Dynamic Programming

DEPARTMENT OF COMPUTER SCIENCE
ST. FRANCIS XAVIER UNIVERSITY

CSCI-531 - Reinforcement Learning

Fall 2025

From Theory to Practice: Computing Optimal Policies

What We Know So Far

- How to model problems as **MDPs**

From Theory to Practice: Computing Optimal Policies

What We Know So Far

- How to model problems as **MDPs**
- Why we need **policies** and **value functions**

From Theory to Practice: Computing Optimal Policies

What We Know So Far

- How to model problems as **MDPs**
- Why we need **policies** and **value functions**
- What **optimal policies** look like mathematically

From Theory to Practice: Computing Optimal Policies

What We Know So Far

- How to model problems as **MDPs**
- Why we need **policies** and **value functions**
- What **optimal policies** look like mathematically

The Remaining Challenge

How do we actually compute these optimal policies?

From Theory to Practice: Computing Optimal Policies

What We Know So Far

- How to model problems as **MDPs**
- Why we need **policies** and **value functions**
- What **optimal policies** look like mathematically

The Remaining Challenge

How do we actually compute these optimal policies?

The Approach Menu

- **Dynamic Programming**: Complete knowledge of MDP

From Theory to Practice: Computing Optimal Policies

What We Know So Far

- How to model problems as **MDPs**
- Why we need **policies** and **value functions**
- What **optimal policies** look like mathematically

The Remaining Challenge

How do we actually compute these optimal policies?

The Approach Menu

- **Dynamic Programming**: Complete knowledge of MDP
- **Monte Carlo Methods**: Learn from experience

From Theory to Practice: Computing Optimal Policies

What We Know So Far

- How to model problems as **MDPs**
- Why we need **policies** and **value functions**
- What **optimal policies** look like mathematically

The Remaining Challenge

How do we actually compute these optimal policies?

The Approach Menu

- **Dynamic Programming**: Complete knowledge of MDP
- **Monte Carlo Methods**: Learn from experience
- **Temporal Difference Learning**: Combines both approaches

Why Start with Dynamic Programming?

Three Key Reasons

1. **Conceptual Foundation**
   - Core principles used in all RL algorithms

Why Start with Dynamic Programming?

Three Key Reasons

1. **Conceptual Foundation**
   - Core principles used in all RL algorithms
2. **Theoretical Clarity**
   - Shows exactly what we're trying to approximate

Why Start with Dynamic Programming?

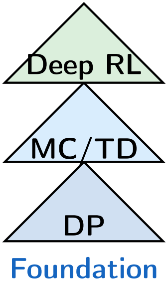
Three Key Reasons

1. **Conceptual Foundation**
   - Core principles used in all RL algorithms
2. **Theoretical Clarity**
   - Shows exactly what we're trying to approximate
3. **Practical Utility**
   - Works perfectly when MDP is known

Why Start with Dynamic Programming?

Three Key Reasons

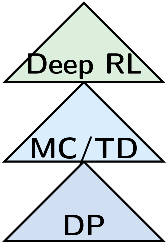
1. **Conceptual Foundation**
   - Core principles used in all RL algorithms
2. **Theoretical Clarity**
   - Shows exactly what we're trying to approximate
3. **Practical Utility**
   - Works perfectly when MDP is known



Why Start with Dynamic Programming?

Three Key Reasons

1. **Conceptual Foundation**
   - Core principles used in all RL algorithms
2. **Theoretical Clarity**
   - Shows exactly what we're trying to approximate
3. **Practical Utility**
   - Works perfectly when MDP is known



Foundation

What do we need to calculate to obtain optimal policies?

Policy Iteration: The Intuitive Approach

Core Idea

Start with any policy, then repeatedly improve it until optimal

Policy Iteration: The Intuitive Approach

Core Idea

Start with any policy, then repeatedly improve it until optimal

Two Alternating Steps

1. **Policy Evaluation**: How good is current policy?

