Learning and Planning with Tabular Methods

Alina Vereshchaka

CSE4/510 Reinforcement Learning
Spring 2010

avereshc@buffalo.edu

February 12, 2020
Overview

1 Definitions

2 Model

3 Recap: Dynamic Programming
Table of Contents

1 Definitions

2 Model

3 Recap: Dynamic Programming
Definitions

Learning: ?
Planning: ?
Learning: the acquisition of knowledge or skills through experience, study, or by being taught.

e.g., we learn value functions from real experience (action/state trajectories) using Monte Carlo methods, or we learn a model (transition function)
**Definitions**

**Learning**: the acquisition of knowledge or skills through experience, study, or by being taught.

e.g., we learn value functions from real experience (action/state trajectories) using Monte Carlo methods, or we learn a model (transition function)

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

e.g., we compute value functions from simulated experience (action/state trajectories)

![Diagram](Model → Planning → Policy)
Planning Examples

- Value iteration
- Policy iteration
- TD-gammon (look-ahead search)
- Alpha-Go (Monte Carlo Tree Search)
- Chess (heuristic search)
Table of Contents

1 Definitions

2 Model

3 Recap: Dynamic Programming
Anything the agent can use to predict how the environment will respond to its actions, concretely, the state transition $T(s'|s, a)$ and reward $R(s, a)$.
Anything the agent can use to predict how the environment will respond to its actions, concretely, the state transition $T(s'|s, a)$ and reward $R(s, a)$. 

![Diagram of model](image)
Distribution vs Sample Models

- Distribution model: ?
- Sample model ?
- **Distribution model**: lists all possible outcomes and their probabilities, $T(s'\mid s, a)$ for all $(s, a, s')$. (We used those in DP)
Distribution model: lists all possible outcomes and their probabilities, $T(s'|s, a)$ for all $(s, a, s')$. (We used those in DP)

Sample model a.k.a. a simulator produces a single outcome (transition) sampled according to its probability of occurring (we will use this in Monte Carlo methods)
Distribution VS Sample Models

- **Distribution model**: lists all possible outcomes and their probabilities, $T(s'|s, a)$ for all $(s, a, s')$. (We used those in DP)

- **Sample model** a.k.a. a simulator produces a single outcome (transition) sampled according to its probability of occurring (we will use this in Monte Carlo methods)

Q: which one is more powerful? Which one is easier to obtain/learn?
Model-free RL
Advantages:

- Model learning transfers across tasks and environment configurations (learning physics)
- Better exploits experience in case of sparse rewards
- Helps exploration: Can reason about model uncertainty
Advantages of Planning (Model-based RL)

Advantages:
- Model learning transfers across tasks and environment configurations (learning physics)
- Better exploits experience in case of sparse rewards
- Helps exploration: Can reason about model uncertainty

Disadvantages:
- First learn model, then construct a value function: Two sources of approximation error
Examples of Models for $T(s'|s, a)$

Table lookup model (tabular): bookkeeping a probability of occurrence for each transition $(s,a,s')$

Transition function is approximated through some function approximator
Examples of Models for $T(s'|s, a)$

Table lookup model (tabular): bookkeeping a probability of occurrence for each transition $(s,a,s')$

Transition function is approximated through some function approximator

This Lecture

Later..
Table Lookup Model

- Model is an explicit MDP \((T, R)\)
- Count visits \(N(s, a)\) to each state-action pair:

\[
\hat{T}(s'|s, a) = \frac{1}{N(s, a)} \sum_{t=1}^{\tau} 1(S_t, A_t, S_{t+1} = s, a, s')
\]
Table Lookup Model

- Model is an explicit MDP \((T, R)\)
- Count visits \(N(s, a)\) to each state-action pair:

\[
\hat{T}(s'|s, a) = \frac{1}{N(s, a)} \sum_{t=1}^{\tau} 1(S_t, A_t, S_{t+1} = s, a, s')
\]

\[
\hat{R}(s, a) = \frac{1}{N(s, a)} \sum_{t=1}^{\tau} 1(S_t, A_t = s, a, )R_t
\]
Table Lookup Model

- Model is an explicit MDP \((T, R)\)
- Count visits \(N(s, a)\) to each state-action pair:

\[
\hat{T}(s'|s, a) = \frac{1}{N(s, a)} \sum_{t=1}^{\tau} 1(S_t, A_t, S_{t+1} = s, a, s')
\]

\[
\hat{R}(s, a) = \frac{1}{N(s, a)} \sum_{t=1}^{\tau} 1(S_t, A_t = s, a) R_t
\]

- Alternatively
  - At each timestep \(t\), record experience tuple \((S_t, A_t, R_{t+1}, S_{t+1})\)
  - To sample model, randomly pick tuple matching \((s, a, .., )\)

Here, model learning means save the experience, memorization == learning
Two states $A, B$; no discounting; 8 episodes of experience

A, 0, B, 0
B, 1
B, 1
B, 1
B, 1
B, 1
B, 0

Construct a model from the experience
Example

Two states $A, B$; no discounting; 8 episodes of experience

A, 0, B, 0
B, 1
B, 1
B, 1
B, 1
B, 1
B, 0

We have constructed a **table lookup model** from the experience.
Given a model: $M_\eta = (T_\eta, R_\eta)$

Solve the MDP: $(S, A, T_\eta, R_\eta)$

Using favorite planning algorithm:

- Value iteration
- Policy iteration
Value iteration

- Experience
- Model
- Planning

Interaction with Environment

Model learning
Simulation

Direct RL methods
Greedification

Value function
Policy
Sample-based Planning

- Use the model only to generate samples, not using its transition probabilities and expected immediate rewards.

