Advanced Machine Learning
(AIE 322)

Lecture 6: Dynamic Programming

Policy Evaluation vs. Control

- Policy evaluation is the task of determining the value function for a specific policy.
- Control is the task of finding a policy to obtain as much reward as possible.
- In other words, finding a policy which maximizes the value function.
- Control is the ultimate goal of reinforcement learning.
- But the task of policy evaluation is usually a necessary first step.
- Dynamic programming algorithms use the Bellman equations to define iterative algorithms for both policy evaluation and control.

Policy evaluation

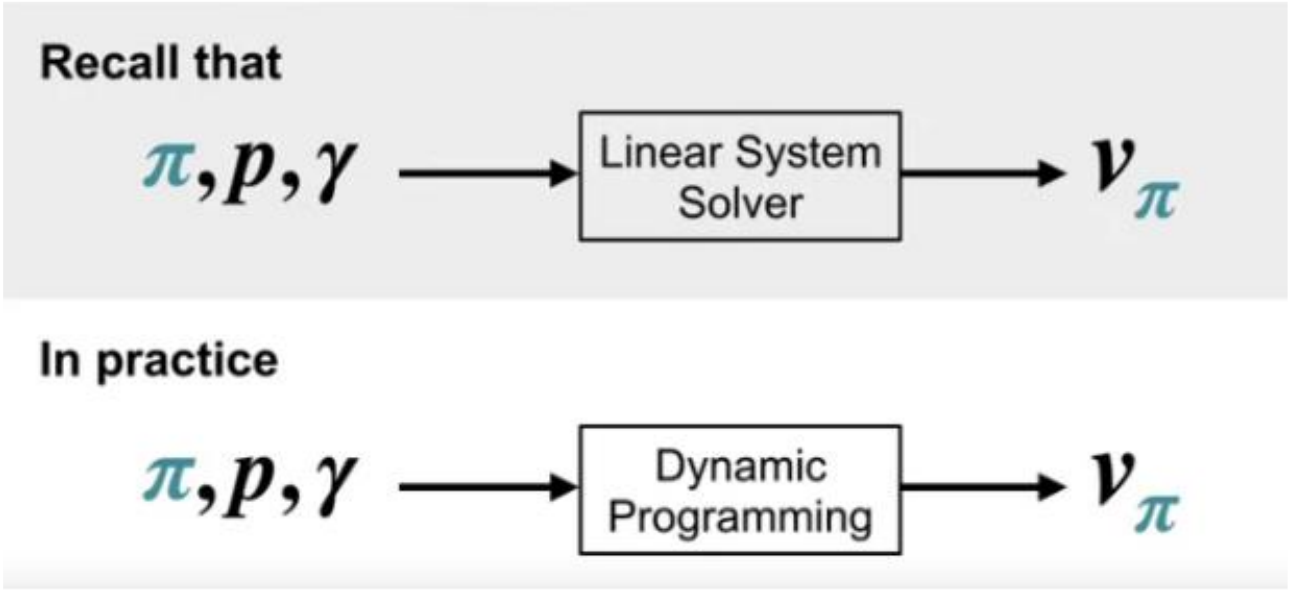
- It is the task of determining the state value function v_π for a particular policy π.

$$\text{Policy evaluation } \pi \rightarrow v_\pi$$

- Remember, the Bellman equation reduces the problem of finding v_π to a system of linear equations, one equation for each state.

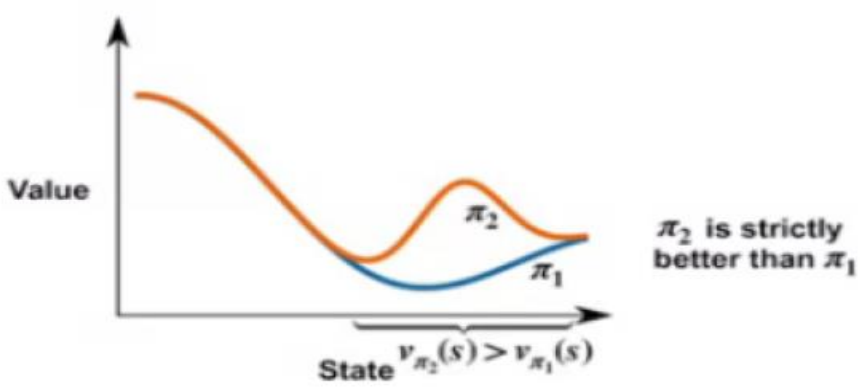
$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a) \left[r + \gamma v_\pi(s')\right]$$

- So the problem of policy evaluation reduces to solving this system of linear equations. But in general this might not be practical when dealing with large MDP

