Previous Lecture

- **RL**: Definitions, interaction API, tasks/challenges
- **MDPs**: Theoretical framework underlying RL, solving MDPs

Today

- **Policy** (continued): How an agents acts at states
- **Value function (Utility)**: How good is a particular state or state-action pair?
- **Algorithms** for solving MDPs (Value Iteration)
- Departure from known rewards and transitions: **Reinforcement Learning (RL), Deep RL**
Markov Decision Processes (MDPs)
- States, Actions, Reward dist., Transition dist., Discount factor (gamma)

Policy:
- Mapping from states to actions (deterministic)
- Distribution of actions given states (stochastic)

$$n = |S|$$
$$m = |A|$$

MDP
$$(S, A, R, T, \gamma)$$

<table>
<thead>
<tr>
<th>State</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>2</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
</tr>
</tbody>
</table>

Recap: MDPs, Policy
Markov Decision Processes (MDPs)
- States, Actions, Reward dist., Transition dist., Discount factor (gamma)

Policy:
- Mapping from states to actions (deterministic)
- Distribution of actions given states (stochastic)

What is a good policy?
- Maximize discounted sum of future rewards
- Discount factor: $\gamma$

Recap: MDPs, Policy
Formally, the **optimal policy** is defined as:

\[ \pi^* = \arg \max_\pi \mathbb{E} \left[ \sum_{t \geq 0} \gamma^t r_t \mid \pi \right] \]
Formally, the \textbf{optimal policy} is defined as:

\[ \pi^* = \arg \max_{\pi} \mathbb{E} \left[ \sum_{t \geq 0} \gamma^t r_t | \pi \right] \]

\[ s_0 \sim p(s_0), a_t \sim \pi(\cdot | s_t), s_{t+1} \sim p(\cdot | s_t, a_t) \]

Expectation over initial state, actions from policy, next states from transition distribution.
Some optimal policies for three different grid world MDPs ($\gamma = 0.99$)

- Varying reward for non-absorbing states (states other than $+1/-1$)

![Diagram with optimal policies for different reward values](Image Credit: Byron Boots, CS 7641)
For example, with an MDP with 5 states as shown, starting at the middle cell:

Actions: (Right, Left)
Deterministic transitions

What is the optimal policy for:

- $\gamma = 1$
- $\gamma = 0.1$

Slides adapted from: Byron Boots, CS 7641
A value function is a prediction of discounted sum of future reward

State value function / V-function / \( V : S \rightarrow \mathbb{R} \)
- How good is this state?
- Am I likely to win/lose the game from this state?

State-Action value function / Q-function / \( Q : S \times A \rightarrow \mathbb{R} \)
- How good is this state-action pair?
- In this state, what is the impact of this action on my future?
For a policy that produces a trajectory sample \((s_0, a_0, s_1, a_1, s_2 \cdots)\)

The **V-function** of the policy at state \(s\), is the expected cumulative reward from state \(s\):

\[
V^\pi(s) = \mathbb{E} \left[ \sum_{t \geq 0} \gamma^t r_t \mid s_0 = s, \pi \right]
\]

\(s_0 \sim p(s_0), a_t \sim \pi^\pi(\cdot \mid s_t), s_{t+1} \sim p(\cdot \mid s_t, a_t)\)
For a policy that produces a trajectory sample \((s_0, a_0, s_1, a_1, s_2 \cdots)\)

The **Q-function** of the policy at state \(s\) and action \(a\), is the expected cumulative reward upon taking action \(a\) in state \(s\) (and following policy thereafter):

\[
Q^\pi(s, a) = E\left[ \sum_{t \geq 0} \gamma^t r_t | s_0 = s, a_0 = a, \pi \right]
\]

\[s_0 \sim p(s_0), a_t \sim \pi(\cdot | s_t), s_{t+1} \sim p(\cdot | s_t, a_t)\]
The V and Q functions corresponding to the optimal policy $\pi^*$

$$V^*(s) = \mathbb{E} \left[ \sum_{t \geq 0} \gamma^t r_t \mid s_0 = s, \pi^* \right]$$

$$Q^*(s, a) = \mathbb{E} \left[ \sum_{t \geq 0} \gamma^t r_t \mid s_0 = s, a_0 = a, \pi^* \right]$$
Recursive Bellman expansion (from definition of Q)

\[ Q^*(s, a) = \mathbb{E}_{a_t \sim \pi^*(\cdot | s_t)} \sum_{t \geq 0} \gamma^t r(s_t, a_t) \mid s_0 = s, a_0 = a \]

