Introduction to Reinforcement Learning

Dynamic Programming - Generalized Policy Iteration

Sarwan Ali

Department of Computer Science
Georgia State University

Generalized Policy Iteration

Today's Learning Journey

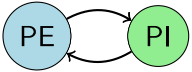
What is Generalized Policy Iteration?

Definition

Generalized Policy Iteration (GPI) is the general framework that combines policy evaluation and policy improvement processes to find optimal policies in Markov Decision Processes.

Key Components:

- Policy Evaluation
- Policy Improvement
- Iterative Process
- Convergence Guarantees



Policy Iteration

Motivation: Why GPI?

The Challenge

How do we systematically find the optimal policy π^* and optimal value function v^* in an MDP?

Traditional Approaches:

- Exhaustive search: Exponential in state/action space
- Random exploration: No convergence guarantees
- Dynamic Programming: Principled, guaranteed convergence

GPI Solution

GPI provides a **systematic framework** that alternates between:

1. Making the value function consistent with current policy
2. Making the policy greedy with respect to current value function

Bellman Equations Recap:

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

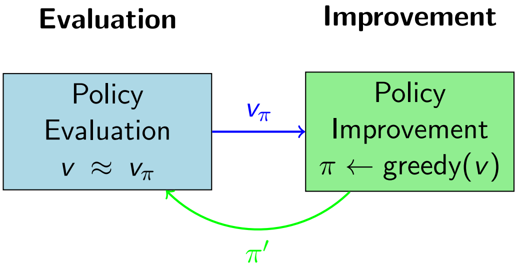
$$v^*(s) = \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v^*(s')] \tag{2}$$

$$q_\pi(s, a) = \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')] \tag{3}$$

$$q^*(s, a) = \sum_{s',r} p(s', r|s, a)[r + \gamma \max_{a'} q^*(s', a')] \tag{4}$$

Policy Improvement Theorem: If $q_\pi(s, \pi'(s)) \geq v_\pi(s)$ for all $s \in \mathcal{S}$, then $\pi' \geq \pi$ (i.e., $v_{\pi'} \geq v_\pi$).

The GPI Framework



Convergence: $v \rightarrow v^*$ and $\pi \rightarrow \pi^*$

Key Insight: These two processes stabilize each other and converge to optimality.

Policy Evaluation: Making Values Consistent

Goal: Given policy π, compute v_π (or approximate it)

Iterative Policy Evaluation

$$v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_k(s')] \tag{5}$$

$$= \mathbb{E}_\pi[R_{t+1} + \gamma v_k(S_{t+1})|S_t = s] \tag{6}$$

Implementation Options:

- **Synchronous:** Update all states simultaneously
- **Asynchronous:** Update states in any order
- **In-place:** Use updated values immediately

Stopping Criterion: $\max_s |v_{k+1}(s) - v_k(s)| < \theta$

Policy Evaluation Algorithm

Iterative Policy Evaluation Algorithm

```
def policy_evaluation(pi, mdp, theta=1e-6, gamma=1.0):
    V = initialize_value_function(mdp.states)

    while True:
        delta = 0
        for s in mdp.states:
            v = V[s]
            V[s] = sum(pi[s][a] * sum(p * (r + gamma * V[s_next])
                           for s_next, r, p in mdp.transitions(s, a))
                           for a in mdp.actions(s))
            delta = max(delta, abs(v - V[s]))

        if delta < theta:
            break

    return V
```

Policy Improvement: Acting Greedily

Goal: Given value function v, find better policy π'

Greedy Policy Improvement

$$\pi'(s) = \arg\max_a q_\pi(s, a) \tag{7}$$

$$= \arg\max_a \sum_{s', r} p(s', r|s, a)[r + \gamma v(s')] \tag{8}$$

Policy Improvement Theorem:

Theorem

Let π and π' be deterministic policies such that for all $s \in \mathcal{S}$: $q_\pi(s, \pi'(s)) \geq v_\pi(s)$ Then $\pi' \geq \pi$, i.e., $v_{\pi'}(s) \geq v_\pi(s)$ for all $s \in \mathcal{S}$.

Proof Intuition: Acting greedily w.r.t. v_π gives at least as good expected return.

Policy Improvement: Stochastic Case

For Stochastic Policies:
If we have a stochastic policy π, we can improve it by:

$$\pi'(a|s) = \begin{cases} 1 & \text{if } a = \arg\max_a q_\pi(s,a) \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

Soft Policy Improvement: For exploration, we might use:

$$\pi'(a|s) = \frac{\exp(q_\pi(s,a)/\tau)}{\sum_{a'} \exp(q_\pi(s,a')/\tau)} \tag{10}$$

where τ is the temperature parameter.

Key Point

Policy improvement is guaranteed to find a better policy unless the current policy is already optimal.