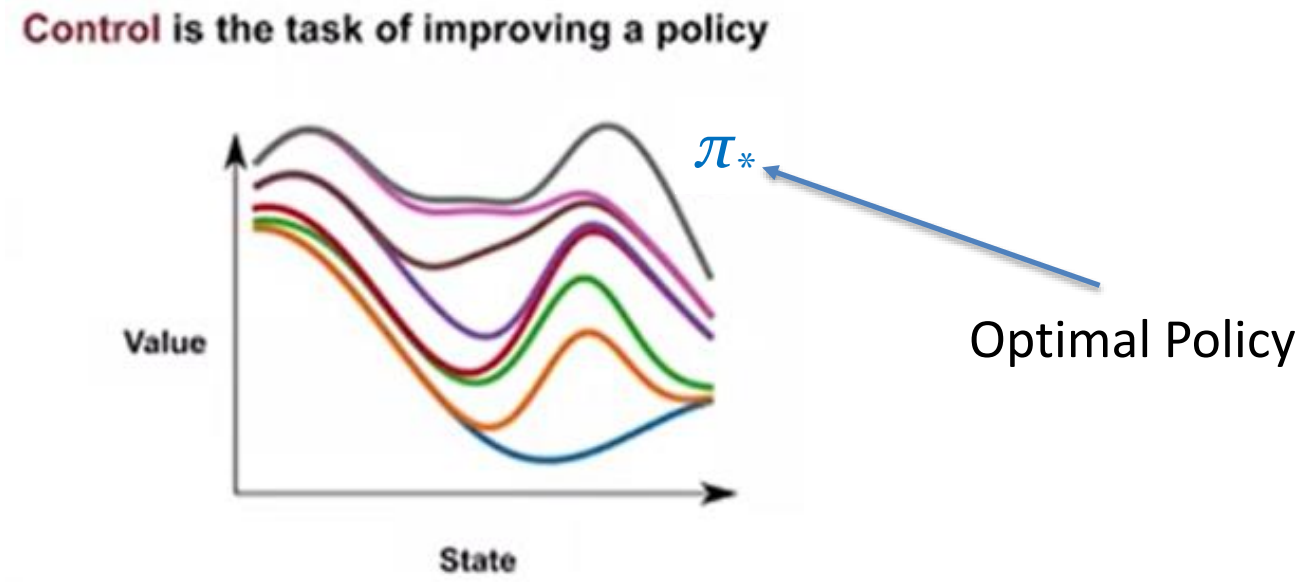


Control

- It the task of improving a policy.
- Remember, here we say π_2 is strictly better than π_1 because π_2 is at least as good as or better than π_1 and there's at least one state where the value under π_2 is strictly greater than the value under π_1.

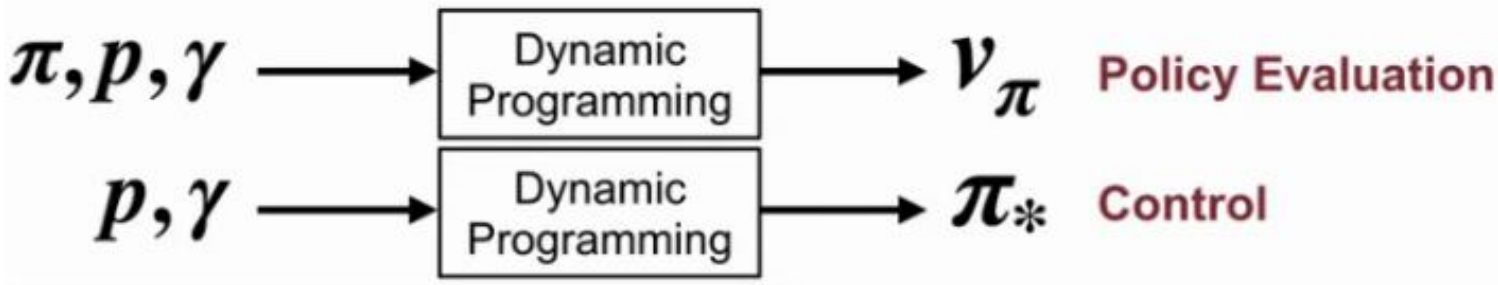


- The goal of the control task is to modify a policy to produce a new one which is strictly better.
- Moreover, we can try to improve the policy repeatedly to obtain a sequence of better and better policies



Dynamic Programming

- Dynamic programming uses the various Bellman equations, along with knowledge of p, to workout value functions and optimal policies.
- Classical dynamic programming does not involve interaction with the environment at all.
- Instead, we use dynamic programming methods to compute value functions and optimal policies given a model of the MDP.



Iterative Policy Evaluation

- The idea of iterative policy evaluation is to take the Bellman equation and directly use it as an update rule.
- This procedure iteratively refine our estimate of the value function.
- This will produce a sequence of better and better approximations to the value function.

