Lecture 4: Control MDP with policy iteration and value iteration

Bolei Zhou

The Chinese University of Hong Kong

bzhou@ie.cuhk.edu.hk

January 15, 2020

Today's Plan

1. Last Time
   - Markov Process, Markov Reward Process, and Markov Decision Process
   - Policy evaluation: evaluate the value of state given a policy

2. This Time: Control in MDP
   - Policy iteration and value iteration

Refresh on Policy Evaluation

1. Estimating value function in MDP using Bellman equation:

$$v_\pi(s) = \sum_{a \in A} \pi(a|s)(R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) v_\pi(s')) \tag{1}$$

2. A live demo: `https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld_dp.html`

Policy Search

1. One option is to enumerate search the best policy
2. Number of deterministic policies is $|\mathcal{A}|^{|\mathcal{S}|}$
3. Other approaches such as policy iteration and value iteration are more efficient

MDP Control

1. Compute the optimal policy

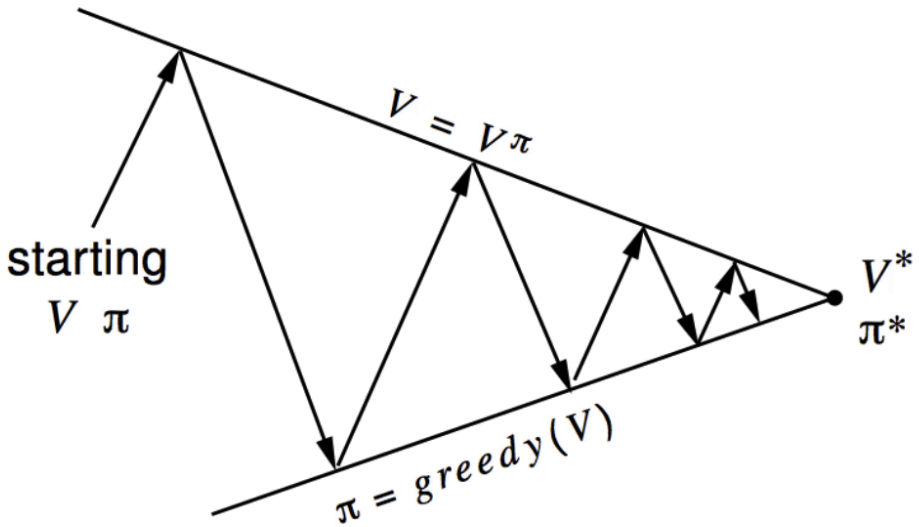
$$\pi_*(s) = \arg\max_{\pi} v_\pi(s) \tag{2}$$

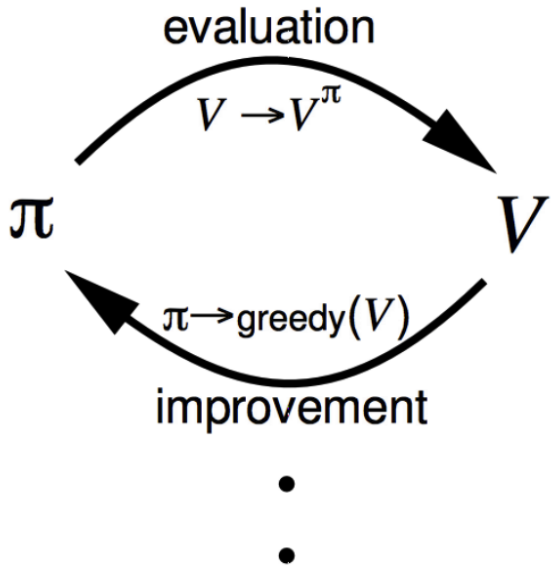
2. Optimal policy for a MDP in an infinite horizon problem (agent acts forever) is
   1. Deterministic
   2. Stationary (does not depend on time step)
   3. Unique? Not necessarily, may have state-actions with identical optimal values

Improve a Policy through Policy Iteration

1. Iterate through the two steps:
   1. Evaluate the policy π (computing v given current π)
   2. Improve the policy by acting greedily with respect to v_π

$$\pi' = \text{greedy}(v_\pi) \tag{3}$$





Policy Improvement

1. compute the state-action value of a policy π:

$$q_{\pi_i}(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) v_{\pi_i}(s') \tag{4}$$

2. Compute new policy π_{i+1} for all $s \in \mathcal{S}$ following

$$\pi_{i+1}(s) = \arg\max_a q_{\pi_i}(s,a) \tag{5}$$

3. We can prove that Monotonic Improvement in Policy.

Monotonic Improvement in Policy

1. Consider a determinisitc policy $a = \pi(s)$
2. We improve the policy through

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

3. This improves the value from any state s over one step,

$$q_\pi(s, \pi'(s)) = \max_{a \in \mathcal{A}} q_\pi(s, a) \geq q_\pi(s, \pi(s)) = v_\pi(s)$$

