Overview

1 Definitions

2 Model

3 Dynamic Programming
Definitions

Learning: ?
Planning: ?
Definitions

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

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

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

1 Definitions

2 Model

3 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 $R(s,a)$. 
Distribution model: ?
Sample model ?
Distribution model: lists all possible outcomes and their probabilities, $T(s'|s, a)$ for all $(s, a, s')$. (We used those in DP)
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 used 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 used this in Monte Carlo methods)

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

- Model learning
- Simulation
- Planning
- Direct RL methods
- Greedification
- Policy

Interaction with Environment

Experience

Value function
Planning (or model-based RL)

- Experience
  - Interaction with Environment
  - Model learning
  - Direct RL methods
  - Greedification

- Value Function
  - Planning
  - Simulation

- Policy
  - Model

- CSE368 Artificial Intelligence, Lecture 7.3
  - June 19, 2019
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.
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 \((s, a, ., .)\)
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

We have constructed a table lookup model from the experience
Planning with a Model

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
Planning with a Model

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
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

\[
Q(S, A) \leftarrow Q(S, A) + \alpha \left[ R + \gamma \max_{a'} Q(S', a') - Q(S, A) \right]
\]

• 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

Interaction with Environment → Experience

Model learning → Model

Simulation → Planning

Value function → Policy

Direct RL methods

Greedification
Table of Contents

1 Definitions

2 Model

3 Dynamic Programming
In tabular value methods we aim to find the solution to Bellman Equation:

$$V_\pi(s) = \sum_A \pi(a|s) \sum_{s', r} p(s', r|S_t = s, A_t = a)(r + \gamma V_\pi(s'))$$
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:**

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

$$\pi'(s) = \arg \max_a Q_\pi(s, a)$$

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

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

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

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

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 $s \in S$:
   \[ old-action \leftarrow \pi(s) \]
   \[ \pi(s) \leftarrow \arg\max_a \sum_{s', r} p(s', r | s, a) [r + \gamma V(s')] \]
   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

$V = v_\pi$
$\pi = \text{greedy}(V)$
$V^* = v_{\pi^*}$
$\pi^* \geq \pi$
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.
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