- Sample experience from model

\[ S_{t+1} \sim T_\eta(S_{t+1} | S_t, A_t) \]
\[ R_{t+1} = R_\eta(R_{t+1} | S_t, A_t) \]

- Apply model-free RL to samples, e.g.:
  - Monte-Carlo control
  - Sarsa \[ Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[ R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t) \right] \]
  - Q-learning \[ Q(s, a) \leftarrow Q(s, a) + \alpha \left[ R + \gamma \max_{a'} Q(S', a') - Q(s, a) \right] \]

- Sample-based planning methods are often more efficient: rather than exhaustive state sweeps we focus on what is likely to happen.
Sample-based planning

Experience -> Value function

Model learning -> Simulation

Planning

Model

Direct RL methods

Greedification

Policy

Interaction with Environment
Table of Contents

1 Definitions

2 Model

3 Recap: Dynamic Programming
Solving the Bellman equation is to find the optimal policy:

**State value function:**

\[
V^*(s) = \max_a \mathbb{E}[R_{t+1} + \gamma V^*(S_{t+1})|S_t = s, A_t = a] \\
= \max_a \sum_{s', r} p(s', r | s, a) [r + \gamma V^*(s')]
\]
Optimal Value Functions

Solving the Bellman equation is to find the optimal policy:

State value function:

\[
V^*(s) = \max_a \mathbb{E}[R_{t+1} + \gamma V^*(S_{t+1}) | S_t = s, A_t = a]
= \max_a \sum_{s', r} p(s', r | s, a) [r + \gamma V^*(s')]
\]

Action-state value function:

\[
Q^*(s, a) = \mathbb{E}[R_{t+1} + \gamma \max_{a'} Q^*(S_{t+1}, a') | S_t = s, A_t = a]
= \sum_{s', r} p(s', r | s, a) [r + \gamma \max_{a'} Q^*(s', a')]
\]
Iterative Policy Evaluation, for estimating $V \approx v_\pi$

Input $\pi$, the policy to be evaluated
Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in S^+$, arbitrarily except that $V(terminal) = 0$

Loop:
\[
\Delta \leftarrow 0
\]
Loop for each $s \in S$:
\[
v \leftarrow V(s)
\]
\[
V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) [r + \gamma V(s')]
\]
\[
\Delta \leftarrow \max(\Delta, |v - V(s)|)
\]
until $\Delta < \theta$
New greedy policy $\pi'$

\[ \pi'(s) = \arg \max_a Q_{\pi}(s, a) \]

\[ = \arg \max_a \mathbb{E}[r_{t+1} + \gamma V_{\pi}(S_{t+1}) | S_t = s, A_t = a] \]

\[ = \arg \max_a \sum_{s', r} p(s', r | s, a) [r + \gamma V_{\pi}(s')] \]
Recap: Dynamic Programming

### Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization
   
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in S$

2. Policy Evaluation
   
   Loop:
   
   $\Delta \leftarrow 0$
   
   Loop for each $s \in S$:
   
   $v \leftarrow V(s)$
   
   $V(s) \leftarrow \sum_{s',r} p(s', r | s, \pi(s)) [r + \gamma V(s')]$
   
   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement
   
   $policy-stable \leftarrow true$
   
   For each $s \in S$:
   
   $old-action \leftarrow \pi(s)$
   
   $\pi(s) \leftarrow \text{argmax}_a \sum_{s',r} p(s', r | s, a) [r + \gamma V(s')]$
   
   If $old-action \neq \pi(s)$, then $policy-stable \leftarrow false$
   
   If $policy-stable$, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2
Policy Improvement

Policy evaluation  Estimate $v_\pi$
Iterative policy evaluation

Policy improvement  Generate $\pi' \geq \pi$
Greedy policy improvement
Recap: Dynamic Programming

Advantages:
Recap: Dynamic Programming

Advantages:

- DP is efficient, it finds optimal policies in polynomial time for most cases
- DP is guaranteed to find optimal policy

Disadvantages:
Recap: Dynamic Programming

Advantages:
- DP is efficient, it finds optimal policies in polynomial time for most cases
- DP is guaranteed to find optimal policy

Disadvantages:
- DP is not suitable for large problems, with millions or more of states
- DP requires the knowledge of the transition probability matrix, however this is an unrealistic requirement for many problems