Policy Iteration Algorithm

Policy Iteration

Initialize: π_0 arbitrarily, $V_0 = 0$

For $k = 0, 1, 2, \ldots$ **until** convergence:

1. **Policy Evaluation:** Solve $v_{\pi_k} = v^{\pi_k}$
2. **Policy Improvement:** $\pi_{k+1}(s) = \arg\max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_{\pi_k}(s')]$
3. **Check:** If $\pi_{k+1} = \pi_k$, then stop

Properties:

- Guaranteed convergence to π^* and v^*
- Computationally expensive - exact policy evaluation
- Finite convergence - at most $|\mathcal{A}|^{|\mathcal{S}|}$ iterations

Value Iteration Algorithm

Value Iteration

Initialize: V_0 arbitrarily (e.g., $V_0 = 0$)

For $k = 0, 1, 2, \ldots$ **until** convergence:

1. **Value Update:**

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

2. **Check:** If $\max_s |V_{k+1}(s) - V_k(s)| < \theta$, then stop

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

Properties:

- More efficient - combines evaluation and improvement
- Guaranteed convergence to v^*
- Geometric convergence rate

Policy vs Value Iteration Comparison

Aspect	**Policy Iteration**	**Value Iteration**
Convergence	Finite steps	Asymptotic
Per iteration cost	High (solve system)	Low (one sweep)
Total iterations	Few	Many
Memory	Two arrays	One array
Practical efficiency	Better for small $\lvert\mathcal{S}\rvert$	Better for large $\lvert\mathcal{S}\rvert$

When to use which?

- **Policy Iteration:** When policy evaluation can be done efficiently
- **Value Iteration:** When state space is large or continuous
- **Modified Policy Iteration:** Compromise between the two

Convergence of GPI

Theorem (GPI Convergence)

Under GPI, both the sequence of value functions $\{v_k\}$ and policies $\{\pi_k\}$ converge to the optimal value function v^* and an optimal policy π^*.

Key Insights:

- **Monotonicity:** $v_{\pi_0} \leq v_{\pi_1} \leq v_{\pi_2} \leq \ldots \leq v^*$
- **Finite Policy Space:** Only finitely many deterministic policies
- **Improvement until Optimal:** If $\pi' \neq \pi$ after improvement, then $v_{\pi'} > v_\pi$

Proof Sketch

1. Policy improvement gives strictly better policy unless optimal
2. Finite policy space $\Rightarrow$ must reach optimal policy
3. Once optimal policy found, policy evaluation converges to v^*

Rate of Convergence

Value Iteration Convergence Rate:

Theorem

For value iteration with discount factor $\gamma < 1$:

$$\|V_k - V^*\|_\infty \leq \gamma^k \|V_0 - V^*\|_\infty$$

Practical Implications:

- **Geometric convergence** with rate γ
- Smaller $\gamma \Rightarrow$ faster convergence
- After k iterations, error bounded by γ^k times initial error

Stopping Criterion

To guarantee $\|V_k - V^*\|_\infty \leq \epsilon$:

$$\|V_{k+1} - V_k\|_\infty \leq \frac{\epsilon(1-\gamma)}{2\gamma}$$

Modified Policy Iteration

Motivation: Balance between policy and value iteration

Algorithm

Initialize: π_0, V_0

Repeat:

1. **Partial Policy Evaluation:** Run m steps of policy evaluation

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

2. **Policy Improvement:** $\pi' = \text{greedy}(V)$

Special Cases:

- $m = \infty$: Standard Policy Iteration
- $m = 1$: Value Iteration
- $m \in [2, \infty)$: Modified Policy Iteration

Advantage: Tunable trade-off between computation per iteration and number of iterations

Asynchronous Dynamic Programming

Key Idea: Update states in any order, potentially multiple times

Asynchronous Value Iteration

At each step, pick some state s and update:

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

Advantages:

- Flexibility in update order
- Can focus on important states
- Online implementation possible
- Parallelization opportunities

Convergence Requirement: Every state must be updated infinitely often in the limit.

Applications: Real-time dynamic programming, prioritized sweeping

Implementation Challenges

State Space Issues:

- Curse of dimensionality: $|\mathcal{S}|$ grows exponentially
- Memory requirements: Store $V(s)$ for all states
- Computation time: $O(|\mathcal{S}|^2|\mathcal{A}|)$ per iteration

Solutions and Approximations:

- Function approximation: $V(s) \approx \hat{V}(s;\theta)$
- State aggregation: Group similar states
- Sampling methods: Monte Carlo approaches
- Approximate DP: Fitted value iteration

Model Requirements:

- Need complete model: $p(s', r|s, a)$
- Model-free alternatives: Temporal difference learning

