# Dynamic Programming in Reinforcement Learning

## Policy Evaluation (Prediction Problem)

Sarwan Ali
Department of Computer Science

Georgia State University

Policy Evaluation in Dynamic Programming

# Today's Learning Journey

1. Introduction to Policy Evaluation
2. Bellman Equation for Policy Evaluation
3. Iterative Policy Evaluation
4. Implementation Considerations
5. Worked Example
6. Applications and Extensions

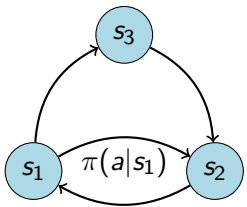
# What is Policy Evaluation?

## Definition

**Policy Evaluation** (also called the *prediction problem*) is the task of computing the state-value function $v_\pi(s)$ for a given policy $\pi$.

**Key Questions:**

- Given a policy $\pi$, how good is each state?
- What is the expected return from state $s$ following $\pi$?
- How do we compute $v_\pi(s)$ efficiently?




Policy $\pi$ determines transitions

# Recall: State-Value Function

## State-Value Function for Policy $\pi$

$$v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s] = \mathbb{E}_\pi \left[ \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s \right]$$

**Components:**

- $G_t$: Return (cumulative discounted reward)
- $\gamma$: Discount factor ($0 \leq \gamma \leq 1$)
- $\mathbb{E}_\pi[\cdot]$: Expectation under policy $\pi$
- $R_{t+k+1}$: Reward at time step $t + k + 1$

## Key Insight

$v_\pi(s)$ tells us the *expected long-term value* of being in state $s$ and following policy $\pi$ thereafter.

# The Bellman Equation for $v_\pi$

## Bellman Equation for State-Value Function

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

**Intuitive Breakdown:**

1. For each possible action $a$ in state $s$:
   - Weight by policy probability $\pi(a|s)$
2. For each possible next state $s'$ and reward $r$:
   - Weight by transition probability $p(s', r|s, a)$
   - Add immediate reward $r$ plus discounted future value $\gamma v_\pi(s')$

## Key Property

This is a **system of linear equations** in the unknowns $v_\pi(s)$ for all $s \in \mathcal{S}$.

# Bellman Equation: Matrix Form

For finite MDPs, we can write the Bellman equation in matrix form:

## Matrix Form

$$\mathbf{v}_\pi = \mathbf{r}_\pi + \gamma \mathbf{P}_\pi \mathbf{v}_\pi$$

**Where:**

- $\mathbf{v}_\pi$: Vector of state values $[v_\pi(s_1), v_\pi(s_2), \ldots, v_\pi(s_n)]^T$
- $\mathbf{r}_\pi$: Vector of expected immediate rewards under $\pi$
- $\mathbf{P}_\pi$: State transition probability matrix under $\pi$

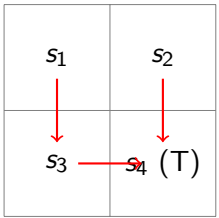
## Closed-Form Solution

$$\mathbf{v}_\pi = (\mathbf{I} - \gamma \mathbf{P}_\pi)^{-1} \mathbf{r}_\pi$$

**Problem:** Matrix inversion is $O(n^3)$ - impractical for large state spaces!

# Example: Simple Grid World

Consider a $2 \times 2$ grid world with deterministic policy:




$r = -1$ except terminal

**Bellman Equations:**

$$v_\pi(s_1) = -1 + \gamma v_\pi(s_3) \quad (1)$$
$$v_\pi(s_2) = -1 + \gamma v_\pi(s_4) \quad (2)$$
$$v_\pi(s_3) = -1 + \gamma v_\pi(s_4) \quad (3)$$
$$v_\pi(s_4) = 0 \text{ (terminal)} \quad (4)$$

**Solution with $\gamma = 0.9$:**

- $v_\pi(s_4) = 0$
- $v_\pi(s_3) = -1 + 0.9 \times 0 = -1$
- $v_\pi(s_2) = -1 + 0.9 \times 0 = -1$
- $v_\pi(s_1) = -1 + 0.9 \times (-1) = -1.9$

# Iterative Policy Evaluation Algorithm

Since matrix inversion is expensive, use **iterative methods**:

## 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 Ideas:**

- Start with arbitrary initial values $v_0(s)$ for all $s$
- Repeatedly apply the Bellman equation as an update rule
- Under certain conditions, $v_k \to v_\pi$ as $k \to \infty$

**Advantages:**

- Simple to implement
- Guaranteed convergence
- Memory efficient

**Considerations:**

- Requires many iterations
- Needs full model knowledge
- Computational cost per iteration

