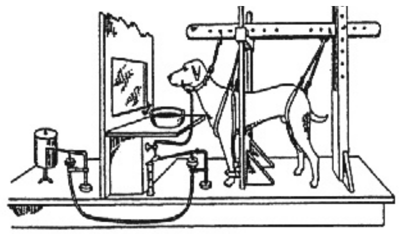
Reinforcement Learning

CSE 573



Ever Feel Like Pavlov's Poor Dog?

Logistics

- Reading for Wed
  AIMA Ch 20 thru 20.3
- Teams for Project 2
- Midterm
  Typical problems – see AIMA exercises
  In class / takehome
  Open book / closed
  Length

573 Topics

Reinforcement Learning
Supervised Learning
Planning
Logic-Based *Probabilistic*
Knowledge Representation & Inference
Search
Problem Spaces
Agency

Pole Demo

Review: MDPs

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

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

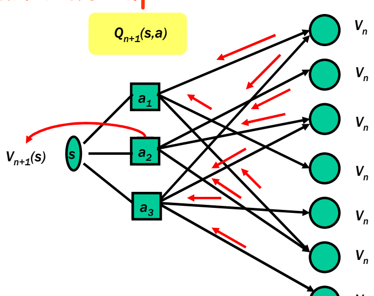
Pr = transition function Pr(s,a,s')
 represented by set of m n x n stochastic matrices (factored into DBNs)
 each defines a distribution over SxS

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

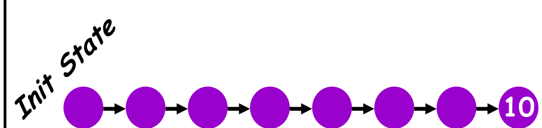
Vt+1(s) = R(s) + Max$_{a\varepsilon A}$ {c(a)+ ***Expected future reward Aver'gd over dest states***

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

Note on Value Iteration

- Order in which one applies Bellman Backups
  Irrelevant!
- Some orders more efficient than others



Action cost = -1. Discount factor, γ = 1

Expand figure

- Initialize all value functions to 0
- First backup sets goal to 10
- Use animation

Policy evaluation

- Given a policy Π:S$\rightarrow$A, find value of each state using this policy.
- $V^{\Pi}(s) = R(s) + c(\Pi(s)) + \gamma[\Sigma_{s' \varepsilon S} Pr(s' | \Pi(s),s)V^{\Pi}(s')]$
- This is a system of linear equations involving |S| variables.

Policy iteration

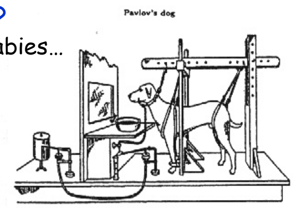
- Start with any policy (Π_0).
- Iterate
  Policy evaluation : For each state find $V^{\Pi_i}(s)$.
  Policy improvement : For each state s, find action a^* that maximizes $Q^{\Pi_i}(a,s)$.
  If $Q^{\Pi_i}(a^*,s) > V^{\Pi_i}(s)$ let $\Pi_{i+1}(s) = a^*$
  else let $\Pi_{i+1}(s) = \Pi_i(s)$
- Stop when $\Pi_{i+1} = \Pi_i$
- Converges faster than value iteration but policy evaluation step is more expensive.

Modified Policy iteration

- Instead of evaluating the actual value of policy by
  - Solving system of linear equations, …
- Approximate it:
  - Value iteration with fixed policy.

Excuse Me…

- MDPs are great, IF…
  - We know the state transition function P(s,a,s')
  - We know the reward function R(s)
- But what if we don't?
  - Like when we were babies…
  - And like our dog…



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?

RL Techniques

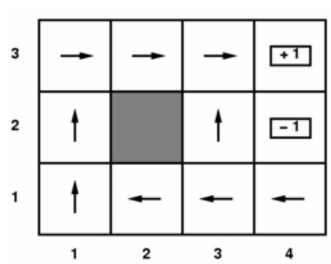
1. Passive RL
2. Adaptive Dynamic Programming
3. Temporal-difference learning
   - Learns a utility function on states
     - treats the difference between expected / actual reward as an error signal, that is propagated backward in time

Concepts

- Exploration functions
  - Balance exploration / exploitation
- Function approximation
  - Compress a large state space into a small one
  - Linear function approximation, neural nets, …
  - Generalization

Example:

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



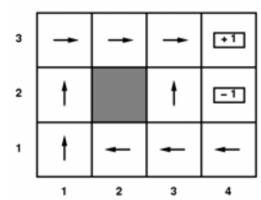
Objective: Value Function

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

Just Like Policy Evaluation

- Except…?

