Reinforcement Learning

Lecture 4: Mathematical Foundations - MDPs and Bellman Equations

Taehoon Kim

Sogang University MIMIC Lab
https://mimic-lab.com

Fall Semester 2025

Learning Objectives

By the end of this lecture, you will:

1. **Understand** the mathematical formulation of MDPs
2. **Master** value functions and their recursive relationships
3. **Derive** and apply Bellman equations
4. **Implement** policy evaluation and improvement
5. **Code** complete Policy Iteration and Value Iteration
6. **Analyze** convergence properties and error bounds

Prerequisites:

- Linear algebra (matrix operations)
- Probability theory basics
- PyTorch tensor operations
- Python programming experience

What is a Markov Decision Process?

An MDP is a mathematical framework for sequential decision-making

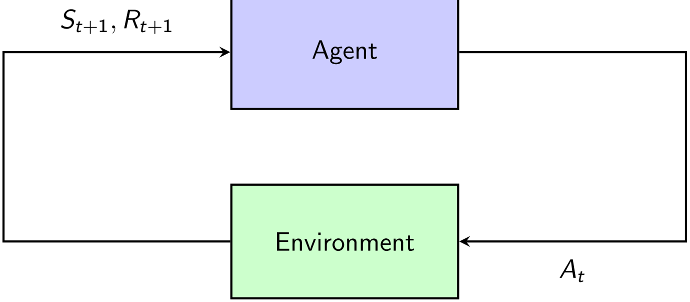
Components: $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, r, \gamma)$

- $\mathcal{S}$: State space (finite or infinite)
- $\mathcal{A}$: Action space (finite or infinite)
- P: Transition dynamics $P(s'|s, a)$
- r: Reward function $r(s, a) = \mathbb{E}[R_{t+1}|S_t = s, A_t = a]$
- γ: Discount factor $\gamma \in [0, 1]$

Key Property: Markov Property

$$P(S_{t+1}|S_t, A_t, S_{t-1}, A_{t-1}, \ldots) = P(S_{t+1}|S_t, A_t) \tag{1}$$

Agent-Environment Interaction



Trajectory: $S_0, A_0, R_1, S_1, A_1, R_2, S_2, \ldots$

Example: GridWorld MDP

GridWorld Environment:

- States: Grid cells
- Actions: {UP, RIGHT, DOWN, LEFT}
- Transitions: Move to adjacent cell
- Rewards: -0.04 per step, +1 at goal, -1 at pit
- Discount: $\gamma = 0.99$

S	.	.	G(+1)
.	#	.	P(-1)
.	.	.	.

S = Start, G = Goal, P = Pit, # = Wall

Transition Dynamics

Deterministic vs Stochastic Transitions

Deterministic:

- Action always leads to same next state
- $P(s'|s,a) \in \{0,1\}$
- Simpler to analyze

Stochastic:

- Action may lead to different states
- $P(s'|s,a) \in [0,1]$
- More realistic

Example with slip probability 0.2:

- Intended direction: probability 0.6
- Perpendicular directions: probability 0.2 each

Policies

A policy π defines the agent's behavior

Types of Policies:

- **Deterministic:** $\pi : \mathcal{S} \to \mathcal{A}$
    - Maps each state to a single action
    - $a = \pi(s)$
- **Stochastic:** $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$
    - Probability distribution over actions
    - $\pi(a|s) = P(A_t = a|S_t = s)$

Goal: Find the optimal policy π^* that maximizes expected return

Returns and Episodes

Return: Sum of (discounted) future rewards

Finite Horizon Return:

$$G_t = R_{t+1} + R_{t+2} + ... + R_T \tag{2}$$

Infinite Horizon Discounted Return:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{3}$$

Why discount?