4. It therefore improves the value function, $v_{\pi'}(s) \geq v_\pi(s)$

$$\begin{aligned} v_\pi(s) \leq & q_\pi(s, \pi'(s)) = \mathbb{E}_{\pi'}[R_{t+1} + \gamma v_\pi(S_{t+1}|S_t = s)] \\ \leq & \mathbb{E}_{\pi'}[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1}))|S_t = s] \\ \leq & \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 q_\pi(S_{t+2}, \pi'(S_{t+2}))|S_t = s] \\ \leq & \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + ...|S_t = s] = v_{\pi'}(s) \end{aligned}$$

Monotonic Improvement in Policy

1. If improvements stop,

$$q_\pi(s, \pi'(s)) = \max_{a \in \mathcal{A}} q_\pi(s, a) = q_\pi(s, \pi(s)) = v_\pi(s)$$

2. Thus the Bellman optimality equation has been satisfied

$$v_\pi(s) = \max_{a \in \mathcal{A}} q_\pi(s, a)$$

3. Therefore $v_\pi(s) = v_*(s)$ for all $s \in \mathcal{S}$, so π is an optimal policy

Bellman Optimality Equation

1. The optimal value functions are reached by the Bellman optimality equations:

$$v_*(s) = \max_a q_*(s,a)$$
$$q_*(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) v_*(s')$$

thus

$$v_*(s) = \max_a R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) v_*(s')$$
$$q_*(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) \max_{a'} q_*(s',a')$$

Value Iteration by turning the Bellman Optimality Equation as update rule

1. If we know the solution to subproblem $v_*(s')$, which is optimal.
2. Then the solution for the optimal $v_*(s)$ can be found by iteration over the following Bellman Optimality backup rule,

$$v(s) \leftarrow \max_{a \in \mathcal{A}} \left(R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) v(s') \right)$$

3. The idea of value iteration is to apply these updates iteratively

Algorithm of Value Iteration

1. Problem: find the optimal policy π
2. Solution: iteration on the Bellman optimality backup
3. Value Iteration algorithm:
   1. initialize $k = 1$ and $v_0(s) = 0$ for all states s
   2. For $k = 1 : H$
      1. for each state s

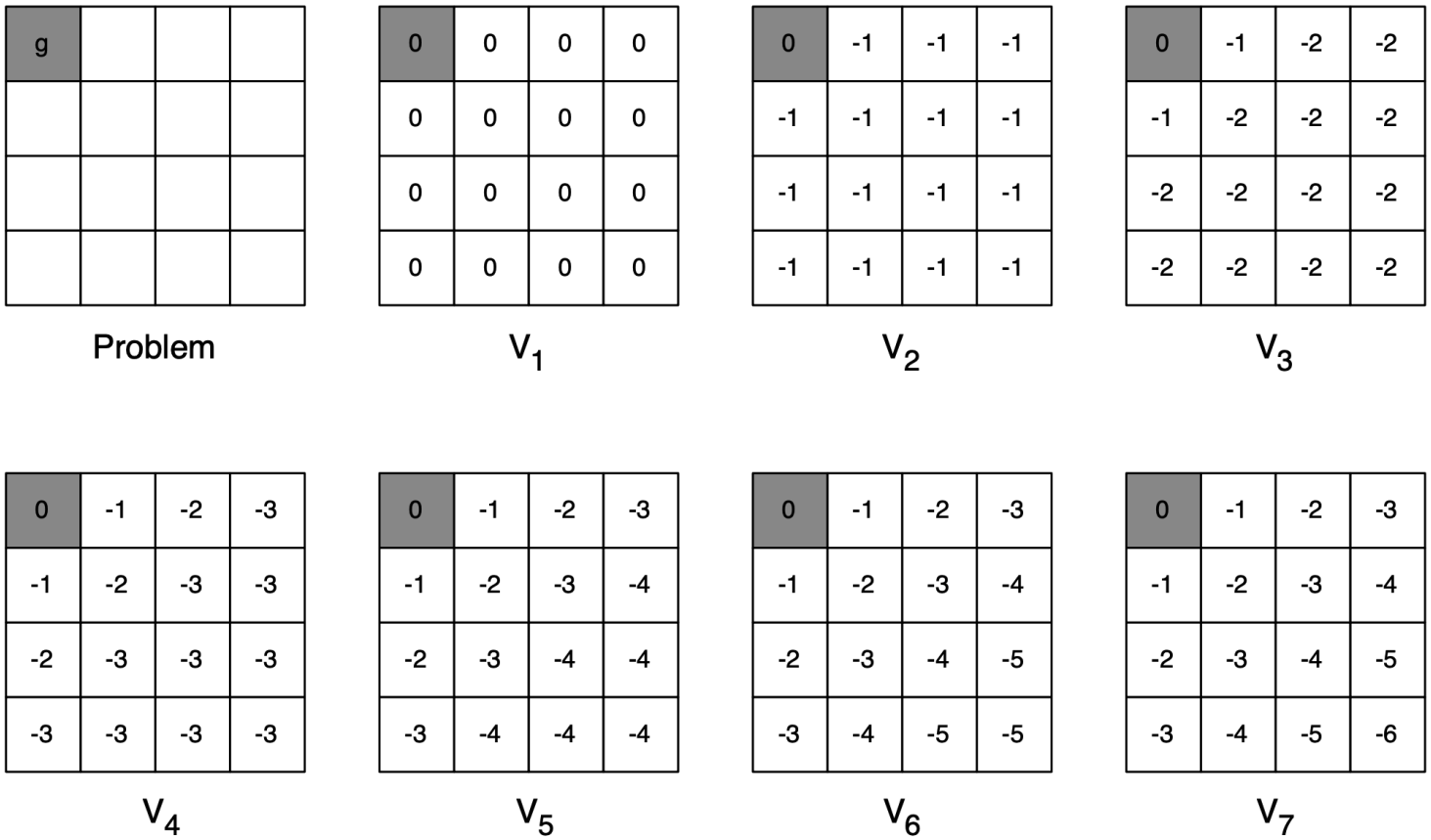
$$q_{k+1}(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) v_k(s') \tag{6}$$

