# Reinforcement Learning

Dr Stephen James

Autumn Term 2025

Imperial College London

# Reinforcement Learning
# Lecture 2: Dynamic Programming

Dr Stephen James

Autumn Term 2025

Imperial College London

# Outline

# From Theory to Practice

1. Introduction
2. Policy Evaluation
3. Policy Improvement
4. Policy Iteration
5. Value Iteration
6. Generalized Policy Iteration

# How do we turn the Bellman equations into working algorithms?

Week 1: The mathematical foundation

Week 2: How do we actually solve MDPs?

## Today's Goals

- Transform Bellman equations into practical algorithms
- Master Policy Iteration and Value Iteration
- Understand the computational foundations of all RL

# What is Dynamic Programming?

**Definition (Dynamic Programming (DP))**
A method for solving complex problems by breaking them down into simpler subproblems, solving each subproblem once, and storing the results.

**Examples of DP in everyday life**

- **Fibbonacci numbers**: Store previous results to avoid recomputation
- **Shortest paths**: Dijkstra's and Bellman-Ford algorithms
- **Reinforcement Learning**: Solving MDPs via Bellman equations

DP gives us exact algorithms for computing optimal policies and value functions

# Policy Evaluation

# Given a policy $\pi$, how do we compute its value function?

This is called the policy evaluation problem.

Recall the Bellman equation for $V^\pi$:

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

The Challenge
This is a system of $|\mathcal{S}|$ linear equations in $|\mathcal{S}|$ unknowns.

Example: 3-state MDP gives us 3 coupled equations:

$$V^\pi(s_1) = \sum_a \pi(a|s_1) \sum_{s',r} P(s', r|s_1, a)[r + \gamma V^\pi(s')]$$

$$V^\pi(s_2) = \sum_a \pi(a|s_2) \sum_{s',r} P(s', r|s_2, a)[r + \gamma V^\pi(s')]$$

$$V^\pi(s_3) = \sum_a \pi(a|s_3) \sum_{s',r} P(s', r|s_3, a)[r + \gamma V^\pi(s')]$$

We could solve it directly, but there's a more elegant iterative approach.

# Given a policy $\pi$, how do we compute its value function?

This is called the policy evaluation problem.

Recall the Bellman equation for $V^\pi$:

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

**The Challenge**
This is a system of $|\mathcal{S}|$ linear equations in $|\mathcal{S}|$ unknowns.

Example: 3-state MDP gives us 3 coupled equations:

$$V^\pi(s_1) = \sum_a \pi(a|s_1) \sum_{s',r} P(s', r|s_1, a)[r + \gamma V^\pi(s')]$$

$$V^\pi(s_2) = \sum_a \pi(a|s_2) \sum_{s',r} P(s', r|s_2, a)[r + \gamma V^\pi(s')]$$

$$V^\pi(s_3) = \sum_a \pi(a|s_3) \sum_{s',r} P(s', r|s_3, a)[r + \gamma V^\pi(s')]$$

We could solve it directly, but there's a more elegant iterative approach.

# Iterative Policy Evaluation: The Algorithm

Turn the Bellman equation into an update rule:

## Iterative Policy Evaluation

1. Initialize $V_0(s) = 0$ for all $s$ (or any arbitrary values)
2. For $k = 1, 2, 3, \ldots$:

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

   for all $s$
3. Stop when $\max_s |V_{k+1}(s) - V_k(s)| < \theta$

here we access values from previous iteration

therefore by the time we will gather the actual reward from the state on the policy.

As $k \to \infty$, $V_k \to V^\pi$

# Should we update all states at once or one at a time?

Two implementation strategies:

## Synchronous Updates (Jacobi)

- Use $V_k(s')$ for all next states
- Update all states simultaneously
- Requires two arrays: old and new values

