Dynamic Programming (DP)

Mohammad **Dehghani**

Mechanical and Industrial Engineering Department

Northeastern University

Feb-20

Introduction

- ## Markov Decision Process
  - Captures these two aspects of real-world problems:
    - Involves an associative aspect—choosing different actions in different situations
    - Actions influence not just immediate rewards, but also subsequent situations

- ## MDP Solvers based on:
  - the MDP model
  - the use if approximation methods

		Exact Representation	
		Yes	No
Known Model	Yes	Dynamic Programming ❶	Approximate DP ❷
	No	Reinforcement Learning ❸	RL with Function Approximation ❹

Introduction

What is Dynamic Programming?

- ***Dynamic*** sequential or temporal component to the problem
- ***Programming*** optimizing a “program”, i.e. a policy
  - c.f. linear programming

DP is a method for solving complex problems, by breaking them down into a series of overlapping sub-problems

- Solve the subproblems
- Combine solutions to subproblems

DP uses the *memorization* technique.

- The technique of storing solutions to subproblems instead of recomputing them is called memorization.

Richard Bellman

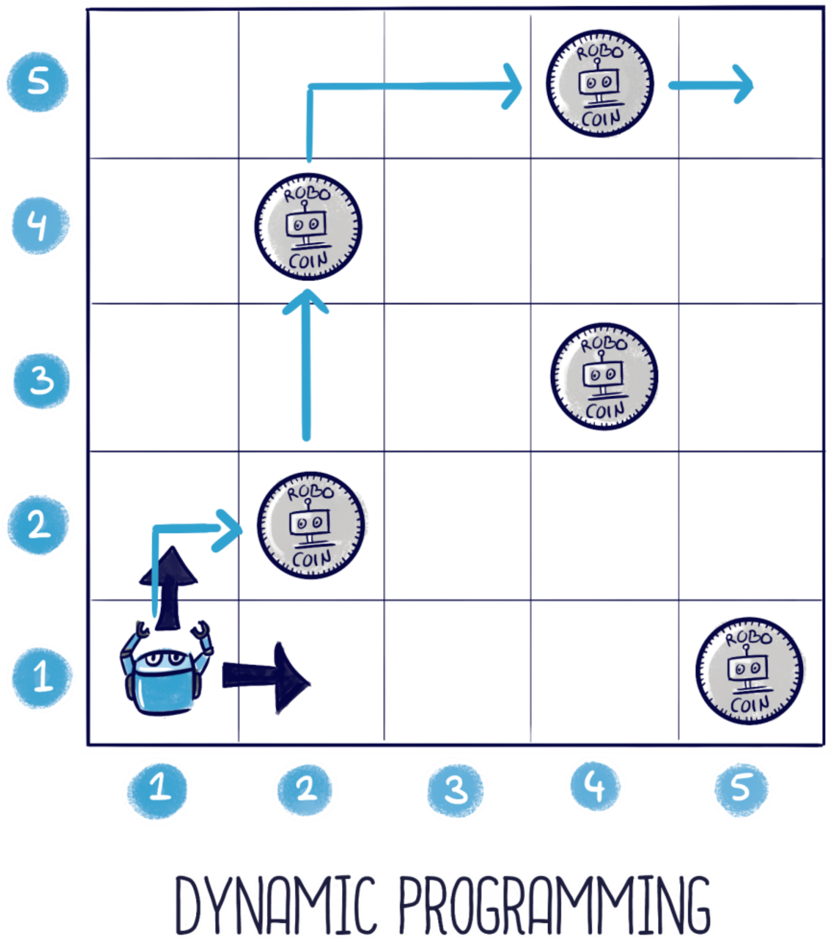
1920-1984

American Applied Mathematician

- Introduced ***Dynamic Programming (DP)*** as a method for solving a complex problem by breaking it down into a collection of simpler subproblems, solving each of those subproblems just once, and storing their solutions.

- Bellman also introduced the ***curse of dimensionality*** which is an expression coined by Bellman to describe the problem caused by the exponential increase in volume associated with adding extra dimensions to a space.

Dynamic Programming



Dynamic Programing Structure

Dynamic Programming is a suitable method for problems with:

1. **Optimal substructure**
   - Principle of optimality applies
   - Optimal solution can be decomposed into subproblems

2. **Overlapping subproblems**
   - Subproblems recur many times
   - Solutions can be cached and reused

- **Markov decision processes satisfy both properties**
  - Bellman equation gives recursive decomposition
  - Value function stores and reuses solutions

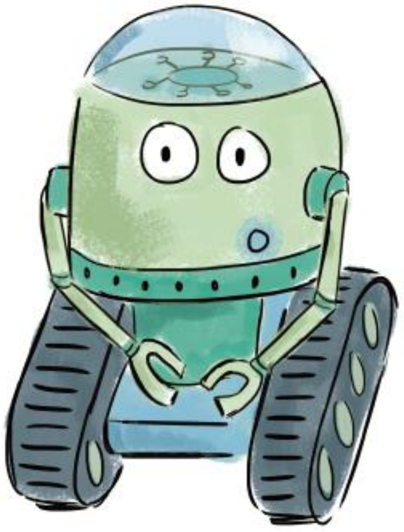
Outline

- **The goal of this lecture is to develop new ways of calculating an optimal policy**

  1. **Policy evaluation**
     - calculating value function for an arbitrary policy
  2. **Policy Improvement**
     - obtain the new policy based on the new selected action
  3. **Policy Iteration**
     - calculating an optimal value function (and policy) by iteratively calculating value function and then improving policy

- **Discuss efficiency and utility of DP**

Policy Evaluation
vs.
Control



Policy Evaluation vs. Control

- **Policy evaluation**
  - The task of determining the value function for a for an arbitrary policy π
- **Control:**
  - The task of finding a policy to obtain as much reward as possible.

	State Values	Action Values
Evaluation	v_π	q_π
Control	v_*	q_*

- **The ultimate goal of is finding a policy which maximizes the value function (control)**
- ***Policy Evaluation* is a medium to reach *Control***

Policy Evaluation vs. Control

Policy evaluation

$$\pi \rightarrow v_\pi$$

state-value function for policy π

According to policy π:

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s\right], \qquad \text{for all } s \in \mathcal{S}$$

- **Bellman equation reduces the problem of finding state-values $v_\pi(s)$ to a system of linear equations**

Bellman equation

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')], \text{ for all } s \in \mathcal{S}$$

