Chapter 9: Planning and Learning

Objectives of this chapter:

- Use of environment models
- Integration of planning and learning methods
The Original Idea

[Diagram]

EVALUATION FUNCTION

Heuristic Reward (scalar)

POLICY

WORLD

Reward (scalar)

State

Action

Sutton, 1990
The Original Idea
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 $s, s', a \in A(s)$
- **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

  
  \[ \text{model} \xrightarrow{\text{planning}} \text{policy} \]

- **Planning in AI**:
  - state-space planning
  - plan-space planning (e.g., partial-order planner)

- We take the following (unusual) view:
  - all state-space planning methods involve computing value functions, either explicitly or implicitly
  - they all apply backups to simulated experience

  
  \[ \text{model} \xrightarrow{\text{simulated experience}} \xrightarrow{\text{backups}} \text{values} \xrightarrow{} \text{policy} \]
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 \( s, a, s', r \):
\[
Q(s, a) \leftarrow Q(s, a) + \alpha [r + \gamma \max_{a'} Q(s', a') - Q(s, a)]
\]

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.
Direct vs. Indirect RL

- **Indirect methods:**
  - make fuller use of experience: get better policy with fewer environment interactions

- **Direct methods**
  - simpler
  - not affected by bad models

But they are very closely related and can be usefully combined:
planning, acting, model learning, and direct RL can occur simultaneously and in parallel
The Dyna Architecture (Sutton 1990)

- **Policy/value functions**
  - Planning update
  - Simulated experience
  - Search control
- **Environment**
  - Real experience
  - Model learning
- **Model**
  - Direct RL update
  - Experience

Direct RL update feeds into real experience, which in turn feeds into policy/value functions. Simulated experience also feeds into policy/value functions. Model learning is connected to the model, which is fed by the environment.
The Dyna-Q Algorithm

Initialize $Q(s, a)$ and $Model(s, a)$ for all $s \in S$ and $a \in A(s)$

Do forever:

(a) $s \leftarrow$ current (nonterminal) state
(b) $a \leftarrow \varepsilon$-greedy($s, Q$)
(c) Execute action $a$; observe resultant state, $s'$, and reward, $r$
(d) $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$
(e) $Model(s, a) \leftarrow s', r$ (assuming deterministic environment)
(f) Repeat $N$ times:
   - $s \leftarrow$ random previously observed state
   - $a \leftarrow$ random action previously taken in $s$
   - $s', r \leftarrow Model(s, a)$
   - $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$

---

direct RL

model learning

planning
Dyna-Q on a Simple Maze

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

\textbf{WITHOUT PLANNING (N=0)}

\begin{figure}[h]
\centering
\includegraphics[width=0.4\textwidth]{without_planning}
\end{figure}

\textbf{WITH PLANNING (N=50)}

\begin{figure}[h]
\centering
\includegraphics[width=0.4\textwidth]{with_planning}
\end{figure}
When the Model is Wrong: Blocking Maze

The changed environment is harder
Shortcut Maze

The changed environment is easier
What is Dyna-Q$^+$?

- Uses an “exploration bonus”:
  - Keeps track of time since each state-action pair was tried for real
  - An extra reward is added for transitions caused by state-action pairs related to how long ago they were tried: the longer unvisited, the more reward for visiting
    \[ r + \kappa \sqrt{n} \]
  - The agent actually “plans” how to visit long unvisited states
Exploration vs. Exploitation

- R-Max (Brafman, Tennenholtz, 2003)
  - Model-based algorithm
  - Classify states as to whether they are sufficiently explored or not (“known”, “unknown”)
  - The optimistic model is one where in unknown states we enter a terminal state with the best possible reward
  - Solve the optimistic model and follow the resulting policy

- UC-RL (Auer, Ortner, 2006)
  - Given the uncertainty in the estimated model picks the world that is consistent with the observations and gives the highest average reward
  - Log-regret bounds
Prioritized Sweeping

