Reinforcement Learning
Statistical Machine Learning Overview

Douglas Aberdeen
Canberra Node, RSISE Building
Australian National University

2nd November 2004
Introduction

What is Reinforcement Learning?
Types of RL

Value-Methods

Model Based
Experience Based
Function Approximation

Partial Observability

Policy-Gradient Methods

Model Based
Experience Based
Reinforcement Learning (RL) in a Nutshell

- RL can learn any function
- RL inherently handles uncertainty
  - Uncertainty in actions (the world)
  - Uncertainty in observations (sensors)
- Directly maximise criteria we care about
- RL copes with delayed feedback
  - Temporal credit assignment problem
Reinforcement Learning (RL) in a Nutshell

- RL can learn any function
- RL inherently handles uncertainty
  - Uncertainty in actions (the world)
  - Uncertainty in observations (sensors)
- Directly maximise criteria we care about
- RL copes with delayed feedback
  - Temporal credit assignment problem
Reinforcement Learning (RL) in a Nutshell

- RL can learn any function
- RL inherently handles uncertainty
  - Uncertainty in actions (the world)
  - Uncertainty in observations (sensors)
- Directly maximise criteria we care about
- RL copes with delayed feedback
  - Temporal credit assignment problem
Reinforcement Learning (RL) in a Nutshell

- RL can learn any function
- RL inherently handles uncertainty
  - Uncertainty in actions (the world)
  - Uncertainty in observations (sensors)
- Directly maximise criteria we care about
- RL copes with delayed feedback
  - Temporal credit assignment problem
RL Can Solve Hard Problems

- Uncertainty = hidden state
- Robot
- Sensor error
- Motor error
- Hidden location
- Perceptual aliasing
- Unknown map = unknown state transition matrix
- Reward
- Goal
- Delayed rewards
- Uncertainty = hidden state

NATIONAL ICT AUSTRALIA
## Examples

### BackGammon: TD-Gammon [2]
- Beat the world champion in individual games
- Can learn things no human ever thought of!
- TD-Gammon opening moves now used by best humans

### Australian Computer Chess Champion [1]
- Australian Champion Chess Player
- RL learns the evaluation function at leaves of min-max search

### Elevator Scheduling
- Crites, Barto 1996
- Optimally dispatch multiple elevators to calls
- Not implemented as far as I know
Partially Observable Markov Decision Processes

- Partial Observability
  - \( \Pr[o|s] \)
  - \( s \)
  - \( o \)

- Value-Methods
  - \( \Pr[s'|s,a] \)
  - \( r(s) \)

- Partially Observable Markov Decision Processes (POMDP)
  - MDP
  - RL

- Policy-Gradient Methods
  - \( \Pr[a|o,w] \)
  - \( w \)
  - \( a \)
RL Axes

- Policy
- Value
- Model Based
- Experience
- DP
- MDP
- POMDP
- Value-Methods
- Partial Observability
- Policy-Gradient Methods
Optimality Criteria

- The value $V(s)$ is a long-term rewards from state $s$
- How do we measure long-term reward??

$$V_\infty(s) = \mathbb{E}_w \left[ \sum_{t=0}^{\infty} r(s_t) | s_0 = s \right]$$

 Ill-conditioned from the decision making point of view

- Sum of discounted rewards

$$V(s) = \mathbb{E}_w \left[ \sum_{t=0}^{\infty} \delta^t r(s_t) | s_0 = s \right]$$

- Finite-horizon

$$V_T(s) = \mathbb{E}_w \left[ \sum_{t=0}^{T-1} r(s_t) | s_0 = s \right]$$
Criteria Continued

- Baseline reward

\[ V_B(s) = \mathbb{E}_w \left[ \sum_{t=0}^{\infty} r(s_t) - \bar{r} \mid s_0 = s \right] \]

- Long-term average is intuitively appealing

\[ \bar{V}(s) = \lim_{T \to \infty} \frac{1}{T} \mathbb{E}_w \left[ \sum_{t=0}^{T-1} r(s_t) \mid s_0 = s \right] \]
Discounted or Average?

