# Reinforcement Learning

## Dynamic Programming


Jordi Casas Roma

**jordi.casas.roma@uab.cat**


September 12, 2025


Universitat Autònoma de Barcelona

# Table of Contents

**Preliminary concepts**:

- **Dynamic Programming** (**DP**) is a collection of algorithms that search for optimal policies based on perfect knowledge of the environment model (Markov Decision Process, MDP).
- These algorithms have a high computational cost, but they are very important and useful from a theoretical point of view.
- DP provides the essential foundation for understanding the methods that will be presented later.
  - We can state that all the methods that we will see in this course are an attempt to achieve the same performance as DP with a lower computational load and without assuming perfect knowledge of the environment.

**Preliminary concepts**:

- The main idea of **DP methods** is to use the value function to organize and structure the search for good policies.
- We can obtain optimal policies once we have found the optimal value functions, $v_*(s)$ or $q_*(s, a)$, that satisfy the Bellman optimality equation:

**Preliminary concepts**:

- The main idea of **DP methods** is to use the value function to organize and structure the search for good policies.
- We can obtain optimal policies once we have found the optimal value functions, $v_*(s)$ or $q_*(s,a)$, that satisfy the Bellman optimality equation:

$$
\begin{aligned}
v_*(s) &= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a)] \\
&= \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_*(s')]
\end{aligned} \tag{1}
$$

**Preliminary concepts**:

- The main idea of **DP methods** is to use the value function to organize and structure the search for good policies.
- We can obtain optimal policies once we have found the optimal value functions, $v_*(s)$ or $q_*(s, a)$, that satisfy the Bellman optimality equation:

$$\begin{aligned} v_*(s) &= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a)] \\ &= \max_a \sum_{s', r} p(s', r | s, a)[r + \gamma v_*(s')] \end{aligned} \tag{1}$$

$$\begin{aligned} q_*(s, a) &= \mathbb{E}[R_{t+1} + \gamma \max_a q_*(S_{t+1}, a') | S_t = s, A_t = a)] \\ &= \sum_{s', r} p(s', r | s, a)[r + \gamma \max_{a'} q_*(s', a')] \end{aligned} \tag{2}$$

# Table of Contents

# Policy Evaluation
## What is Policy Evaluation?

**What is "Policy Evaluation"?**

### Policy Evaluation

Policy evaluation focuses on how to calculate the **value function** of a state $v_\pi(s)$ for an arbitrary policy $\pi$.

**What is "Policy Evaluation"?**

Actually, we want to implement the following equation:

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[G_t|S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s] \\ &= \sum_a \pi(a, s) \sum_{s', r} p(s', r|s, a)[r + \gamma v_\pi(s')] \end{aligned} \tag{3}$$

- where $\pi(a, s)$ is the probability of taking an action $a$ in a state $s$ under policy $\pi$.

# Policy Evaluation

## What is Policy Evaluation?

**What is “Policy Evaluation”?**

### Policy Evaluation

If the **dynamics of the environment are completely known**, equation (3) is a system of $|S|$ simultaneous linear equations in $|S|$ with variables $v_\pi(s), s \in S$.

Therefore, its solution is simple, although the computing time can be high.

# Policy Evaluation
## Iterative Policy Evaluation

**Iterative Policy Evaluation**:

- Iterative method to find the optimal policy.

The **general scheme** of an iterative method is:

1. We consider a sequence of approximate value functions $v_0, v_1, v_2, \ldots$, where each one maps $S$ to $\mathbb{R}$ (real numbers).
2. The initial approximation, $v_0$, is chosen arbitrarily.
3. And each successive approximation is obtained using the Bellman equation for $v_\pi$ (equation 3) as the update rule.
4. The sequence $v_k$ converges to $v_\pi$ with $k \to \infty$, under the same conditions that guarantee the existence of $v_\pi$.