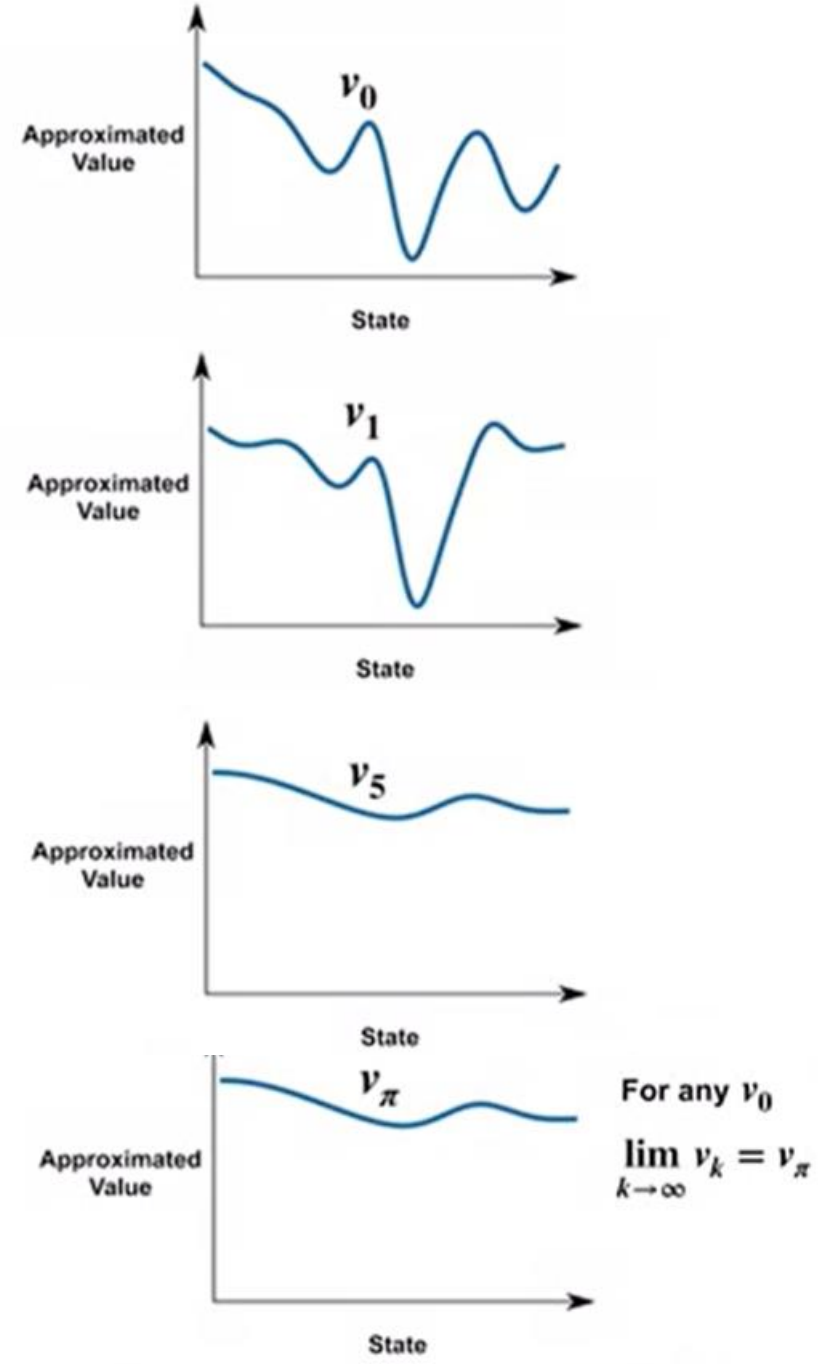
$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)\left[r + \gamma v_\pi(s')\right]$$

Update Rule

$$v_{k+1}(s) \leftarrow \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)\left[r + \gamma v_k(s')\right]$$

