An Introduction to COMPUTATIONAL REINFORCEMENT LEARNING

Andrew G. Barto
Department of Computer Science
University of Massachusetts – Amherst

UPF Lecture 3
The Overall Plan

- Lecture 1:
  - What is Computational Reinforcement Learning?
  - Learning from evaluative feedback
  - Markov decision processes
- Lecture 2:
  - Dynamic Programming
  - Basic Monte Carlo methods
  - Temporal Difference methods
  - A unified perspective
  - Connections to neuroscience
- Lecture 3:
  - Function approximation
  - Model-based methods
  - Abstraction and hierarchy
  - Intrinsically motivated RL
The Overall Plan

- Lecture 1:
  - What is Computational Reinforcement Learning?
  - Learning from evaluative feedback
  - Markov decision processes

- Lecture 2:
  - Dynamic Programming
  - Basic Monte Carlo methods
  - Temporal Difference methods
  - A unified perspective
  - Connections to neuroscience

- Lecture 3:
  - Function approximation
  - Model-based methods
  - Abstraction and hierarchy
  - Intrinsically motivated RL
Lecture 3, Part 1:
Generalization and Function Approximation

Objectives of this part:

- Look at how experience with a limited part of the state set be used to produce good behavior over a much larger part
- Overview of function approximation (FA) methods and how they can be adapted to RL
Value Prediction with FA

As usual: Policy Evaluation (the prediction problem):
for a given policy \( \pi \), compute the state-value function \( V^\pi \)

In earlier chapters, value functions were stored in lookup tables.

Here, the value function estimate at time \( t \), \( V_t \), depends on a parameter vector \( \theta_t \), and only the parameter vector is updated.

\[ \theta_t \]
\[ \text{e.g.,} \, \theta_t \text{ could be the vector of connection weights of a neural network.} \]
Adapt Supervised Learning Algorithms

Training Info = desired (target) outputs

Inputs \rightarrow \text{Supervised Learning System} \rightarrow \text{Outputs}

Training example = \{input, target output\}

Error = (target output \ – \ actual output)
Backups as Training Examples

e.g., the TD(0) backup:

\[ V(s_t) \leftarrow V(s_t) + \alpha [r_{t+1} + \gamma V(s_{t+1}) - V(s_t)] \]

As a training example:

\[ \{ \text{description of } s_t, \quad r_{t+1} + \gamma V(s_{t+1}) \} \]

input \hspace{2cm} target output
Any FA Method?

- In principle, yes:
  - artificial neural networks
  - decision trees
  - multivariate regression methods
  - etc.

- But RL has some special requirements:
  - usually want to learn while interacting
  - ability to handle nonstationarity
  - other?
Linear Methods

Represent states as feature vectors:
for each $s \in S$:
$$\phi_s = (\phi_s(1), \phi_s(2), \ldots, \phi_s(n))^T$$

$$V_t(s, \theta_t) = \theta_t^T \phi_s = \sum_{i=1}^{n} \theta_t(i) \phi_s(i)$$
Nice Properties of Linear FA Methods

- The gradient is very simple: \( \nabla_\theta V_t(s) = \tilde{\phi}_s \)
- For MSE, the error surface is simple: quadratic surface with a single minimum.
- Linear gradient descent TD(\( \lambda \)) converges:
  - Step size decreases appropriately
  - On-line sampling (states sampled from the on-policy distribution)
  - Converges to parameter vector \( \theta_\infty \) with property:
    \[
    MSE(\theta_\infty) \leq \frac{1 - \gamma \lambda}{1 - \gamma} MSE(\theta^*)
    \]
    (Tsitsiklis & Van Roy, 1997)
Tile Coding

- Binary feature for each tile
- Number of features present at any one time is constant
- Binary features mean weighted sum easy to compute
- Easy to compute indices of the features present

Shape of tiles ⇒ Generalization
#Tilings ⇒ Resolution of final approximation

Radial Basis Functions (RBFs)

e.g., Gaussians

\[ \phi_s(i) = \exp\left(-\frac{\|s - c_i\|^2}{2\sigma_i^2}\right) \]

