Dynamic Programming in Reinforcement Learning

Policy Improvement and Policy Iteration

Sarwan Ali

Department of Computer Science
Georgia State University

Policy Improvement & Policy Iteration

Today's Learning Journey

Quick Review: Policy Evaluation

Policy Evaluation Problem

Given a policy π, compute the state-value function $v_\pi(s)$ for all states $s \in \mathcal{S}$.

Bellman Equation for v_π:

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

Iterative Policy Evaluation:

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

Key Insight

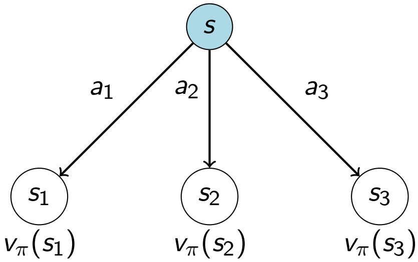
Policy evaluation tells us *how good* a given policy is, but not how to *improve* it.

The Policy Improvement Problem

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

Intuition:

- We know how good each state is under policy π
- Can we make better action choices?
- Look ahead one step and be greedy!



Action-Value Function (Q-Function)

Definition: The action-value function $q_\pi(s, a)$ gives the expected return when taking action a in state s and then following policy π.

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

Relationship to State-Value Function:

$$v_\pi(s) = \sum_{a} \pi(a|s) q_\pi(s, a)$$

Key Insight

$q_\pi(s, a)$ tells us the value of taking action a in state s under policy π. This is exactly what we need for policy improvement!

The Policy Improvement Theorem

Theorem (Policy Improvement Theorem)

Let π and π' be two 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).

Proof Idea:

$$v_\pi(s) \leq q_\pi(s, \pi'(s)) \quad \text{(given)} \tag{1}$$
$$= \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = \pi'(s)] \tag{2}$$
$$\leq \mathbb{E}[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) | S_t = s, A_t = \pi'(s)] \tag{3}$$
$$= \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 v_\pi(S_{t+2}) | S_t = s, A_t = \pi'(s)] \tag{4}$$
$$\leq \ldots \leq v_{\pi'}(s) \tag{5}$$

Greedy Policy Improvement

Greedy Policy: Choose the action that maximizes the action-value function.

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

Policy Improvement Step

Given policy π and its value function v_π:

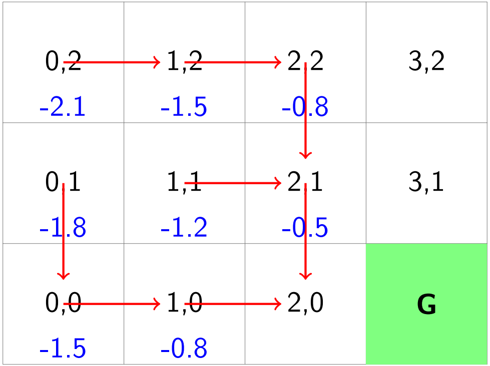
1. Compute $q_\pi(s, a)$ for all state-action pairs
2. Set $\pi'(s) = \arg\max_a q_\pi(s, a)$ for all states
3. The new policy π' is guaranteed to be at least as good as π

When does improvement stop?

When $\pi' = \pi$, we have found the optimal policy π^*!

Policy Improvement: Visual Example

Current Policy π



Value Function v_π

Policy Improvement: Look at each state and ask: "Is there a better action than what the current policy suggests?"

Policy Iteration: The Complete Algorithm

Policy Iteration Algorithm

1. **Initialization:** Choose arbitrary policy π_0 and set $k = 0$
2. **Policy Evaluation:** Compute v_{π_k} (solve the system of linear equations or use iterative method)
3. **Policy Improvement:** For each state s:

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

4. **Convergence Check:** If $\pi_{k+1} = \pi_k$, stop and return $\pi^* = \pi_k$
5. Otherwise, set $k = k + 1$ and go to step 2

Guaranteed Convergence

Policy iteration converges to the optimal policy π^* in a finite number of steps!

Policy Iteration: Convergence Properties

Why does Policy Iteration converge?