**Ergodic MDP**
- Irreducible: all states reachable
- Positive recurrent: finite return times
- Aperiodic: GCD of return times = 1

- If the Markov system is *ergodic* then $\bar{V}(s) = \eta$ for all $s$, i.e., $\eta$ is constant over $s$
- Convert from discounted to long-term average

$$\eta = \mathbb{E}_s \frac{V(s)}{1 - \delta}$$

- We will focus on discounted $V(s)$
Dynamic Programming

- How do we compute $V(s)$?
- Solution to the unique fixed point

$$V^*(s) = r(s) + \delta \sum_{a \in A} \sum_{s' \in S} \Pr[s'|s, a] \Pr[a|s, w] V^*(s')$$

- In matrix form with vectors $V^*$ and $r$:
  - Define stochastic transition matrix for current policy

$$P = \sum_{a \in A} \Pr[s'|s, a] \Pr[a|s, w]$$

- Now

$$V^* = r + \delta P V^*$$
Analytic Solution

\[ V^* = r + \delta PV^* \]
\[ V^* - \delta PV^* = r \]
\[ (I - \delta P)V^* = r \]
\[ Ax = b \]

- Computes \( V(s) \) for fixed policy (fixed \( w \))
- No solution unless \( \delta \in [0, 1) \)
- \( O(|S|^3) \) solution... not feasible
Value Iteration

- Avoid the matrix inverse with the iteration
  \[ V_{t+1}(s) = r(s) + \delta \sum_{a \in A} \sum_{s' \in S} \Pr[s'|s, a] \Pr[a|s, w] V_t(s') \]

- Asymptotically converges to \( V_{\infty} = V^* \)

- Interpretation as \( t \)-step to go cost

Initialise \( V_0 = 0 \)

\[ V_1(s) = r(s) \]

\[ V_2(s) = r(s) + \delta \sum_{a \in A} \sum_{s' \in S} \Pr[s'|s, a] \Pr[a|s, w] V_1(s') \]

\[ V_3(s) = r(s) + \delta \sum_{a \in A} \sum_{s' \in S} \Pr[s'|s, a] \Pr[a|s, w] V_2(s') \]

\[ \ldots \]
Value Iteration Continued

- We really want is the value of the optimal policy
- Optimal policy chooses the maximising action

\[ V_{t+1}(s) = r(s) + \delta \max_a \sum_{s' \in S} \Pr[s'|s, a] V_t(s') \]

- The maximising action will change as \( V(s) \) evolves
- Value iteration repeats until the \( \| V_{t+1} - V_t \| < \epsilon \)
- The final policy is given by

\[ \Pr[\arg\max_a \sum_{s' \in S} \Pr[s'|s, a] V_t(s')|s] = 1 \]

- The parameters \( w \) are the table mapping \( s \) to an \( a \)
- What should we do about continuous state spaces?
Value Iteration Convergence

\[ \| V_T - V^* \|_\infty \leq \frac{2\delta^{T+1}}{(1 - \delta)^2} \| r \|_\infty \]

- Each iteration is \( O(|S|^2) \)
- Could take a while!
- Less than \( O(|S|^3) \)?
Policy Iteration

- Actually pre-dates value iteration

Policy Iteration Algorithm

1. Pick an initial deterministic policy (state to action table) \( w \)
2. Evaluate \( V(s) \) for policy, exact or approximate
3. Compute new policy \( w \) by choosing maximising actions
4. If policy has changed, goto 2
5. Return (optimal) policy \( w \)

- Can use exact policy evaluation or approximate
- Similar convergence to value iteration
Policy Iteration, Pros and Cons

- **Pros**
  - Stopping criteria: continue until policy does not change
  - If exact evaluation by $V^* = (I - \delta P)^{-1} r$, policy is optimal

- **Cons**
  - Wastes effort computing true value of mostly non-optimal policies

- Somewhat reminiscent of the EM algorithm
Further computation wasted

\[ t < |S|? \]

\[ \|V^*(s)\| \]

\[ \|V_t(s)\| \]
Our Progress...