Control with FA

- Learning state-action values
  - Training examples of the form:
    \[
    \{\text{description of } (s_t, a_t), v_t\}
    \]
  - The general gradient-descent rule:
    \[
    \theta_{t+1} = \theta_t + \alpha [v_t - Q_t(s_t, a_t)] \nabla_\theta Q(s_t, a_t)
    \]
- Gradient-descent Sarsa(\(\lambda\)) (backward view):
  \[
  \theta_{t+1} = \theta_t + \alpha \delta_t e_t
  \]
  where
  \[
  \delta_t = r_{t+1} + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t)
  \]
  \[
  e_t = \gamma \lambda e_{t-1} + \nabla_\theta Q_t(s_t, a_t)
  \]
Mountain-Car Task
Baird’s Counterexample
Baird’s Counterexample Cont.

Parameter values, $\theta_k(i)$
(log scale, broken at ±1)
Summary

- Generalization
- Adapting supervised-learning function approximation methods
  - Linear methods
    - Radial basis functions
    - Tile coding
    - Kanerva coding
- Nonlinear gradient-descent methods? Backpropagation?
- Subleties involving function approximation, bootstrapping and the on-policy/off-policy distinction

The Overall Plan

- Lecture 1:
  - What is Computational Reinforcement Learning?
  - Learning from evaluative feedback
  - Markov decision processes
- Lecture 2:
  - Dynamic Programming
  - Basic Monte Carlo methods
  - Temporal Difference methods
  - A unified perspective
  - Connections to neuroscience
- Lecture 3:
  - Function approximation
  - Model-based methods
  - Abstraction and hierarchy
  - Intrinsically motivated RL
Lecture 3, Part 2: Model-Based Methods

Objectives of this part:

- Use of environment models
- Integration of planning and learning methods
Lecture 3, Part 2: Model-Based Methods

Objectives of this part:

- Use of environment models
- Integration of planning and learning methods
Models

- **Model**: anything the agent can use to predict how the environment will respond to its actions

- **Distribution model**: description of all possibilities and their probabilities
  - e.g., $P_{ss'}^a$ and $R_{ss'}^a$, for all $s$, $s'$, and $a \in A(s)$

- **Sample model**: produces sample experiences
  - e.g., a simulation model

- Both types of models can be used to produce simulated experience

- Often sample models are much easier to come by
Planning

- **Planning**: any computational process that uses a model to create or improve a policy

```
model  |  planning  |  policy
```
Planning Cont.

- Classical DP methods are state-space planning methods
- Heuristic search methods are state-space planning methods
- A planning method based on Q-learning:

```
Do forever:
1. Select a state, \( s \in S \), and an action, \( a \in A(s) \), at random
2. Send \( s, a \) to a sample model, and obtain a sample next state, \( s' \),
   and a sample next reward, \( r \)
3. Apply one-step tabular Q-learning to \( s, a, s', r \):
   \[ Q(s, a) \leftarrow Q(s, a) + \alpha \left[ r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right] \]
```

Random-Sample One-Step Tabular Q-Planning
Learning, Planning, and Acting

- Two uses of real experience:
  - model learning: to improve the model
  - direct RL: to directly improve the value function and policy
- Improving value function and/or policy via a model is sometimes called indirect RL or model-based RL. Here, we call it planning.
The Dyna Architecture (Sutton 1990)
Dyna-Q on a Simple Maze

rewards = 0 until goal, when = 1
Dyna-Q Snapshots: Midway in 2nd Episode

WITHOUT PLANNING ($N=0$)

WITH PLANNING ($N=50$)
Full vs. Sample Backups

\[ \text{RMS error in value estimate} \]

\[ \text{Number of max } Q(s', a') \text{ computations} \]

\( b \) successor states, equally likely; initial error = 1; assume all next states’ values are correct

