Recap of Last Module (“The RL Problem”)

- The Agent-Environment Interface
- Goals, Rewards, Returns
- The Markov Property
- The Markov Decision Process
- Value Functions
- Optimal Value Functions
- Optimality and Approximation

MDP Recap

- Finite MDP: {S, A, R, p, γ}
- Model: p(s', r | s, a)

Value Functions Recap

- State-value function: $v_\pi(s)$
- Action-value function: $q_\pi(s, a)$
- Optimal state-value function: $v_*(s)$
- Optimal action-value function: $q_*(s, a)$

Bellman Equations - Summary

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')]$$

$$q_\pi(s,a) = \sum_{s',r} p(s',r|s,a)[r + \gamma \sum_{a'} p(a'|s') q_\pi(s',a')]$$

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

$$q_*(s,a) = \sum_{s',r} p(s',r|s,a)[r + \gamma \max_{a'} q_*(s',a')]$$

Microsoft

Dynamic Programming

ROLAND FERNANDEZ

Researcher, MSR AI

Instructor, AI School

Outline

- What is Dynamic Programming?
- Policy Evaluation
- Policy Improvement
- Policy Iteration
- Value Iteration
- Asynchronous DP
- GPI
- Efficiency of DP

Outline

- What is Dynamic Programming?
- Policy Evaluation
- Policy Improvement
- Policy Iteration
- Value Iteration
- Asynchronous DP
- GPI
- Efficiency of DP

What is Dynamic Programming?

- Aka Dynamic Optimization
- General Technique
- Overlapping Subproblems

Dynamic Programming for finite MDPs

- **Key idea:** use values function to organize and structure the search for good policies
- **Key idea**: can turn Bellman equations into *iterative updates*

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')]$$

- The overlapping subproblems on the value functions on the right-hand side
- This is aka *planning* – since it uses complete model of MDP (vs, environment interaction)

Outline

- What is Dynamic Programming?
- Policy Evaluation
- Policy Improvement
- Policy Iteration
- Value Iteration
- Asynchronous DP
- GPI
- Efficiency of DP

Policy Evaluation

- Goal:
  - Given a policy, compute the long term value of each state
  - Formally: given policy π, compute $v_\pi(s)$ (for all $s \in S$)
  - Also called the *prediction problem* of planning

Policy Evaluation

- Method: *Iterative policy evaluation*:

$$v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_k(s')]$$

  - Two array vs. in-place updating
  - Called *expected* (vs. *sampled*) updates
  - This is an example of *bootstrapping*

Policy Evaluation

- Convergence
  - Converges when $v_{k+1}(s) = v_k(s)$
  - Convergence guaranteed if $\gamma < 1$ or termination is guaranteed
  - In-place updating: state order affects convergence rate

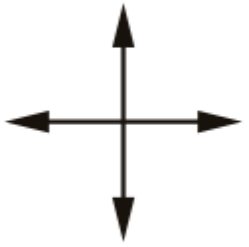
Algorithm: Iterative Policy Evaluation

Iterative policy evaluation

Input π, the policy to be evaluated
Initialize an array $V(s) = 0$, for all $s \in \mathcal{S}^+$
Repeat
    $\Delta \leftarrow 0$
    For each $s \in \mathcal{S}$:
        $v \leftarrow V(s)$
        $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\left[r + \gamma V(s')\right]$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)
Output $V \approx v_\pi$