# Policy Evaluation

## Iterative Policy Evaluation

**Iterative Policy Evaluation**:

**Algorithm 1** Pseudocode of Iterative Policy Evaluation method ($V \approx v_\pi$)

**Require:** Policy ($\pi$)
**Require:** Threshold ($\theta$)

```
Initialize V(s) ∀s ∈ S randomly, except V(target) = 0
while Δ > θ do
    Δ ← 0
    for all s ∈ S do
        v ← V(s)
        V(s) ← Σ_a π(a|s) Σ_{s',r} p(s',r|s,a)[r + γV(s')]
        Δ ← max(Δ, |v − V(s)|)
    end for
end while
return V
```

# Policy Evaluation

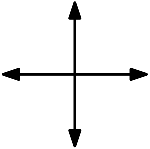
## Example: 4x4 grid

- Start position: random $\in \{1, 2, \dots, 14\}$
- The reward is $-1$ for all transitions.
- The terminal states are those in gray.

$$R_t = -1$$

(all transitions)

| | | | |
|---|---|---|---|
| | 1 | 2 | 3 |
| 4 | 5 | 6 | 7 |
| 8 | 9 | 10 | 11 |
| 12 | 13 | 14 | |



Actions

# Policy Evaluation
## Example: 4x4 grid

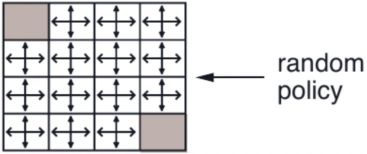
Iteration: $K = 0$

$v_k$ for the Random Policy

$k = 0$

| | | | |
|---|---|---|---|
| 0.0 | 0.0 | 0.0 | 0.0 |
| 0.0 | 0.0 | 0.0 | 0.0 |
| 0.0 | 0.0 | 0.0 | 0.0 |
| 0.0 | 0.0 | 0.0 | 0.0 |

Greedy Policy w.r.t. $v_k$



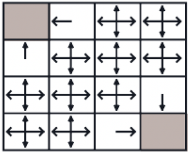

Sutton & Barto, 2018

# Policy Evaluation
## Example: 4x4 grid

Iteration: $K = 1$

$k = 1$

| | | | |
|---|---|---|---|
| 0.0 | -1.0 | -1.0 | -1.0 |
| -1.0 | -1.0 | -1.0 | -1.0 |
| -1.0 | -1.0 | -1.0 | -1.0 |
| -1.0 | -1.0 | -1.0 | 0.0 |



Sutton & Barto, 2018

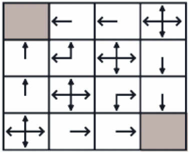
# Policy Evaluation

## Example: 4x4 grid

Iteration: $K = 2$

$k = 2$

| | | | |
|---|---|---|---|
| 0.0 | -1.7 | -2.0 | -2.0 |
| -1.7 | -2.0 | -2.0 | -2.0 |
| -2.0 | -2.0 | -2.0 | -1.7 |
| -2.0 | -2.0 | -1.7 | 0.0 |



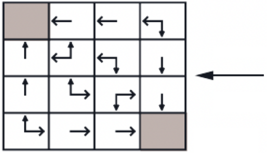
Sutton & Barto, 2018

# Policy Evaluation
## Example: 4x4 grid

Iteration: $K = 3$

$k = 3$

| | | | |
|---|---|---|---|
| 0.0 | -2.4 | -2.9 | -3.0 |
| -2.4 | -2.9 | -3.0 | -2.9 |
| -2.9 | -3.0 | -2.9 | -2.4 |
| -3.0 | -2.9 | -2.4 | 0.0 |



optimal policy

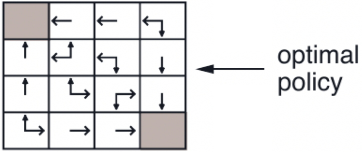
Sutton & Barto, 2018

# Policy Evaluation
## Example: 4x4 grid

Iteration: $K = 10$

$k = 10$

| | | | |
|---|---|---|---|
| 0.0 | -6.1 | -8.4 | -9.0 |
| -6.1 | -7.7 | -8.4 | -8.4 |
| -8.4 | -8.4 | -7.7 | -6.1 |
| -9.0 | -8.4 | -6.1 | 0.0 |




Sutton & Barto, 2018

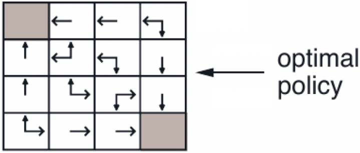
# Policy Evaluation

## Example: 4x4 grid

Iteration: $K = \infty$

$k = \infty$

| | | | |
|---|---|---|---|
| 0.0 | -14. | -20. | -22. |
| -14. | -18. | -20. | -20. |
| -20. | -20. | -18. | -14. |
| -22. | -20. | -14. | 0.0 |




Sutton & Barto, 2018

# Table of Contents

# Policy Improvement
## What is Policy Improvement?

**What is "Policy Improvement"?**

### Policy Improvement

Policy improvement focuses on how to **improve an arbitrary policy** $\pi$ through get better $q_\pi(s, a)$ for some $s \in S$, and producing a new better policy $\pi'$.