Policy Iteration: The Intuitive Approach

Core Idea

Start with any policy, then repeatedly improve it until optimal

Two Alternating Steps

1. **Policy Evaluation**: How good is current policy?
2. **Policy Improvement**: Update policy based on values

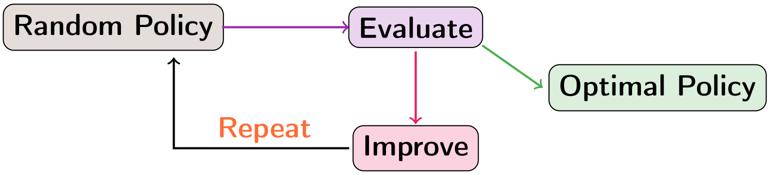
Policy Iteration: The Intuitive Approach

Core Idea

Start with any policy, then repeatedly improve it until optimal

Two Alternating Steps

1. **Policy Evaluation**: How good is current policy?
2. **Policy Improvement**: Update policy based on values



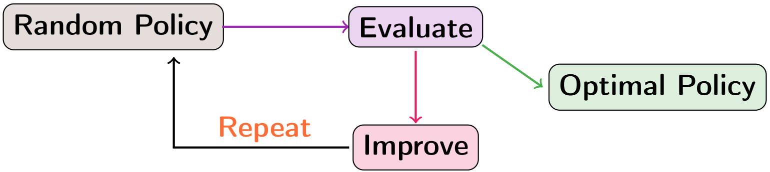
Policy Iteration: The Intuitive Approach

Core Idea

Start with any policy, then repeatedly improve it until optimal

Two Alternating Steps

1. **Policy Evaluation**: How good is current policy?
2. **Policy Improvement**: Update policy based on values



Each improvement step makes the policy better (or keeps it optimal)

Policy Evaluation: Computing Value Functions

The Challenge

Given policy π, compute $v_\pi(s)$ for all states using:

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

Policy Evaluation: Computing Value Functions

The Challenge

Given policy π, compute $v_\pi(s)$ for all states using:

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

The Problem

- This equation is **circular**!

Policy Evaluation: Computing Value Functions

The Challenge

Given policy π, compute $v_\pi(s)$ for all states using:

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

The Problem

- This equation is **circular**!
- Value of state s depends on values of future states s'

Policy Evaluation: Computing Value Functions

The Challenge

Given policy π, compute $v_\pi(s)$ for all states using:

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

The Problem

- This equation is **circular**!
- Value of state s depends on values of future states s'
- We have a system of $|S|$ equations with $|S|$ unknowns

Policy Evaluation: Computing Value Functions

The Challenge

Given policy π, compute $v_\pi(s)$ for all states using:

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

Policy Evaluation: Computing Value Functions

The Challenge

Given policy π, compute $v_\pi(s)$ for all states using:

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

The Solution: Iterative Approximation

1. Start with initial guesses $v_0(s)$ (often zeros)

Policy Evaluation: Computing Value Functions

The Challenge

Given policy π, compute $v_\pi(s)$ for all states using:

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

The Solution: Iterative Approximation

1. Start with initial guesses $v_0(s)$ (often zeros)
2. Iteratively improve estimates using Bellman equation

Policy Evaluation: Computing Value Functions

The Challenge

Given policy π, compute $v_\pi(s)$ for all states using:

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

The Solution: Iterative Approximation

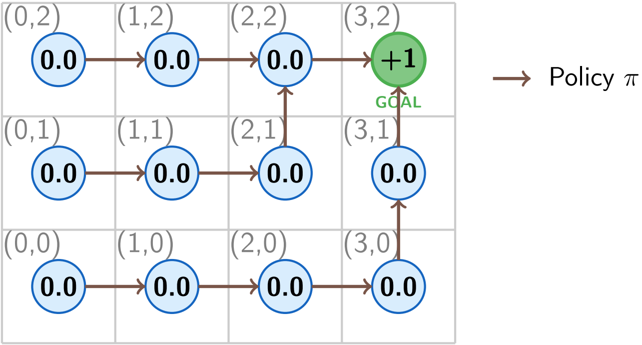
1. Start with initial guesses $v_0(s)$ (often zeros)
2. Iteratively improve estimates using Bellman equation
3. Continue until values stabilize