(Expected) return from \( t = 0 \)

\[ = r(s, a) + \gamma \mathbb{E}_{s' \sim p(s' | s, a)} \left[ V^*(s') \right] \]

\[ = \mathbb{E}_{s' \sim p(s' | s, a)} \left[ r(s, a) + \gamma V^*(s') \right] \]
Equations relating optimal quantities

\[ V^*(s) = \max_a Q^*(s, a) \quad \text{and} \quad \pi^*(s) = \arg \max_a Q^*(s, a) \]

Recursive Bellman optimality equation

\[
Q^*(s, a) = \mathbb{E}_{s' \sim p(s' | s, a)} \left[ r(s, a) + \gamma V^*(s') \right] \\
= \sum_{s'} p(s' | s, a) \left[ r(s, a) + \gamma V^*(s') \right] \\
= \sum_{s'} p(s' | s, a) \left[ r(s, a) + \gamma \max_{a'} Q^*(s', a') \right]
\]

\[
V^*(s) = \max_a \sum_{s'} p(s' | s, a) \left[ r(s, a) + \gamma V^*(s') \right]
\]
Based on the bellman optimality equation

$$V^*(s) = \max_a \sum_{s'} p(s'|s,a) \left[ r(s,a) + \gamma V^*(s') \right]$$

Algorithm

- Initialize values of all states
- While not converged:
  - For each state: $$V_{i+1}(s) \leftarrow \max_a \sum_{s'} p(s'|s,a) \left[ r(s,a) + \gamma V^i(s') \right]$$
- Repeat until convergence (no change in values)

Time complexity per iteration $$O(|S|^2|A|)$$
Value Iteration Update:

\[ V^{i+1}(s) \leftarrow \max_a \sum_{s'} p(s'|s,a) \left[ r(s,a) + \gamma V^i(s') \right] \]

Q-Iteration Update:

\[ Q^{i+1}(s,a) \leftarrow \]
Policy iteration: Start with arbitrary $\pi_0$ and refine it.

Involves repeating two steps:
- **Policy Evaluation**: Compute $V^{\pi}$ (similar to Value Iteration)
- **Policy Refinement**: Greedily change actions as per $V^{\pi}$

Why do policy iteration?
- $\pi_0$ often converges to $\pi^*$ much sooner than $V^{\pi_0}$ to $V^{\pi^*}$
For Value Iteration:

Time complexity per iteration $O(|S|^2|A|)$

- 3x4 Grid world?
- Chess/Go?
- Atari Games with integer image pixel values [0, 255] of size 16x16 as state?
- **Value Iteration**
  - Bellman update to state value estimates

- **Q-Value Iteration**
  - Bellman update to (state, action) value estimates

- **Policy Iteration**
  - Policy evaluation + refinement

Summary: MDP Algorithms
Reinforcement Learning, Deep RL
Recall RL assumptions:

- $\mathbb{P}(s, a, s')$ unknown, how actions affect the environment.
- $\mathcal{R}(s, a, s')$ unknown, what/when are the good actions?

But, we can learn by trial and error.
- Gather experience (data) by performing actions.
- Approximate unknown quantities from data.

Reinforcement Learning
- Old Dynamic Programming Demo
  - [https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld_dp.html](https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld_dp.html)

- RL Demo
  - [https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld_td.html](https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld_td.html)
In addition to not knowing the environment, sometimes the state space is too large.

Recall: Value iteration not scalable (chess, RGB images as state space, etc)

Solution: Deep Learning! … more precisely, function approximation.
- Use deep neural networks to learn state representations
- Useful for continuous action spaces as well

Deep Reinforcement Learning
In today’s class, we looked at:

- **Dynamic Programming** for solving MDPs
  - Value, Q-Value Iteration
  - Policy Iteration

- **Reinforcement Learning (RL)**
  - The challenges of (deep) learning based methods

Next class:

- **Value-based RL algorithms**
  - Deep Q-Learning
- **Policy-based RL algorithms** (policy gradients)