- **Finite MDP:** Only finitely many deterministic policies exist
- **Monotonic Improvement:** Each iteration produces a strictly better policy (unless already optimal)
- **No Cycles:** Cannot return to a previously visited policy

Convergence Sequence:

$$\pi_0 \to v_{\pi_0} \to \pi_1 \to v_{\pi_1} \to \pi_2 \to \ldots \to \pi^* \to v^*$$

Time Complexity

- **Policy Evaluation:** $O(|\mathcal{S}|^3)$ per iteration (solving linear system)
- **Policy Improvement:** $O(|\mathcal{S}||\mathcal{A}|)$ per iteration
- **Number of Iterations:** At most $|\mathcal{A}|^{|\mathcal{S}|}$ (usually much fewer)

Policy Iteration vs Value Iteration

Policy Iteration	**Value Iteration**
Alternates between policy evaluation and policy improvement	Updates value function directly using Bellman optimality equation
Each policy evaluation step solves exactly for v_π	Each step performs one backup of the value function
Fewer iterations, but each iteration is more expensive	More iterations, but each iteration is cheaper
$O(\|\mathcal{S}\|^3)$ per policy evaluation	$O(\|\mathcal{S}\|\|\mathcal{A}\|)$ per iteration
Converges in finite steps	Converges asymptotically

When to use which?

- **Policy Iteration:** When policy evaluation is not too expensive
- **Value Iteration:** When states space is large or when we want faster per-iteration updates

Example: Simple Grid World

Actions: $\{\uparrow, \downarrow, \leftarrow, \rightarrow\}$

S1	S2	S3
S4	S5	**+1**

Setup:

- Reward: $+1$ at goal, 0 elsewhere
- $\gamma = 0.9$
- Deterministic transitions

Initial Policy π_0: Move right in all states

Iteration 1 - Policy Evaluation:

$$v_{\pi_0}(S1) = 0 + 0.9 \cdot v_{\pi_0}(S2) \tag{6}$$

$$v_{\pi_0}(S2) = 0 + 0.9 \cdot v_{\pi_0}(S3) \tag{7}$$

$$v_{\pi_0}(S3) = 0 + 0.9 \cdot 0 = 0 \tag{8}$$

$$v_{\pi_0}(S4) = 0 + 0.9 \cdot v_{\pi_0}(S5) \tag{9}$$

$$v_{\pi_0}(S5) = 0 + 0.9 \cdot 1 = 0.9 \tag{10}$$

Solving: $v_{\pi_0} = [0, 0, 0, 0.81, 0.9]$

Policy Improvement for π_0:
For each state, compute $q_{\pi_0}(s, a)$ for all actions and choose the best:

State S1:

$$q_{\pi_0}(S1, \rightarrow) = 0 + 0.9 \cdot 0 = 0 \tag{11}$$
$$q_{\pi_0}(S1, \downarrow) = 0 + 0.9 \cdot 0.81 = 0.729 \tag{12}$$

$\pi_1(S1) = \downarrow$ (improved!)

State S4:

$$q_{\pi_0}(S4, \rightarrow) = 0 + 0.9 \cdot 0.9 = 0.81 \tag{13}$$
$$q_{\pi_0}(S4, \uparrow) = 0 + 0.9 \cdot 0 = 0 \tag{14}$$

$\pi_1(S4) = \rightarrow$ (no change)
Continue for all states...

Modified Policy Iteration

Sometimes full policy evaluation is expensive. We can modify the algorithm:

Modified Policy Iteration

1. **Initialization:** Choose arbitrary policy π_0 and value function V_0
2. **Partial Policy Evaluation:** Perform k steps of iterative policy evaluation:

$$V_{i+1} = T^{\pi} V_i \quad \text{for } i = 0, 1, \ldots, k-1$$

3. **Policy Improvement:** $\pi_{new}(s) = \arg\max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V_k(s')]$
4. Repeat steps 2-3 until convergence

Special Cases:

- $k = 1$: Value Iteration
- $k = \infty$: Standard Policy Iteration
- $k =$ intermediate: Compromise between the two

Implementation Tips

Policy Evaluation Implementation:

- **Iterative Method:** Use convergence threshold θ
- **In-place Updates:** Can speed up convergence
- **Linear System:** For small state spaces, solve $V = R + \gamma PV$ directly

Policy Improvement Implementation:

- Store policy as array: $\pi[s] = a$
- Handle ties in $\arg\max$ consistently
- Check for policy stability: $\pi_{new} = \pi_{old}$