Policy Evaluation Algorithm

Algorithm Policy Evaluation

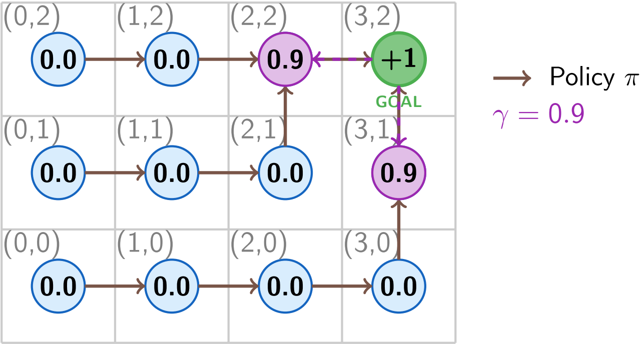
Initialize: $V(s) \in \mathbb{R}$ arbitrarily, $V(\text{terminal}) = 0$
repeat
    $\Delta \leftarrow 0$
    for each state s **do**
        $v \leftarrow V(s)$
        $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s'} p(s'|s,a)[r + \gamma V(s')]$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
    end for
until $\Delta < \theta$ (small threshold)

Policy Evaluation: Intuition



Iteration 0: Initialize all values to 0

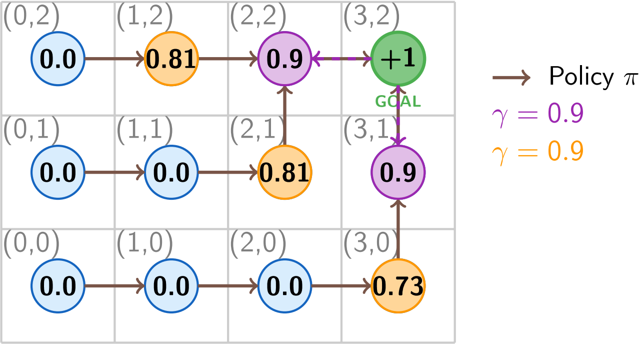
Policy Evaluation: Intuition



Iteration 0: Initialize all values to 0

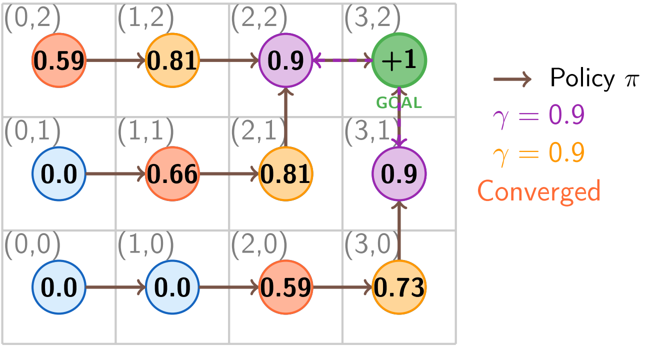
Iteration 1: Values propagate to immediate neighbors

Policy Evaluation: Intuition



Iteration 0: Initialize all values to 0
Iteration 1: Values propagate to immediate neighbors
Iteration 2: Values continue propagating (discounted)

Policy Evaluation: Intuition



Iteration 0: Initialize all values to 0

Iteration 1: Values propagate to immediate neighbors

Iteration 2: Values continue propagating (discounted)

Convergence: Optimal values for given policy

Policy Improvement: Making Better Decisions

The Question

Given policy π and its value function v_π, can we find a better policy?

Policy Improvement: Making Better Decisions

The Question

Given policy π and its value function v_π, can we find a better policy?

The Approach: Be Greedy!

For each state s, examine all possible actions:

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

"What happens if I take action a in state s, then follow policy π?"

Policy Improvement: Making Better Decisions

The Question

Given policy π and its value function v_π, can we find a better policy?

The Approach: Be Greedy!

For each state s, examine all possible actions:

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

"What happens if I take action a in state s, then follow policy π?"

Improved Policy

Create new policy π' by being greedy:

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

Policy Improvement Theorem

Theorem

If π and π' are policies such that for all $s \in S$:

$$q_\pi(s, \pi'(s)) \geq v_\pi(s)$$

Then policy π' must be as good as or better than π:

$$v_{\pi'}(s) \geq v_\pi(s) \text{ for all } s \in S$$

Policy Improvement Theorem

Theorem

If π and π' are policies such that for all $s \in S$:

$$q_\pi(s, \pi'(s)) \geq v_\pi(s)$$

Then policy π' must be as good as or better than π:

$$v_{\pi'}(s) \geq v_\pi(s) \text{ for all } s \in S$$

Intuition

- **Acting optimally for one step, then following old policy**

Policy Improvement Theorem

Theorem

If π and π' are policies such that for all $s \in S$:

$$q_\pi(s, \pi'(s)) \geq v_\pi(s)$$

Then policy π' must be as good as or better than π:

$$v_{\pi'}(s) \geq v_\pi(s) \text{ for all } s \in S$$

Intuition

- **Acting optimally for one step, then following old policy**
- Can only make things better (or stay the same)