$$v_{k+1}(s) = \max_a q_{k+1}(s,a) \tag{7}$$

      2. $k \leftarrow k + 1$
   3. To retrieve the optimal policy after the value iteration:

$$\pi(s) = \arg\max_a R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) v_{k+1}(s') \tag{8}$$

Example: Shortest Path



After the optimal values are reached, we run policy extraction to retrieve the optimal policy.

Difference between Policy Iteration and Value Iteration

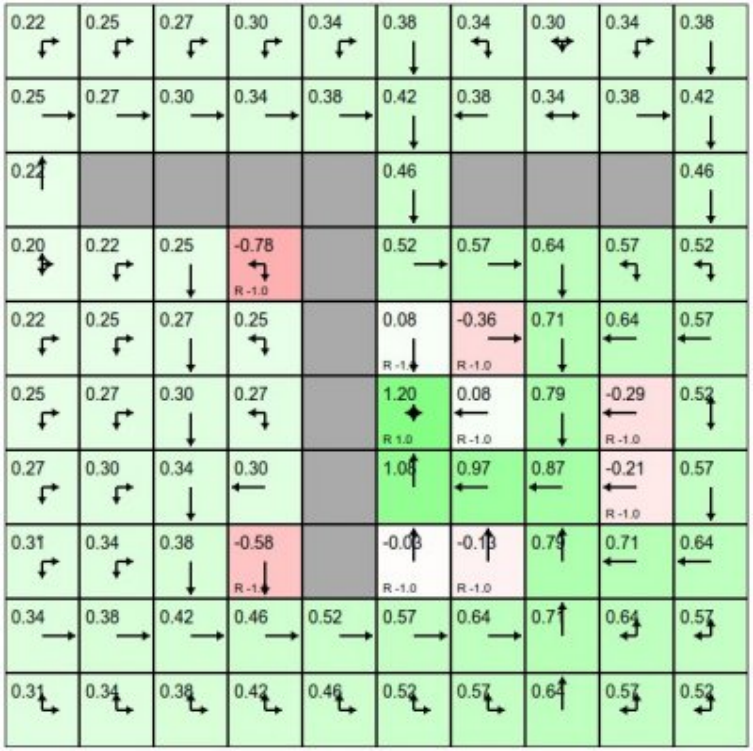
1. Policy iteration includes: **policy evaluation** + **policy improvement**, and the two are repeated iteratively until policy converges.
2. Value iteration includes: **finding optimal value function** + **one policy extraction**. There is no repeat of the two because once the value function is optimal, then the policy out of it should also be optimal (i.e. converged).
3. Finding optimal value function can also be seen as a combination of policy improvement (due to max) and truncated policy evaluation (the reassignment of v(s) after just one sweep of all states regardless of convergence).