Policy Evaluation vs. Control

Policy evaluation

$$\pi \rightarrow v_\pi$$

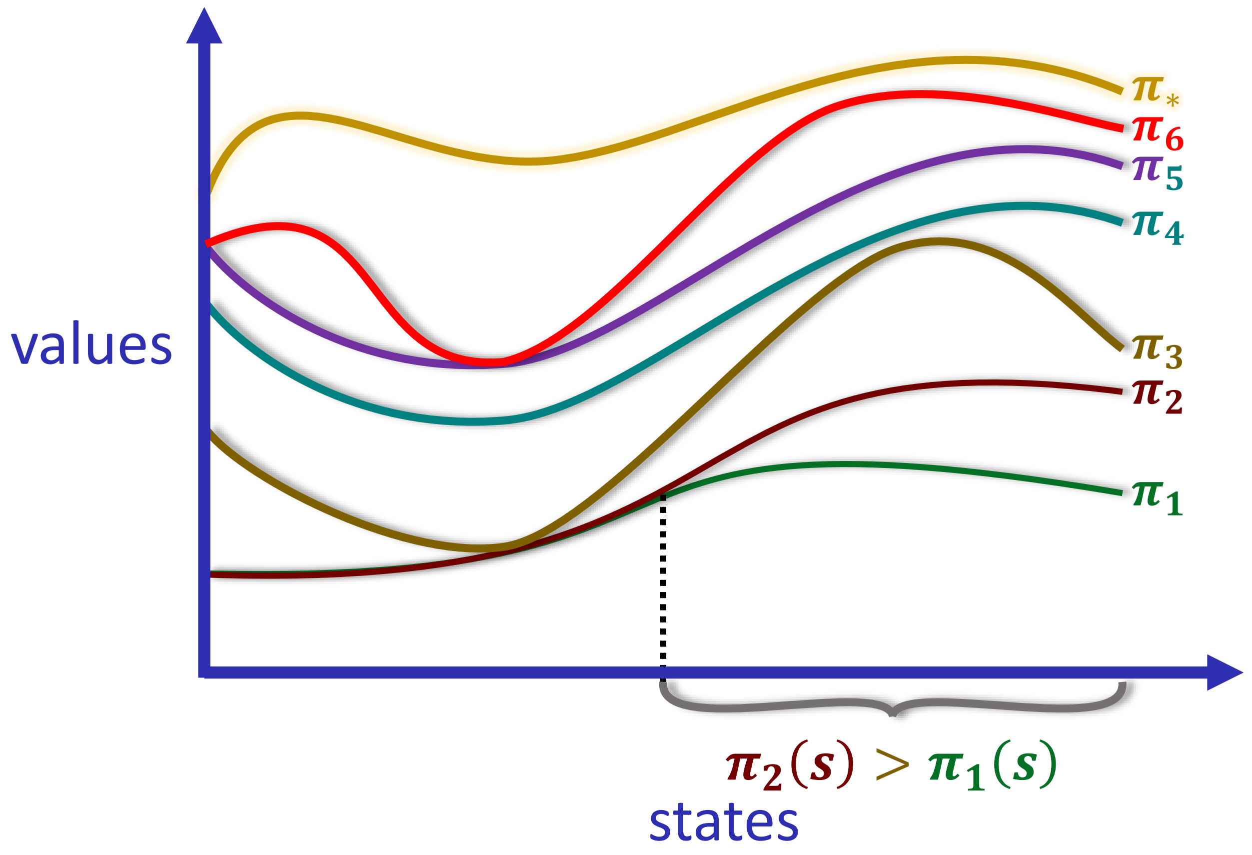
Simple MDPs: Linear Programing

π, p, γ → Linear System Solver → v_π

General MDP: Dynamic Programing

π, p, γ → Dynamic Programming → v_π

- **Control** corresponds to the operation of RL's policy-improvement algorithms.



π_2 strictly better than π_1

Policy Evaluation vs. Control

Dynamic Programming Requirements: MDP model

- **Dynamic programming assumes full knowledge of the MDP**
- **It is used for planning in an MDP**

- For ***Prediction***:
  - Input: MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \gamma \rangle$ and policy π
  - Output: value function $\boldsymbol{v_\pi}$

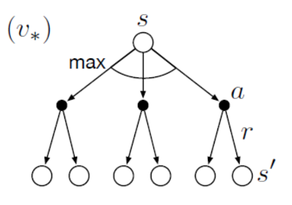
- For ***Control***:
  - Input: MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \gamma \rangle$
  - Output:
    - Optimal value function $\boldsymbol{v}_*$
    - Optimal policy $\boldsymbol{\pi}_*$

- The key idea of DP, and of RL generally, is the use of ***value functions*** to organize and structure the search for good policies.

Definition

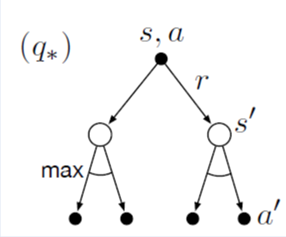
- **The *value of a state* under an optimal policy must equal the expected return for the *best* action from that state**

$$v_*(s) = \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma\, v_*(s')], \text{ for all } s \in \mathcal{S}$$

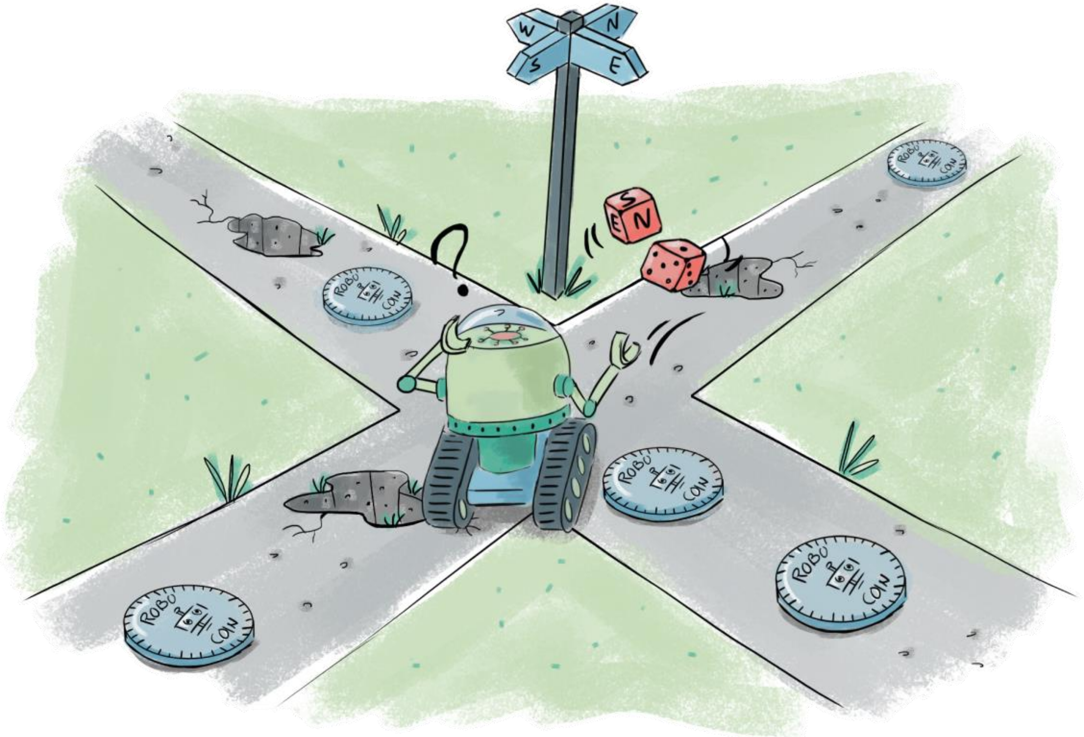


- **The *state-action pair value* under an optimal policy must equal the expected return for the *best* actions from the next state**

$$q_*(s,a) = \sum_{s',r} p(s',r|s,a)\left[r + \gamma \max_{a'} q_*(s',a')\right]$$



Policy Evaluation



POLICY EVALUATION

Policy Evaluation

Iterative policy evaluation.

Update Rule for Bellman Equation (state values)

Consider a sequence of approximate value functions $v_0, v_1, v_2, \cdots$, each mapping $\mathcal{S}^+$ to $\mathbb{R}$ (the real numbers).