Policy Improvement Theorem

Theorem

If π and π' are policies such that for all $s \in S$:

$$q_\pi(s, \pi'(s)) \geq v_\pi(s)$$

Then policy π' must be as good as or better than π:

$$v_{\pi'}(s) \geq v_\pi(s) \text{ for all } s \in S$$

Intuition

- **Acting optimally for one step, then following old policy**
- Can only make things better (or stay the same)

Guaranteed improvement or stability!

Policy Iteration Algorithm

Complete Process

1. **Initialize**: Start with arbitrary policy π_0

Policy Iteration Algorithm

Complete Process

1. **Initialize**: Start with arbitrary policy π_0
2. **Policy Evaluation**: Compute v_π for current policy

Policy Iteration Algorithm

Complete Process

1. **Initialize**: Start with arbitrary policy π_0
2. **Policy Evaluation**: Compute v_π for current policy
3. **Policy Improvement**: Compute improved policy π' from v_π

Policy Iteration Algorithm

Complete Process

1. **Initialize**: Start with arbitrary policy π_0
2. **Policy Evaluation**: Compute v_π for current policy
3. **Policy Improvement**: Compute improved policy π' from v_π
4. **Check**: If $\pi' = \pi$, STOP (found π^*)

Policy Iteration Algorithm

Complete Process

1. **Initialize**: Start with arbitrary policy π_0
2. **Policy Evaluation**: Compute v_π for current policy
3. **Policy Improvement**: Compute improved policy π' from v_π
4. **Check**: If $\pi' = \pi$, STOP (found π^*)
5. **Otherwise**: Set $\pi = \pi'$ and go to step 2

Policy Iteration Algorithm

Complete Process

1. **Initialize**: Start with arbitrary policy π_0
2. **Policy Evaluation**: Compute v_π for current policy
3. **Policy Improvement**: Compute improved policy π' from v_π
4. **Check**: If $\pi' = \pi$, STOP (found π^*)
5. **Otherwise**: Set $\pi = \pi'$ and go to step 2

Convergence Guarantee

- MDPs have finite number of deterministic policies

Policy Iteration Algorithm

Complete Process

1. **Initialize**: Start with arbitrary policy π_0
2. **Policy Evaluation**: Compute v_π for current policy
3. **Policy Improvement**: Compute improved policy π' from v_π
4. **Check**: If $\pi' = \pi$, STOP (found π^*)
5. **Otherwise**: Set $\pi = \pi'$ and go to step 2

Convergence Guarantee

- MDPs have finite number of deterministic policies
- Each improvement makes policy strictly better (unless optimal)

Policy Iteration Algorithm

Complete Process

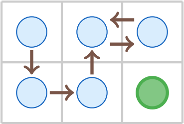
1. **Initialize**: Start with arbitrary policy π_0
2. **Policy Evaluation**: Compute v_π for current policy
3. **Policy Improvement**: Compute improved policy π' from v_π
4. **Check**: If $\pi' = \pi$, STOP (found π^*)
5. **Otherwise**: Set $\pi = \pi'$ and go to step 2

Convergence Guarantee

- MDPs have finite number of deterministic policies
- Each improvement makes policy strictly better (unless optimal)
- Must converge to optimal policy in finite steps