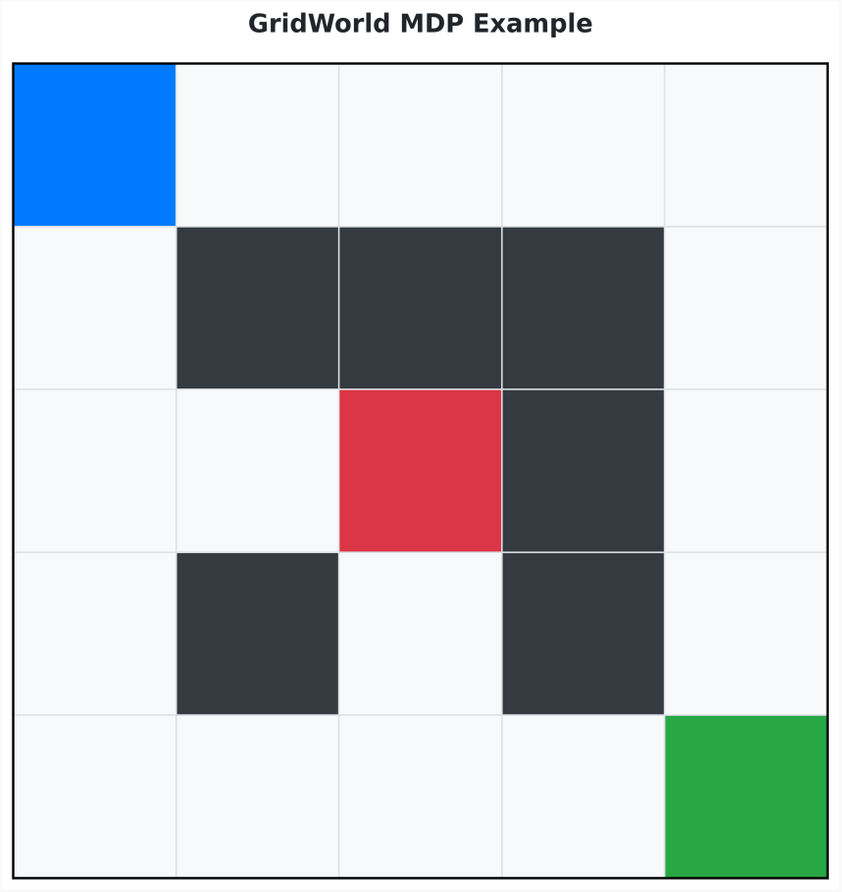
$$V_{\text{new}}(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} P(s', r|s, a)[r + \gamma V_{\text{old}}(s')]$$

## In-Place Updates (Gauss-Seidel)

- Use most recent values available
- Update states one at a time
- Can use single array, often converges faster

