CS 4803 / 7643
Deep Learning, Fall 2020

Reinforcement Learning: Module 2/3

Presented by Nirbhay Modhe

Georgia Tech

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**

Recap & Overview

Georgia Tech

Recap: MDPs, Policy

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

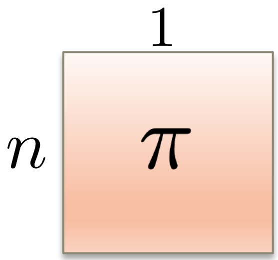
MDP

$$(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathbb{T}, \gamma)$$

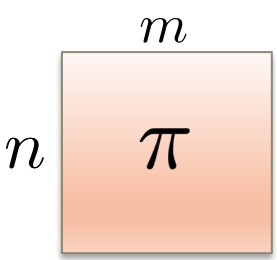
- Policy:
  - Mapping from states to actions (deterministic)
  - Distribution of actions given states (stochastic)

State	Action
A ⟶	2
B ⟶	1

$$n = |\mathcal{S}|$$
$$m = |\mathcal{A}|$$



Deterministic



Stochastic

Recap: MDPs, Policy

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

MDP
$(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathbb{T}, \gamma)$

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

State	Action
A →	2
B →	1

- What is a good policy?
  - Maximize **discounted sum of future rewards**
    - Discount factor: γ

1 Worth Now | γ Worth Next Step | γ^2 Worth In Two Steps

- Formally, the **optimal policy** is defined as:

discounted sum of future rewards

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

?

Solving MDPs: Optimal policy

Georgia Tech

- Formally, the **optimal policy** is defined as:

discounted sum of future rewards

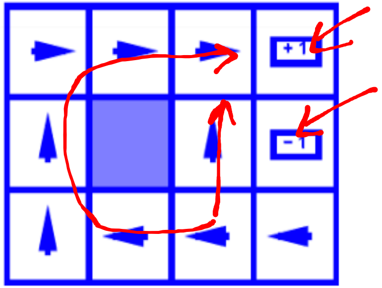
$$\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t \middle| \pi\right]$$

$$\boldsymbol{s}_0 \sim p(\boldsymbol{s}_0), a_t \sim \pi(\cdot|\boldsymbol{s}_t), \boldsymbol{s}_{t+1} \sim p(\cdot|\boldsymbol{s}_t, a_t)$$

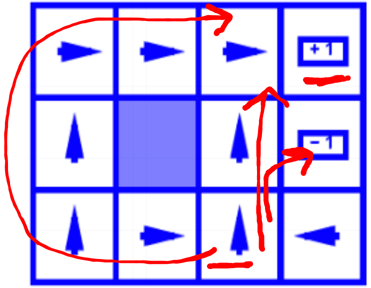
Expectation over initial state, actions from policy,
next states from transition distribution

Solving MDPs: Optimal policy

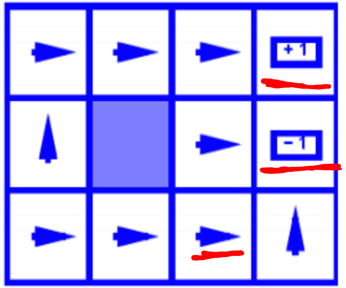
- Some optimal policies for three different grid world MDPs (gamma=0.99)
  - Varying reward for non-absorbing states (states other than +1/-1)



R(s) = -0.03



R(s) = -0.4



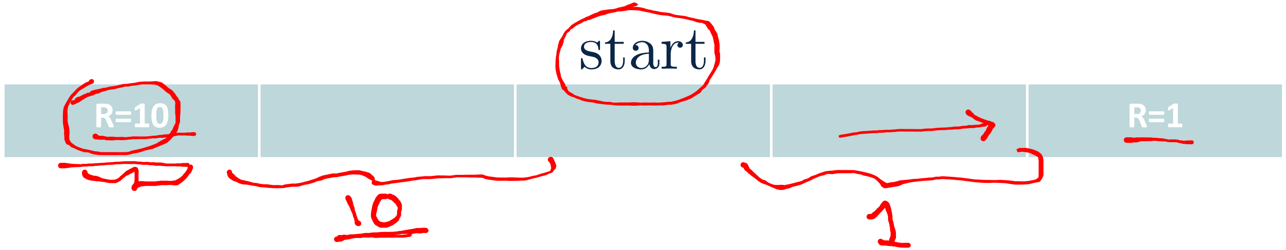
R(s) = -2.0

Image Credit: Byron Boots, CS 7641

Optimal policy examples

Georgia Tech

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

10

Slides adapted from: Byron Boots, CS 7641

Discounting future rewards

- A **value function** is a prediction of discounted sum of future reward

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

- **State-Action** value function / **Q**-function / $Q : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$
  - How good is this state-action pair?
  - In this state, what is the impact of this action on my future?

