Reinforcement Learning
Sampler

CS 536: Machine Learning
Littman (Wu, TA)
Administration

• Programming projects returned
• Final project feedback available.
• Project paper should be in NIPS format, 5 pages: http://www-2.cs.cmu.edu/Groups/NIPS/NIPS2001/formatting.html
• The hard part will be making it only 5 pages.
• Include at least two references and one figure or table.
• Make a comparison: What question are you asking and how does your work address it?
Recommended Reading


Subtleties & Ongoing Research

- Replace $\hat{Q}$ table with neural net or other generalizer
- Handle case where state only partially observable
- Design optimal exploration strategies
- Extend to continuous action, state
- Learn and use $\delta: S \times A \rightarrow S$
- Relationship to dynamic programming and heuristic search
Value Function Approximation

Central problem: Learning $Q: S \times A \rightarrow \mathbb{R}$.

Again, $Q(s,a)$ is the expected total discounted reward for taking $a$ from $s$, then behaving optimally (maximizing cumulative discounted expected reward).

How do we represent $Q$?
Representing $Q$

Many options:

• table
• decision tree (Kaelbling & Chapman)
• neural network (Tesauro)
• nearest neighbor (Atkeson & Moore)
• even Naïve Bayes! (McCallum)
Training Data Issues

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

- input: $s$, $a$; output: best guess

Non-stationary
- Target improves (?!?) with experience
- Need to downweight or toss old data

Bootstrapped
- Training value depends on current estimate for Q function
- Errors can be magnified through generalization!
Simple Scenario: Fitted VI

Trying to find minimum cost to goal. Randomly generate a set of states $D$. Estimates proceed in rounds:

- Let $J_i(s)$ be our estimate for the optimal cost in round $i$.
- Generate training data for round $i+1$ by $J_{i+1}(s) \approx \min_a (r(s,a) + E_s [J(s')])$, $\forall \ s \in D$
- Use supervised learning on $<s, J_{i+1}(s)>$. 
What Happens?

If $J_i$ is a table and $D=S$, the algorithm is known as value iteration (a form of dynamic programming, described by Bellman) and it converges to the optimal cost to goal function.

For many other function approximators (“neuro–dynamic programming”), bad things can happen...
Example Gridworld

From Boyan and Moore
Discretized Value Iteration

Value function rises away from the goal.
Quadratic Regression VFA

Value function diverges: can’t fit intermediate value functions.
Another Example

Value function discontinuous: rock up the hill
Backprop Flops

Complete collapse!
Boyan and Moore suggest “safe” (non-bootstrapping) training sets to avoid magnifying errors.
A Safe VFA Method

Averagers (Gordon 1995):

- Let $D$ be a set of (anchor) states.
- $\hat{V}(s) = \max_a (r(s,a) + \mathbb{E}_{s'}[\hat{V}(s')])$
- $\hat{V}(s) = \sum_{s' \in N(s,k,D)} \frac{V(s')}{k}$

where $N(s,k,D)$ is the $k$ “closest” states to $s$ in $D$ (a la $k$NN).

Can’t diverge, can prove bounds on accuracy of approximation.

**Proof:** Like a composite MDP formed from true transitions from $s$ in $D$ and uniform transitions to the $k$ NN that result.
Argument

\[ V(s) = \max_a (R(s,a) + \sum_s T(s,a,s') \hat{V}(s')) \]
\[ \hat{V}(s') = \sum_{s'' \in N(s',k,D)} \hat{V}(s'') / k \]

Let \( T(s,a,s'') = \sum_s T(s,a,s') / k, \)
if \( s'' \in N(s',k,D). \)

Solving the MDP with \( \hat{T} \) results in the same solution as the averager.
Bandit Problems

Consider the 2-armed bandit problem. We want to maximize our “take”.
Let’s say arm A has a return of $a=\frac{2}{5}$ and arm B has a return of $b=\frac{4}{5}$.

- Gibb’s (probability match) strategy?
- Bayes optimal strategy?
Exploration vs. Exploitation

In the learning setting, we don’t know the payoffs $a$ and $b$.

We’ve pulled A $n_a$ times ($s_a$ successes) and pulled B $n_b$ times ($s_b$ successes).

$A: \frac{4}{10} = .40$

$B: \frac{1}{3} = .33$

A looks better, but how much do we believe this difference?
Approaches to Exploration

Greedy
- Always choose the max (problem?)

Interval estimation
- Choose higher 95% confidence interval

Gittins index
- Elegant exact solution

Tsetlin automata
- Counts payoffs for each arm

PAC Algorithm
- After polynomial pulls, pick arm with \( \epsilon \) of maximal with probability at least \( 1 - \delta \).
Is This Relevant to RL?