Policy Iteration Example

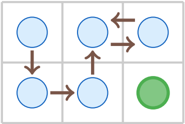
Step 1: Random Policy



Random actions

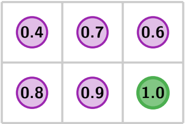
Policy Iteration Example

Step 1: Random Policy



Random actions

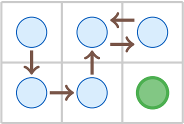
Step 2: Evaluate



Computed values

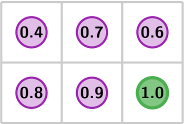
Policy Iteration Example

Step 1: Random Policy



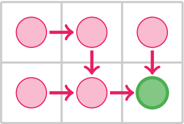
Random actions

Step 2: Evaluate



Computed values

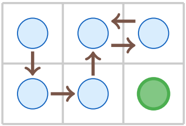
Step 3: Improve



Greedy actions

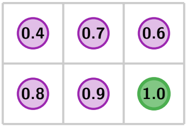
Policy Iteration Example

Step 1: Random Policy



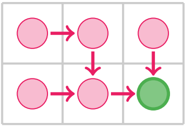
Random actions

Step 2: Evaluate



Computed values

Step 3: Improve



Greedy actions

Key Insight

Each iteration makes the policy strictly better until optimal

Value Iteration: Cutting the Bottleneck

Policy Iteration's Bottleneck

- Full policy evaluation requires multiple sweeps through all states

Value Iteration: Cutting the Bottleneck

Policy Iteration's Bottleneck

- Full policy evaluation requires multiple sweeps through all states
- Each evaluation step can be **computationally expensive**

Value Iteration: Cutting the Bottleneck

Policy Iteration's Bottleneck

- Full policy evaluation requires multiple sweeps through all states
- Each evaluation step can be **computationally expensive**
- We do complete evaluation before any improvement

Value Iteration: Cutting the Bottleneck

Policy Iteration's Bottleneck

- Full policy evaluation requires multiple sweeps through all states
- Each evaluation step can be **computationally expensive**
- We do complete evaluation before any improvement

Value Iteration's Insight

What if we combine evaluation and improvement?

Value Iteration: Cutting the Bottleneck

Policy Iteration's Bottleneck

- Full policy evaluation requires multiple sweeps through all states
- Each evaluation step can be **computationally expensive**
- We do complete evaluation before any improvement

Value Iteration's Insight

What if we combine evaluation and improvement?

- Don't need full evaluation before improving

Value Iteration: Cutting the Bottleneck

Policy Iteration's Bottleneck

- Full policy evaluation requires multiple sweeps through all states
- Each evaluation step can be **computationally expensive**
- We do complete evaluation before any improvement

Value Iteration's Insight

What if we combine evaluation and improvement?

- Don't need full evaluation before improving
- One step of evaluation gives useful information

Value Iteration: Cutting the Bottleneck

Policy Iteration's Bottleneck

- Full policy evaluation requires multiple sweeps through all states
- Each evaluation step can be **computationally expensive**
- We do complete evaluation before any improvement

Value Iteration's Insight

What if we combine evaluation and improvement?

- Don't need full evaluation before improving
- One step of evaluation gives useful information
- Always act greedily based on current value estimates

Value Iteration: Cutting the Bottleneck

Policy Iteration's Bottleneck

- Full policy evaluation requires multiple sweeps through all states
- Each evaluation step can be **computationally expensive**
- We do complete evaluation before any improvement

Value Iteration's Insight

What if we combine evaluation and improvement?

- Don't need full evaluation before improving
- One step of evaluation gives useful information
- Always act greedily based on current value estimates

Skip the separate policy - work directly with values!

Value Iteration Formula

Key Difference

Instead of following a fixed policy, always choose the best action:

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

Value Iteration Formula

Key Difference

Instead of following a fixed policy, always choose the best action:

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

Comparison

Policy Iteration says:

- "Fully evaluate my current strategy, then improve it"

Value Iteration says:

Value Iteration Formula

Key Difference

Instead of following a fixed policy, always choose the best action:

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

Comparison

Policy Iteration says:

- "Fully evaluate my current strategy, then improve it"

Value Iteration says:

- "Always act greedily based on current value estimates"

Value Iteration Formula

Key Difference

Instead of following a fixed policy, always choose the best action:

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

Value Iteration Formula

Key Difference

Instead of following a fixed policy, always choose the best action:

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

Connection to Bellman Optimality

This directly solves the Bellman optimality equation:

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

Value Iteration Algorithm

Value Iteration Algorithm

1. **Initialize**: $V(s) \in \mathbb{R}$ arbitrarily for all $s \in S$, $V(\text{terminal}) = 0$

Value Iteration Algorithm

Value Iteration Algorithm

1. **Initialize**: $V(s) \in \mathbb{R}$ arbitrarily for all $s \in S$, $V(\text{terminal}) = 0$
2. **Repeat** until convergence:
   - $\Delta \leftarrow 0$

