Reinforcement Learning

CSE 473

Heli Flying

Review: MDPs

S = set of states set (|S| = n)

A = set of actions (|A| = m)

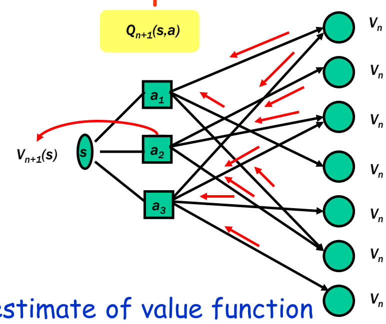
Pr = transition function Pr(s,a,s')
represented by set of m n x n stochastic matrices
each defines a distribution over SxS

R(s) = bounded, real-valued reward fun
represented by an n-vector

Goal for an MDP

- Find a ***policy*** which:
  maximizes ***expected discounted reward***
  over an ***infinite horizon***
  for a ***fully observable***
  Markov decision process.

Bellman Backup



Improve estimate of value function

Vt+1(s) = R(s) +
$\text{Max}_{a\epsilon A}$ {c(a)+ ***Expected future reward Aver'gd over dest states***

Value Iteration

- Assign arbitrary values to each state
  (or use an admissible heuristic).
- Iterate over all states
  Improving value funct via Bellman Backups
- Stop the iteration when converges
  (V_t approaches V^* as $t \rightarrow \infty$)
- Dynamic Programming

How is learning to act possible when…

- Actions have non-deterministic effects
  Which are initially unknown
- Rewards / punishments are infrequent
  Often at the end of long sequences of actions
- Learner must decide what actions to take
- World is large and complex

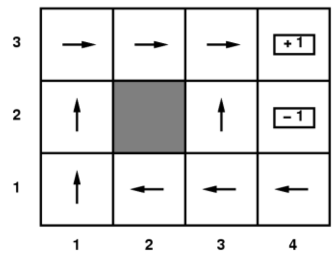
Naïve Approach

1. Act Randomly for a while
   (Or systematically explore all possible actions)
2. Learn
   Transition function
   Reward function
3. Use value iteration, policy iteration, …

Problems?

Example:

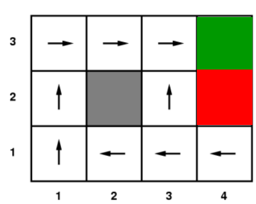
- Suppose given policy
- Want to determine how good it is



Objective: Value Function

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

Passive RL



- Given policy π,
  estimate $U^\pi(s)$
- Not given
  transition matrix, nor
  reward function!
- Epochs: training sequences

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

Approach 1

- Direct estimation
  Estimate $U^\pi(s)$ as average total reward of epochs containing s (calculating from s to end of epoch)
- Pros / Cons?

Requires huge amount of data
doesn't exploit **Bellman constraints!**

Expected utility of a state =
its own reward +
expected utility of successors

Temporal Difference Learning

Do backups on a per-action basis
Don't try to estimate entire transition function!
For each transition from s to s', update:

α = *Learning rate*

$$U^\pi(s) \leftarrow U^\pi(s) + \alpha(R(s) + \gamma U^\pi(s') - U^\pi(s))$$

γ = *Discount rate*

Notes

- Once U is learned, updates become 0:

$$\text{when } U^\pi(s) = R(s) + \gamma U^\pi(s')$$

Adjusts state to 'agree' with observed successor
- Not *all* possible successors

Doesn't require M, model of transition function

Notes II

- "TD(0)"
  One step lookahead

$$U^\pi(s) \leftarrow U^\pi(s) + \alpha(R(s) + \gamma U^\pi(s') - U^\pi(s))$$

Can do 2 step, 3 step…

TD(λ)

- Or, … take it to the limit!
- Compute weighted average of all future states

$$U^\pi(s_t) \leftarrow U^\pi(s_t) + \alpha(R(s_t) + \gamma U^\pi(s_{t+1}) - U^\pi(s_t))$$

becomes

$$U^\pi(s_t) \leftarrow U^\pi(s_t) + \alpha(R(s_t) + \gamma(1-\lambda)\sum_{i=0}^{\infty} \lambda^i U^\pi(s_{t+i+1}) - U^\pi(s_t))$$

weighted average

- Implementation
  Propagate current weighted TD onto past states
  Must memorize states visited from start of epoch