- Using the Bellman equation, each successive approximation for state value is obtained by using the update rule as follows:

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

$$= \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma v_k(s')], \text{ for } all\ s \in \mathcal{S}$$

where $v_k = v_\pi$ is a fixed point for this update rule because the Bellman equation for v_π assures us of equality in this case

- The initial approximation, v_0, is chosen arbitrarily.
- The terminal state, if any, must be given value 0

Policy Evaluation

Iterative policy evaluation

Update Rule for Bellman Equation (state values)

$$v_{k+1}(s) \doteq \sum_{a} \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma v_k(s')], \text{ for } all\ s \in \mathcal{S}$$

- $v_{k+1}(s)$ is the estimate of value function, which iteratively gets updated sequentially
- The sequence $\{v_k\}$ can be shown in general to converge to v_π as $k \to \infty$
- The existence and uniqueness of v_π are guaranteed as long as
  - $K \to \infty$
  - eventual termination is guaranteed from all states under the policy π

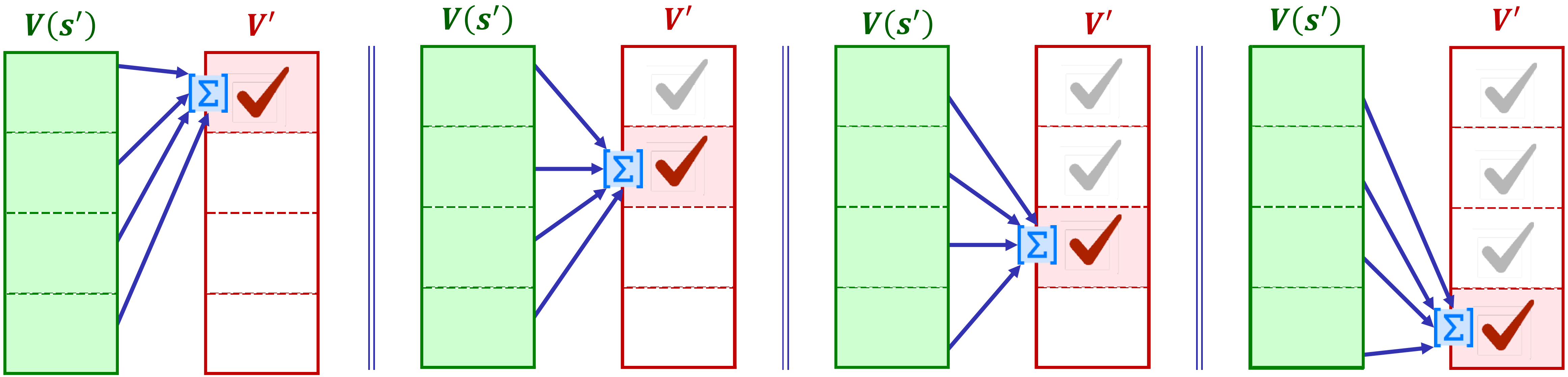
$$\lim_{k \to \infty} v_k = v_\pi, \qquad \forall\ v_0$$

Policy Evaluation

Iterative policy evaluation

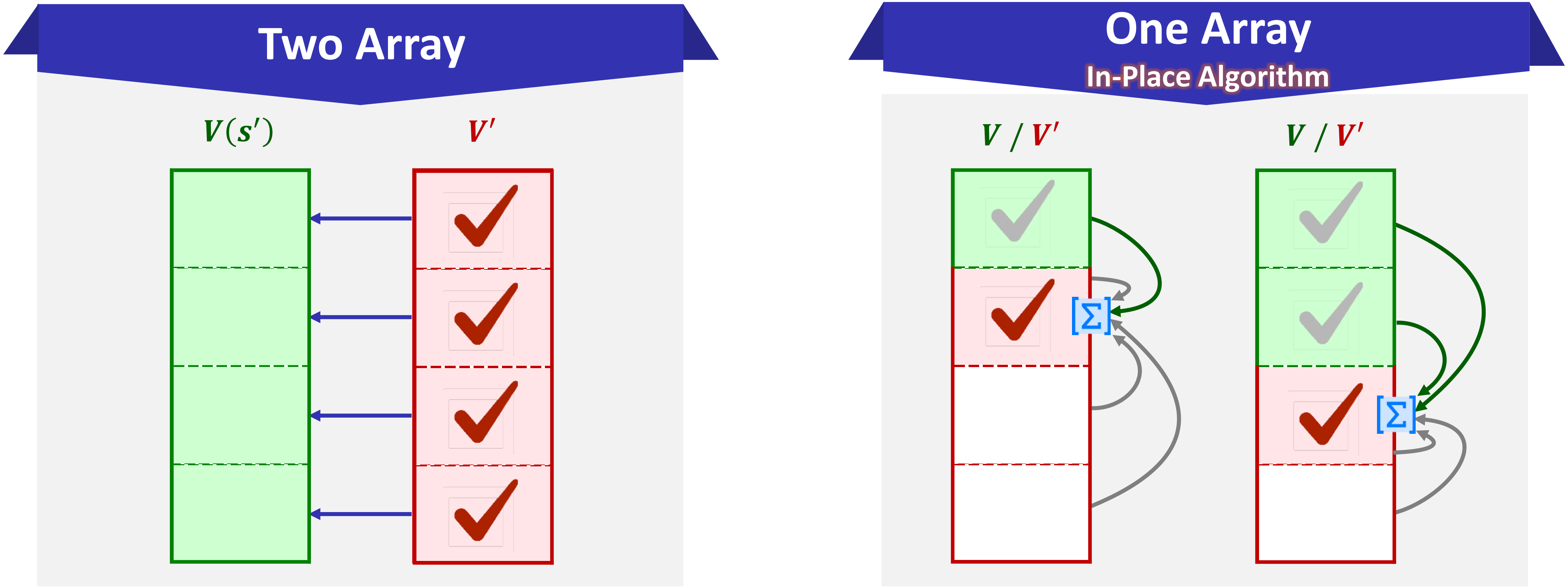
Update Rule for Bellman Equation (state values)

$$\underbrace{V'(s)}_{\text{Current state updated value}} \doteq \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma \underbrace{V(s')}_{\text{successive states (old values)}}], \text{ for } all\ s \in \mathcal{S}$$



Policy Evaluation

Iterative policy evaluation



- ***In-Place Algorithm usually converges faster than the two-array version***
  - it uses new data as soon as they are available

Policy Evaluation

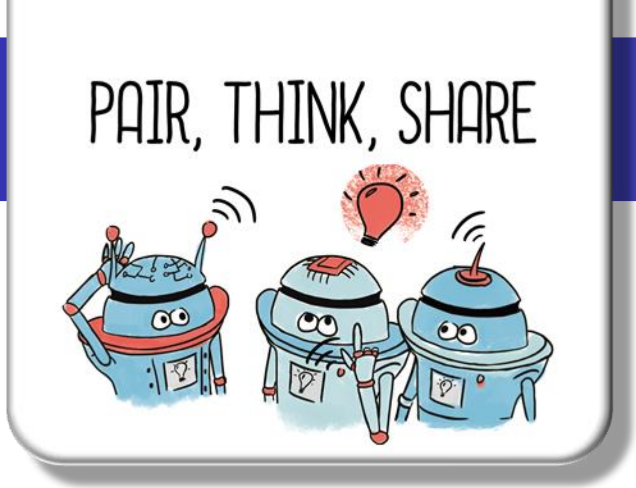
Iterative policy evaluation.