Image Credit: Sutton and Barto, Reinforcement Learning, An Introduction 2017

Iterative Policy Evaluation: Gridworld Example



actions

	1	2	3
4	5	6	7
8	9	10	11
12	13	14	

$R_t = -1$
on all transitions

$$v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_k(s')]$$

$$v_{k+1}(s) = \sum_a .25 \sum_{s',r} [r + v_k(s')]$$

Image Credit: Sutton and Barto, Reinforcement Learning, An Introduction 2017

Gridworld Example

$$v_{k+1}(s) = \sum_a .25 \sum_{s',r} [r + v_k(s')]$$

	1	2	3
4	5	6	7
8	9	10	11
12	13	14	

k = 0

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

k = 1

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	0

Gridworld Example

$$v_{k+1}(s) = \sum_a .25 \sum_{s',r} [r + v_k(s')]$$

	1	2	3
4	5	6	7
8	9	10	11
12	13	14	

k = 1

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	0

k = 2

0	-1.75	-2	-2
-1.75	-2	-2	-2
-2	-2	-2	-1.75
-2	-2	-1.75	0

Gridworld Example

$$v_{k+1}(s) = \sum_a .25 \sum_{s',r} [r + v_k(s')]$$

	1	2	3
4	5	6	7
8	9	10	11
12	13	14	

k = 113

0	-14	-20	-22
-14	-18	-20	-20
-20	-20	-18	-14
-22	-20	-14	0

Outline

- What is Dynamic Programming?
- Policy Evaluation
- Policy Improvement
- Policy Iteration
- Value Iteration
- Asynchronous DP
- GPI
- Efficiency of DP

Policy Improvement

- How can we compare two policies to find which is better?
- Policy Improvement Theorem:
  - For all states, if the value of following the new policy for 1 step and then following the current policy >= the value of following the current policy, then the new policy is better than or equal to the current policy
  - Formally:

$$\forall s \in S, q_\pi(s, \pi'(s)) \geq v_\pi(s) \implies v'_\pi(s) \geq v_\pi(s)$$

  - This is Policy Improvement, aka the *control problem* of planning

Policy Improvement

- By policy improvement theorem, *greedy policy* will be better than or equal to our current policy:

$$\pi'(s) = \arg\max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')]$$

- We will use greedy policy as our policy improvement method
- If greedy policy doesn't improve our policy, our policy is optimal

Outline

- What is Dynamic Programming?
- Policy Evaluation
- Policy Improvement
- Policy Iteration
- Value Iteration
- Asynchronous DP
- GPI
- Efficiency of DP

Policy Iteration

- We have seen:
  - Given initial policy, we can find $v_\pi(s)$ using Iterative Policy Evaluation
  - Given $v_\pi(s)$, we can find improved policy π' using Policy Improvement
- Repeat this process:
  - monotonically improving policies and values functions

$$\pi_0 \xrightarrow{\text{E}} v_{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} v_{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} v_*,$$

Algorithm: Policy Iteration

Policy iteration (using iterative policy evaluation)

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

2. Policy Evaluation
   Repeat
   $\Delta \leftarrow 0$
   For each $s \in \mathcal{S}$:
   $v \leftarrow V(s)$
   $V(s) \leftarrow \sum_{s',r} p(s', r \mid s, \pi(s)) \left[r + \gamma V(s') \right]$
   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   old-action $\leftarrow \pi(s)$
   $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma V(s') \right]$
   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

Image Credit: Sutton and Barto, Reinforcement Learning, An Introduction 2017

Outline

- What is Dynamic Programming?
- Policy Evaluation
- Policy Improvement
- Policy Iteration
- Value Iteration
- Asynchronous DP
- GPI
- Efficiency of DP

Value Iteration

- What's wrong with Policy Iteration?
  - We have to wait for each round of Policy Evaluation to converge
- Solutions
  - Can approx. value function by stopping after N state sweeps of Policy Evaluation
  - Convergence still guaranteed for discounted, finite MDPs
  - Stop after 1 sweep = *Value Iteration*
  - Single update to combine Policy Improvement with truncated Policy Evaluation:

$$v_{k+1}(s) = \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_k(s')]$$

Algorithm: Value Iteration

Value iteration

Initialize array V arbitrarily (e.g., $V(s) = 0$ for all $s \in \mathcal{S}^+$)

Repeat
  $\Delta \leftarrow 0$
  For each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, $\pi \approx \pi_*$, such that
  $\pi(s) = \arg\max_a \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$

Image Credit: Sutton and Barto, Reinforcement Learning, An Introduction 2017

Outline

- What is Dynamic Programming?
- Policy Evaluation
- Policy Improvement
- Policy Iteration
- Value Iteration
- Asynchronous DP
- GPI
- Efficiency of DP