Value & Pol Iteration

MDP

POMDP

Model Based Experience

Value Policy

Value-Methods

Partial Observability

Policy-Gradient Methods
Recall that \( P = \sum_{a \in A} \Pr[s' | s, a] \Pr[a | s, w] \)

Q. What happens if we don’t know \( P \) (dynamics)?

A:
- Estimate \( P \) from experience, apply value/policy iteration
- Monte-Carlo estimation of \( V^* \)
- Generate a trajectory \( \{s_0, a_0, s_1, a_1, s_2, a_2, \ldots \} \) according to some policy
- Use trajectory to learn model, or do Monte-Carlo estimate
- Monte-Carlo implicitly learns \( P \) during value estimation
Temporal Differences (TD)[?] 

**Temporal difference**

- Key observation is that direction of $V(s)$ error is given by

$$\Delta = V_t \underbrace{- (r(s) + \delta V_t(s'))}_{\text{estimate after one-step update}}$$

- This is a temporal difference

- Think of $\Delta$ as one possible branch of the expectation

$$V^*(s) = r(s) + \delta \sum_{a \in A} \sum_{s' \in S} \Pr[s'|s, a] \Pr[a|s, w] V^*(s')$$

- Observing many branches gives back expectation
Temporal Differences Cont.

- Update in the direction of $\Delta$ yields estimate $V_{t+1}$

$$V_{t+1}(s) = V_t(s) + \alpha_t \left( V_t(s) - (r(s) + \delta V_t(s')) \right)$$

- Looks like gradient ascent (because it is)
- Normal issues with choosing $\alpha$

Stochastic approximation conditions

$$\sum_{t=0}^{\infty} \alpha_t = \infty, \quad \sum_{t=0}^{\infty} \alpha_t^2 < \infty$$

- Call this the one-step backup for state $s$
**TD(0)**

1. Initialise $V_0$ to anything
2. Set $s'$ randomly
3. Set $s \leftarrow s'$
4. Receive reward $r(s)$
5. Choose an action $a$
6. Observe new state $s'$
7. Compute $\Delta = V_t(s) - (r(s) + \delta V_t(s'))$
8. Update $V_{t+1}(s) = V_t(s) + \alpha_t \Delta$
9. If not converged, goto 3
Introducing $Q$-Values

- $Q(s, a)$ is the value of taking action $a$ in $s$, then acting normally
- Sometimes known as **Quality** function

$$Q_{t+1}(s, a) = r(s) + \delta \sum_{s' \in S} \Pr[s'|s, a] \Pr[a|s, w] V_t(s')$$

- Greedy policy given $Q(s, a)$ is trivial to compute

$$\Pr[\arg \max_a Q(s, a)|s] = 1$$

- Use of $Q$-values does not imply we are doing $Q$-learning
- The SARSA(0) algorithm learns $Q$-values directly
Exploration versus Exploitation

- How do we select actions during TD(0)?
- All states must be visited tried
- We should exploit our current knowledge of good actions
- Trade off is tricky
- Rule of thumb.... always allow some exploration
- Common action selection policies:
  - Epsilon soft: Greedy with prob. $1 - \epsilon$, random otherwise
  - Soft-max:

$$
\Pr[a|Q_t(s, \cdot)] = \frac{\exp(Q_t(s, a)/k)}{\sum_{a'} \exp(Q_t(s, a')/k)},
$$

where $k$ is a temperature parameter.