**What is "Policy Improvement"?**

Let $\pi$ and $\pi'$ be any two deterministic policies.

If $\forall s \in S$ it turns out that:

$$q_\pi(s, \pi'(s)) \geq v_\pi(s) \tag{4}$$

- If applying policy $\pi'$ in state $s$ obtains a better reward than policy $\pi$, then modify policy $\pi$ so that it executes action $q_\pi(s, \pi'(s))$ in state $s$ implies an improvement of policy $\pi$.

**What is "Policy Improvement"?**

Let $\pi$ and $\pi'$ be any two deterministic policies.

If $\forall s \in S$ it turns out that:

$$q_\pi(s, \pi'(s)) \geq v_\pi(s) \tag{4}$$

- If applying policy $\pi'$ in state $s$ obtains a better reward than policy $\pi$, then modify policy $\pi$ so that it executes action $q_\pi(s, \pi'(s))$ in state $s$ implies an improvement of policy $\pi$.

Then policy $\pi'$ must be as good or better than $\pi$. Or, another way, $\pi'$ should get equal or larger expected returns in all states $s \in S$:

$$v_{\pi'}(s) \geq v_\pi(s) \tag{5}$$

- $\pi$ and $\pi'$ are identical, except that $\pi'(s) = a \neq \pi(s)$.

**What is "Policy Improvement"?**

- So far we have seen how, given a policy and its value function, we can easily evaluate a change in policy in a given state with a particular action.
- The natural extension is to consider changes in all states and in all possible actions, and selecting in each state the action that seems best based on $q_\pi(s, a)$.
- In other words, consider the **new policy greedy** $\pi'$ **determined by**:

$$
\begin{aligned}
\pi'(s) &= \arg\max_a q_\pi(s, a) \\
&= \arg\max_a \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = a] \\
&= \arg\max_a \sum_{s', r} p(s', r | s, a)[r + \gamma v_\pi(s')],
\end{aligned} \tag{6}
$$

**Policy Improvement**

- The policy *greedy* takes the action that seems best in the near future (after a step forward) based on $v_\pi$.
- The process of making a new policy that progressively improves the original policy is called **policy improvement**.
- Policy improvement must strictly give us a better policy, except when the original policy is optimal.

# Policy Evaluation

## What is Policy Improvement?

**Policy Improvement in stochastic environments**

- We have discussed the special case of **deterministic policies**.
- In the general case, a **stochastic policy** $\pi$ specifies the probabilities $\pi(a|s)$ of choosing each action $a$ in each state $s$. We can conclude that all the ideas in this section easily extend to stochastic policies.
- The *policy improvement* theorem is formulated exactly the same in the case of stochastic policies.
- Also, if we have multiple actions that maximize the expected value, then we do not need to select a single action.
  - Each action that maximizes the reward value can be a portion of the probability of being selected in a new *greedy* policy.

# Table of Contents

**What is "Policy Iteration"?**

### Policy Iteration

**Policy iteration** is the process of generating a new policy $\pi'$ that improves the original policy $\pi$, iteratively. That is, the process continues to generate a second improved policy, $\pi''$, from policy $\pi'$, and so on.

We will thus obtain a sequence of policies and value functions that improve monotonically:

$$\pi_0 \xrightarrow{E} v_{\pi 0} \xrightarrow{I} \pi_1 \xrightarrow{E} v_{\pi 1} \xrightarrow{I} \pi_2 \xrightarrow{E} \ldots \xrightarrow{I} \pi_* \xrightarrow{E} v_{\pi_+} \tag{7}$$

- where $\xrightarrow{E}$ indicates an **evaluation** of the policy and $\xrightarrow{I}$ indicates an **improvement** of the policy.