Value Function

- 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:

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

n x 1

$$\boldsymbol{s}_0 \sim p(\boldsymbol{s}_0), a_t \sim \pi(\cdot|\boldsymbol{s}_t), \boldsymbol{s}_{t+1} \sim p(\cdot|\boldsymbol{s}_t, a_t)$$

Value Function

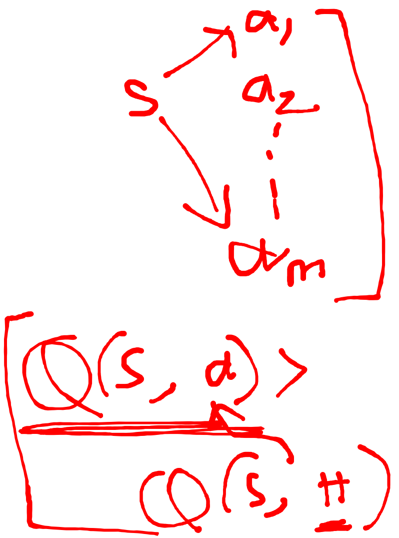
Action-Value Function

- 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):

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

$$\boldsymbol{s}_0 \sim p\left(\boldsymbol{s}_0\right), a_t \sim \pi\left(\cdot | \boldsymbol{s}_t\right), \boldsymbol{s}_{t+1} \sim p\left(\cdot | \boldsymbol{s}_t, a_t\right)$$



- The V and Q functions corresponding to the optimal policy $\pi^\star$

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

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

Optimal V & Q functions

- Recursive Bellman expansion (from definition of Q)

(Expected) return from t = 0

$$Q^*(s,a) = \mathop{\mathbb{E}}_{\substack{a_t \sim \pi^*(\cdot|s_t) \\ s_{t+1} \sim p(\cdot|s_t,a_t)}} \left[\sum_{t\geq 0} \gamma^t r(s_t,a_t) \mid s_0 = s, a_0 = a \right]$$

(Reward at t = 0) + gamma * (Return from expected state at t=1)

$$= \gamma^0 r(s,a) + \mathop{\mathbb{E}}_{s' \sim p(\cdot|s,a)} \left[\gamma \mathop{\mathbb{E}}_{\substack{a_t \sim \pi^*(\cdot|s_t) \\ s_{t+1} \sim p(\cdot|s_t,a_t)}} \left[\sum_{t\geq 1} \gamma^{t-1} r(s_t,a_t) \mid s_1 = s' \right] \right]$$

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

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

Bellman Optimality Equations

- Equations relating optimal quantities

$$V^*(s) = \max_a Q^*(s, a)$$

$$\pi^*(s) = \arg\max_a Q^*(s, a)$$

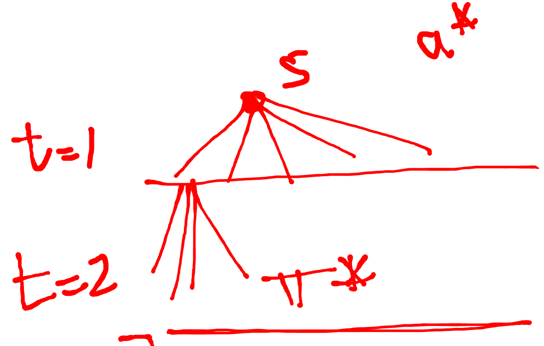
- Recursive Bellman optimality equation

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

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

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

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

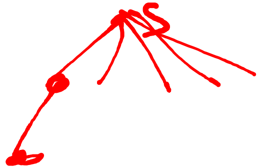


Bellman Optimality Equations

Value Iteration

- Based on the bellman optimality equation

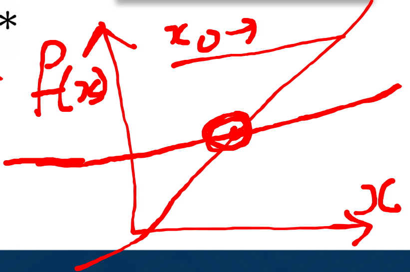
$$V^*(s) = \max_a \sum_{s'} p(s'|s,a)\left[r(s,a) + \gamma V^*(s')\right]$$



- 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)

$$V^0 \to V^1 \to V^2 \to \quad \cdots \to V^i \to \ldots \quad \to V^*$$

Homework



Time complexity per iteration $O(|\mathcal{S}|^2|\mathcal{A}|)$

- Value Iteration Update:

$V(s) = \max_a Q(s,a)$

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

- Q-Iteration Update:

$$Q^{i+1}(s,a) \leftarrow$$

- Policy iteration: Start with arbitrary π_0 and refine it.

$$\pi_0 \rightarrow \pi_1 \rightarrow \pi_2 \rightarrow \quad \ldots \quad \rightarrow \pi^*$$

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

$$\pi_0 \longrightarrow V^{\pi_0} \longrightarrow \pi_1 \longrightarrow V^{\pi_1} \longrightarrow \quad \ldots \quad \longrightarrow \pi^* \longrightarrow V^{\pi^*}$$

- Why do policy iteration?
  - π_i often converges to π^* much sooner than V^{π_i} to V^{π^*}

Policy Iteration

For Value Iteration:

Time complexity per iteration $O(|\mathcal{S}|^2|\mathcal{A}|)$

- 3x4 Grid world? 12 4
- Chess/Go? ← 10^{120}
- Atari Games with integer image pixel values [0, 255] of size 16x16 as state? 256^{16x16}

State Spaces & Time Complexity

- **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:
  - $\mathbb{T}(s, a, s')$ 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
- RL Demo
  - https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld_td.html

Slide credit: Dhruv Batra

- 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)

Summary