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

# Let's see policy evaluation in action on GridWorld



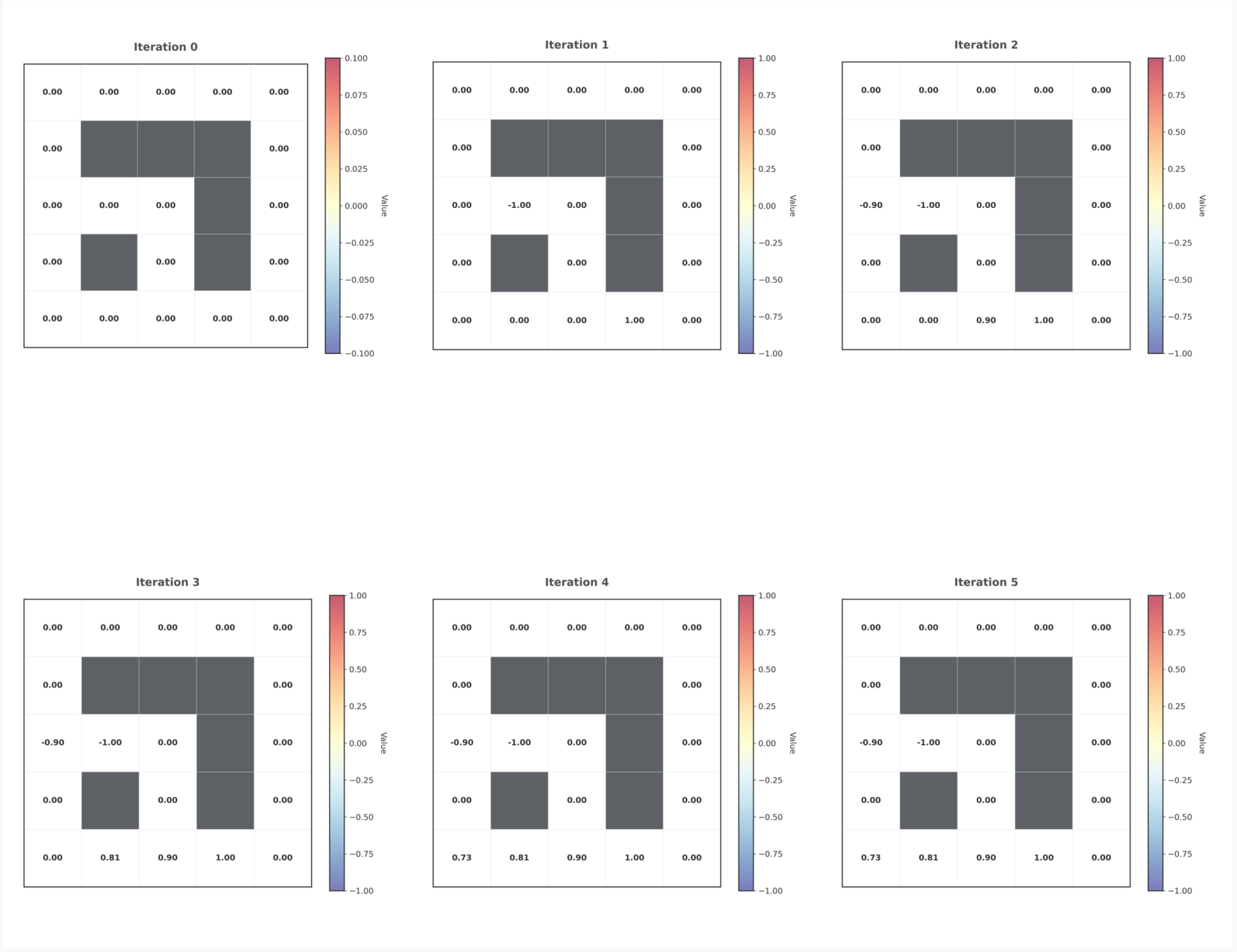

Simple $5 \times 5$ GridWorld

**MDP Setup:**

- $\mathcal{S}$: Grid positions $(i, j)$
- $\mathcal{A}$: {Up, Down, Left, Right}
- $P$: Deterministic movement
- $R$: Goal=+1, Trap=-1, Step=0.0

**Policy to Evaluate:** Always go right

# Policy Evaluation Results: Values Propagate Backwards




Policy: Always go right ($\gamma = 0.9$)

# menti.com - Code: **3814 1358**

What happens if we initialize $V_0(s) = 1000$ for all states in policy evaluation?

- A) The algorithm won't converge
- B) It will converge to the wrong values
- C) It will converge to the correct values, just from a different starting point
- D) The Bellman equation becomes invalid
- E) We must always initialize to zero

# menti.com - Code: **3814 1358**

In-place updates (Gauss-Seidel) often converge faster than synchronous updates (Jacobi) because:

- A) They use less memory
- B) They immediately use the most recent value estimates
- C) They require fewer iterations by definition
- D) They are more mathematically rigorous
- E) They work better with stochastic policies

# menti.com - Code: **3814 1358**

In the policy evaluation update
$V_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} P(s', r|s, a)[r + \gamma V_k(s')]$,
what does $V_k(s')$ represent?

- A) The immediate reward from state $s'$
- B) Our current estimate of the value of the next state
- C) The optimal value of state $s'$
- D) The probability of reaching $s'$
- E) The policy's action probability at $s'$

# 🎮 Mentimeter Poll: All clear so far?

menti.com — Code: **3814 1358**

Clarity on Policy Evaluation?

- A) All clear
- B) Iterative Policy Evaluation unclear
- C) Synchronous vs in-place updates unclear

# Policy Improvement

# Given a value function $V^\pi$, how do we make the policy better?

This is the policy improvement problem.

**Intuitive Idea**
If we know how valuable each state is under the current policy, we should choose actions that lead to more valuable states.

The key insight: Greedy Policy Improvement

$$\pi'(s) = \text{argmax}_a Q_\pi(s, a)$$

But we need to prove this actually works...

# The Policy Improvement Theorem

**Theorem (Policy Improvement Theorem)**
*Let $\pi$ and $\pi'$ be deterministic policies such that:*

$$Q_\pi(s, \pi'(s)) \geq V_\pi(s) \text{ for all } s$$

for the rest, we follow $\pi$. Therefore just the one step is different

*Then $\pi'$ is at least as good as $\pi$:*

$$V_{\pi'}(s) \geq V_\pi(s) \text{ for all } s$$

**What this means:** If taking action $\pi'(s)$ and then following $\pi$ gives at least as much value as just following $\pi$, then always following $\pi'$ is better.

# Why does the Policy Improvement Theorem work?

Proof by "unrolling" the future:

the proof only for one certain step take the action from different policy $\pi'$

$$
\begin{aligned}
V^{\pi}(s) &\leq Q_{\pi}(s, \pi'(s)) && \text{(assumption)} \\
&= \mathbb{E}[R_{t+1} + \gamma V^{\pi}(S_{t+1}) | S_t = s, A_t = \pi'(s)] && \text{(Q-function def)} \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma V^{\pi}(S_{t+1}) | S_t = s] && \text{(expectation under } \pi') \\
&\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma Q_{\pi}(S_{t+1}, \pi'(S_{t+1})) | S_t = s] && \text{(apply assumption again)} \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma \mathbb{E}[R_{t+2} + \gamma V^{\pi}(S_{t+2}) | S_{t+1}, A_{t+1} = \pi'(S_{t+1})] | S_t = s] \\
& && \text{(expand Q)} \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 V^{\pi}(S_{t+2}) | S_t = s] && \text{(combine expectations)} \\
&\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 V^{\pi}(S_{t+3}) | S_t = s] && \text{(continue pattern)} \\
&\vdots \\
&\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \ldots | S_t = s] \\
&= V_{\pi'}(s) && \text{(definition of } V_{\pi'})
\end{aligned}
$$