- Mathematical convergence
- Uncertainty about the future
- Economic interpretation (interest rates)
- Bounded returns: $|G_t| \leq \frac{R_{max}}{1-\gamma}$

State-Value Function

Value of a state under policy π:

$$v_\pi(s) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \middle| S_0 = s \right] \tag{4}$$

Interpretation:

- Expected return starting from state s
- Following policy π thereafter
- Accounts for all future rewards (discounted)

Properties:

- Bounded: $|v_\pi(s)| \leq \frac{R_{max}}{1-\gamma}$
- Unique for a given policy and MDP

Action-Value Function (Q-Function)

Value of taking action a in state s under policy π:

$$q_\pi(s,a) = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \,\middle|\, S_0 = s, A_0 = a\right] \tag{5}$$

Relationship to state-value:

$$v_\pi(s) = \sum_a \pi(a|s)\, q_\pi(s,a) \tag{6}$$

Q-function tells us:

- How good is action a in state s?
- Basis for action selection
- Central to Q-learning algorithms

Value Function Examples

Random Policy Values:

0.52	0.55	0.61	+1
0.48	#	0.43	-1
0.44	0.41	0.38	0.35

Optimal Policy Values:

0.81	0.87	0.92	+1
0.76	#	0.66	-1
0.71	0.66	0.61	0.39

Observation: Optimal values are higher (better policy)

Bellman Expectation Equation for v_π

Recursive decomposition of value:

$$v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s] \tag{7}$$

Expanded form:

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s'} P(s'|s,a)[r(s,a) + \gamma v_\pi(s')] \tag{8}$$

Key insight:

- Value = immediate reward + discounted future value
- Self-consistent system of equations
- One equation per state

Bellman Expectation Equation for q_π

Recursive decomposition:

$$q_\pi(s,a) = r(s,a) + \gamma \sum_{s'} P(s'|s,a) \sum_{a'} \pi(a'|s') q_\pi(s',a') \tag{9}$$

Relationship between V and Q:

$$v_\pi(s) = \sum_{a} \pi(a|s) q_\pi(s,a) \tag{10}$$

$$q_\pi(s,a) = r(s,a) + \gamma \sum_{s'} P(s'|s,a) v_\pi(s') \tag{11}$$

These equations form the basis for policy evaluation

Matrix Form of Bellman Expectation

For deterministic policy π:
Define:

- $\mathsf{v}_\pi \in \mathbb{R}^{|\mathcal{S}|}$: value vector
- $\mathsf{r}_\pi \in \mathbb{R}^{|\mathcal{S}|}$: reward vector
- $\mathsf{P}_\pi \in \mathbb{R}^{|\mathcal{S}| \times |\mathcal{S}|}$: transition matrix

Bellman equation in matrix form:

$$\mathsf{v}_\pi = \mathsf{r}_\pi + \gamma \mathsf{P}_\pi \mathsf{v}_\pi \tag{12}$$

Solution:

$$\mathsf{v}_\pi = (\mathsf{I} - \gamma \mathsf{P}_\pi)^{-1} \mathsf{r}_\pi \tag{13}$$

Note: Direct inversion is $O(n^3)$ - iterative methods preferred

Bellman Expectation Operator

Define operator $T_\pi : \mathbb{R}^{|\mathcal{S}|} \to \mathbb{R}^{|\mathcal{S}|}$:

$$(T_\pi v)(s) = r_\pi(s) + \gamma \sum_{s'} P_\pi(s'|s) v(s') \tag{14}$$

Properties:

- T_π is a γ-contraction in $\|\cdot\|_\infty$
- $\|T_\pi v - T_\pi w\|_\infty \leq \gamma \|v - w\|_\infty$
- Has unique fixed point v_π
- $v_\pi = T_\pi v_\pi$ (Bellman equation)

Banach Fixed-Point Theorem:

- Starting from any v_0
- Sequence $v_{k+1} = T_\pi v_k$ converges to v_π
- Convergence rate: geometric with factor γ

Optimal Value Functions

Optimal state-value function:

$$v_*(s) = \max_\pi v_\pi(s) \tag{15}$$

Optimal action-value function:

$$q_*(s, a) = \max_\pi q_\pi(s, a) \tag{16}$$

Properties:

- Unique for a given MDP
- Defines the best possible performance
- Independent of initial state distribution

Optimal policy: Any policy achieving v_* is optimal

Bellman Optimality Equation for v_*

The optimal value satisfies:

$$v_*(s) = \max_a \left\{ r(s,a) + \gamma \sum_{s'} P(s'|s,a) v_*(s') \right\} \tag{17}$$

Key difference from expectation equation:

- Max over actions instead of expectation
- Non-linear due to max operator
- Harder to solve than expectation equation

Interpretation:

- Optimal value = best immediate reward + discounted future
- Greedy action selection

Bellman Optimality Equation for q_*

The optimal Q-function satisfies:

$$q_*(s, a) = r(s, a) + \gamma \sum_{s'} P(s'|s, a) \max_{a'} q_*(s', a') \tag{18}$$

Relationship between v_* and q_*:

$$v_*(s) = \max_{a} q_*(s, a) \tag{19}$$

$$q_*(s, a) = r(s, a) + \gamma \sum_{s'} P(s'|s, a) v_*(s') \tag{20}$$

Optimal policy extraction:

$$\pi_*(s) \in \arg\max_{a} q_*(s, a) \tag{21}$$

Bellman Optimality Operator

Define operator $T_* : \mathbb{R}^{|\mathcal{S}|} \to \mathbb{R}^{|\mathcal{S}|}$**:**

$$(T_* v)(s) = \max_a \left\{ r(s,a) + \gamma \sum_{s'} P(s'|s,a) v(s') \right\} \tag{22}$$

Properties:

- T_* is a γ-contraction
- Unique fixed point v_*
- Monotonic: if $v \leq w$ then $T_* v \leq T_* w$
- Non-linear due to max

Value Iteration: $v_{k+1} = T_* v_k$ converges to v_*

Dynamic Programming for MDPs

Requirements:

- Complete model of environment (P and r known)
- Finite state and action spaces
- Computational resources

Two main algorithms:

1. **Policy Iteration:** Alternates evaluation and improvement
2. **Value Iteration:** Directly applies optimality operator

Both algorithms:

- Converge to optimal policy
- Based on Bellman equations
- Exact solutions for finite MDPs

Policy Evaluation

Given: Policy π, MDP model
Find: v_π

Algorithm 1 Iterative Policy Evaluation

```
Initialize V(s) = 0 for all s ∈ S
repeat
    Δ ← 0
    for each s ∈ S do
        v ← V(s)
        V(s) ← Σ_a π(a|s) Σ_s' P(s'|s,a)[r(s,a) + γV(s')]
        Δ ← max(Δ, |v − V(s)|)
    end for
until Δ < ε
return V
```

Policy Improvement

Given: Value function v_π
Find: Better policy π'
Greedy policy with respect to v_π:

$$\pi'(s) = \arg\max_a \left\{ r(s,a) + \gamma \sum_{s'} P(s'|s,a) v_\pi(s') \right\} \tag{23}$$

Policy Improvement Theorem:

- If π' is greedy w.r.t. v_π
- Then $v_{\pi'}(s) \geq v_\pi(s)$ for all s
- Equality holds iff π is optimal

This guarantees monotonic improvement!

Policy Iteration

Algorithm 2 Policy Iteration

```
Initialize π arbitrarily
repeat
    Policy Evaluation:
        Compute v_π using iterative evaluation
    Policy Improvement:
        π_old ← π
    for each s ∈ S do
        π(s) ← arg max_a q_π(s, a)
    end for
until π = π_old
return π
```

Convergence: Finite number of iterations for finite MDPs

Value Iteration

Algorithm 3 Value Iteration

Initialize $V(s) = 0$ for all $s \in \mathcal{S}$
repeat
    $\Delta \leftarrow 0$
    for each $s \in \mathcal{S}$ **do**
        $v \leftarrow V(s)$
        $V(s) \leftarrow \max_a \sum_{s'} P(s'|s,a)[r(s,a) + \gamma V(s')]$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
    end for
until $\Delta < \epsilon$
Extract policy: $\pi(s) = \arg\max_a q(s,a)$
return π