Trajectory Sampling

- **Trajectory sampling**: perform backups along simulated trajectories
- This samples from the on-policy distribution
- Advantages when function approximation is used
- Focusing of computation: can cause vast uninteresting parts of the state space to be (usefully) ignored:

![Diagram showing initial states, reachable under optimal control, and irrelevant states.](image-url)
Summary

- Emphasized close relationship between planning and learning
- Important distinction between distribution models and sample models
- Looked at some ways to integrate planning and learning
  - synergy among planning, acting, model learning
- Size of backups: full vs. sample; deep vs. shallow
- Distribution of backups: focus of the computation
  - trajectory sampling: backup along trajectories
The Overall Plan

- Lecture 1:
  - What is Computational Reinforcement Learning?
  - Learning from evaluative feedback
  - Markov decision processes

- Lecture 2:
  - Dynamic Programming
  - Basic Monte Carlo methods
  - Temporal Difference methods
  - A unified perspective
  - Connections to neuroscience

- Lecture 3:
  - Function approximation
  - Model-based methods
  - Abstraction and hierarchy
  - Intrinsically motivated RL
The “Macro” Idea

- A sequence of operations with a name; can be invoked like a primitive operation
  - Can invoke other macros... hierarchy
  - But: an open-loop policy
- Closed-loop macros
  - A decision policy with a name; can be invoked like a primitive control action
  - behavior (Brooks, 1986), skill (Thrun & Schwartz, 1995), mode (e.g., Grudic & Ungar, 2000), activity (Harel, 1987), temporally-extended action, option (Sutton, Precup, & Singh, 1997), schema (Piaget, Arbib)
Options (Precup, Sutton, & Singh, 1997)

A generalization of actions to include temporally-extended courses of action

An option is a triple \( o = < I, \pi, \beta > \)

- **\( I \)**: initiation set: the set of states in which \( o \) may be started
- **\( \pi \)**: is the policy followed during \( o \)
- **\( \beta \)**: termination conditions: gives the probability of terminating in each state

Example: robot docking

- **\( I \)**: all states in which charger is in sight
- **\( \pi \)**: pre-defined controller
- **\( \beta \)**: terminate when docked or charger not visible
Options cont.

- Policies can select from a set of options & primitive actions
- Generalizations of the usual concepts:
  - Transition probabilities (“option models”)
  - Value functions
  - Learning and planning algorithms
- Intra-option off-policy learning:
  - Can simultaneously learn policies for many options from same experience
Canonical Illustration: Rooms Example

4 rooms
4 hallways
4 unreliable primitive actions
8 multi-step options
(to each room's 2 hallways)
Goal states are given a terminal value of 1
All rewards zero
\( \gamma = .9 \)

Given goal location, quickly plan shortest route
Fail 33% of the time
Intra-Option Off-Policy Learning

After each primitive action, update all the options that could have taken that action.

On every transition: \( s_t \xrightarrow{a_t} r_t \xrightarrow{a} s_{t+1} \)

For all options \( o \) that would have chosen \( a_t \) in \( s_t \), update their values:

\[
Q(s_t, o) \leftarrow Q(s_t, o) + \alpha \left[ r_t + \gamma \left( (1 - \beta_t)Q(s_{t+1}, o) + \beta_t \max_{o'} Q(s_{t+1}, o') \right) - Q(s_t, o) \right]
\]

Guaranteed to converge to correct values for all options, for any "non-starving" behavior policy.
Where do Options come from?

- Dominant approach: hand-crafted from the start
- How can an agent create useful options for itself?
  - Several different approaches (McGovern, Digney, Hengst, ...). All involve defining subgoals of various kinds
Rewards and Pseudo-Rewards

- For each subgoal option there is a pseudo-reward function: e.g. zero everywhere except at the subgoal.
- Pseudo-rewards are used to learn the option policy, but typically do not influence agent’s behavior.
The Overall Plan

- Lecture 1:
  - What is Computational Reinforcement Learning?
  - Learning from evaluative feedback
  - Markov decision processes

- Lecture 2:
  - Dynamic Programming
  - Basic Monte Carlo methods
  - Temporal Difference methods
  - A unified perspective
  - Connections to neuroscience

- Lecture 3:
  - Function approximation
  - Model-based methods
  - Abstraction and hierarchy
  - Intrinsically motivated RL
Motivation

“Forces” that energize an organism to act and that direct its activity.