We can hop around, estimate rewards and transitions. What action should we take? Can act to maximize reward given what we seen, or perhaps venture off to learn more about some of the states.
Polynomial Time RL

Let $M$ be a Markov decision process over $N$ states. Let $P(T,\varepsilon, M)$ be the set of all policies that get within $\varepsilon$ of their true return in the first $T$ steps, and that $\text{opt}(P(T,\varepsilon, M))$ is the optimal asymptotic expected undiscounted return achievable in $P(T,\varepsilon, M)$. There exists an algorithm $A$, taking inputs $\varepsilon, \delta, N, T$ and $\text{opt}(P(T,\varepsilon, M))$ such that after a total number of actions and computation time bounded by a polynomial in $1/\varepsilon, 1/\delta, N, T,$ and $R_{\text{max}}$, with probability at least $1-\delta$, the total undiscounted return will be at least $\text{opt}(P(T,\varepsilon, M)) - \varepsilon$. 
Explicit Explore Exploit

(Initialization) Initially, the set $S$ of known states is empty.

(Balanced Wandering) Any time the current state is not in $S$, the algorithm performs balanced wandering.

(Discovery of New Known States) Any time a state $i$ has been visited $m_{\text{known}}$ times during balanced wandering, it enters the known set $S$, and no longer participates in balanced wandering.

(Off–line optimizations) Compute optimal policies for $M_r$ (maximize reward, avoiding unknown states) and $M_d$ (minimize steps to unknown state).

Execute $M_r$ if it is within $\epsilon/2$ of optimal, otherwise $M_d$ is likely to quickly discover a state out of $S$. 
non-Markovian Examples

Can you solve them?
Recall MDP:

- finite set of states $S$
- set of actions $A$
- at each discrete time agent observes state $s_t \in S$ and chooses action $a_t \in A$
- receives reward $r_t$, and state changes to $s_{t+1}$
- Markov assumption: $s_{t+1} = \delta(s_t, a_t)$ and $r_t = r(s_t, a_t)$
  - $r_t$ and $s_{t+1}$ depend only on current state and action
  - functions $\delta$ and $r$ may be nondeterministic
  - functions $\delta$ and $r$ not necessarily known to agent
Partially Observable MDPs

Same as MDP, but additional observation function $\omega$ that translates the state into what the learner can observe: $o_t = \omega(s_t)$

Transitions and rewards still depend on state, but learner only sees a “shadow”.

How can we learn what to do?
State Approaches to POMDPs

Q learning (dynamic programming)

states:
- observations
- short histories
- learn POMDP model: most likely state
- learn POMDP model: information state
- learn predictive model: predictive state
- experience as state

Advantages, disadvantages?
Learning a POMDP

Input: history (action–observation seq).
Output: POMDP that “explains” the data.
EM, iterative algorithm (Baum et al. 70; Chrisman 92)

E: Forward-backward

POMDP model

state occupation probabilities

M: Fractional counting
EM Pitfalls

Each iteration increases data likelihood.

Local maxima. (Shatkay & Kaelbling 97; Nikovski 99)

Rarely learns good model.

Hidden states are truly unobservable.
Information State

Assumes:

• objective reality
• known “map”

Also belief state: represents location. Vector of probabilities, one for each state.

Easily updated if model known. (Ex.)
Plan with Information States

Now, learner is 50% here and 50% there instead of in any particular state.

**Good news**: Markov in these vectors

**Bad news**: States continuous

**Good news**: Can be solved

**Bad news**: ...slowly, typically

**More bad news**: Model is approximate!
Predictions as State

Idea: Key information from distance past, but never too far in the future. (Littman et al. 02)

start at blue: down red (left red)_{odd} up _?  

history: forget  

predict: up blue? left red?
Experience as State

Nearest sequence memory (McCallum 1995)
Relate current episode to past experience.
$k$ longest matches considered to be the same for purposes of estimating value and updating.
Current work: Extend TD($\lambda$), extend notion of similarity (allow for soft matches, sensors)
Classification Dialog

User to travel to Roma, Torino, or Merino?
Actions:
- QC (What city?),
- QR, QT, QM (Going to $X$?),
- R, T, M (I think $X$).
Observations:
- Yes, no (more reliable), R, T, M (T/M confusable).
Objective:
- Reward for correct class, cost for questions.
Optimal plan varies with priors ($S_R = S_M$).
$S_T = 0.00$
\( S_T = 0.02 \)
$S_T=0.22$
$S_T = 0.76$
$S_T=0.90$
Current Project: Diagnosis

Can’t reach | How come?
• DNS failure
• Web site is down
• route is down
• etc.

Use known tests to find out quickly and repair the problem.
Learned Output

(Joint work with Nishkam and others)
Real Reinforcement Learning

Classical RL:
• complete observability
• enumerated observations
Most real applications: complex sensors.

Research Opportunity:
• Exciting new area of study
• Solve the AI problem.
Find the Ball
Wrap Up

Reinforcement learning: Get the right answer without being told. Hard, less developed than supervised learning.