Iterative Policy Evaluation, for estimating $V \approx v_\pi$

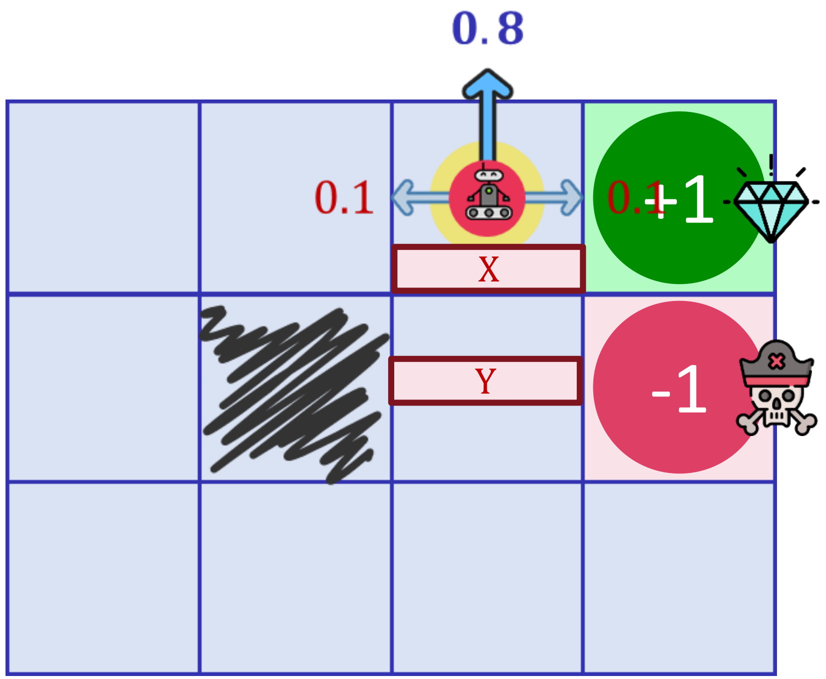
Input π, the policy to be evaluated
Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:
  $\Delta \leftarrow 0$
  Loop for each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

One Sweep



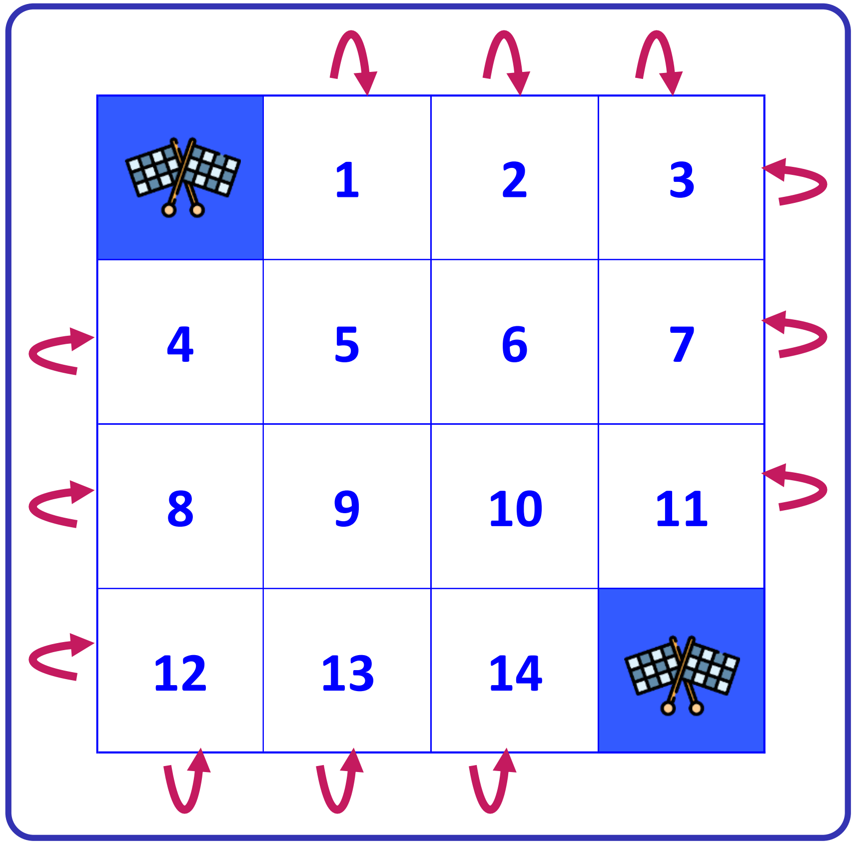
- **Recall the Gridworld Example**
- Use the evaluate iteration to calculate the state value of x for its 1st and 2nd iteration
  - $v_1(X) = ?$
  - $v_2(X) = ?$



- **Parameters:**
  - $\gamma = 0.5$
  - $R(S) = -0.04$
  - $v_0(s) = 0$
- **Policy**
  - Random Policy

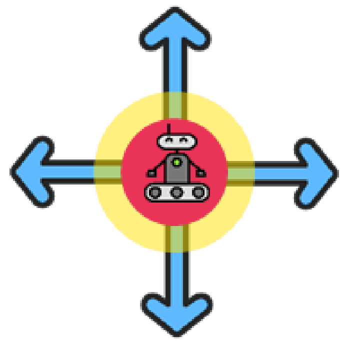
Policy Evaluation

Example: 4×4 Gridworld



States

$$\pi(n|\cdot) = \pi(e|\cdot) = \pi(w|\cdot) = \pi(s|\cdot) = 0.25$$



Actions

$$R_t = -1$$

$$\gamma = 1$$

Reward

Policy Evaluation

Example: 4×4 Gridworld

V

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

» $k = 1$ «

$$v_1(1) = \uparrow 0.25 \times (-1 + 0) + \downarrow 0.25 \times (-1 + 0) + \leftarrow 0.25 \times (-1 + 0) + \rightarrow 0.25 \times (-1 + 0)$$

$$v_1(1) = -1$$

V'

	-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	

$$\underbrace{v_{k+1}(s)}_{\text{updated state value}} = \sum_a \pi(a|s) \sum_{s',r} \underbrace{p(s',r|s,a)}_{1}[\underbrace{r}_{-1} + \gamma \underbrace{v_k(s')}_{\text{old values of the successor states}}],$$

Policy Evaluation

Example: 4×4 Gridworld

V

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	0

» $k = 2$ «

$$v_2(1) = \uparrow 0.25 \times (-1 + 1 \times -1) +$$
$$\downarrow 0.25 \times (-1 + 1 \times -1) +$$
$$\leftarrow 0.25 \times (-1 + 1 \times 0) +$$
$$\rightarrow 0.25 \times (-1 + 1 \times -1) +$$

$$v_2(1) = -1.75$$

V'

	-1.75	-2	-2
-1.75	-2	-2	-2
-2	-2	-2	-1.75
-2	-2	-1.75	

$$\underbrace{v_{k+1}(s)}_{\text{updated state value}} = \sum_a \pi(a|s) \sum_{s',r} \underbrace{p(s',r|s,a)}_{1}[\underbrace{r}_{-1} + \gamma \underbrace{v_k(s')}_{\text{old values of the successor states}}],$$

Policy Evaluation

Example: 4×4 Gridworld

$v_k(S)$

$k = 0$

	0.00	0.00	0.00
0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00
0.00	0.00	0.00	

$k = 1$

	-1.75	-2.00	-2.00
-1.75	-2.00	-2.00	-2.00
-2.00	-2.00	-2.00	-1.75
-2.00	-2.00	-1.75	

$k = 3$

	-2.44	-2.94	-3.00
-2.44	-2.88	-3.00	-2.94
-2.94	-3.00	-2.88	-2.44
-3.00	-2.94	-2.44	

$k = 10$

	-6.14	-8.35	-8.97
-6.14	-7.74	-8.43	-8.35
-8.35	-8.43	-7.74	-6.14
-8.97	-8.35	-6.14	

$k = \infty$

	-14.00	-20.00	-22.00
-14.00	-18.00	-20.00	-20.00
-20.00	-20.00	-18.00	-14.00
-22.00	-20.00	-14.00	