# Why does greedy improvement work?

**Let's verify that greedy policy improvement satisfies the theorem:**

Recall a few slides ago the greedy policy $\pi'(s) = \text{argmax}_a Q_\pi(s, a)$:

$$
\begin{aligned}
Q_\pi(s, \pi'(s)) &= Q_\pi(s, \text{argmax}_a Q_\pi(s, a)) && \text{(substitute greedy action)} \\
&= \max_a Q_\pi(s, a) && \text{(definition of argmax)} \\
&\geq Q_\pi(s, \pi(s)) && \text{(max is at least as large as any element)} \\
&= V^\pi(s) && \text{(definition of state value function)}
\end{aligned}
$$

So indeed: $Q_\pi(s, \pi'(s)) \geq V^\pi(s)$ for all $s$!

Therefore: $V_{\pi'}(s) \geq V_\pi(s)$ by the Policy Improvement Theorem!

# menti.com - Code: **3814 1358**

In the Policy Improvement Theorem proof, we repeatedly apply the assumption that $Q_\pi(s, \pi'(s)) \geq V^\pi(s)$. This "**unrolling**" works because:

- A) The greedy policy is always optimal
- B) We can apply the same logic at each future state
- C) The Markov property ensures independence
- D) The discount factor is less than 1
- E) We only consider deterministic policies

# 🎮 Mentimeter Poll: All clear so far?

menti.com — Code: **3814 1358**

Clarity on Policy Improvement?

- A) All clear
- B) Greedy policy improvement unclear
- C) Policy Improvement Theorem proof unclear

# Policy Iteration

# What if we alternate between evaluation and improvement?

## Policy Iteration: Combine evaluation and improvement

## Policy Iteration Algorithm

1. **Initialize**: Arbitrary policy $\pi_0$
2. **Repeat until convergence**:
   - **Policy Evaluation**: Compute $V_k^\pi$ using iterative evaluation
   - **Policy Improvement**: Set $\pi_{k+1}(s) = \text{argmax}_a Q_{\pi_k}(s, a)$
3. **Stop when**: $\pi_{k+1} = \pi_k$ (policy is stable)

$$\pi_0 \xrightarrow{\text{Eval}} V_0^{\pi_0} \xrightarrow{\text{Improve}} \pi_1 \xrightarrow{\text{Eval}} V_1^{\pi_1} \xrightarrow{\text{Improve}} \cdots \rightarrow \pi^*$$

# Why does Policy Iteration find the optimal policy?

Two key properties guarantee optimality:

**Property 1: Monotonic Improvement**
Each iteration either strictly improves the policy or the policy is already optimal:

$$V_{k+1}^{\pi}(s) \geq V_k^{\pi}(s) \text{ for all } s$$

**Property 2: Finite Convergence**

- Finite number of possible policies: $|\mathcal{A}|^{|\mathcal{S}|}$
- Each improvement is strict (until optimality)
- Must reach optimal policy in finite steps

Result: Policy Iteration always finds $\pi^*$ in finite time!

# menti.com - Code: **3814 1358**

Policy Iteration is guaranteed to converge in finite steps because:

- A) The Bellman equation is linear
- B) There are finitely many policies and each improvement is monotonically better
- C) The discount factor is less than 1
- D) We use synchronous updates
- E) The MDP is episodic

# 🎮 Mentimeter Poll: All clear so far?

menti.com — Code: **3814 1358**

Clarity on Policy Iteration?

- A) All clear
- B) Why we alternate evaluation and improvement unclear
- C) Why it finds optimal policy unclear
- D) Difference from policy evaluation unclear

# Value Iteration

# Why do full policy evaluation if we're going to change the policy anyway?

**The inefficiency in Policy Iteration:**

**What Policy Iteration Does**

1. **Policy Evaluation**: Run iterative evaluation until convergence
2. **Policy Improvement**: Update policy greedily
3. **Repeat**: Go back to step 1 with new policy

**The key insight:** Why wait for full convergence in step 1?

# What if we don’t wait for full convergence?

## The Computational Waste

- Policy evaluation might take 100+ iterations to converge
- We immediately throw away this policy after improvement
- Early iterations of evaluation give us the ”general shape” of values
- Perfect accuracy isn’t needed if we’re changing the policy anyway

## The Natural Question

What’s the extreme case? What if we do just **one step** of policy evaluation before improving?