- With exploration and table of values, TD(0) finds global maximum
TD(0) $\delta = 0.5, \alpha = 1.0$ Trial 1
TD(0) \( \delta = 0.5, \alpha = 1.0 \) Trial 2
TD(0) $\delta = 0.5$, $\alpha = 1.0$ Trial 3
**TD(0) $\delta = 0.5, \alpha = 1.0$ Trial 4**

<table>
<thead>
<tr>
<th>v=0</th>
<th>0</th>
<th>0</th>
<th>0.25</th>
<th>0.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0.25</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

$v=0$ moves to the next state, which is rewarded with $r=1$. The value function update is calculated using $\delta = 0.5$ and $\alpha = 1.0$. The value function $v$ is updated as follows:

$v_{t+1} = v_t + \alpha (r + \delta v_{t+1} - v_t)$
**TD(0)** $\delta = 0.5$, $\alpha = 1.0$ Trial 5

<table>
<thead>
<tr>
<th></th>
<th>0.03125</th>
<th>0.0625</th>
<th>0.125</th>
<th>0.25</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0625</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.125</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.25</td>
<td></td>
<td></td>
<td></td>
<td>r=1</td>
</tr>
</tbody>
</table>

$r=1$
Problem With One-Step Backup

- Suppose we have $r(s) = 0$ except for a $r(goal) = 1$
- All values are initially 0
- Sample trajectories from a start point to the goal
- If $n$ steps to the goal, we need \textit{at least} $n$ trajectories!
- Goal reward propagates one step further back on each trajectory
- Can we update all steps that lead to goal with one trajectory?
Multi-Step TD
Forward View

- We can define multiple step temporal differences

\[ \Delta_1 = V(s) - (r(s_t) + \delta V(s_{t+1})) \]
\[ \Delta_2 = V(s) - (r(s_t) + \delta r(s_{t+1}) + \delta^2 V(s_{t+2})) \]
\[ \vdots \]
\[ \Delta_n = V(s) - (r(s_t) + \delta r(s_{t+1}) + \delta^2 r(s_{t+2}) + \cdots + \delta^n V(s_n)) \]

- TD(0) takes only \( \Delta_1 \) as the error estimate
  - Tends to be low variance and high bias
- A Monte-Carlo approach for \( T \) step episodes uses \( \Delta_T \)
  - Tends to be low bias but high variance
Exponentially Weighting $n$-Step Errors

- Weight $n$-step errors exponentially, combining
  - Low bias of Monte-Carlo
  - Low variance of one-step backup

- Weight factor $\lambda$:

$$\Delta = (1 - \lambda)\Delta_1 + \lambda(1 - \lambda)\Delta_2 + \cdots + \lambda^{n-1}(1 - \lambda)\Delta_n + \ldots$$

- Exercise: show that infinite sum of weight terms

$$\sum_{n=1}^{\infty} \lambda^{n-1}(1 - \lambda) = 1$$
Q: How do we implement an algorithm that needs future returns?

A: Eligibility Traces

- When state $s$ is visited, add 1 to $e(s)$
- Discount vector $e$ by $\lambda \delta$ after each step
- $\lambda$ helps convergence, $\delta$ is a part of the domain
- $e$ is the eligibility of each state for update
- This leads to the TD($\lambda$) algorithm...
TD($\lambda$)

1. Initialise $V_0$ to anything
2. Set $s'$ randomly
3. Set $s \leftarrow s'$
4. Receive reward $r(s)$
5. Choose an action $a$
6. Observe new state $s'$
7. Compute $\Delta = V_t(s) - (r(s) + \delta V_t(s'))$
8. $e(s) = e(s) + 1$
9. Update $V_{t+1} = V_t + \alpha_t \Delta e_t$
10. $e = \delta \lambda e$
11. If not converged, goto 3

This algorithm is equivalent to forward view (\lambda weighted multi-step backup)
$\text{TD}(0.5)$ $\delta = 0.5$, $\alpha = 1.0$ Trial 1
TD(0.5) $\delta = 0.5$, $\alpha = 1.0$ Trial 2

$v = ??$

<table>
<thead>
<tr>
<th>0.0039</th>
<th>0.0156</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0156</td>
<td>0</td>
<td>0.063 + 0.188 + 0.063 = 0.313</td>
<td>0.5 (\triangle = 0.25)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0.25</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

$r = 1$
Function Approximation

**Bellman’s Curse of Dimensionality**

- Suppose there are \( v \) binary variables \( x_1, \ldots, x_v \)
- Number of states is \( 2^v \)
- 20 variables = 1,048,576 states