Summary for Prediction and Control in MDP

Table: Dynamic Programming Algorithms

Problem	Bellman Equation	Algorithm
Prediction	Bellman Expectation Equation	Iterative Policy Evaluation
Control	Bellman Expectation Equation	Policy Iteration
Control	Bellman Optimality Equation	Value Iteration

Demo using REINFORCEjs library



1. Iteration of policy evaluation and policy improvement(update)
2. Value iteration
3. Demo link: `https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld_dp.html`

Example for Understanding MDP, policy iteration, and value iteration.
OpenAI FrozenLake-v0 (https://gym.openai.com/envs/FrozenLake-v0/)
"SFFF",
"FHFH",
"FFFH",
"HFFG"
corresponding to Environment state :
" 0, 1, 2, 3",
" 4, 5, 6, 7",
" 8, 9,10,11",
"12,13,14,15",
Action: "0, 1, 2, 3" is "left, down, right, up".
Condition:
The episode ends when you reach the goal or fall in a hole.
You receive a reward of 1 if you reach the goal, and zero otherwise.
Slippery:
Go Right from state 1, it has three possible states to go,
not deterministically to state 2

Policy iteration and value iteration on FrozenLake

1. https://github.com/metalbubble/RLexample/tree/master/MDP

Asynchronous Dynamic Programming

1. A major drawback to the DP methods is that they involve operations over the entire state set of the MDP, that is, they require sweeps of the state set.
2. If the state set is very large, for example, the game of backgammon has over 10^{20} states. Thousands of years to be taken to finish one sweep.
3. Asychronous DP algorithms are in-place iterative DP that are not organized in terms of systematic sweeps of the state set
4. The values of some states may be updated several times before the values of others are updated once.

Improving Dynamic Programming

Synchronoous dynamic programming is usually slow. Three simple ideas to extend DP for asynchronous dynamic programming:

1. In-place dynamic programming
2. Prioritized sweeping
3. Real-time dynamic programming

In-Places Dynamic Programming

1. Synchronous value iteration stores two copies of value function:
   for all s in $\mathcal{S}$
   $$v_{new}(s) \leftarrow \max_{a \in \mathcal{A}} \left(R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) v_{old}(s') \right)$$
   $v_{old} \leftarrow v_{new}$
2. In-place value iteration only stores one copy of value function:
   for all s in $\mathcal{S}$
   $$v(s) \leftarrow \max_{a \in \mathcal{A}} \left(R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) v(s') \right)$$

Prioritized Sweeping

1. Use magnitude of Bellman error to guide state selection, e.g.

$$\left| \max_{a \in \mathcal{A}} \left(R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) v(s') \right) - v(s) \right|$$

2. Backup the state with the largest remaining Bellman error
3. Update Bellman error of affected states after each backup
4. Can be implemented efficiently by maintaining a priority queue

Real-Time Dynamic Programming

1. To solve a given MDP, we can run an iterative DP algorithm at the same time that an agent is actually experiencing the MDP
2. The agent's experience can be used to determine the states to which the DP algorithm applies its updates
3. We can apply updates to states as the agent visits them. So focus on the parts of the state set that are most relevant to the agent
4. After each time-step S_t, A_t, backup the state S_t,

$$v(S_t) \leftarrow \max_{a \in \mathcal{A}} \left(R(S_t, a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|S_t, a) v(s') \right)$$

Sample Backups

1. The key design for RL algorithms such as Q-learning and SARSA in next lectures
2. Using sample rewards and sample transition pairs $< S, A, R, S' >$, rather than the reward function $\mathcal{R}$ and transition dynamics $\mathcal{P}$
3. Benefits:
   1. Model-free: no advance knowledge of MDP required
   2. Break the curse of dimensionality through sampling
   3. Cost of backup is constant, independent of $n = |\mathcal{S}|$

Approximate Dynamic Programming

1. Using a function approximator $\hat{v}(s, \mathbf{w})$
2. Fitted value iteration repeats at each iteration k,
   1. Sample state s from the state cache $\tilde{\mathcal{S}}$

$$\tilde{v}_k(s) = \max_{a \in \mathcal{A}} \left(R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) \hat{v}(s', \mathbf{w}_k) \right)$$

   2. Train next value function $\hat{v}(s', \mathbf{w}_{k+1})$ using targets $< s, \tilde{v}_k(s) >$.
3. Key idea behind the Deep Q-Learning

End

1. Summary: policy evaluation, policy iteration, and value iteration are introduced
2. Practice: try code in FrozenLake: `https://github.com/cuhkrlcourse/RLexample/tree/master/MDP`
3. Homework 1 is released at `https://github.com/cuhkrlcourse/ierg6130-assignment`
    1. Due: 23:59, Feb.5, 2020 (The week after CNY)
    2. Please start working on Section 1 and 2
4. Next Week: Model-free methods
    1. Reading: Textbook Chapter 5 and 6