## This leads us directly to Value Iteration!

# From Policy Iteration to Value Iteration

## The Spectrum of Truncation

- **Policy Iteration**: Complete evaluation + improvement
- **Truncated Policy Iteration**: Partial evaluation + improvement
- **Value Iteration**: One-step evaluation + improvement

**One-step policy evaluation + greedy improvement gives us:**

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

for all $s$

# The Value Iteration Algorithm

## Value Iteration

1. Initialize $V_0(s) = 0$ for all $s$
2. For $k = 1, 2, 3, \ldots$:

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

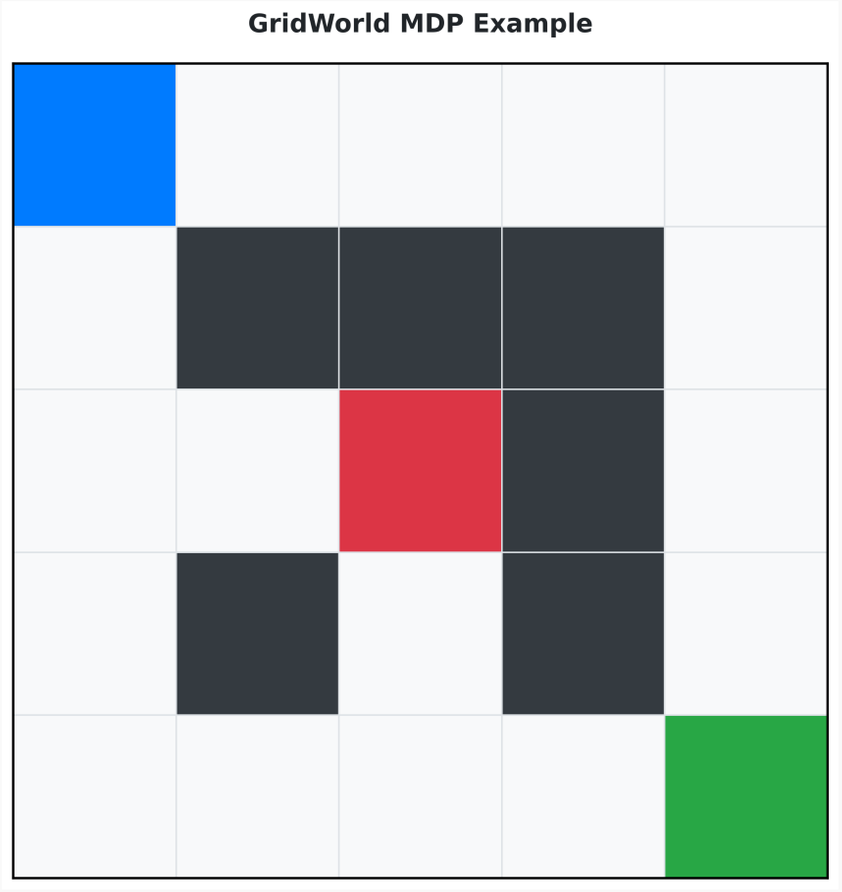
   for all $s$
3. Stop when $\max_s |V_{k+1}(s) - V_k(s)| < \theta$
4. Extract policy: $\pi^*(s) = \text{argmax}_a \sum_{s',r} P(s', r|s, a)[r + \gamma V^*(s')]$

**Key difference from Policy Evaluation:** The $\max$ operator instead of weighted sum over policy

# The Moment You've Been Waiting For...

Learning our first optimal policy and value function in GridWorld!

