Chapter 1: Introduction

Artificial Intelligence

Psychology

Reinforcement Learning (RL)

Control Theory and Operations Research

Neuroscience

Artificial Neural Networks
What is Reinforcement Learning?

- Learning from interaction
- Goal-oriented learning
- Learning about, from, and while interacting with an external environment
- Learning what to do—how to map situations to actions—so as to maximize a numerical reward signal
Supervised Learning

Training Info = desired (target) outputs

Error = (target output – actual output)
Reinforcement Learning

Training Info = evaluations ("rewards" / "penalties")

Objective: get as much reward as possible
Key Features of RL

- Learner is not told which actions to take
- Trial-and-Error search
- Possibility of delayed reward
  - Sacrifice short-term gains for greater long-term gains
- The need to *explore* and *exploit*
- Considers the whole problem of a goal-directed agent interacting with an uncertain environment
Complete Agent

- Temporally situated
- Continual learning and planning
- Object is to affect the environment
- Environment is stochastic and uncertain

R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction
Elements of RL

- **Policy**: what to do
- **Reward**: what is good
- **Value**: what is good because it predicts reward
- **Model**: what follows what
An Extended Example: Tic-Tac-Toe

Assume an imperfect opponent:
—he/she sometimes makes mistakes
An RL Approach to Tic-Tac-Toe

1. Make a table with one entry per state:

<table>
<thead>
<tr>
<th>State</th>
<th>$V(s)$ – estimated probability of winning</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td>.5  ?</td>
</tr>
<tr>
<td>#</td>
<td>.5  ?</td>
</tr>
<tr>
<td>. .</td>
<td></td>
</tr>
<tr>
<td>. .</td>
<td></td>
</tr>
<tr>
<td>x o</td>
<td>1  win</td>
</tr>
<tr>
<td>. .</td>
<td></td>
</tr>
<tr>
<td>. .</td>
<td></td>
</tr>
<tr>
<td>#</td>
<td>0  loss</td>
</tr>
<tr>
<td>. .</td>
<td></td>
</tr>
<tr>
<td>. .</td>
<td></td>
</tr>
<tr>
<td>#</td>
<td>0  draw</td>
</tr>
</tbody>
</table>

2. Now play lots of games.
   To pick our moves, look ahead one step:

   Just pick the next state with the highest estimated prob. of winning — the largest $V(s)$; a greedy move.

   But 10% of the time pick a move at random; an exploratory move.
**RL Learning Rule for Tic-Tac-Toe**

```
• our move
  • opponent's move
  • our move
  • opponent's move
  • our move
```

Starting position

```
s = the state before our greedy move
s' = the state after our greedy move
```

We increment each $V(s)$ toward $V(s')$ — a **backup**:

$$V(s) \leftarrow V(s) + \alpha[V(s') - V(s)]$$

A small positive fraction, e.g., $\alpha = .1$

The **step-size parameter**
How can we improve this T.T.T. player?

- Take advantage of symmetries
  - representation/generalization
  - How might this backfire?
- Do we need “random” moves? Why?
  - Do we always need a full 10%?
- Can we learn from “random” moves?
- Can we learn offline?
  - Pre-training from self play?
  - Using learned models of opponent?
- . . .
e.g. Generalization

<table>
<thead>
<tr>
<th>Table</th>
<th>Generalizing Function Approximator</th>
</tr>
</thead>
<tbody>
<tr>
<td>State</td>
<td>V</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>$s_1$</td>
<td></td>
</tr>
<tr>
<td>$s_2$</td>
<td></td>
</tr>
<tr>
<td>$s_3$</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>$s_N$</td>
<td></td>
</tr>
</tbody>
</table>

Train here
How is Tic-Tac-Toe Too Easy?

- Finite, small number of states
- One-step look-ahead is always possible
- State completely observable
- ...

R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction
Some Notable RL Applications

- **TD-Gammon**: Tesauro
  - world’s best backgammon program

- **Elevator Control**: Crites & Barto
  - high performance 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
  - high performance assignment of radio channels to mobile telephone calls
TD-Gammon

Start with a random network
Play very many games against self
Learn a value function from this simulated experience

This produces arguably the best player in the world

Tesauro, 1992–1995
Elevator Dispatching

10 floors, 4 elevator cars

STATES: button states; positions, directions, and motion states of cars; passengers in cars & in halls

ACTIONS: stop at, or go by, next floor

REWARDS: roughly, –1 per time step for each person waiting

Conservatively about $10^{22}$ states

Crites and Barto, 1996
Performance Comparison
## Some RL History

<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 (Ψ) 1911</td>
<td>Secondary reinforcement (Ψ)</td>
<td>Hamilton (Physics) 1800s</td>
</tr>
<tr>
<td>Minsky</td>
<td>Samuel</td>
<td>Shannon</td>
</tr>
<tr>
<td>Klopf</td>
<td>Holland</td>
<td>Bellman/Howard (OR)</td>
</tr>
<tr>
<td>Barto et al.</td>
<td>Witten</td>
<td>Werbos</td>
</tr>
<tr>
<td></td>
<td>Sutton</td>
<td>Watkins</td>
</tr>
</tbody>
</table>
MENACE (Michie 1961)

“Matchbox Educable Noughts and Crosses Engine”
The Book

- Part I: The Problem
  - Introduction
  - Evaluative Feedback
  - The Reinforcement Learning Problem
- Part II: Elementary Solution Methods
  - Dynamic Programming
  - Monte Carlo Methods
  - Temporal Difference Learning
- Part III: A Unified View
  - Eligibility Traces
  - Generalization and Function Approximation
  - Planning and Learning
  - Dimensions of Reinforcement Learning
  - Case Studies
The Course

- One chapter per week (with some exceptions)
- Read the chapter for the first class devoted to that chapter
- Written homeworks: basically all the non-programming assignments in each chapter. Due second class on that chapter.
- Programming exercises (not projects!): each student will do approximately 3 of these, including one of own devising (in consultation with instructor and/or TA).
- Closed-book, in-class midterm; closed-book 2-hr final
- Grading: 40% written homeworks; 30% programming homeworks; 15% final; 10% midterm; 5% class participation
- See the web for more details
Next Class

- Introduction continued and some case studies
- Read Chapter 1
- Hand in exercises 1.1 — 1.5