Reinforcement Learning: Lessons for AI

Richard S. Sutton
University of Massachusetts
www.cs.umass.edu/~rich

Outline

• Definitions of RL
• History of RL
• State of the Art
• Lessons
• RL at the Knowledge Level

RL is Learning from Interaction

Environment

perception

rewards

action

Agent

complete agent
• temporally situated
• continual learning & planning
• object is to affect environment
• env stochastic & uncertain

RL is like life!

More Formally:
Markov Decision Problems (MDPs)

An MDP is defined by \((S, A, P, R, \gamma)\)

\(S\) – set of states of the environment
\(A(s)\) – set of actions possible in state \(s \in S\)
\(P(s,s',a)\) – probability of transition from \(s\) to \(s'\) given action \(a\)
\(R(s,s',a)\) – expected reward on transition \(s\) to \(s'\) given \(a\)
\(\gamma\) – discount rate for delayed reward

\(s_t\)

\(a_t\)

\(r_t\)

\(s_{t+1}\)

\(s_{t+2}\)

\(s_{t+3}\)

\(\ldots\)

The Objective

• Find a way of behaving that gets a lot of reward in the long run

• Find a policy \(\pi: s \in S \rightarrow a \in A(s)\) (could be stochastic)

that maximizes the value (expected future reward) of each \(s\):

\[ V(s) = E \{ r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots | s = s, \pi \} \]

and each \(s, a\) pair:

\[ Q(s,a) = E \{ r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots | s = s, q = a, \pi \} \]

These are called value functions - cf. evaluation functions

Radical Generality
of the RL problem

RL

General stochastic dynamics
General goals
Uncertainty
Reactive decision-making

Classical

Determinism
Goals of achievement
Complete knowledge
Closed world
Unlimited deliberation
Definitions of Reinforcement Learning
Learning about, from, and while interacting with an external environment to achieve a goal.
Learning what to do by trial and error.
Any way of solving the MDP problem
i.e., of finding a policy that maximizes long-term reward
An approach to AI emphasizing the above.

AI's Founding Fathers would find RL Familiar
Minsky's PhD thesis is earliest work in RL (1954)
SNARC - "Stochastic Neural-Analog Reinforcement" Calculator
Samuel (1959) learned evaluation functions using a temporal-difference method
State/action/reward ideas commonplace in early AI and in animal learning psychology, and in optimal control theory

In the 1970s, RL Died Out
• Learning fell out of fashion in AI
• Learning fell out of fashion in Psychology
• RL was confused with supervised learning, pattern recognition
• Little work on genuine trial-and-error learning
  Exceptions: Michie, Andreas, Learning Automata, Klopf

Rebirth of RL in the 1980s
• Realization that trial-and-error learning had been lost – Klopf (1972), Barto & Sutton (1981)
• TD(λ) – Sutton (1988)
• Q-learning and connections to dynamic programming – Watkins (1989)
• TD-Gammon – Tesauro (1992)

Modern RL
Very active area, centered in
• Machine learning
• Neural Networks
• Operations Research
• MDP planning in AI
Also spin-offs in Psychology and Neuroscience
Not a separate field

Strands of History of RL

<table>
<thead>
<tr>
<th>Trial-and-error learning</th>
<th>Temporal-difference learning</th>
<th>Optimal control, value functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thorndike (1911)</td>
<td>Secondary reinforcement (1960)</td>
<td>Hamilton (Physics) 1800s</td>
</tr>
<tr>
<td>Minsky</td>
<td>Samuel</td>
<td>Shannon</td>
</tr>
<tr>
<td>Klopf</td>
<td>Witten</td>
<td>Bellman/Howard (OR)</td>
</tr>
<tr>
<td>Barto et al</td>
<td>Sutton</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Watkins</td>
</tr>
</tbody>
</table>
Outline

- Definitions of RL
- History of RL
- State of the Art
- Lessons
- RL at the Knowledge Level

What RL Algorithms Do

All RL Algorithms work by Interaction of Policy and Value

Dimensions of RL algorithms

- Function Approximation
- On-line Experience or Simulated Experience
- Amount of Search in Action Selection
- Exploration Method
- Kind of Backups
  - Deep backups
  - Shallow backups
  - Sample backups
  - Bootstrapping backups
  - Dynamic programming
  - Exhaustive backups

1-Step Tabular Q-Learning

On each state transition:

\[ Q(s, a) \leftarrow Q(s, a) + \alpha \left[ r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s, a) \right] \]

Update:

\[ Q(s, a) \leftarrow Q(s, a) + \alpha \left[ r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s, a) \right] \]

TD Error

\[ \lim Q(s, a) \rightarrow Q^*(s, a) \]

\[ \lim \pi \rightarrow \pi^* \]

Optimal behavior found without a model of the environment!