Greedy Policy w.r.t v_k

$k = 0$

	↑↓←→	↑↓←→	↑↓←→
↑↓←→	↑↓←→	↑↓←→	↑↓←→
↑↓←→	↑↓←→	↑↓←→	↑↓←→
↑↓←→	↑↓←→	↑↓←→	

$k = 1$

	←	←	↓←
↑	↑←	↓←	↓
↑	↑→	↓→	↓
↑→	→	→	

$k = 3$

	←	←	↓←
↑	↑←	↓←	↓
↑	↑→	↓→	↓
↑→	→	→	

$k = 10$

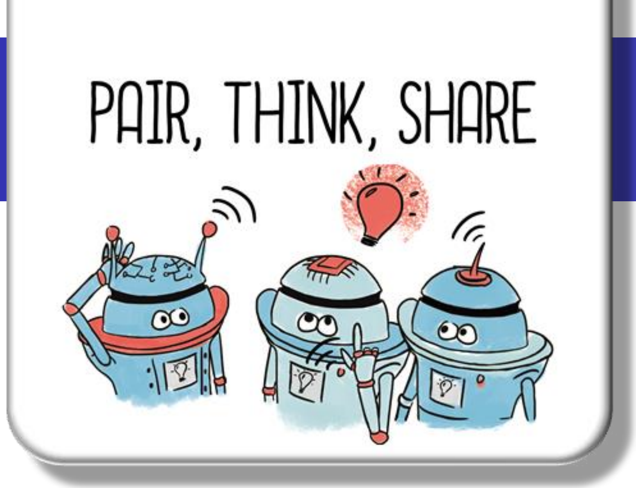
	←	←	↓←
↑	↑←	↓←	↓
↑	↑→	↓→	↓
↑→	→	→	

$k = \infty$

	←	←	↓←
↑	↑←	↓←	↓
↑	↑→	↓→	↓
↑→	→	→	

Random Policy
equiprobable

Optimal Policy



Recall the Gridworld Example

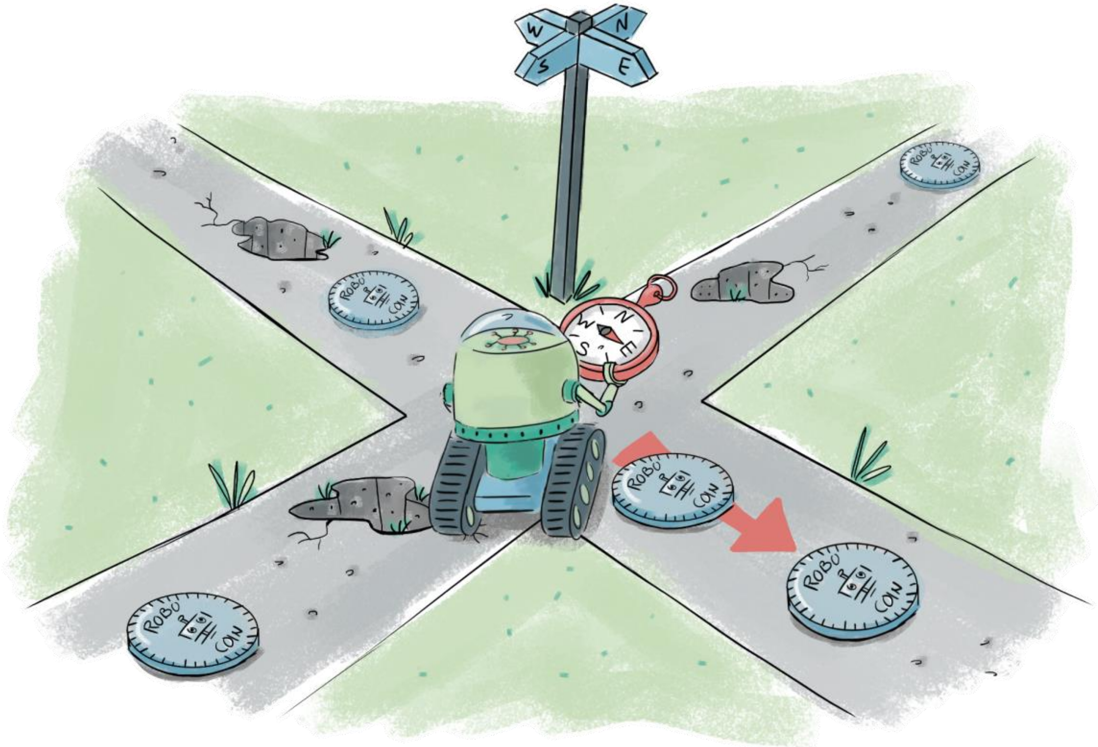
1. Why the model converges even with $\gamma = 1.0$

2. What is the simple interpretation of the values of states when the policy evaluation converges?

$k = \infty$

	-14.0	-20.0	-22.0
-14.0	-18.0	-20.0	-20.0
-20.0	-20.0	-18.0	-14.0
-22.0	-20.0	-14.0	

Policy Improvement



POLICY IMPROVEMENT

Policy Improvement

Policy Evaluation helps us to compute value function for a policy $\boldsymbol{v_\pi(s)}$

- $\boldsymbol{v_\pi(s)}$: indicates know how good it is to follow the current policy from s
- $\boldsymbol{\pi \rightarrow \pi'}$: results for policy evaluation can help develop better policy

Policy Change is applied after policy evaluation is performed

1. $\boldsymbol{a \neq \pi(s)}$: change in the policy at a single state to a particular action
2. $\boldsymbol{\pi'}$: obtain the new policy based on the new selected action
3. $\boldsymbol{v_{\pi'}(s)}$: evaluate the new changed policy

Policy improvement theorem

- Let π and π' be any pair of deterministic policies such that

$$\underbrace{q_\pi(s, \boldsymbol{\pi}'(\boldsymbol{s}))}_{\text{New Value}} \geq \underbrace{q_\pi(s, \boldsymbol{\pi}(\boldsymbol{s}))}_{\text{Old value}}, \text{ for all } \boldsymbol{s} \in \mathcal{S},$$

  - then the policy $\boldsymbol{\pi}'$ must be as good as, or better than, $\boldsymbol{\pi}$.

$$\boldsymbol{\pi}' \geq \boldsymbol{\pi}$$

  - The new policy must obtain greater or equal expected return from all states $\boldsymbol{s} \in \mathcal{S}$:

$$v_{\pi'}(s) \geq \boldsymbol{v}_{\boldsymbol{\pi}}(\boldsymbol{s}), \text{ for all } \boldsymbol{s} \in \mathcal{S}$$

- The new policy is a ***strict*** improvement over π, if and only if:

$$q_\pi(s, \boldsymbol{\pi}'(\boldsymbol{s})) > q_\pi(s, \boldsymbol{\pi}(\boldsymbol{s})) \text{ for at least one } s \in \mathcal{S} \quad \rightarrow \quad \boldsymbol{\pi}' > \boldsymbol{\pi}$$

How to select actions to improve the policy?

Greedy policy

Greedy policy considers changes at all states and to all possible actions, by selects at each state the action that appears best according to $q_\pi(s, a)$

$$\begin{aligned}\boldsymbol{\pi}'(\boldsymbol{s}) &\doteq \underset{\boldsymbol{a}}{\mathbf{argmax}}\, q_\pi(s, a) \\ &\doteq \underset{\boldsymbol{a}}{\mathbf{argmax}} \sum_{\boldsymbol{s}', \boldsymbol{r}} \boldsymbol{p}(\boldsymbol{s}', \boldsymbol{r} | \boldsymbol{s}, \boldsymbol{a})[\boldsymbol{r} + \boldsymbol{\gamma} \boldsymbol{v}_{\boldsymbol{\pi}}(\boldsymbol{s}')],\end{aligned}$$

- By construction, the greedy policy meets the conditions of the policy improvement theorem
- We know that the new policy is as good as, or better than, the original policy.