Computational Optimizations:

- **Sparse Representations:** For large, sparse transition matrices
- **Prioritized Sweeping:** Focus updates on important states
- **Asynchronous Updates:** Update states in different orders

Pseudocode: Policy Iteration

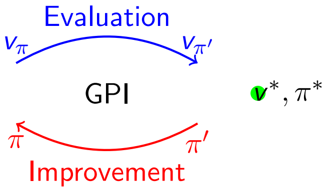
```
def policy_iteration(mdp, gamma=0.9, theta=1e-6):
    # Initialize
    V = np.zeros(mdp.num_states)
    pi = np.random.choice(mdp.num_actions, mdp.num_states)

    while True:
        # Policy Evaluation
        while True:
            delta = 0
            for s in range(mdp.num_states):
                v = V[s]
                V[s] = sum(mdp.P[s][pi[s]][s_prime] *
                           (mdp.R[s][pi[s]][s_prime] + gamma * V[s_prime])
                           for s_prime in range(mdp.num_states))
                delta = max(delta, abs(v - V[s]))
            if delta < theta:
                break

        # Policy Improvement
        policy_stable = True
        for s in range(mdp.num_states):
            old_action = pi[s]
            pi[s] = np.argmax([sum(mdp.P[s][a][s_prime] *
                                   (mdp.R[s][a][s_prime] + gamma * V[s_prime])
                                   for s_prime in range(mdp.num_states))
                               for a in range(mdp.num_actions)])
            if old_action != pi[s]:
                policy_stable = False
        if policy_stable:
            return V, pi
```

Generalized Policy Iteration (GPI)

Key Idea: Policy evaluation and policy improvement can interact in more flexible ways.

GPI Principle:

- Evaluation: Make value function consistent with current policy
- Improvement: Make policy greedy w.r.t current value function
- These processes can be interleaved in various ways



Examples of GPI:

- Policy Iteration
- Value Iteration
- Modified Policy Iteration
- Asynchronous DP methods

Convergence Guarantee

Under GPI, both the value function and policy converge to optimal values, regardless of the specific interleaving pattern.

Stochastic Policies

So far we considered deterministic policies. What about stochastic policies?
Stochastic Policy: $\pi(a|s)$ = probability of taking action a in state s

Policy Improvement for Stochastic Policies:

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

Policy Improvement Theorem (General)

For any stochastic policies π and π', if $q_\pi(s, a) \geq v_\pi(s)$ for all s, a such that $\pi'(a|s) > 0$, then $v_{\pi'}(s) \geq v_\pi(s)$ for all s.

Practical Note: In finite MDPs, there always exists a deterministic optimal policy, so we can focus on deterministic policies.

Summary: Policy Improvement & Policy Iteration

Key Concepts:

- **Policy Improvement:** Given v_π, create better policy by acting greedily
- **Policy Iteration:** Alternate between evaluation and improvement
- **Convergence:** Guaranteed to find optimal policy in finite steps
- **GPI:** General framework for interleaving evaluation and improvement

Important Formulas:

$$q_\pi(s,a) = \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')] \tag{15}$$

$$\pi'(s) = \arg\max_a q_\pi(s,a) \tag{16}$$

$$v_{\pi'}(s) \geq v_\pi(s) \quad \text{(Policy Improvement Theorem)} \tag{17}$$

Computational Complexity:

- Policy Evaluation: $O(|\mathcal{S}|^3)$ or $O(|\mathcal{S}|^2|\mathcal{A}|)$ per iteration
- Policy Improvement: $O(|\mathcal{S}||\mathcal{A}|)$ per iteration
- Number of policy iterations: $\leq |\mathcal{A}|^{|\mathcal{S}|}$ (typically much smaller)

Next Steps

What's Coming Next:

- **Value Iteration:** Direct optimization of value function
- **Asynchronous Dynamic Programming:** Flexible update schedules
- **Approximate Dynamic Programming:** Handling large state spaces
- **Model-Free Methods:** When we don't know the MDP model

Practice Problems:

- Implement policy iteration for grid world problems
- Compare convergence rates of policy iteration vs value iteration
- Analyze the effect of discount factor on convergence
- Implement modified policy iteration with different k values

? Questions?