(Watkins, 1989)
Assumes finite MDP
Summary of Convergence theory
Asymptotic results only – almost no rate results

Lesson #1:
Approximate the Solution, Not the Problem

Lesson #2:
The Power of Learning from Experience

Lesson #3:
The Central Role of Value Functions

World-Class Applications of RL

- TD-Gammon and Jellyfish – Tesauro, Dahl
  World’s best backgammon player

- Elevator Control – Crites & Barto
  World’s best down-peak elevator controller

- Inventory Management – Van Roy, Bertsekas, Lee & Tsitsiklis
  10-15% improvement over industry standard methods

- Dynamic Channel Assignment – Singh & Bertsekas, Nie & Haykin
  World’s best assigner of radio channels to mobile telephone calls

All these applications are large, stochastic, optimal control problems
- too hard for conventional methods to solve exactly
- require problem to be simplified
RL just finds an approximate solution . . . which can be much better!

Expert examples are expensive and scarce
Experience is cheap and plentiful!
And teaches the real solution

TD-Gammon
Tesauro, 1992–1995
Start with a random network
Play millions of games against self
Learn a value function from this simulated experience
This produces arguably the best player in the world

...of modifiable moment-by-moment estimates of how well things are going
All RL methods learn value functions
All state-space planners compute value functions
Both are based on “backing up” value
Recognizing and reacting to the ups and downs of life is an important part of intelligence
Lesson #4: Learning and Planning can be radically similar

Historically, planning and trial-and-error learning have been seen as opposites. But RL treats both as processing of experience.

1-Step Tabular Q-Planning

1. Generate a state, \(s\), and action, \(a\).
2. Consult model for next state, \(s'\), and reward, \(r\).
3. Learn from the experience, \(s,a,r,s'\):
   \[
   Q(s,a) \leftarrow Q(s,a) + \alpha [r + \gamma \max_a Q(s',a') - Q(s,a)]
   \]
4. Go to 1.

With function approximation and cleverness in search control (Step 1), this is a viable, perhaps even superior, approach to state-space planning.

Lesson #5: Accretive Computation "Solves" Planning Dilemmas

"Solves" Planning Dilemmas

Reactivity/Deliberation dilemma
"solved" simply by not opposing search and memory

Intractability of planning
"solved" by anytime improvement of solution

RL at the Knowledge Level

RL has a form of planning/reasoning but it seems too low-level, too flat.

RL learns values and models but needs to re-learn values when goal changes.

How can we learn re-usable components?
How can we learn to do A, B, and C and then recombine them in new ways?

Need Actions at a Higher Level

E.g., open-the-door rather than twitch-muscle-17, walk-to-work rather than 1-step-forward

Macro Actions
- take variable number of time steps
- specified by a whole (sub)policy
- and by a termination condition

state \(s_i\) \(\rightarrow\) action \(a_j\) \(\rightarrow\) reward along the way \(r_{i+1} + \gamma r_{i+2} + \cdots + \gamma^{k-1} r_{i+k}\)

"next" state \(s_{i+k}\)

can be treated much like primitive actions

Rooms Example

4 unreliable primitive actions
Far from the free
8 macro actions (to each room's 2 hallways)
4 Learning Problems for Macro Actions

Selection among macro actions
  Treat them as regular actions, learn their values
Learn models of macro actions
  predict outcome of executing macro action
Subgoal Credit Assignment
  Learn the policy inside a macro action
    e.g., reward the way you did something,
    while punishing the decision to do it
Discovery of suitable macro actions
  Which subgoals? Utility issues

Value Iteration in Rooms Example

\[
V_0(s) = 0 \quad \forall s \in S \\
V_0(\text{goal}) = 1 \\
V_{k+1}(s) = \max_a \sum_{s'} P(s,s',a)[R(s,s',a) + \gamma V_k(s')] \quad \forall s \in S
\]

Value Iteration with Models of Macro Actions

Example with Goal-Subgoal

using models of both primitive and macro actions

Models of Macro Actions

- Produce guaranteed correct plans
- Can be learned by TD methods
- Can be based on subgoals
  policies to achieve subgoals can be learned
  user can provide subgoals
  agent can propose own subgoals
- Enable learning at the Knowledge level
  re-usable knowledge with a clear semantics
  for a general context: stochastic, closed-loop, reward goals

Lesson #6: Generality is no impediment to working with higher-level knowledge

Summary of Lessons

1. Approximate the Solution, Not the Problem
2. The Power of Learning from Experience
3. The Central Role of Value Functions in finding optimal sequential behavior
4. Learning and Planning can be Radically Similar
5. Accretive Computation "Solves Dilemmas"
6. A General Approach need not be Flat, Low-level