Value Iteration Algorithm

Value Iteration Algorithm

1. **Initialize**: $V(s) \in \mathbb{R}$ arbitrarily for all $s \in S$, $V(\text{terminal}) = 0$
2. **Repeat** until convergence:
   - $\Delta \leftarrow 0$
   - **For** each state $s \in S$:
     - $v \leftarrow V(s)$

Value Iteration Algorithm

Value Iteration Algorithm

1. **Initialize**: $V(s) \in \mathbb{R}$ arbitrarily for all $s \in S$, $V(\text{terminal}) = 0$
2. **Repeat** until convergence:
   - $\Delta \leftarrow 0$
   - **For** each state $s \in S$:
     - $v \leftarrow V(s)$
     - $V(s) \leftarrow \max_a \sum_{s'} p(s'|s,a)[r + \gamma V(s')]$

Value Iteration Algorithm

Value Iteration Algorithm

1. **Initialize**: $V(s) \in \mathbb{R}$ arbitrarily for all $s \in S$, $V(\text{terminal}) = 0$
2. **Repeat** until convergence:
   - $\Delta \leftarrow 0$
   - **For** each state $s \in S$:
     - $v \leftarrow V(s)$
     - $V(s) \leftarrow \max_a \sum_{s'} p(s'|s,a)[r + \gamma V(s')]$
     - $\Delta \leftarrow \max(\Delta, |v - V(s)|)$

Value Iteration Algorithm

Value Iteration Algorithm

1. **Initialize**: $V(s) \in \mathbb{R}$ arbitrarily for all $s \in S$, $V(\text{terminal}) = 0$
2. **Repeat** until convergence:
   - $\Delta \leftarrow 0$
   - **For** each state $s \in S$:
     - $v \leftarrow V(s)$
     - $V(s) \leftarrow \max_a \sum_{s'} p(s'|s,a)[r + \gamma V(s')]$
     - $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
3. **Until** $\Delta < \theta$ (convergence threshold)

Value Iteration Algorithm

Value Iteration Algorithm

1. **Initialize**: $V(s) \in \mathbb{R}$ arbitrarily for all $s \in S$, $V(\text{terminal}) = 0$
2. **Repeat** until convergence:
   - $\Delta \leftarrow 0$
   - **For** each state $s \in S$:
     - $v \leftarrow V(s)$
     - $V(s) \leftarrow \max_a \sum_{s'} p(s'|s,a)[r + \gamma V(s')]$
     - $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
3. **Until** $\Delta < \theta$ (convergence threshold)
4. **Extract policy**: $\pi(s) = \arg\max_a \sum_{s'} p(s'|s,a)[r + \gamma V(s')]$

Value Iteration Algorithm

Value Iteration Algorithm

1. **Initialize**: $V(s) \in \mathbb{R}$ arbitrarily for all $s \in S$, $V(\text{terminal}) = 0$
2. **Repeat** until convergence:
   - $\Delta \leftarrow 0$
   - **For** each state $s \in S$:
     - $v \leftarrow V(s)$
     - $V(s) \leftarrow \max_a \sum_{s'} p(s'|s,a)[r + \gamma V(s')]$
     - $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
3. **Until** $\Delta < \theta$ (convergence threshold)
4. **Extract policy**: $\pi(s) = \arg\max_a \sum_{s'} p(s'|s,a)[r + \gamma V(s')]$

Key Advantages

- **Simpler**: No separate policy evaluation phase

Value Iteration Algorithm

Value Iteration Algorithm

1. **Initialize**: $V(s) \in \mathbb{R}$ arbitrarily for all $s \in S$, $V(\text{terminal}) = 0$
2. **Repeat** until convergence:
   - $\Delta \leftarrow 0$
   - **For** each state $s \in S$:
     - $v \leftarrow V(s)$
     - $V(s) \leftarrow \max_a \sum_{s'} p(s'|s,a)[r + \gamma V(s')]$
     - $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
3. **Until** $\Delta < \theta$ (convergence threshold)
4. **Extract policy**: $\pi(s) = \arg\max_a \sum_{s'} p(s'|s,a)[r + \gamma V(s')]$

Key Advantages

- Simpler: No separate policy evaluation phase
- More efficient: Often faster per iteration