# Reminder of GridWorld Setup

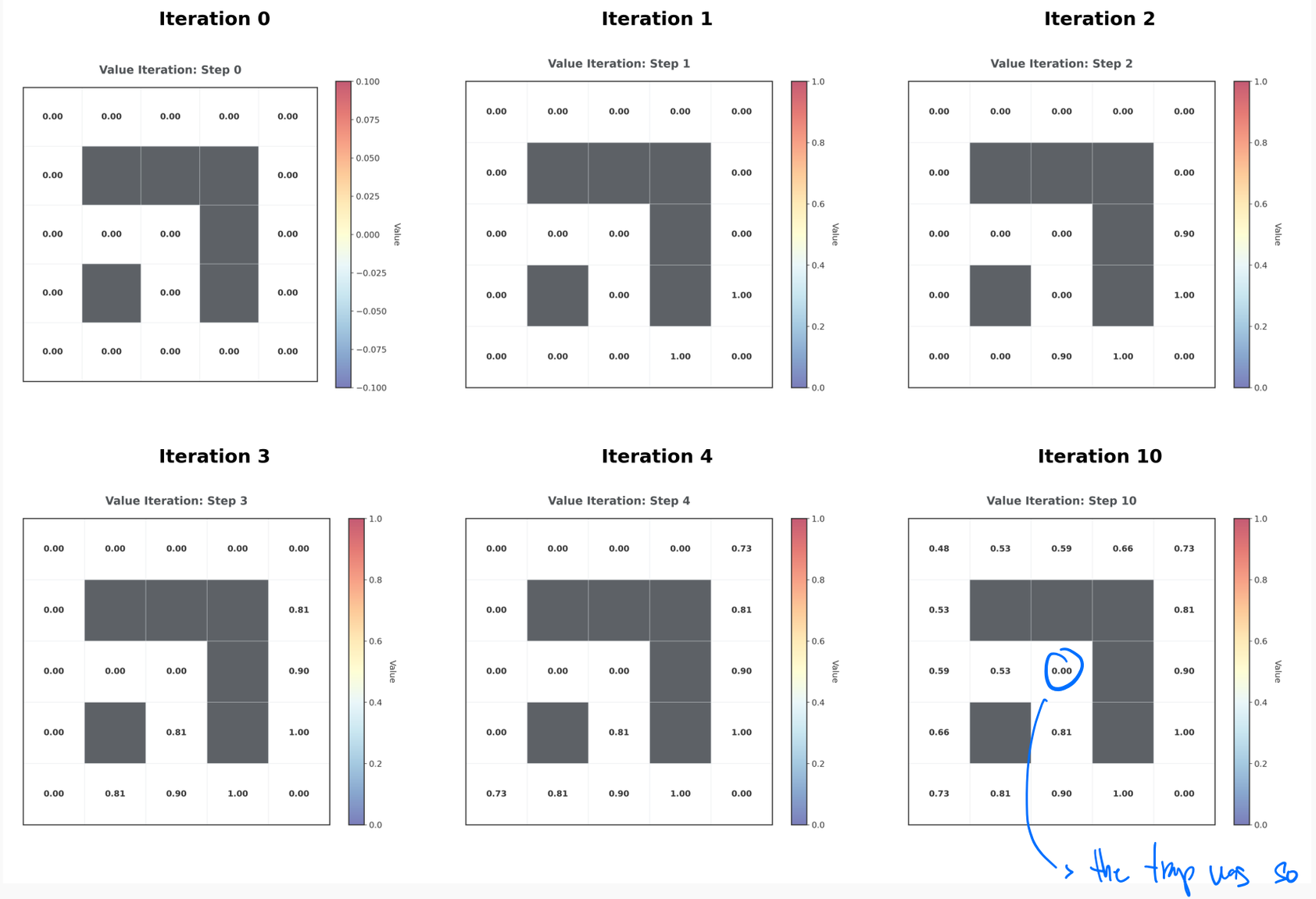



Simple $5 \times 5$ GridWorld

**MDP Setup:**

- $\mathcal{S}$: Grid positions $(i, j)$
- $\mathcal{A}$: {Up, Down, Left, Right}
- $P$: Deterministic movement
- $R$: Goal=+1, Trap=-1, Step=0.0