- Which states or state-action pairs should be generated during planning?
- Work backwards from states whose values have just changed:
  - Maintain a queue of state-action pairs whose values would change a lot if backed up, prioritized by the size of the change
  - When a new backup occurs, insert predecessors according to their priorities
  - Always perform backups from first in queue
- Moore and Atkeson 1993; Peng and Williams, 1993
- Improved prioritized sweeping (McMahan & Gordon 2005)
Prioritized Sweeping

Initialize $Q(s, a)$, $Model(s, a)$, for all $s, a$, and $PQueue$ to empty
Do forever:
(a) $s \leftarrow$ current (nonterminal) state
(b) $a \leftarrow policy(s, Q)$
(c) Execute action $a$; observe resultant state, $s'$, and reward, $r$
(d) $Model(s, a) \leftarrow s', r$
(e) $p \leftarrow |r + \gamma \max_{a'} Q(s', a') - Q(s, a)|$.
(f) if $p > \theta$, then insert $s, a$ into $PQueue$ with priority $p$
(g) Repeat $N$ times, while $PQueue$ is not empty:
   $s, a \leftarrow first(PQueue)$
   $s', r \leftarrow Model(s, a)$
   $Q(s, a) \leftarrow Q(s, a) + \alpha [r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$
Repeat, for all $\bar{s}, \bar{a}$ predicted to lead to $s$:
   $\bar{r} \leftarrow$ predicted reward
   $p \leftarrow |\bar{r} + \gamma \max_a Q(s, a) - Q(\bar{s}, \bar{a})|$
   if $p > \theta$ then insert $\bar{s}, \bar{a}$ into $PQueue$ with priority $p$
Prioritized Sweeping vs. Dyna-Q

Both use N=5 backups per environmental interaction
Rod Maneuvering (Moore and Atkeson 1993)
Full and Sample (One-Step) Backups

<table>
<thead>
<tr>
<th>Value estimated</th>
<th>Full backups (DP)</th>
<th>Sample backups (one-step TD)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V^\pi(s)$</td>
<td><img src="image1" alt="Diagram" /></td>
<td><img src="image2" alt="Diagram" /></td>
</tr>
<tr>
<td>$V^*(s)$</td>
<td><img src="image3" alt="Diagram" /></td>
<td><img src="image4" alt="Diagram" /></td>
</tr>
<tr>
<td>$Q^\pi(a,s)$</td>
<td><img src="image5" alt="Diagram" /></td>
<td><img src="image6" alt="Diagram" /></td>
</tr>
<tr>
<td>$Q^*(a,s)$</td>
<td><img src="image7" alt="Diagram" /></td>
<td><img src="image8" alt="Diagram" /></td>
</tr>
</tbody>
</table>

- $V^\pi(s)$: Policy Evaluation
- $V^*(s)$: Value Iteration
- $Q^\pi(a,s)$: Q-policy Evaluation
- $Q^*(a,s)$: Q-value Iteration
- TD(0):
- Q-learning:
- Sarsa:
Full vs. Sample Backups

$b$ 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 (Chapter 8)
- Focusing of computation: can cause vast uninteresting parts of the state space to be (usefully) ignored:

  ![Diagram showing initial states, reachable states under optimal control, and irrelevant states]

Initial states

Reachable under optimal control

Irrelevant states
Trajectory Sampling Experiment

- one-step full tabular backups
- uniform: cycled through all state-action pairs
- on-policy: backed up along simulated trajectories
- 200 randomly generated undiscounted episodic tasks
- 2 actions for each state, each with \( b \) equally likely next states
- .1 prob of transition to terminal state
- expected reward on each transition selected from mean 0 variance 1 Gaussian
Heuristic Search

- Used for action selection, not for changing a value function (=heuristic evaluation function)
- Backed-up values are computed, but typically discarded
- Extension of the idea of a greedy policy — only deeper
- Also suggests ways to select states to backup: smart focusing:

UCT: Kocsis&Szepesvari 2006 “The” algorithm used in all the best go programs as of 2007, 500 ELO increase, MOGO, ..
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
- Distribution of backups: focus of the computation
  - trajectory sampling: backup along trajectories
  - prioritized sweeping
  - heuristic search
- Size of backups: full vs. sample; deep vs. shallow