Combines evaluation and improvement in single update

Policy Iteration vs Value Iteration

Aspect	**Policy Iteration**	**Value Iteration**
Updates	Alternating	Combined
Convergence	Fewer iterations	More iterations
Per iteration cost	Higher (full evaluation)	Lower (single backup)
Memory	Store policy + values	Store values only
Early stopping	Natural (policy stable)	Needs error bound
Implementation	More complex	Simpler

In practice:

- VI often preferred for simplicity
- PI can be faster for some problems
- Modified PI: partial evaluation

Contraction Mapping Theorem

Theorem: If T is a γ-contraction on complete metric space, then:

1. T has unique fixed point v^*
2. For any v_0, sequence $v_{k+1} = Tv_k$ converges to v^*
3. $\|v_k - v^*\|_\infty \leq \gamma^k \|v_0 - v^*\|_\infty$

Application to RL:

- Both T_π and T_* are contractions
- Guarantees convergence of DP algorithms
- Provides convergence rate

Convergence rate:

- Geometric with factor γ
- Higher $\gamma \to$ slower convergence
- Number of iterations $\propto \frac{1}{1-\gamma}$

Error Bounds for Value Iteration

After k iterations of value iteration:

$$\|V_k - v_*\|_\infty \leq \gamma^k \|V_0 - v_*\|_\infty \tag{24}$$

Using Bellman residual $\delta_k = \|V_{k+1} - V_k\|_\infty$:

$$\|V_k - v_*\|_\infty \leq \frac{\gamma}{1-\gamma}\delta_k \tag{25}$$

Stopping criterion for ϵ-optimal solution:

- Want: $\|V_k - v_*\|_\infty \leq \epsilon$
- Stop when: $\delta_k < \frac{(1-\gamma)\epsilon}{2\gamma}$

These bounds are tight and computable!

Computational Complexity

Per iteration complexity:

- Policy Evaluation: $O(|\mathcal{S}|^2|\mathcal{A}|)$ per iteration
- Policy Improvement: $O(|\mathcal{S}|^2|\mathcal{A}|)$
- Value Iteration: $O(|\mathcal{S}|^2|\mathcal{A}|)$ per iteration

Number of iterations:

- Policy Iteration: $O(|\mathcal{A}|^{|\mathcal{S}|})$ worst case (rarely reached)
- Value Iteration: $O(\frac{1}{1-\gamma} \log \frac{1}{\epsilon})$

Space complexity:

- Transition matrix: $O(|\mathcal{S}|^2|\mathcal{A}|)$
- Value function: $O(|\mathcal{S}|)$
- Policy: $O(|\mathcal{S}|)$

Implementation: Setup

```python
import torch
import numpy as np

def setup_seed(seed=42):
    torch.manual_seed(seed)
    np.random.seed(seed)
    if torch.cuda.is_available():
        torch.cuda.manual_seed_all(seed)

device = torch.device(
    'cuda' if torch.cuda.is_available()
    else 'mps' if torch.backends.mps.is_available()
    else 'cpu'
)

# MDP components shapes
# P: [S, A, S] - transition probabilities
# R: [S, A]    - rewards
# V: [S]       - values
# Q: [S, A]    - Q-values
# pi: [S]      - deterministic policy
```

GridWorld MDP Implementation

```python
class GridWorldMDP:
    def __init__(self, grid, terminal_rewards,
                 step_cost=-0.04, slip_prob=0.1):
        self.height = len(grid)
        self.width = len(grid[0])

        # Build state mapping
        self.state_to_pos = {}
        self.pos_to_state = {}
        state_idx = 0
        for r in range(self.height):
            for c in range(self.width):
                if grid[r][c] != '#':  # Not a wall
                    self.state_to_pos[state_idx] = (r, c)
                    self.pos_to_state[(r, c)] = state_idx
                    state_idx += 1

        self.n_states = len(self.state_to_pos)
        self.n_actions = 4  # UP, RIGHT, DOWN, LEFT
```