Q-Learning

- Version of TD-learning where
  instead of learning value funct on states
  we learn funct on [state,action] pairs

$$U^{\pi}(s) \leftarrow U^{\pi}(s) + \alpha(R(s) + \gamma U^{\pi}(s') - U^{\pi}(s))$$

becomes

$$Q(a,s) \leftarrow Q(a,s) + \alpha(R(s) + \gamma \max_{a'} Q(a',s') - Q(a,s))$$

- [Helpful for model-free policy learning]

Part II

- So far, we've assumed agent ***had*** policy
- Now, suppose agent must learn it
  While acting in uncertain world

Active Reinforcement Learning

Suppose agent must make policy while learning

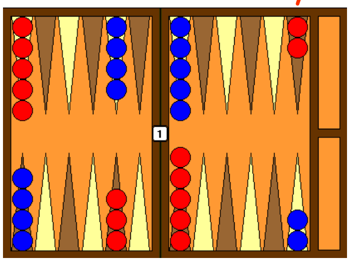
First approach:
Start with arbitrary policy
Apply Q-Learning
New policy:
In state *s*,
Choose action *a* that maximizes *Q(a,s)*

Problem?

Utility of Exploration

- Too easily stuck in non-optimal space
  "Exploration versus exploitation tradeoff"
- Solution 1
  With fixed probability perform a random action
- Solution 2
  Increase est expected value of infrequent states

~Worlds Best Player



- Neural network with 80 hidden units
  - Used computed features
- 300,000 games against self

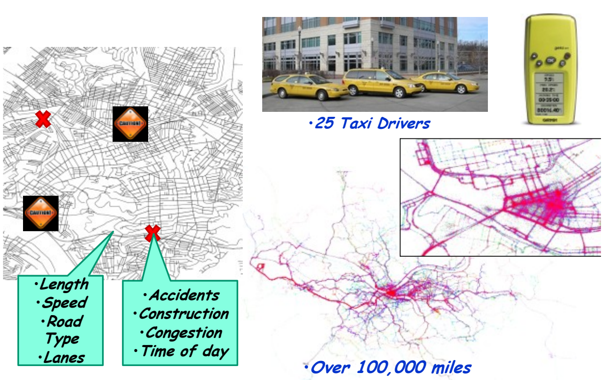
Imitation Learning

- What can you do if you have a teacher?
- People are often …
  - … good at demonstrating a system
  - … bad at specifying exact rewards / utilities

- Idea: Learn the reward function that best "explains" demonstrated behavior
- That is, learn reward such that demonstrated behavior is optimal wrt. It
- Also called apprenticeship learning, inverse RL

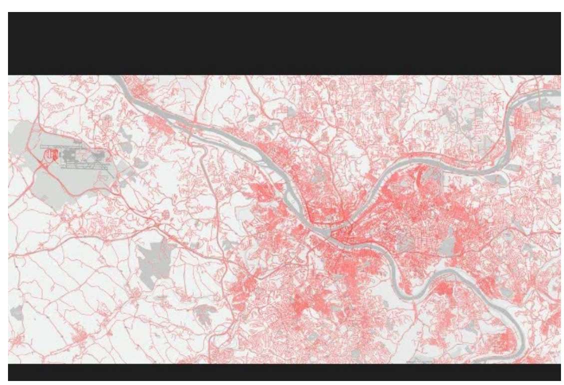
Courtesy of B. Ziebart

Data Collection



Courtesy of B. Ziebart

Destination Prediction



Heli Airshow

Courtesy of P. Abbeel

Summary

- Use reinforcement learning when
  - Model of world is unknown and/or rewards are delayed
- Temporal difference learning
  - Simple and efficient training rule
- Q-learning eliminates need for explicit T model
- Large state spaces can (sometimes!) be handled
  - Function approximation, using linear functions or neural nets