Asynchronous DP

- The problem
  - Normal DP requires multiple sweeps of state space
  - For some problems, cannot do even a single state sweep
    - Backgammon: 10**20 states, > 1 thousand years / sweep
- Asynchronous DP
  - In-place iterative DP algorithms that don’t use systematic state sweeps
  - States updated in any order, multiple times
  - For convergence, all states must be updated eventually

Outline

- What is Dynamic Programming?
- Policy Evaluation
- Policy Improvement
- Policy Iteration
- Value Iteration
- Asynchronous DP
- GPI
- Efficiency of DP

GPI (Generated Policy Iteration)

- Generalizing the interaction of Policy Evaluation and Policy Improvement processes:
  - Sync vs. Async
  - Various levels of granularity between interaction
  - Competition and cooperation

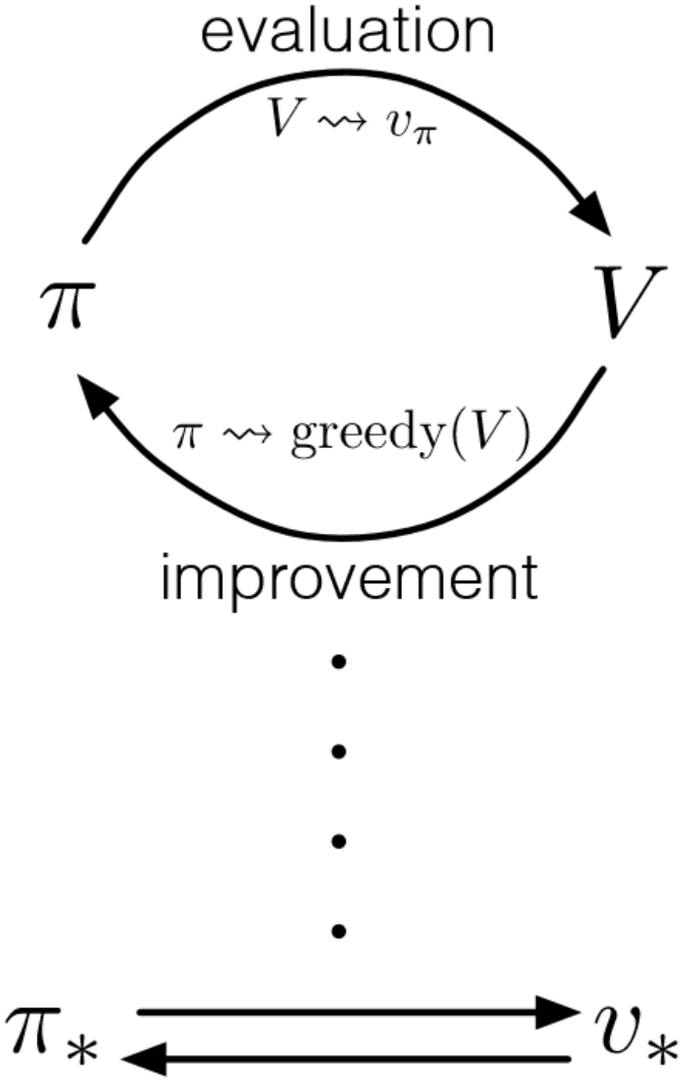


Image Credit: Sutton and Barto, Reinforcement Learning, An Introduction 2017

Outline

- What is Dynamic Programming?
- Policy Evaluation
- Policy Improvement
- Policy Iteration
- Value Iteration
- Asynchronous DP
- GPI
- Efficiency of DP

Efficiency of DP

- Not Practical for Large Problems
- Efficient compared to other MDP methods:
  - Polynomial in number of states and actions
- Today's computers can solve DP models with millions of states
- Approximate DP methods used for large problems

Summary

- What is Dynamic Programming?
- Components:
  - Policy Evaluation
  - Policy Improvement
- Algorithms:
  - Policy Iteration
  - Value Iteration
  - Asynchronous DP
- Observations:
  - GPI
  - Efficiency of DP