Building Transition and Reward Matrices

```python
def build_dynamics(self):
    P = torch.zeros((self.n_states, self.n_actions,
                     self.n_states), device=device)
    R = torch.full((self.n_states, self.n_actions),
                   self.step_cost, device=device)

    for s in range(self.n_states):
        if self.is_terminal[s]:
            P[s, :, s] = 1.0  # Absorbing state
            R[s, :] = 0.0
            continue

        for a in range(self.n_actions):
            # Stochastic transitions with slip
            for actual_a, prob in self.get_transitions(a):
                next_s = self.get_next_state(s, actual_a)
                P[s, a, next_s] += prob
                R[s, a] += prob * self.get_reward(next_s)

    return P, R
```

Policy Evaluation in PyTorch

```python
def policy_evaluation(P, R, pi, gamma=0.99, tol=1e-8):
    """Evaluate a policy using matrix operations"""
    S = P.shape[0]

    # Convert policy to transition matrix
    if pi.dim() == 1:  # Deterministic
        P_pi = P[torch.arange(S), pi]  # [S, S]
        R_pi = R[torch.arange(S), pi]  # [S]
    else:  # Stochastic
        P_pi = torch.einsum('sa,sas->ss', pi, P)
        R_pi = torch.einsum('sa,sa->s', pi, R)

    V = torch.zeros(S, device=P.device)

    for _ in range(1000):
        V_new = R_pi + gamma * (P_pi @ V)
        if torch.max(torch.abs(V_new - V)) < tol:
            break
        V = V_new

    return V
```

Policy Improvement in PyTorch

```python
def policy_improvement(P, R, V, gamma=0.99):
    """Extract greedy policy from value function"""
    # Compute Q-values
    # Q(s,a) = R(s,a) + gamma * sum_s' P(s,a,s') * V(s')
    Q = R + gamma * torch.einsum('sas,s->sa', P, V)

    # Extract greedy policy
    pi = torch.argmax(Q, dim=1)

    return pi, Q

def compute_q_values(P, R, V, gamma=0.99):
    """Compute Q-values from value function"""
    return R + gamma * torch.einsum('sas,s->sa', P, V)
```

Complete Policy Iteration

```python
def policy_iteration(P, R, gamma=0.99, tol=1e-8):
    S, A = P.shape[0], P.shape[1]
    pi = torch.zeros(S, dtype=torch.long)  # Start arbitrary

    for iteration in range(100):
        # Policy Evaluation
        V = policy_evaluation(P, R, pi, gamma, tol)

        # Policy Improvement
        pi_new, Q = policy_improvement(P, R, V, gamma)

        # Check convergence
        if torch.equal(pi_new, pi):
            print(f"Converged in {iteration + 1} iterations")
            break

        pi = pi_new

    return pi, V
```

Value Iteration in PyTorch

```python
def value_iteration(P, R, gamma=0.99, tol=1e-8):
    S = P.shape[0]
    V = torch.zeros(S, device=P.device)

    for iteration in range(1000):
        V_old = V.clone()

        # Bellman optimality update
        Q = compute_q_values(P, R, V_old, gamma)
        V = torch.max(Q, dim=1)[0]

        # Check convergence
        delta = torch.max(torch.abs(V - V_old))
        if delta < tol:
            print(f"Converged in {iteration + 1} iterations")
            break

    # Extract optimal policy
    Q_final = compute_q_values(P, R, V, gamma)
    pi = torch.argmax(Q_final, dim=1)

    return pi, V
```

Experiment 1: Environment Setup

File: `exp01_setup.py`
Objectives:

- Verify PyTorch installation
- Test device selection (CUDA/MPS/CPU)
- Initialize MDP tensors
- Verify tensor shapes and operations