# Value Iteration in Action: GridWorld



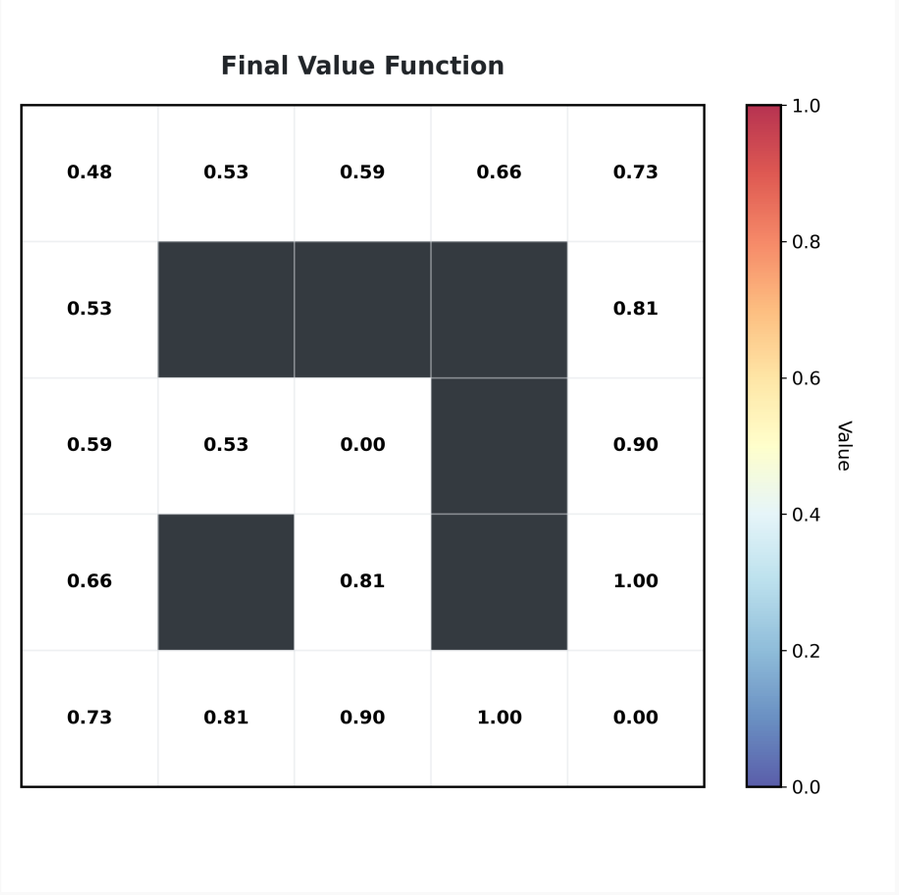

→ the trap was so bad that we haven't even stepped into it.

Key Observations:

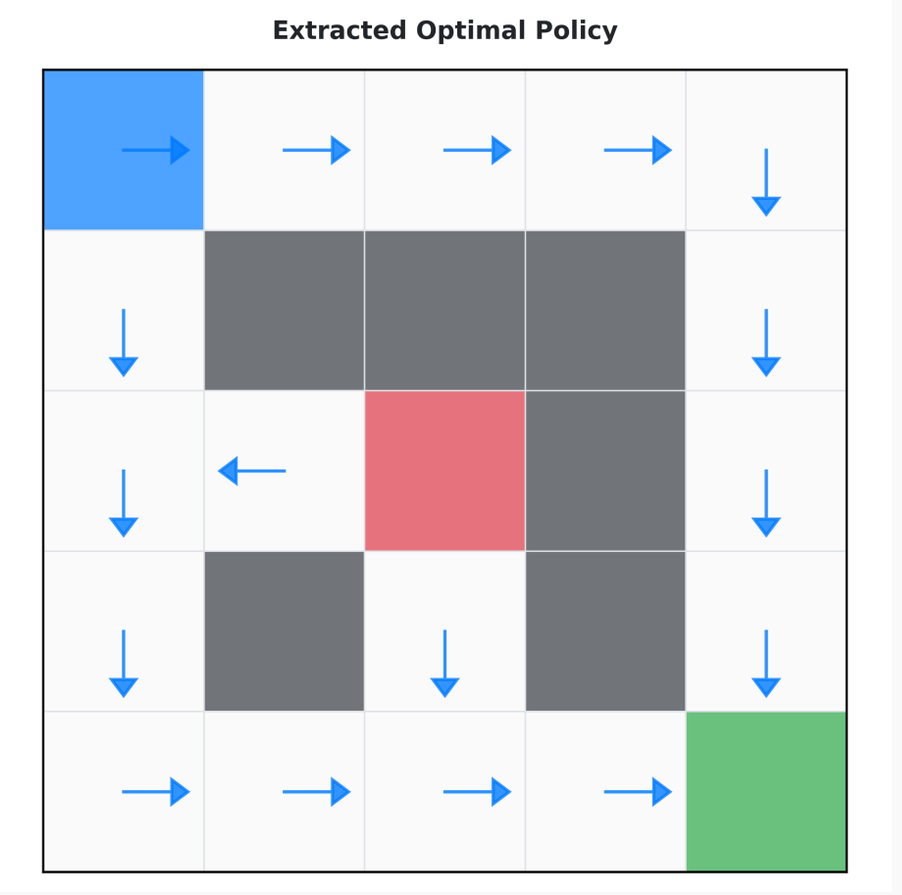
- Values propagate faster than policy evaluation
- Each state chooses its best action at each step

# Value Iteration Results: Optimal Values and Policy

## Optimal Value Function $V^*$




## Extracted Optimal Policy $\pi^*$




Value iteration finds both $V^*$ and $\pi^*$ in one algorithm!

# menti.com - Code: **3814 1358**

The max operator in Value Iteration directly implements:

- A) Policy evaluation
- B) Policy improvement
- C) Both evaluation and improvement simultaneously
- D) The Bellman expectation equation
- E) Random action selection

that is indeed
why it is that great

# 🎮 Mentimeter Poll: All clear so far?

menti.com — Code: **3814 1358**

Clarity on Value Iteration?