Practical Implementation Tips

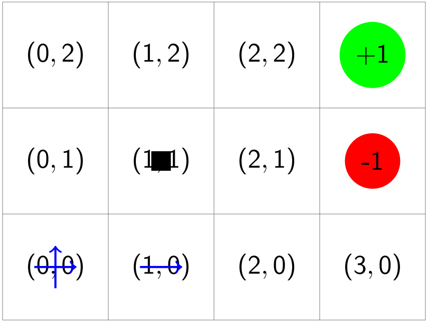
Numerical Considerations:

```
# Use appropriate data types
V = np.zeros(n_states, dtype=np.float64)

# Vectorized operations when possible
V_new = np.max(R + gamma * P @ V, axis=1)

# Careful with stopping criteria
delta = np.max(np.abs(V_new - V))
if delta < theta * (1 - gamma) / (2 * gamma):
    break
```

Debugging Tips:

- Verify Bellman equations hold at convergence
- Check policy improvement actually improves value
- Monitor convergence curves
- Test on small, known problems first

Example: Grid World



Setup:

- 4$\times$3 grid, agent starts at (0,0)
- Actions: Up, Down, Left, Right
- Rewards: +1 at (3,2), -1 at (3,1), -0.04 otherwise
- Walls at (1,1)

Grid World: Value Iteration Results

Value Function after Convergence:

0.81	0.87	0.92	1.0
0.76		0.66	-1.0
0.70	0.66	0.61	0.39

Observations:

- Values decrease with distance from goal
- Policy avoids the -1 terminal state
- Small negative rewards encourage shorter paths

Optimal Policy:

→	→	→	
↑		→	
→	→	↑	↑

Applications of GPI

Classical Applications:

- **Inventory Management:** Optimal ordering policies
- **Resource Allocation:** CPU scheduling, bandwidth allocation
- **Financial Planning:** Portfolio optimization, option pricing
- **Manufacturing:** Production planning, quality control

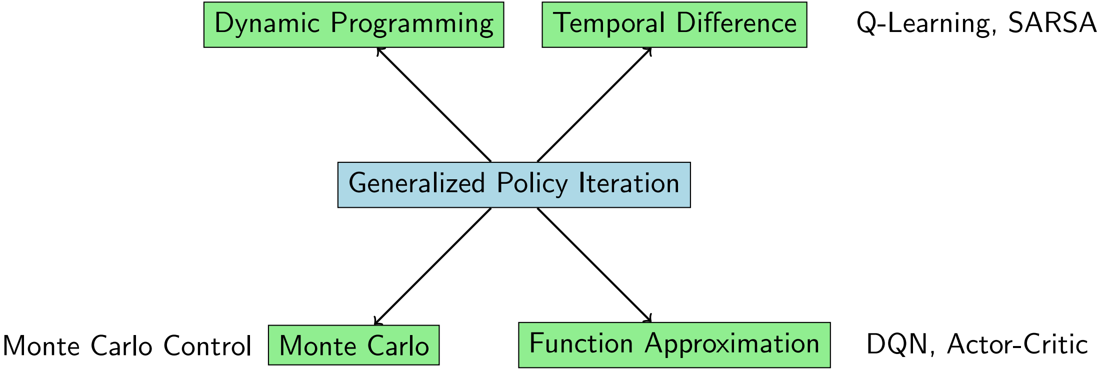
Modern AI Applications:

- **Game Playing:** Chess, Go, video games
- **Robotics:** Path planning, manipulation
- **Autonomous Vehicles:** Decision making, route planning
- **Recommendation Systems:** Sequential recommendations

Limitations:

- Requires complete model of environment
- Computational complexity for large state spaces
- Discrete state and action spaces

GPI as Foundation for RL



Key Insight: Almost all RL algorithms can be viewed as implementations of GPI under different assumptions:

- **Model-free:** Estimate values from experience
- **Online:** Learn while interacting with environment
- **Approximate:** Handle large/continuous state spaces

Summary: Key Takeaways

What We Learned

- **GPI Framework:** Systematic approach to finding optimal policies
- **Two Key Processes:** Policy evaluation + Policy improvement
- **Convergence Guarantees:** Mathematically proven optimality
- **Two Algorithms:** Policy iteration vs Value iteration
- **Practical Challenges:** Computational complexity, model requirements

Practical Impact:

- Foundation for modern reinforcement learning
- Provides theoretical guarantees for convergence
- Template for model-free and approximate methods

Remember

GPI is not just an algorithm—it's a **general principle** that underlies most of reinforcement learning!

Next Steps in RL

Limitations of Dynamic Programming:

- Model-based: Requires complete knowledge of MDP
- Computational: Curse of dimensionality
- Discrete: Limited to finite state/action spaces

Coming Up:

- **Monte Carlo Methods:** Model-free learning from episodes
- **Temporal Difference Learning:** Online, incremental learning
- **Function Approximation:** Handling large state spaces
- **Policy Gradient Methods:** Direct policy optimization

The Journey Continues

Each new method will build upon the GPI framework while addressing specific limitations of dynamic programming.

Questions?

Let's discuss the concepts, applications,
or any clarifications needed!

Think about:

- How would you apply GPI to a real-world problem?
- What challenges might arise in practice?
- How does this connect to machine learning you've seen before?

Thank You!

sali85@student.gsu.edu

Keep iterating towards optimality!