- Usually too many states to store \( V(s) \) for each \( s \)
- Let’s use our knowledge of training function approximators
- Approximate \( V(s) \) with \( \hat{V}(s, w) \), parameterised by \( w \)
- E.g., linear approximation (perceptron)

\[
\hat{V}(s, w) = \sum_{i=1}^{v} w_i x_i = w^\top x
\]

- Change to TD(\( \lambda \)) line 8: \( e = e + \nabla_w \hat{V}(s, w) \)
Eligibility trace is now a vector of length $v$
- It stores eligibility of each parameter for update
- Generally destroys convergence guarantees
- If $\lambda = 0$, and perceptron, can converge to best
- If fixed policy, and perceptron approximator

$$\|V^* - \hat{V}\|_\pi \leq \frac{1 - \delta \lambda}{1 - \delta} \|\hat{V}^* - V^*\|_\pi$$

- final error
- best possible error

- If $\lambda = 1$ can find best approximation $\hat{V}^*$
Q-Learning

- Recall $Q$-function learns value of action $a$ in state $s$
- Can learn $Q$-function using SARSA($\lambda$) variant of TD($\lambda$)
- But both TD and SARSA learn value of current policy
- What if we could learn the optimal $Q$-function **off-policy**
  - Learn the best policy while exploring
  - Re-use expensive real-world experience
- $Q$-learning does this
- Slow because it ignores off-policy transitions
### Partial Observability

- We have assumed so far that $o = s$, full observability
- What if we don’t know? Markov assumption violated
  - Ostrich approach (SARSA works well in practice)
  - Exact methods
  - Direct policy search: bypass values, local convergence

- Exact policy is based on full history

$$
\Pr[a_t|o_t, a_{t-1}, o_{t-1}, \ldots, a_1, o_1]
$$

- **Belief states** summarise history sufficiently for optimal decisions

$$
b_{t+1}(s') = \sum_{s' \in S} b_t(s) \Pr[s'|s, a]
$$

- Probability of each world state computed from history
Value Iteration For Belief States

- Do normal VI, but replace states with belief state $b$
  \[
  V(b) = r(b) + \sum_b \sum_a \Pr[b' | b, a] V(b')
  \]

- Expanding out $b$
  \[
  V(b) = \sum_{s \in S} b(s) r(s) + \sum_{a \in A} \sum_{o \in O} \sum_{s \in S} \sum_{s' \in S} \Pr[s' | s, a] \Pr[o, s] \Pr[a | o, w] b(s) V(s')
  \]

- What is $V(b)$?
  \[
  V(b) = \max_{l \in \mathcal{L}} l^T b
  \]
Piecewise Linear Representation

- Action $u$
- Belief state space
- $V(b)$
- $b_0 = 1 - b_1$
- Useful hyperplane

Common action $u$
Complexity

**High Level Value Iteration for POMDPs**

1. Initialise $b_0$ (uniform/set state)
2. Receive observation $o$
3. Update belief state $b$
4. Find maximising hyperplane $l$ for $b$
5. Choose action $a$
6. Generate new $l$ for each observation and future action
7. While not converged, goto 2

- Specifics of 4 generate lots of algorithms
- Number of hyperplanes grows exponentially: P-space hard
- Infinite horizon problems *might* need infinite hyperplanes
- Approximations mostly learn value of representative belief states
Introduction

Value-Methods

Partial Observability

Policy-Gradient Methods

Progress...

- Exact VI
- SARSA?
- Value & Pol Iteration
- MDP
- POMDP

Model Based

Experience
Jonathan Baxter, Andrew Tridgell, and Lex Weaver.
KnightCap: A chess program that learns by combining TD(λ) with game-tree search.

Richard S. Sutton and Andrew G. Barto.
*Reinforcement Learning: An Introduction.*

Gerald Tesauro.
TD-Gammon, a self-teaching backgammon program, achieves master-level play.