Value Iteration vs Policy Iteration

Aspect	Policy Iteration	Value Iteration
Approach	Evaluate then improve	Combined eval+improve
Per iteration	Slower (full eval)	Faster (one step)
Total iterations	Fewer	More
Memory	Store policy + values	Store values only
Implementation	More complex	Simpler
Convergence	Finite steps	Asymptotic

Value Iteration vs Policy Iteration

Aspect	**Policy Iteration**	**Value Iteration**
Approach	Evaluate then improve	Combined eval+improve
Per iteration	Slower (full eval)	Faster (one step)
Total iterations	Fewer	More
Memory	Store policy + values	Store values only
Implementation	More complex	Simpler
Convergence	Finite steps	Asymptotic

When to Use Which?

- **Policy Iteration**: When actions are expensive to compute

Value Iteration vs Policy Iteration

Aspect	Policy Iteration	Value Iteration
Approach	Evaluate then improve	Combined eval+improve
Per iteration	Slower (full eval)	Faster (one step)
Total iterations	Fewer	More
Memory	Store policy + values	Store values only
Implementation	More complex	Simpler
Convergence	Finite steps	Asymptotic

When to Use Which?

- **Policy Iteration**: When actions are expensive to compute
- **Value Iteration**: When state space is large

Value Iteration vs Policy Iteration

Aspect	Policy Iteration	Value Iteration
Approach	Evaluate then improve	Combined eval+improve
Per iteration	Slower (full eval)	Faster (one step)
Total iterations	Fewer	More
Memory	Store policy + values	Store values only
Implementation	More complex	Simpler
Convergence	Finite steps	Asymptotic

When to Use Which?

- **Policy Iteration**: When actions are expensive to compute
- **Value Iteration**: When state space is large
- **In practice**: Value iteration often preferred for simplicity

Computational Complexity Analysis

Time Complexity per Iteration

For both algorithms, each iteration requires:

- Loop over all states: $O(|S|)$

Computational Complexity Analysis

Time Complexity per Iteration

For both algorithms, each iteration requires:

- Loop over all states: $O(|S|)$
- For each state, loop over all actions: $O(|A|)$

Computational Complexity Analysis

Time Complexity per Iteration

For both algorithms, each iteration requires:

- Loop over all states: $O(|S|)$
- For each state, loop over all actions: $O(|A|)$
- For each action, loop over possible next states: $O(|S|)$

Computational Complexity Analysis

Time Complexity per Iteration

For both algorithms, each iteration requires:

- Loop over all states: $O(|S|)$
- For each state, loop over all actions: $O(|A|)$
- For each action, loop over possible next states: $O(|S|)$
- **Total per iteration**: $O(|S|^2|A|)$

Computational Complexity Analysis

Time Complexity per Iteration

For both algorithms, each iteration requires:

- Loop over all states: $O(|S|)$
- For each state, loop over all actions: $O(|A|)$
- For each action, loop over possible next states: $O(|S|)$
- **Total per iteration**: $O(|S|^2|A|)$

Space Complexity

- **Value Iteration**: $O(|S|)$ for value function

Computational Complexity Analysis

Time Complexity per Iteration

For both algorithms, each iteration requires:

- Loop over all states: $O(|S|)$
- For each state, loop over all actions: $O(|A|)$
- For each action, loop over possible next states: $O(|S|)$
- **Total per iteration**: $O(|S|^2|A|)$

Space Complexity

- **Value Iteration**: $O(|S|)$ for value function
- **Policy Iteration**: $O(|S|)$ for values + $O(|S|)$ for policy

Computational Complexity Analysis

Time Complexity per Iteration

For both algorithms, each iteration requires:

- Loop over all states: $O(|S|)$
- For each state, loop over all actions: $O(|A|)$
- For each action, loop over possible next states: $O(|S|)$
- **Total per iteration**: $O(|S|^2|A|)$

Space Complexity

- **Value Iteration**: $O(|S|)$ for value function
- **Policy Iteration**: $O(|S|)$ for values + $O(|S|)$ for policy
- Both need to store MDP: $O(|S|^2|A|)$ for transition probabilities

Computational Complexity Analysis

Scalability Issues

- **Curse of dimensionality**: State space grows exponentially
- **Memory requirements**: Storing full transition model
- **Exact methods**: Limited to small-medium problems