# Iterative Policy Evaluation: Pseudocode

**Algorithm 1** Iterative Policy Evaluation

**Require:** Policy $\pi$ to be evaluated

**Require:** Small threshold $\theta > 0$ determining accuracy

Initialize $V(s) = 0$ for all $s \in \mathcal{S}^+$

**repeat**

  $\Delta \leftarrow 0$

  **for** each $s \in \mathcal{S}$ **do**

    $v \leftarrow V(s)$

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

    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$

  **end for**

**until** $\Delta < \theta$

**return** $V \approx v_\pi$

**Time Complexity:** $O(|\mathcal{S}|^2|\mathcal{A}|)$ per iteration

**Space Complexity:** $O(|\mathcal{S}|)$

# Convergence Properties

## Theorem (Convergence of Iterative Policy Evaluation)

*For any policy $\pi$ and any initial value function $v_0$, the sequence $\{v_k\}$ generated by iterative policy evaluation converges to $v_\pi$ as $k \to \infty$.*

**Why does this work?**

- The Bellman operator $T_\pi$ defined by:

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

- Is a **contraction mapping** with contraction factor $\gamma$
- By Banach Fixed-Point Theorem, has unique fixed point $v_\pi$

## Rate of Convergence

$$\|v_k - v_\pi\|_\infty \leq \gamma^k \|v_0 - v_\pi\|_\infty$$

Convergence is **exponentially fast** with rate $\gamma$.

# In-Place vs. Two-Array Updates

**Two-Array Method:**

- Use separate arrays for $v_k$ and $v_{k+1}$
- Update all states simultaneously
- Requires $2|\mathcal{S}|$ memory

**Update Rule**

$$v_{k+1}(s) = T_\pi v_k(s)$$

**In-Place Method:**

- Use single array
- Update states one at a time
- Use most recent values
- Requires $|\mathcal{S}|$ memory

**Update Rule**

$$v(s) \leftarrow T_\pi v(s)$$

**Key Insight**

In-place updates typically converge **faster** because they use more up-to-date information, though convergence is still guaranteed.

# Stopping Criteria

**How do we know when to stop iterating?**

1. **Maximum Change Criterion:**

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

2. **Relative Change Criterion:**

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

3. **Fixed Number of Iterations:** Simply run for $N$ iterations

## Error Bound

If we stop when $\max_s |v_{k+1}(s) - v_k(s)| < \theta$, then:

$$\|v_k - v_\pi\|_\infty \leq \frac{\theta}{1-\gamma}$$

# Computational Complexity Analysis

**Per Iteration Complexity:**

- For each state $s \in \mathcal{S}$: $O(|\mathcal{A}| \times |\mathcal{S}|)$
- Total per iteration: $O(|\mathcal{S}|^2|\mathcal{A}|)$

**Number of Iterations:**

- Depends on discount factor $\gamma$ and desired accuracy $\theta$
- Approximately $O(\log(1/\theta)/\log(1/\gamma))$ iterations

**Total Complexity:**

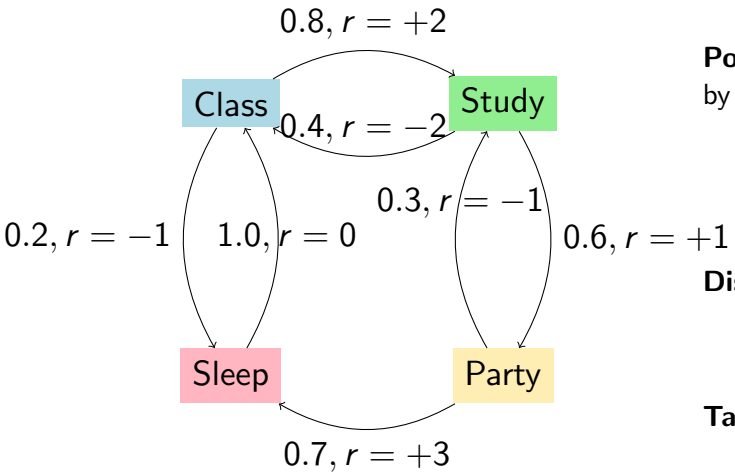
$$O\left(\frac{|\mathcal{S}|^2|\mathcal{A}|\log(1/\theta)}{\log(1/\gamma)}\right)$$

## Practical Implications

- Higher $\gamma$ (closer to 1) $\Rightarrow$ slower convergence
- Smaller $\theta$ (higher accuracy) $\Rightarrow$ more iterations
- Quadratic in number of states $\Rightarrow$ challenging for large state spaces

# Example: Student MDP

Consider a simplified model of a student's day:




**Policy $\pi$:** Deterministic policy shown by transition probabilities

**Discount factor:** $\gamma = 0.9$

**Task:** Compute $v_\pi(s)$ for each state

# Student MDP: Bellman Equations

**Setting up the system:**

$$v_\pi(\text{Class}) = 0.8[2 + 0.9 \cdot v_\pi(\text{Study})] + 0.2[-1 + 0.9 \cdot v_\pi(\text{Sleep})] \quad (5)$$
$$v_\pi(\text{Study}) = 0.4[-2 + 0.9 \cdot v_\pi(\text{Class})] + 0.6[1 + 0.9 \cdot v_\pi(\text{Party})] \quad (6)$$
$$v_\pi(\text{Party}) = 0.7[3 + 0.9 \cdot v_\pi(\text{Sleep})] + 0.3[-1 + 0.9 \cdot v_\pi(\text{Study})] \quad (7)$$
$$v_\pi(\text{Sleep}) = 1.0[0 + 0.9 \cdot v_\pi(\text{Class})] \quad (8)$$

**Simplifying:**

$$v_\pi(C) = 1.6 + 0.72 \cdot v_\pi(S) - 0.18 \cdot v_\pi(Sl) \quad (9)$$
$$v_\pi(S) = -0.8 + 0.36 \cdot v_\pi(C) + 0.54 \cdot v_\pi(P) \quad (10)$$
$$v_\pi(P) = 2.1 + 0.63 \cdot v_\pi(Sl) + 0.27 \cdot v_\pi(S) \quad (11)$$
$$v_\pi(Sl) = 0.9 \cdot v_\pi(C) \quad (12)$$

# Student MDP: Iterative Solution

**Iteration 0:** Initialize $v_0(s) = 0$ for all states

| Iteration | Class | Study | Party | Sleep |
|---|---|---|---|---|
| 0 | 0.00 | 0.00 | 0.00 | 0.00 |
| 1 | 1.60 | -0.80 | 2.10 | 0.00 |
| 2 | 1.03 | 1.35 | 2.10 | 1.44 |
| 3 | 2.57 | 1.50 | 3.01 | 0.93 |
| 4 | 2.42 | 2.55 | 3.65 | 2.31 |
| 5 | 3.43 | 2.85 | 4.14 | 2.18 |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| **Converged** | **4.12** | **3.68** | **5.02** | **3.71** |

**Interpretation:**

- Party state has highest value (5.02) - most rewarding long-term
- Class state has good value (4.12) - leads to productive outcomes
- Study and Sleep have similar moderate values

# When Do We Use Policy Evaluation?

**Direct Applications:**

- **Policy Assessment:** Evaluate how good a given policy is
- **Comparison:** Compare multiple policies
- **Debugging:** Understand policy behavior in different states

**As Building Block for:**

- **Policy Iteration:** Alternates between policy evaluation and improvement
- **Value Iteration:** Truncated policy evaluation (1 step)
- **Actor-Critic Methods:** Continuous policy evaluation
- **Monte Carlo Methods:** Sample-based policy evaluation

**Key Insight**

Policy evaluation is the **prediction** component of reinforcement learning. Most RL algorithms need some form of value function estimation.

# Limitations and Challenges

**Computational Limitations:**

- **Curse of Dimensionality:** $O(|\mathcal{S}|^2)$ complexity
- **Full Model Required:** Need $p(s', r|s, a)$ for all transitions
- **Memory Requirements:** Store value for every state

**Practical Challenges:**

- **Large State Spaces:** Millions or billions of states
- **Continuous States:** Infinite state spaces
- **Unknown Dynamics:** Model-free environments

**Solutions:**

- **Function Approximation:** Neural networks, linear functions
- **Sampling Methods:** Monte Carlo, Temporal Difference
- **State Aggregation:** Group similar states

# Summary and Key Takeaways

## Policy Evaluation: Core Concepts

- **Goal:** Compute $v_\pi(s)$ for a given policy $\pi$
- **Method:** Solve Bellman equation $v_\pi = r_\pi + \gamma P_\pi v_\pi$
- **Algorithm:** Iterative application of Bellman operator
- **Convergence:** Guaranteed for $\gamma < 1$

**Practical Implementation:**

- Use in-place updates for faster convergence
- Choose appropriate stopping criteria
- Consider computational complexity

**Next Steps:**

- **Policy Improvement:** How to find better policies
- **Policy Iteration:** Combining evaluation and improvement
- **Value Iteration:** More efficient dynamic programming

# Questions and Discussion

**Questions?**

sali85@student.gsu.edu

*Next: Policy Improvement and Policy Iteration*