Policy Improvement

The process of making a new policy that improves on an original policy, by making it greedy with respect to the value function of the original policy, is called policy improvement.

$$v_{\pi'}(s) = \max_a \mathbb{E}[R_{t+1} + \gamma v_{\pi'}(S_{t+1}) | S_t = s, A_t = a]$$

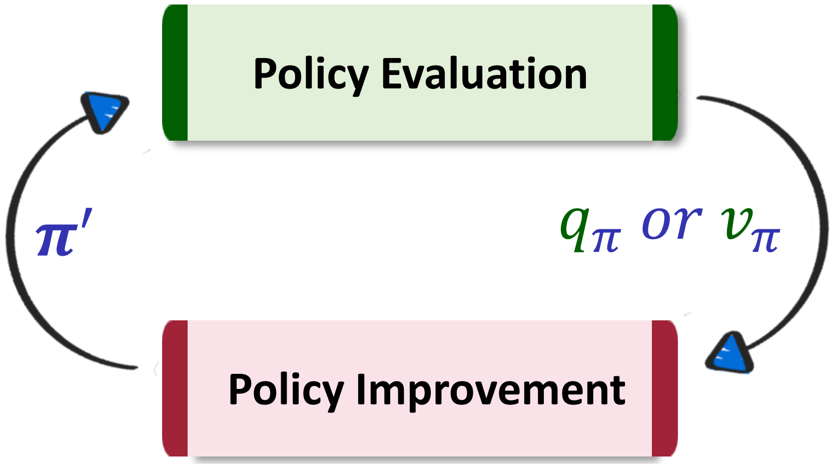
$$v_{\pi'}(s) = \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma v_{\pi'}(s')],$$

- Policy improvement is same as the Bellman optimality equation, and therefore:
  - $v_{\pi'}(s)$ must be v_*
  - **Both** $\boldsymbol{\pi}'$ and $\boldsymbol{\pi}$ must be optimal policies
- Policy improvement thus must give us a strictly better policy except when the original policy is already optimal.

Policy Iteration

Policy Iteration

- **Policy iteration:** is the process combining policy evaluation, updating the value of the policy, and policy improvement, obtaining the best policy available



$\xrightarrow{I}$ **I**mprovement

$\xrightarrow{E}$ **E**valuation

$$\pi_0 \xrightarrow{E} v_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} v_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \cdots \xrightarrow{I} \pi_* \xrightarrow{E} v_*$$

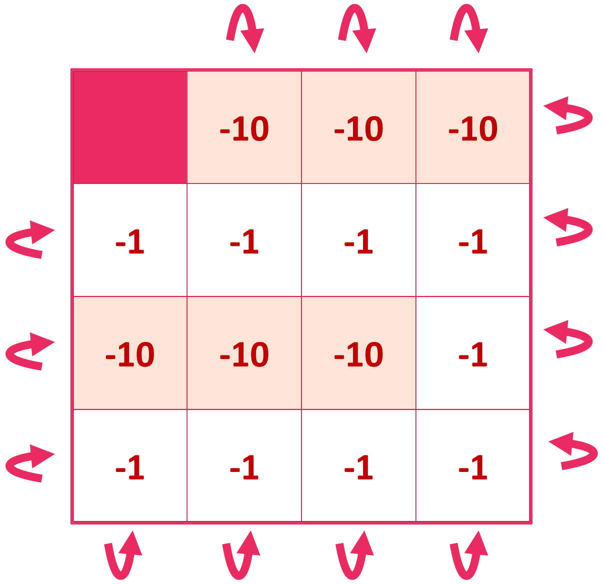
- Each policy is guaranteed to be a strict improvement over the previous one
- Because a finite MDP:
  - has only a finite number of policies
  - this process must converge to an optimal policy and optimal value function in a finite number of iterations

Policy Iteration

The Gridworld Example: New Version

- In Small Gridworld improved policy was optimal, $\pi' =$
- In general, need more iterations of improvement / evaluation
- But this process of policy iteration always converges to π_*

$R(s)$



Excel Lab

Policy Iteration

The Gridworld Example: New Version

Initial Policy

	←→↑↓	←→↑↓	←→↑↓
←→↑↓	←→↑↓	←→↑↓	←→↑↓
←→↑↓	←→↑↓	←→↑↓	←→↑↓
←→↑↓	←→↑↓	←→↑↓	←→↑↓

→

State Vlaues

	-137	-210	-239
-129	-181	-221	-238
-194	-214	-232	-242
-218	-228	-238	-242

→

New Policy

	←	←	←
↑	←	←	←
↑	↑	↑	↑
↑	←	←	←

Policy Iteration

The Gridworld Example: New Version

Old Policy

	←	←	←
↑	←	←	←
↑	↑	↑	↑
↑	←	←	←

State Vlaues

	-1	-11	-21
-1	-2	-3	-4
-2	-3	-4	-5
-12	-13	-14	-15

New Policy

	←	↓	↓
↑	←	←	←
↑	↑	↑	↑
↑	←↑	←↑	↑

Policy Iteration

The Gridworld Example: New Version

- **We repeat iterations until the policy becomes stable**

Old Policy

	←	↓	↓
↑	←	←	←
↑	↑	↑	↑
→	→	→	↑

=

Optimal Policy

	←	↓	↓
↑	←	←	←
↑	↑	↑	↑
→	→	→	↑

Optimal State Vlaues

	-1	-4	-5
-1	-2	-3	-4
-2	-3	-4	-5
-9	-8	-7	-6

Policy Iteration

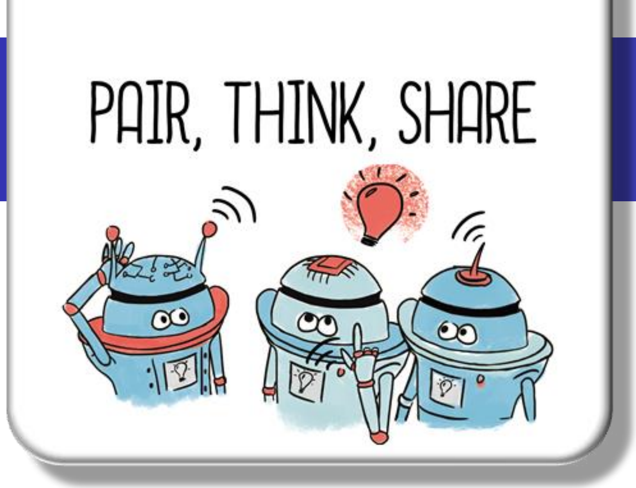
Pseudocode

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Loop:
   $\quad \Delta \leftarrow 0$
   $\quad$ Loop for each $s \in \mathcal{S}$:
   $\quad\quad v \leftarrow V(s)$
   $\quad\quad V(s) \leftarrow \sum_{s',r} p(s', r \mid s, \pi(s))\big[r + \gamma V(s')\big]$
   $\quad\quad \Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   $\quad$ *old-action* $\leftarrow \pi(s)$
   $\quad \pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma V(s')\big]$
   $\quad$ If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2



Policy Iteration pseudocode

The policy iteration algorithm on page 80 has a subtle bug in that it may never terminate if the policy continually switches between two or more policies that are equally good.

Describe qualitatively how we can modify the pseudocode so that convergence is guaranteed?