Handling Large State Spaces

The Problem

- Tabular DP requires storing $V(s)$ for every state

Handling Large State Spaces

The Problem

- Tabular DP requires storing $V(s)$ for every state
- Real problems often have millions/billions of states

Handling Large State Spaces

The Problem

- Tabular DP requires storing $V(s)$ for every state
- Real problems often have millions/billions of states
- **Solution**: Approximate value function

Handling Large State Spaces

The Problem

- Tabular DP requires storing $V(s)$ for every state
- Real problems often have millions/billions of states
- Solution: Approximate value function

Function Approximation Approaches

- **Linear**: $V(s) \approx \theta^T \phi(s)$
  - Feature vector $\phi(s)$, learned weights θ

Handling Large State Spaces

The Problem

- Tabular DP requires storing $V(s)$ for every state
- Real problems often have millions/billions of states
- **Solution**: Approximate value function

Function Approximation Approaches

- **Linear**: $V(s) \approx \theta^T \phi(s)$
  - Feature vector $\phi(s)$, learned weights θ
- **Neural Networks**: $V(s) \approx f_\theta(s)$
  - Deep networks for complex state representations

Handling Large State Spaces

The Problem

- Tabular DP requires storing $V(s)$ for every state
- Real problems often have millions/billions of states
- **Solution**: Approximate value function

Function Approximation Approaches

- **Linear**: $V(s) \approx \theta^T \phi(s)$
  - Feature vector $\phi(s)$, learned weights θ
- **Neural Networks**: $V(s) \approx f_\theta(s)$
  - Deep networks for complex state representations
- **Decision Trees**: Partition state space

From Dynamic Programming to Model-Free RL

DP Limitations in Practice

- **Model requirement**: Often don't know T and R

From Dynamic Programming to Model-Free RL

DP Limitations in Practice

- **Model requirement**: Often don't know T and R
- **Computational cost**: Exponential in state space

From Dynamic Programming to Model-Free RL

DP Limitations in Practice

- **Model requirement**: Often don't know T and R
- **Computational cost**: Exponential in state space

From Dynamic Programming to Model-Free RL

DP Limitations in Practice

- **Model requirement**: Often don't know T and R
- **Computational cost**: Exponential in state space

The Next Step: Model-Free Methods

- **Monte Carlo**: Learn from complete episodes

From Dynamic Programming to Model-Free RL

DP Limitations in Practice

- **Model requirement**: Often don't know T and R
- **Computational cost**: Exponential in state space

The Next Step: Model-Free Methods

- **Monte Carlo**: Learn from complete episodes
- **Temporal Difference**: Learn from single steps

From Dynamic Programming to Model-Free RL

DP Limitations in Practice

- **Model requirement**: Often don't know T and R
- **Computational cost**: Exponential in state space

The Next Step: Model-Free Methods

- **Monte Carlo**: Learn from complete episodes
- **Temporal Difference**: Learn from single steps
- **Q-Learning**: Off-policy value learning

From Dynamic Programming to Model-Free RL

DP Limitations in Practice

- **Model requirement**: Often don't know T and R
- **Computational cost**: Exponential in state space

The Next Step: Model-Free Methods

- **Monte Carlo**: Learn from complete episodes
- **Temporal Difference**: Learn from single steps
- **Q-Learning**: Off-policy value learning
- **SARSA**: On-policy value learning

From Dynamic Programming to Model-Free RL

DP Limitations in Practice

- **Model requirement**: Often don't know T and R
- **Computational cost**: Exponential in state space

The Next Step: Model-Free Methods

- **Monte Carlo**: Learn from complete episodes
- **Temporal Difference**: Learn from single steps
- **Q-Learning**: Off-policy value learning
- **SARSA**: On-policy value learning

Key Insight

These methods use the same core principles as DP, but learn from experience instead of using a model

Summary

What We've Learned

- **Policy Iteration**: Alternate evaluation and improvement
- **Value Iteration**: Combined evaluation and improvement
- **Theoretical guarantees**: Convergence to optimal policies
- **Practical considerations**: Complexity and scalability

Next Topics

- Monte Carlo methods for model-free learning
- Temporal difference learning (TD, Q-Learning, SARSA)
- Function approximation and deep reinforcement learning
- Policy gradient methods