- A) All clear
- B) Why one-step evaluation works unclear
- C) Difference from Policy Iteration unclear
- D) How to extract the policy unclear

# Generalized Policy Iteration

# What's the big picture pattern behind these algorithms?

## Generalized Policy Iteration (GPI)

**Definition (Generalized Policy Iteration)**
The general idea of letting policy evaluation and policy improvement processes interact, independent of the granularity and other details.

## The GPI Pattern

- **Evaluation**: Make value function consistent with current policy
- **Improvement**: Make policy greedy with respect to current value function
- These processes work toward the same goal from different directions

Almost all RL algorithms follow the GPI pattern!

# Examples of GPI in Action

We've already seen GPI in different forms:

GPI Spectrum

- **Policy Iteration**: Complete evaluation, then improvement
- **Value Iteration**: One-step evaluation, then improvement
- **Asynchronous DP**: Update states individually, any order

Future algorithms also follow GPI:

- **Q-Learning**: Implicit evaluation via Q-updates, $\epsilon$-greedy improvement
- **Actor-Critic**: Continuous evaluation and improvement
- **Policy Gradient**: Direct policy improvement with value estimation

# Can we be more flexible about which states to update?

## Asynchronous Dynamic Programming

**Key Insight**
We don't need to update all states in each iteration - we can focus computation where it matters most.

**Asynchronous Strategies:**

- **Random**: Pick states uniformly at random
- **Prioritized**: Update states with largest value changes first
- **Real-time**: Update states along simulated trajectories

**Convergence guarantee:** Still works if all states updated infinitely often

# Why can't we just use Dynamic Programming for everything?

## Three Fundamental Limitations

1. **Perfect Model Required**: Need exact $P(s'|s, a)$ and $R(s, a, s')$
2. **Computational Complexity**: Must iterate over all states and actions
3. **Curse of Dimensionality**: State space grows exponentially as $|S|$ and $|A|$ increases

## Real-world challenges:

- Robotics: Need perfect physics simulation
- Finance: Need perfect market model
- Games: Environment rules might be unknown

# The Curse of Dimensionality: Why DP Doesn't Scale

State space explosion:

Examples

- **Backgammon**: $\approx 10^{20}$ states
- **Go**: $\approx 10^{170}$ states
- **Robot (10 joints)**: $\approx 10^{30}$ states
- **Atari games**: $256^{210 \times 160} \approx 10^{1,000,000}$ states

We need methods that don't require storing values for every state!

# How do modern RL methods overcome these limitations?

**The path forward:**

**Model-Free Methods**
Learn directly from experience without knowing $P$ or $R$

- Monte Carlo methods
- Temporal Difference learning (Q-learning)

**Function Approximation**
Use neural networks instead of tables

**The math stays the same - only the representation changes!**

# menti.com - Code: **3814 1358**

What is the key relationship in Generalized Policy Iteration?

- A) Evaluation and improvement compete against each other
- B) Evaluation and improvement work toward the same goal from different directions
- C) Evaluation must complete before improvement begins
- D) Improvement is always faster than evaluation
- E) Both processes must be synchronous

# 🎮 Mentimeter Poll: All clear so far?

menti.com — Code: **3814 1358**

Clarity on GPI and limitations of DP?

- A) All clear
- B) GPI concept unclear
- C) Why DP doesn't scale unclear
- D) Curse of dimensionality unclear

# What have we learned today?

Key Takeaways

- **Dynamic Programming**: Exact algorithms for solving MDPs
- **Policy Evaluation**: Compute value functions for given policies
- **Policy Improvement**: Make policies better using value functions
- **Policy Iteration**: Alternates evaluation and improvement
- **Value Iteration**: Finds optimal values directly using one-step updates
- **GPI**: Unifies the evaluation-improvement pattern

**Next week:** Model-Free Methods How to learn optimal policies without knowing the environment model

# Reading and Resources

## 📚 Essential Reading

- **Sutton & Barto Chapter 4**: Dynamic Programming
- **Focus on**: Policy Evaluation (4.1), Policy Improvement (4.2)
- **Core algorithms**: Policy Iteration (4.3), Value Iteration (4.4)
- **Extensions**: Asynchronous DP (4.5), GPI (4.6)

# menti.com - Code: **3814 1358**

Q1: If you got lost today, at what point did you lose track?

- A) Policy Evaluation
- B) Policy Improvement
- C) Policy Iteration
- D) Value Iteration
- E) Generalized Policy Iteration
- F) None of the above, I followed everything

Q2: Generic feedback - what worked well, what could be improved?

- e.g. ”I found the section on policies particularly helpful.”
- e.g. ”Pace was too fast.”
- e.g. ”Not enough examples.”