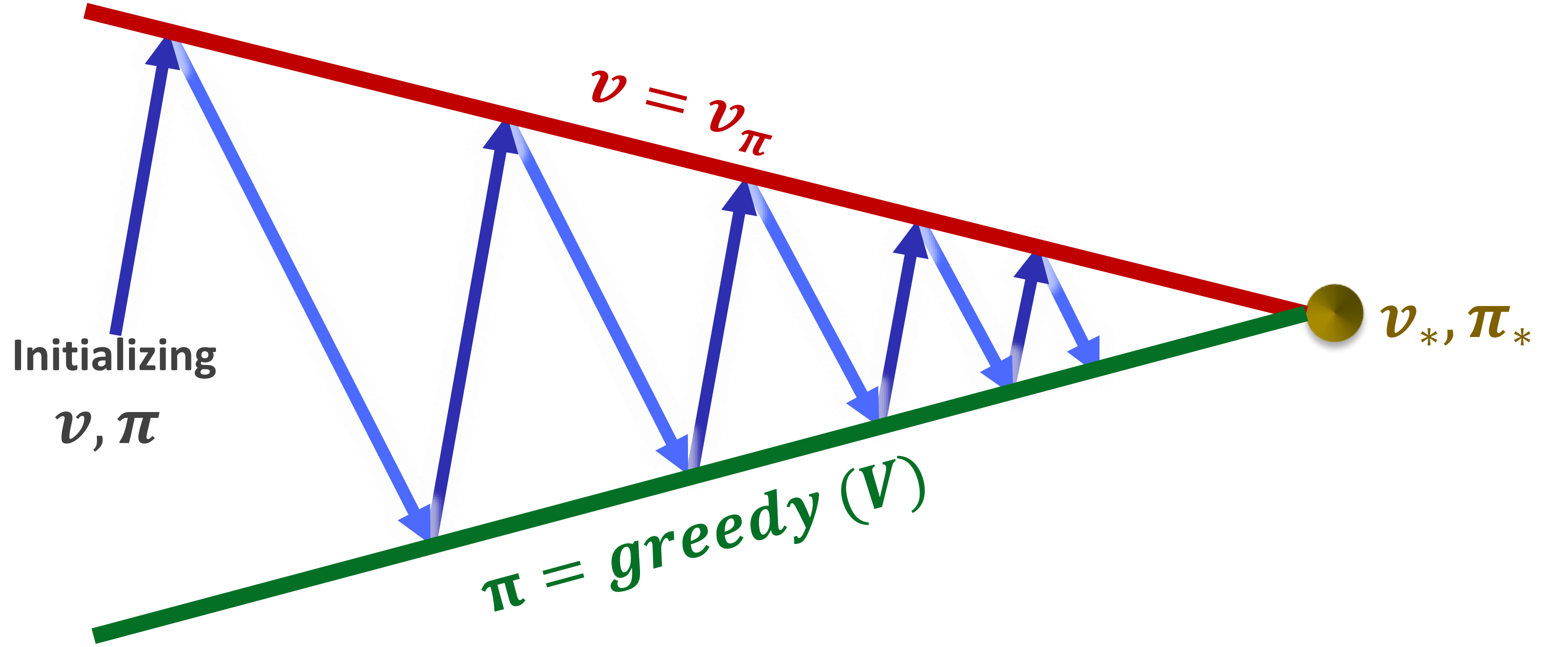
Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

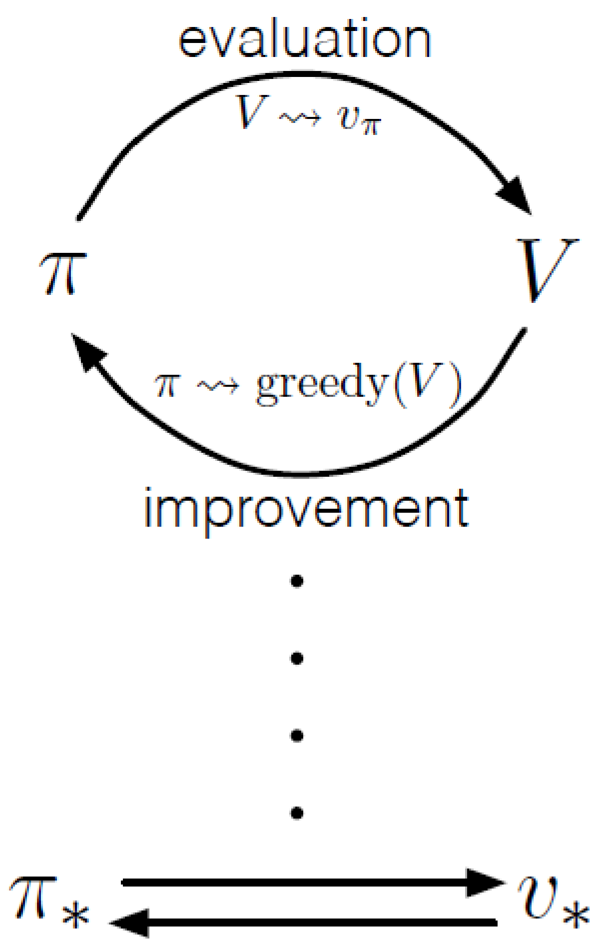
1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Loop:
       $\Delta \leftarrow 0$
       Loop for each $s \in \mathcal{S}$:
           $v \leftarrow V(s)$
           $V(s) \leftarrow \sum_{s',r} p(s',r|s,\pi(s))\left[r + \gamma V(s')\right]$
           $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

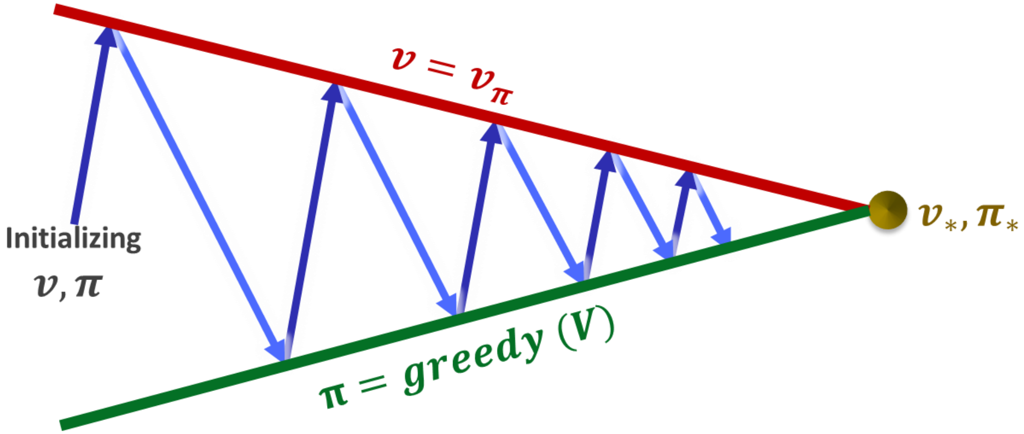
3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
       old-action $\leftarrow \pi(s)$
       $\pi(s) \leftarrow \operatorname{argmax}_a \sum_{s',r} p(s',r|s,a)\left[r + \gamma V(s')\right]$
       If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

Generalized Policy Iteration (GPI)





- Any policy evaluates to a unique value function which can be greedified to produce a better policy
- That in turn evaluates to a value function which can in turn be greedified
- Each policy is *strictly better* than the previous, until *eventually both are optimal*
- The dance converges in a finite number of steps, usually very few



- **Competing:**
  They compete in the sense that they pull in opposing directions. Making the policy greedy with respect to the value function typically makes the value function incorrect for the changed policy, and making the value function consistent with the policy typically causes that policy no longer to be greedy.