Extrinsic Motivation: being moved to do something because of some external reward ($$, a prize, etc.).

Intrinsic Motivation: being moved to do something because it is inherently enjoyable.
Intrinsic Motivation

- An activity is intrinsically motivated if the agent does it for its own sake rather than as a step toward solving a specific problem
- Curiosity, Exploration, Manipulation, Play, Learning itself . .
A few of the classics

  - Competence: an organism’s capacity to interact effectively with its environment
  - Critique of Freudian and Hullian drive theories
  - “The motivation needed to obtain competence cannot be wholly derived from sources of energy currently conceptualized as drives or instincts.”
  - Made a case for exploratory motive as an independent primary drive

“As knowledge accumulated about the conditions that govern exploratory behavior and about how quickly it appears after birth, it seemed less and less likely that this behavior could be a derivative of hunger, thirst, sexual appetite, pain, fear of pain, and the like, or that stimuli sought through exploration are welcomed because they have previously accompanied satisfaction of these drives.”
Computational Curiosity


- “The direct goal of curiosity and boredom is to improve a world model. The indirect goal is to ease the learning of new goal-directed action sequences.”

- “Curiosity Unit”: reward is a function of the mismatch (Euclidean distance) between model’s current predictions and actuality. There is positive reinforcement whenever the system fails to correctly predict the environment.

- “Thus the usual credit assignment process … encourages certain past actions in order to repeat situations similar to the mismatch situation.”
Schmidhuber Cont.

“The important point is: The same complex mechanism which is used for ‘normal’ goal-directed learning is used for implementing curiosity and boredom. There is no need for devising a separate system which aims at improving the world model.”

Problems with that earlier idea (expectation mismatch as reward):

- It will focus on parts of environment that are inherently unpredictable. So it will be rewarded even though the model cannot improve.
- It won’t try to learn easier parts before learning hard parts

Instead of learning to predict errors, learn to predict cumulative prediction error changes.
Comp. Curiosity Cont.

More Schmidhuber:

- “My adaptive explorer continually wants ... to focus on those novel things that seem easy to learn, given current knowledge. It wants to ignore (1) previously learned, predictable things, (2) inherently unpredictable ones (such as details of white noise on the screen), and (3) things that are unexpected but not expected to be easily learned (such as the contents of an advanced math textbook beyond the explorer’s current level.”

Vygotsky’s “Zone of Proximal Development” and Optimal Level Theories of motivation: e.g. Dember and Earl’s, 1957 “pacer principle” etc.
From my son’s 4th grade classroom
Comp. Curiosity Cont.

  - For each state and action, add a value to the usual immediate reward called the **exploration bonus**.
  - It is proportional to a measure of how uncertain the system is about the value of doing that action in that state.
  - Uncertainty is assessed by keeping track of the *time since that action was last executed in that state*. The longer the time, the greater the assumed uncertainty.
  - “…why not expect the system to *plan* an action sequence to go out and test the uncertain state-action pair”
A Less Misleading View

external sensations

internal sensations

memory

reward

state

actions

RL agent
So What is IMRL?

- Key distinction:
  - Extrinsic reward = problem specific
  - Intrinsic reward = problem independent

- Learning phases:
  - Developmental Phase: gain general competence
  - Mature Phase: learn to solve specific problems

- Toward open-ended learning via hierarchical exploration
Task-Independent Subgoals

- “Bottlenecks”, “Hubs”, “Access States”, …

- Surprising events
- Novel events
- Incongruous events
- Etc. …
Option Creation and Intrinsic Reward