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

Approach 2

Adaptive Dynamic Programming
Requires fully observable environment
Estimate transition function *M* from training data
Solve Bellman eqn w/ modified policy iteration

$$U^\pi = R(s) + \gamma \sum_{s'} M^\pi_{s,s'} U^\pi(s')$$

Pros / Cons:

Requires complete observations
Don't usually need value of ***all*** states

Approach 3

- 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:

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

- Similar to ADP
  - Adjusts state to 'agree' with observed successor
    - Not ***all*** possible successors
  - Doesn't require M, model of transition function
  - Intermediate approach: use M to generate "Pseudo experience"

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

Notes III

- Online: update immediately after actions
  - Works even if epochs are infinitely long
- Offline: wait until the end of an epoch
  - Can perform updates in any order
    - E.g. backwards in time
  - Converges faster if rewards come at epoch end
    - Why?!?
- ADP Prioritized sweeping heuristic
  - **Bound # of value iteration steps (small Δ ave)**
  - Only update states whose successors have $\Uparrow\Delta$
  - **Sample complexity ~ADP**
  - **Speed ~ TD**

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]

Baseball

CMU Robotics

Puma arm learning to throw
training involves 100 throws
(video is lame; learning is good)

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

$U^+(s) \leftarrow R(s) + \gamma \max_a \mathbf{f}(\Sigma_{s'} P(s' \mid a,s)\, U^+(s'),\ N(a,s))$

Properties of f(u, n) ??

If $n > N_e$	U	i.e. normal utility
Else,	R^+	i.e. max possible reward

Part III

- Problem of large state spaces remain
  Never enough training data!
  Learning takes too long
- What to do??

Function Approximation

- Never enough training data!
  Must generalize what learning to new situations
- Idea:
  Replace large state table by a smaller, parameterized function
  Updating the value of state will change the value assigned to many other similar states

Linear Function Approximation

- Represent *U(s)* as a weighted sum of features (basis functions) of *s*

$$\hat{U}_\theta(s) = \theta_1 f_1(s) + \theta_2 f_2(s) + \ldots + \theta_n f_n(s)$$

- Update each parameter separately, *e.g:*

$$\theta_i \leftarrow \theta_i + \alpha(R(s) + \gamma \hat{U}_\theta(s') - \hat{U}_\theta(s)) \frac{\partial \hat{U}_\theta(s)}{\partial \theta_i}$$

Example

- $U(s) = \theta_0 + \theta_1 x + \theta_2 y$
- Learns good approximation

						10

But What If…

- $U(s) = \theta_0 + \theta_1 x + \theta_2 y + \theta_3 z$
- Computed Features

$z = \sqrt{(x_g - x)^2 + (y_g - y)^2}$

			10			

Neural Nets

- *Can create powerful function approximators*
  - Nonlinear
  - Possibly unstable
- For TD-learning, apply difference signal to neural net output and perform back-propagation

Policy Search

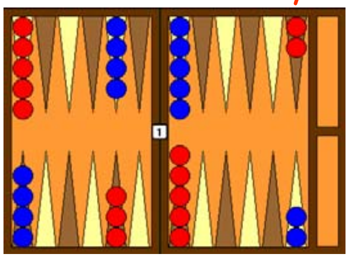
- Represent policy in terms of Q functions
- Gradient search
  - Requires differentiability
  - Stochastic policies; softmax
- Hillclimbing
  - Tweak policy and evaluate by running
- Replaying experience

Walking Demo

UT AustinVilla

Aibo walking – before & after 1000 training instances (across field)
… yields fastest known gait!

~Worlds Best Player

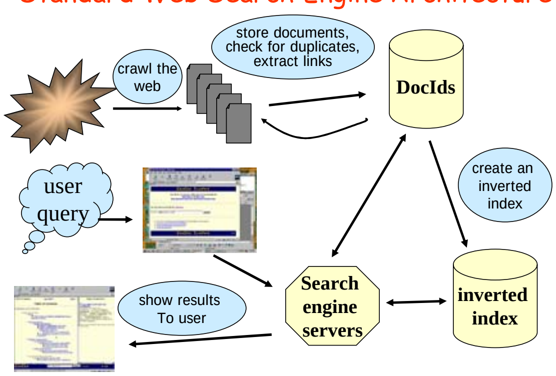


- Neural network with 80 hidden units
  - Used computed features
- 300,000 games against self

Applications to the Web
Focused Crawling

- Limited resources
  - Fetch most ***important*** pages first
- Topic specific search engines
  - Only want pages which are ***relevant*** to topic
- Minimize stale pages
  - Efficient re-fetch to keep index timely
  - How track the rate of change for pages?

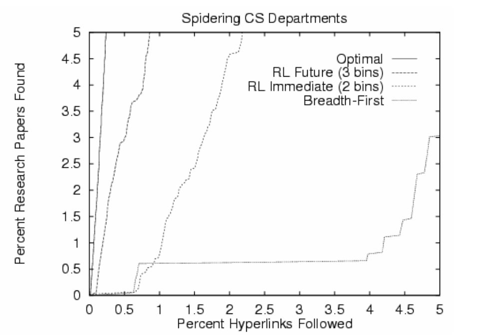
Standard Web Search Engine Architecture



Slide adapted from Marty Hearst / UC Berkeley]

Performance

Rennie & McCallum (ICML-99)



Methods

- Agent Types
  - Utility-based
  - Action-value based (Q function)
  - Reflex
- Passive Learning
  - Direct utility estimation
  - Adaptive dynamic programming
  - Temporal difference learning
- Active Learning
  - Choose random action $1/n^{th}$ of the time
  - Exploration by adding to utility function
  - Q-learning (learn action/value f directly – model free)
- Generalization
  - Function approximation (linear function or neural networks)
- Policy Search
  - Stochastic policy repr / Softmax
  - Reusing past experience

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