# Policy Iteration

## Pseudocode of Policy Iteration

**Algorithm 2** Policy Iteration to estimate $\pi \approx \pi^*$

**Require:** Threshold ($\theta$); Random values: $V(s) \in \mathbb{R}$ and $\pi(s) \in A(s)$, $\forall s \in S$

Step 1) Policy Evaluation<br>
**while** $\Delta > \theta$ **do**<br>
    $\Delta \leftarrow 0$<br>
    **for all** $s \in S$ **do**<br>
        $v \leftarrow V(s)$<br>
        $V(S_t) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))[r + \gamma V(s')]$<br>
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$<br>
    **end for**<br>
**end while**<br>
Step 2) Policy Improvement<br>
*stable-policy* $\leftarrow$ *true*<br>
**for all** $s \in S$ **do**<br>
    *old-action* $\leftarrow \pi(s)$<br>
    $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$<br>
    **if** *old-action* $\neq \pi(s)$ **then**<br>
        *stable-policy* $\leftarrow$ *false*<br>
    **end if**<br>
**end for**<br>
**if** *stable-policy* **then**<br>
    **return** $V \approx v_*$ y $\pi \approx \pi_*$<br>
**else**<br>
    GOTO Step 1)<br>
**end if**

# Table of Contents

**What is "Value Iteration"?**

## Value Iteration

**Value Iteration** is an algorithm, based on policy iteration, that runs for one cycle.

We will understand by "cycle" an update for each state of the environment, i.e. $\forall s \in S$.

$$\begin{aligned} v_{k+1}(s) &= \max_a \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) | S_t = s, A_t = a] \\ &= \max_a \sum_{s', r} p(s', r | s, a)[r + \gamma v_k(s')] \end{aligned} \tag{8}$$

# Value Iteration

## Pseudocode of Value Iteration

**Algorithm 3** Value Iteration to estimate $\pi \approx \pi_*$

**Require:** Threshold $(\theta > 0)$

Initialize $V(s)\ \forall s \in S^+$ randomly, except $V(\textit{terminal}) = 0$

**while** $\Delta > \theta$ **do**

  $\Delta \leftarrow 0$

  **for all** $s \in S$ **do**

    $v \leftarrow V(s)$

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

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

  **end for**

**end while**

**return** Deterministic policy $\pi \approx \pi_*$,
where $\pi(s) = \arg\max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$

# Table of Contents

**What is "Generalized Policy Iteration"?**

- **Policy iteration** consists of two simultaneous processes that interact with each other:
  - one makes the value function consistent with the current policy (policy evaluation),
  - the other executes the policy *greedy* with respect to the current value of the value function (policy improvement).

### Generalized Policy Iteration

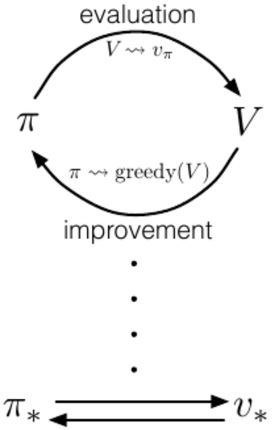
We use the term **generalized policy iteration** to refer to the general idea of interacting policy evaluation and policy improvement methods, regardless of granularity and of other details of the two processes.

**What is "Generalized Policy Iteration"?**

- We can affirm that almost all reinforcement learning processes can be described with the term "**generalized policy iteration**".
- That is, they all have identifiable policies and value functions, in which the policy is always improved with respect to the value functions, and the value function is always guided towards the value function for that policy.

# Generalized Policy Iteration
## What is Generalized Policy Iteration?

**What is "Generalized Policy Iteration"?**



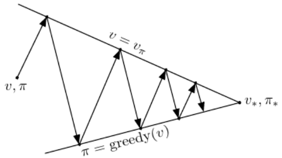

Sutton & Barto, 2018

# Generalized Policy Iteration

## What is Generalized Policy Iteration?

**What is “Generalized Policy Iteration”?**

- Evaluation and improvement processes can be seen not only as cooperative but also competitive processes.
- They compete because they are both “pushing” in opposite directions:
  - The policy *greedy* causes the value function to be incorrect for the new policy.
  - On the other hand, making the value function consistent with the policy typically causes the policy to no longer be *greedy*.
- In the long run, however, these two processes interact to find a unique joint solution.




Sutton & Barto, 2018

# Table of Contents

# Biliography
## References

Some relevant references:


1. **R. S. Sutton, A. G. Barto**. (2018). *Reinforcement Learning: An Introduction (Second edition)*. MIT Press, Cambridge, MA.
2. **M. Lapan**. (2024). *Deep Reinforcement Learning Hands-On (Third Edition)*. Packt Publishing.
3. **Dimitri P. Bertsekas** (2017). *Dynamic Programming and optimal Control, Vol. I (4th edition)*. Athena Scientific. ISBN: 1-886529-43-4.
4. **Dimitri P. Bertsekas** (2012). *Dynamic Programming and Optimal Control, Vol. II: Approximate Dynamic Programming (4th edition)*. Athena Scientific. ISBN: 1-886529-44-2.