- **Cooperating:**
  In the long run, however, these two processes interact to find a single joint solution

Value Iteration

Value Iteration

- **One drawback to policy iteration is that each of its iterations involves policy evaluation**
  - Iterative computation requiring multiple sweeps through the state set

- **Must we wait for exact convergence, or can we stop short of that?**

$k = 3$

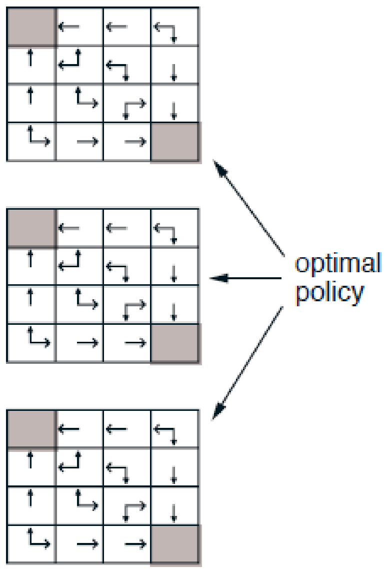
0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

$k = 10$

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

$k = \infty$

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



In this example, policy evaluation iterations beyond the first three have no effect on the corresponding greedy policy.

Value Iteration

- **One drawback to policy iteration is that each of its iterations involves policy evaluation**
  - Iterative computation requiring multiple sweeps through the state set

- **In fact, the policy evaluation step of policy iteration can be truncated**
  - i.e. stop policy evaluation just one sweep (one update of each state).
  - This algorithm is called value iteration

Update Rule for Bellman Equation (action values)

$$v_{k+1}(s) \doteq \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma\, v_k(s')], \text{ for } all\ s \in \mathcal{S}$$

Value Iteration

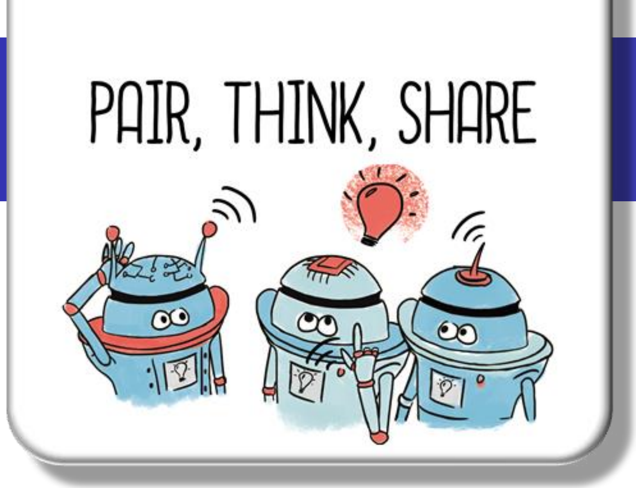
Pseudocode

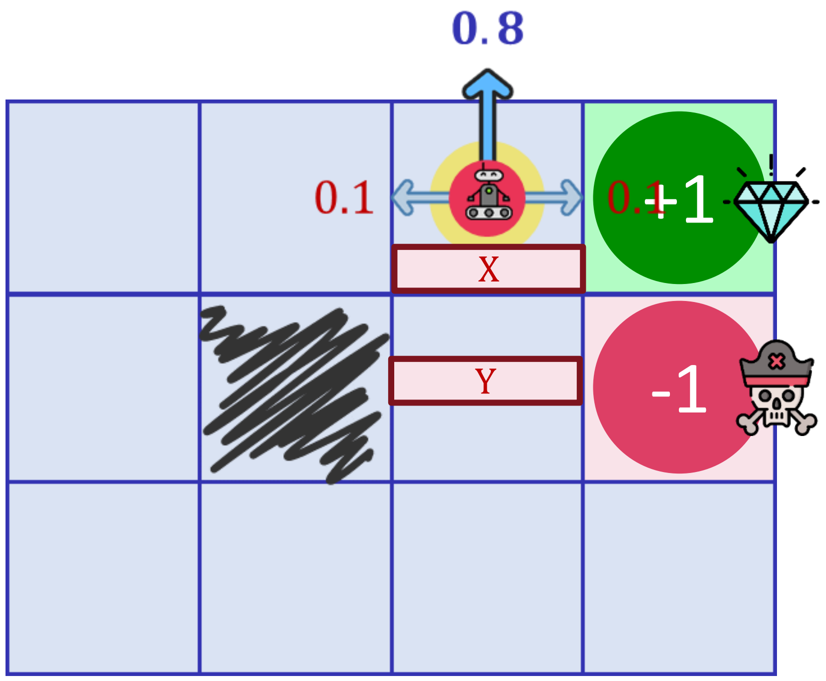
Value Iteration, for estimating $\pi \approx \pi_*$

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:
| $\Delta \leftarrow 0$
| Loop for each $s \in \mathcal{S}$:
| $v \leftarrow V(s)$
| $V(s) \leftarrow \max_a \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma V(s')\big]$
| $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Output a deterministic policy, $\pi \approx \pi_*$, such that
$\pi(s) = \arg\max_a \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma V(s')\big]$

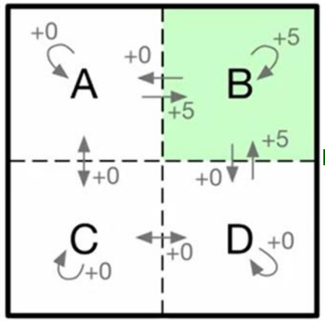




- **Recall the Gridworld Example**
- Use the evaluate iteration to calculate the state value of x for its 1st and 2nd iteration
  - $v_1(X) = ?$
  - $v_2(X) = ?$
- **Parameters:**
  - $\gamma = 0.5$
  - $R(S) = -0.04$
  - $v_0(s) = 0$
- **Policy**
  - Based on the Value Iteration?

The curse of dimensionality

- **We can DP is appropriate when the optimal policy is polynomial in the number of states**



Polynomial Policy → DP

- **the number of states is often astronomical, e.g., often growing exponentially with the number of state** variables (what Bellman called "the curse of dimensionality")

- More than 2^{10} states
- 21 dice combos
- 20 legal Move/combo

→ Asynchronous DP

Asynchronous DP

- DP methods described so far used synchronous backups
  - i.e. all states are backed up in parallel

- Asynchronous DP backs up states individually, in any order
  - For each selected state, apply the appropriate backup
  - Can significantly reduce computation
  - Guaranteed to converge if all states continue to be selected
    - To converge correctly, however, an asynchronous algorithm must continue to update the values of all the states: it can’t ignore any state after some point in the computation.
  - take advantage of this flexibility by selecting the states to which we apply updates so as to improve the algorithm’s rate of progress

Summary

- **DP in RL follows these 3 steps:**
  - Policy Evaluation:
  - Policy Improvement
  - Policy Iteration

- **The improvement theorem is also valid for stochastic policy $\pi(a|s)$ cases**
  - In the general case, a stochastic policy $\boldsymbol{\pi}$ specifies probabilities, $\boldsymbol{\pi}(\boldsymbol{a}|\boldsymbol{s})$, for taking each
  - action, a, in each state, s.
  - In particular, the policy improvement theorem carries through as stated for the stochastic case
  - If there are ties in policy improvement steps, each maximizing action can be given a portion of the probability of being selected in the new greedy policy

References

- [1] R. S. Sutton and A. G. Barto, Reinforcement learning: An introduction. MIT press, 2018.