- Subgoals: events that are “intrinsically interesting”; not in the service of any specific task
- Create options to achieve them, together with pseudo-reward functions
- Intrinsic reward generated to guide agent behavior during learning option policy
- Once option is well learned, the triggering event loses its intrinsic rewarding quality (though pseudo-reward remains)
- Previously learned options are available as actions for learning policies of new options
- When facing a specific problem: extract a “working set” of actions (primitive and abstract) for planning and learning
An Example

- Built-in salient stimuli: changes in lights and sounds
- Intrinsic reward generated by each salient event:
  - Proportional to the error in prediction of that event according to the option model for that event (“surprise”)
- Motivated in part by novelty responses of dopamine neurons
Example Continued

- Upon first occurrence of salient event: create an option, its pseudo-reward function and initialize:
  - Initiation set
  - Policy
  - Termination condition
  - Option model
- All options and option models updated all the time using intra-option learning (based on pseudo-rewards)
- Intrinsic reward added to extrinsic reward, if present, to control behavior
A not-quite-so-simple domain: Playroom

Agent has eye, hand, visual marker

Actions:
- move eye to hand
- move eye to marker
- move eye N, S, E, or W
- move eye to random object
- move hand to eye
- move hand to marker
- move marker to eye
- move marker to hand

If both eye and hand are on object: turn on light, push ball, etc.
The Playroom Domain cont.

Switch controls room lights
Bell rings and moves one square if ball hits it
Press blue/red block turns music on/off
Lights have to be on to see colors
Can push blocks
Monkey cries out if bell and music both sound in dark room
Skills

- To make monkey cry out:
  - Move eye to switch
  - Move hand to eye
  - Turn lights on
  - Move eye to blue block
  - Move hand to eye
  - Turn music on
  - Move eye to switch
  - Move hand to eye
  - Turn light off
  - Move eye to bell
  - Move marker to eye
  - Move eye to ball
  - Move hand to ball
  - Kick ball to make bell ring

- Using skills (options)
  - Turn lights on
  - Turn music on
  - Turn lights off
  - Ring bell
Reward for Salient Events

Speed of Learning Various Skills

![Graph showing the speed of learning various skills](image-url)

Learning to Make the Monkey Cry Out

Limitations

- Hand-crafted for our purposes
- Pre-defined salience
- Completely observable
- Little state abstraction
- Deterministic (mostly)
- No un-caused salient events
- Etc.
Playroom States
Explore Now / Exploit Later

If the agent’s task is to get as much reward as possible, then what motivates it to build a widely-defined policy to be exploited later?

- As continuing task with a usual RL algorithm?
- Episodic, e.g. with random restarts?
- Optimistic initial values?
- Counter based?
- But what we want is a kind of on-line prioritized sweeping
Explore Now / Exploit Later

- Our approach: Design an Intrinsic Reward mechanism to create a policy (behavior policy) that is efficient for learning a policy that is optimal for a different reward function (a task policy)

- Two value functions are maintained:
  - One to solve the “real” MDP (in the future): $V^T$
  - Another one used to select actions now: $V^B$
  - $V^B$ predicts intrinsic reward, $r^I$, which we have to define to make this work
Connects with Previous RL Work

- Schmidhuber
- Thrun and Moller
- Sutton
- Kaplan and Oudeyer
- Marshall, Blank, & Meeden
- Duff
- Others….

But these did not have the option framework and related algorithms available
Conclusions

- Intrinsically motivated behavior is important for creating behavioral building blocks
- Can dramatically simplify and accelerate learning
- RL+Options+Intrinsic reward is a natural way to do this
- Theory?
- Behavior?
- Neuroscience?
The Overall Plan

- Lecture 1:
  - What is Computational Reinforcement Learning?
  - Learning from evaluative feedback
  - Markov decision processes

- Lecture 2:
  - Dynamic Programming
  - Basic Monte Carlo methods
  - Temporal Difference methods
  - A unified perspective
  - Connections to neuroscience

- Lecture 3:
  - Function approximation
  - Model-based methods
  - Abstraction and hierarchy
  - Intrinsically motivated RL

THANKS FOR LISTENING!