Key checks:

- Transition probabilities sum to 1
- Reward bounds are finite
- Bellman operator properties

Expected output: System info, device selection, tensor verification

Experiment 2: GridWorld MDP

File: `exp02_gridworld.py`
Tasks:

- Build complete GridWorld environment
- Create transition matrix P[s,a,s']
- Create reward matrix R[s,a]
- Handle walls and terminal states
- Add stochastic transitions (slip)

Verification:

- P matrix: proper probability distribution
- Terminal states are absorbing
- Transitions respect walls

Experiment 3: Policy Evaluation

File: `exp03_policy_evaluation.py`
Implement and test:

- Iterative policy evaluation
- Contraction property verification
- Compare different policies (random, fixed)
- Analyze convergence rates with different γ

Key observations:

- Geometric convergence
- Higher $\gamma \rightarrow$ slower convergence
- Unique fixed point

Experiment 4: Policy Improvement

File: `exp04_policy_improvement.py`
Tasks:

- Compute Q-values from V
- Extract greedy policy
- Verify improvement theorem
- Compare deterministic vs soft improvement

Verification:

- $v_{\pi'} \geq v_{\pi}$ for all states
- Monotonic improvement
- Convergence to optimal

Experiment 5: Policy Iteration

File: `exp05_policy_iteration.py`
Complete implementation:

- Full policy iteration algorithm
- Track policy evolution
- Compare different initial policies
- Visualize convergence

Analysis:

- Number of iterations to convergence
- Policy changes per iteration
- Final optimal policy

Experiment 6: Value Iteration

File: `exp06_value_iteration.py`
Implement:

- Value iteration algorithm
- Compare with policy iteration
- Test different initializations
- Analyze Bellman optimality operator

Comparisons:

- VI vs PI convergence speed
- Computational efficiency
- Memory requirements

Experiment 7: Stopping Criteria

File: `exp07_stopping_criteria.py`
Explore:

- Different stopping criteria
- Error bounds computation
- Trade-offs: accuracy vs computation
- Early stopping strategies

Key results:

- Theoretical bounds are tight
- Policy often converges before values
- Span seminorm alternative

Experiment 8: Algorithmic Optimizations

File: `exp08_optimizations.py`
Compare:

- Synchronous vs asynchronous updates
- Prioritized sweeping
- GPU acceleration
- Memory efficiency

Performance metrics:

- Iterations to convergence
- Wall-clock time
- Memory usage
- Scalability

Experiment 9: Integrated Test

File: `exp09_integrated_test.py`
Complete pipeline:

- Multiple test scenarios
- All algorithms comparison
- Comprehensive visualization
- Performance benchmarking

Deliverables:

- Optimal policies for each scenario
- Algorithm comparison report
- Convergence verification
- Reproducibility guarantee

Key Takeaways

Theory:

- MDPs provide mathematical framework for sequential decisions
- Value functions satisfy Bellman equations
- Bellman operators are contractions $\rightarrow$ unique solutions
- DP algorithms solve MDPs exactly

Practice:

- Implemented complete GridWorld MDP
- Policy evaluation converges geometrically
- Policy improvement guarantees monotonic improvement
- PI and VI converge to same optimal policy

Insights:

- Model-based methods are sample efficient
- Computation scales with state/action space
- Foundation for model-free methods

Mathematical Foundations Summary

Concept	**Key Equation**
State Value	$v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s]$
Action Value	$q_\pi(s,a) = \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a]$
Bellman Expectation	$v_\pi = r_\pi + \gamma P_\pi v_\pi$
Bellman Optimality	$v_* = \max_a \{r_a + \gamma P_a v_*\}$
Policy Evaluation	$v_{k+1} = T_\pi v_k$
Value Iteration	$v_{k+1} = T_* v_k$
Greedy Policy	$\pi(s) = \arg\max_a q(s,a)$

Remember: These equations are the heart of RL!