beta \max_a \sum_{s'} T(s,a,s')V(s')$$ Standard VI

- Optimistic value iteration computes an optimistic value function V^+ using following updates

$$V^+(s) \leftarrow R(s) + \beta \max_a \begin{cases} V^{\max}, & N(s,a) < N_e \\ \sum_{s'} T(s,a,s')V^+(s'), & N(s,a) \geq N_e \end{cases}$$

- The agent will behave initially as if there were wonderful rewards scattered all over around– **optimistic** .
- But after actions are tried enough times we will perform standard "non-optimistic" value iteration

41

Summary RL

- Bellman Equation
- Value iteration
- Credit assignment problem
- Explortion / exploitation tradeoff
  - Greedy in limit of infinite exploration
  - Optomistic exploration