Introduction to Reinforcement Learning

Dynamic Programming - Value Iteration

Sarwan Ali

Department of Computer Science
Georgia State University

Dynamic Programming Methods

Today's Learning Journey

Bellman Equations Recap

State Value Function:

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[G_t | S_t = s] && (1) \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} | S_t = s] && (2) \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')] && (3) \end{aligned}$$

Optimal State Value Function:

$$\begin{aligned} v_*(s) &= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a] && (4) \\ &= \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_*(s')] && (5) \end{aligned}$$

Key Insight

The Bellman optimality equation provides the foundation for value iteration

Value Iteration: Core Idea

Principle

Value Iteration turns the Bellman optimality equation into an iterative update rule

Instead of solving: $v_*(s) = \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_*(s')]$
We iterate: $v_{k+1}(s) = \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_k(s')]$

Key Properties:

- Model-based approach
- Guaranteed convergence
- Finds optimal policy

Requirements:

- Complete MDP model
- Finite state/action spaces
- Discount factor $\gamma < 1$

Value Iteration Algorithm

Algorithm 1 Value Iteration

Require: MDP (S, A, P, R, γ), threshold $\theta > 0$

Initialize $V(s) = 0$ for all $s \in S$
repeat
    $\Delta \leftarrow 0$
    for each state $s \in S$ **do**
        $v \leftarrow V(s)$
        $V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
    end for
until $\Delta < \theta$
Output: Optimal policy $\pi_*(s) = \arg\max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$

Time Complexity

$O(|S|^2|A|)$ per iteration, polynomial convergence

Value Iteration: Step-by-Step Breakdown

Step 1: Initialization

- Set $V_0(s) = 0$ for all states s
- Choose convergence threshold θ

Step 2: Value Update (Bellman Backup)

$$V_{k+1}(s) = \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V_k(s')] \tag{6}$$

Step 3: Convergence Check

- Compute $\Delta = \max_s |V_{k+1}(s) - V_k(s)|$
- Stop if $\Delta < \theta$

Step 4: Policy Extraction

$$\pi_*(s) = \arg\max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V_*(s')] \tag{7}$$

Convergence Guarantees

Theorem (Value Iteration Convergence)

For any finite MDP with $\gamma < 1$, value iteration converges to the unique optimal value function v_*.

Proof Sketch:

1. The Bellman operator T is a contraction mapping
2. For any value functions u, v: $||Tu - Tv||_\infty \leq \gamma ||u - v||_\infty$
3. By Banach Fixed Point Theorem, T has unique fixed point v_*
4. Value iteration: $v_{k+1} = Tv_k$ converges to v_*

Rate of Convergence

$$||v_k - v_*||_\infty \leq \gamma^k ||v_0 - v_*||_\infty \tag{8}$$

Geometric convergence with rate γ

Error Bounds and Stopping Criteria

Practical Question: When should we stop the algorithm?

Error Bound

If $||v_{k+1} - v_k||_\infty < \theta$, then:

$$||v_k - v_*||_\infty \leq \frac{\theta}{1-\gamma} \tag{9}$$

Policy Loss Bound:

- Let π_k be greedy policy w.r.t. v_k
- If $||v_k - v_*||_\infty \leq \epsilon$, then:

$$||v_* - v_{\pi_k}||_\infty \leq \frac{2\gamma\epsilon}{1-\gamma} \tag{10}$$

Practical Insight

Small value function errors lead to near-optimal policies

Synchronous vs Asynchronous Updates

Synchronous Value Iteration:

- Update all states simultaneously
- Use V_k to compute V_{k+1}
- Requires two arrays
- Standard algorithm

Asynchronous Value Iteration:

- Update states one at a time
- Use most recent values
- In-place updates
- Often faster convergence

Pseudocode

```
for s in states:
  V_new[s] = max_a Q(s,a)
V = V_new
```

Pseudocode

```
for s in states:
  V[s] = max_a Q(s,a)
// immediately use new V[s]
```

Note

Both versions converge to v_*, but asynchronous often faster in practice

Value Iteration Variants

1. **Gauss-Seidel Value Iteration**
   - Update states in systematic order
   - Use newest available values
   - Faster convergence than Jacobi method
2. **Prioritized Sweeping**
   - Update states with largest Bellman error first
   - Maintain priority queue of states
   - Much faster for sparse problems
3. **Real-Time Value Iteration**
   - Update only visited states
   - Useful for large state spaces
   - Agent-centric approach

Key Insight

All variants maintain convergence guarantees while improving efficiency

Grid World Example

S1	S2	T
S3	X	S4
S	S6	S7

Problem Setup:

- 3×3 grid world
- Start: S5, Goal: T (reward $+10$)
- Obstacle: X (impassable)
- Actions: Up, Down, Left, Right
- Step reward: -1
- $\gamma = 0.9$

Transition Model:

- 0.8 prob: intended direction
- 0.1 prob: each perpendicular direction
- Hit wall: stay in place

Grid World: Value Iteration Steps

Iteration 0:

0	0	0
0	X	0
0	0	0

Iteration 1:

-1	-1	9
-1	X	9
-1	-1	-1

Iteration 2:

-1.9	6.4	9
-1.9	X	6.4
-1.9	-1.9	-1.9

Iteration 5:

4.6	6.4	9
2.3	X	6.4
0.4	2.3	4.6

Value Update Example (S6):

$$V_1(S6) = \max\{Q(S6, \text{up}), Q(S6, \text{down}), Q(S6, \text{left}), Q(S6, \text{right})\} \quad (11)$$

$$= \max\{-1, -1, -1, 6.4\} = 6.4 \quad (12)$$

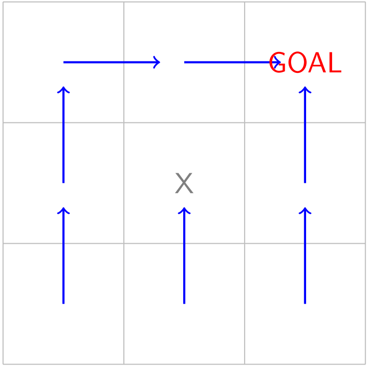
Where: $Q(S6, \text{right}) = -1 + 0.9 \times [0.8 \times 9 + 0.1 \times (-1) + 0.1 \times (-1)] = 6.4$

Grid World: Optimal Policy

Final Values:

6.1	7.4	10
4.3	X	7.4
2.4	4.3	6.1

Optimal Policy:



Policy Extraction:

$$\pi_*(s) = \arg\max_a \sum_{s'} P(s'|s,a)[R + \gamma V(s')] \tag{13}$$

Complexity Analysis

Time Complexity per Iteration:

- For each state s: $O(|A| \times |S|)$ operations
- Total per iteration: $O(|S| \times |A| \times |S|) = O(|S|^2|A|)$

Number of Iterations:

- Convergence rate: $O(\log(1/\epsilon)/\log(1/\gamma))$
- For $\epsilon = 10^{-6}$, $\gamma = 0.9$: ≈ 131 iterations

Total Complexity:

$$O\left(\frac{|S|^2|A|}{\log(1/\gamma)}\log\left(\frac{1}{\epsilon}\right)\right) \tag{14}$$

Curse of Dimensionality

For n state variables with k values each: $|S| = k^n$ (exponential!)

Memory Requirements

Space Complexity:

- Value function: $O(|S|)$
- Transition model: $O(|S|^2|A|)$
- Policy: $O(|S|)$

Optimization Techniques:

1. **In-place updates:** Reduce memory by half
2. **Sparse representations:** For structured problems
3. **Function approximation:** For large state spaces
4. **State abstraction:** Group similar states

Practical Limits

- Tabular VI: $\sim 10^6$ states (modern computers)
- Beyond this: Need approximation methods

Value Iteration: Strengths

Advantages

- **Guaranteed convergence** to optimal solution
- **Simple to implement** and understand
- **No policy needed** during learning
- **Anytime algorithm:** can stop early for approximate solution
- **Parallelizable:** states can be updated independently
- **Memory efficient:** only needs value function

Mathematical Elegance

- Direct implementation of Bellman optimality equation
- Clear convergence theory and error bounds
- Foundation for many advanced RL algorithms

Value Iteration: Limitations

Major Limitations

- **Requires complete MDP model** (P, R)
- **Curse of dimensionality:** $O(|S|^2|A|)$ per iteration
- **Only works for finite state/action spaces**
- **Slow for large discount factors** $\gamma \approx 1$
- **No online learning:** must know environment model

When VI Struggles:

- Continuous state/action spaces
- Unknown environment dynamics
- Very large state spaces ($> 10^6$ states)
- Real-time applications requiring fast responses

Solutions

Function approximation, sampling methods, model-free approaches

Value Iteration vs Policy Iteration

Aspect	**Value Iteration**	**Policy Iteration**
Updates	Value function only	Policy + Value function
Convergence	$O(\log(1/\epsilon))$	Fewer iterations
Per iteration	$O(\|S\|^2\|A\|)$	$O(\|S\|^3 + \|S\|^2\|A\|)$
Memory	$O(\|S\|)$	$O(\|S\|)$
Implementation	Simpler	More complex
Early stopping	Good approx. policy	Poor policy

When to use each:

- **Value Iteration:** When you want simplicity, anytime behavior
- **Policy Iteration:** When exact solution needed, small state spaces

Value Iteration Extensions

1. **Approximate Value Iteration**
   - Use function approximation: $V(s) \approx \hat{V}(s;\theta)$
   - Neural networks, linear functions, etc.
   - Enables large/continuous state spaces
2. **Asynchronous Value Iteration**
   - Update states in any order
   - Real-time dynamic programming
   - Prioritized sweeping variants
3. **Multi-objective Value Iteration**
   - Vector-valued rewards and values
   - Pareto-optimal policies
   - Applications in robotics, finance

Real-World Applications

1. **Game Playing**
   - Chess, Go, Backgammon endgames
   - Perfect information games
   - Tablebase generation
2. **Robotics**
   - Path planning in discrete grids
   - Navigation with known maps
   - Manipulation planning
3. **Operations Research**
   - Inventory management
   - Queueing systems optimization
   - Maintenance scheduling
4. **Finance**
   - Portfolio optimization
   - Options pricing (American options)
   - Risk management

Key Takeaways

Value Iteration Algorithm

- **Core idea:** Iteratively apply Bellman optimality operator
- **Convergence:** Guaranteed for finite MDPs with $\gamma < 1$
- **Complexity:** $O(|S|^2|A|)$ per iteration

Theoretical Foundation

- Based on contraction mapping theorem
- Geometric convergence rate γ
- Clear error bounds and stopping criteria

Practical Considerations

- Works well for moderate-sized problems
- Requires complete environment model
- Foundation for more advanced RL methods

Next Steps

What's Coming Next:

- **Policy Iteration:** Alternative DP approach
- **Generalized Policy Iteration:** Unified framework
- **Monte Carlo Methods:** Model-free approaches
- **Temporal Difference Learning:** Online model-free methods

Practice Problems:

- Implement value iteration for small grid worlds
- Analyze convergence for different γ values
- Compare with policy iteration on same problems
- Explore asynchronous update schemes

Questions?