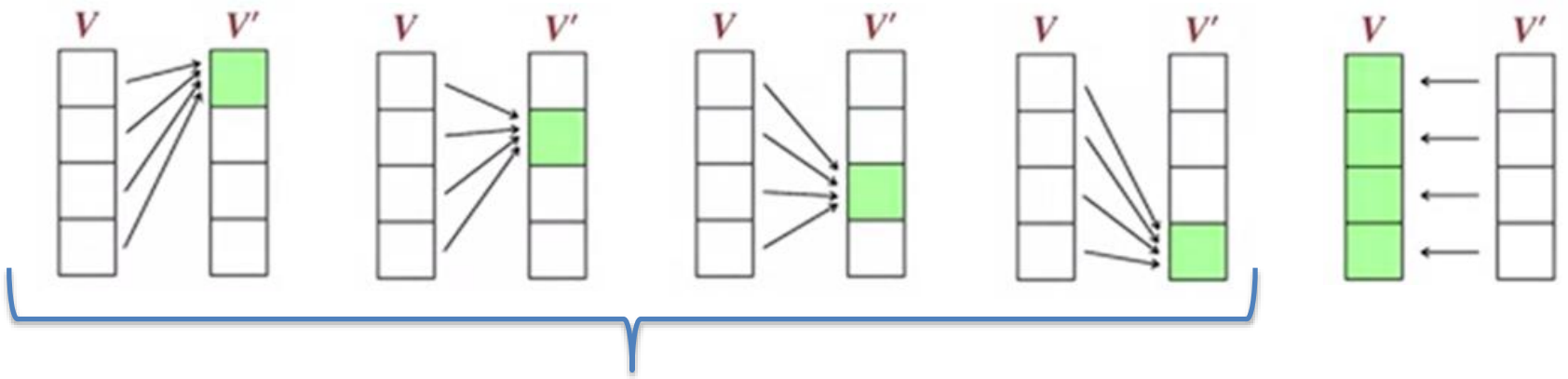
Iterative Policy Evaluation

- We begin with an arbitrary initialization for our approximate value function, let's call this V_o.
- Each iteration then produces a better approximation by using the update rule.
- Each iteration applies this updates to every state, s, in the state space, which we call a sweep.
- Applying this update repeatedly leads to a better and better approximation to the state value function V_π.
- If this update leaves the value function approximation unchanged, that is, if V_{k+1} equals V_k for all states, then V_k equals V_π, and we have found the value function.



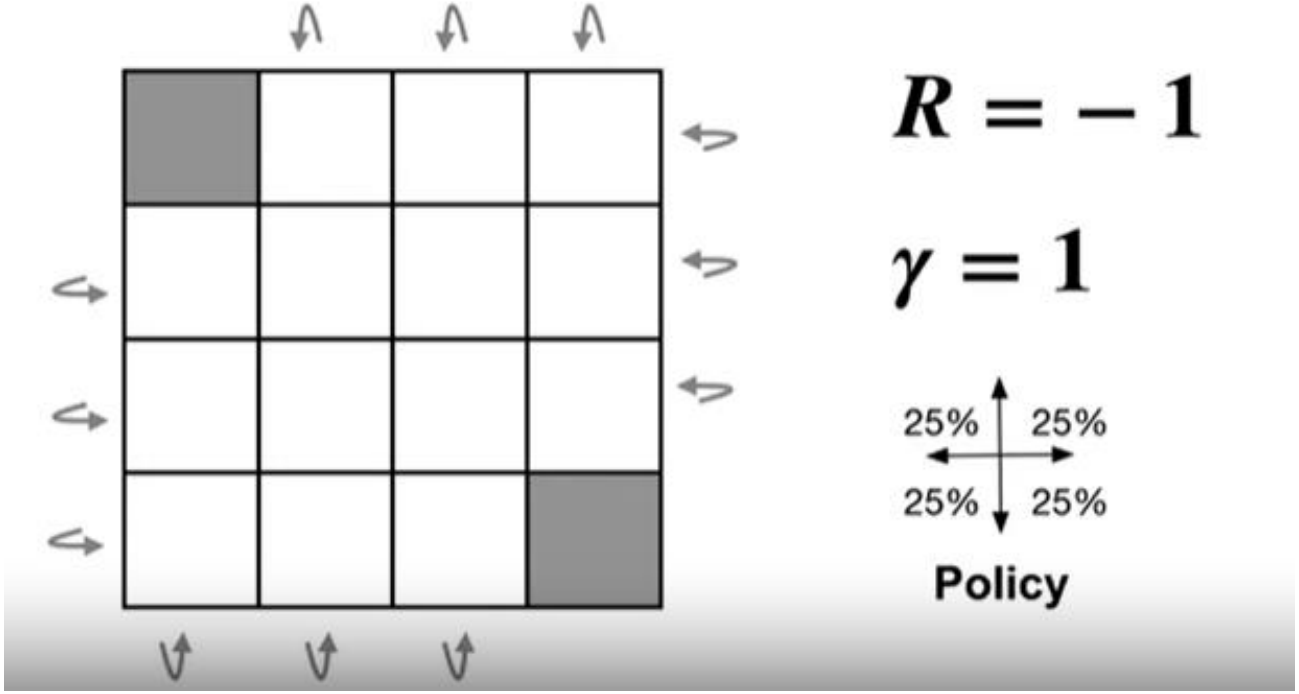
Iterative Policy Evaluation Implementation

- We store two arrays, each has one entry for every state.
- One array, which we label V stores the current approximate value function. Another array, V', stores the updated values.
- By using two arrays, we can compute the new values from the old one state at a time without the old values being changed in the process.
- At the end of a full sweep, we can write all the new values into V ; then we do the next iteration



Example: Gridworld

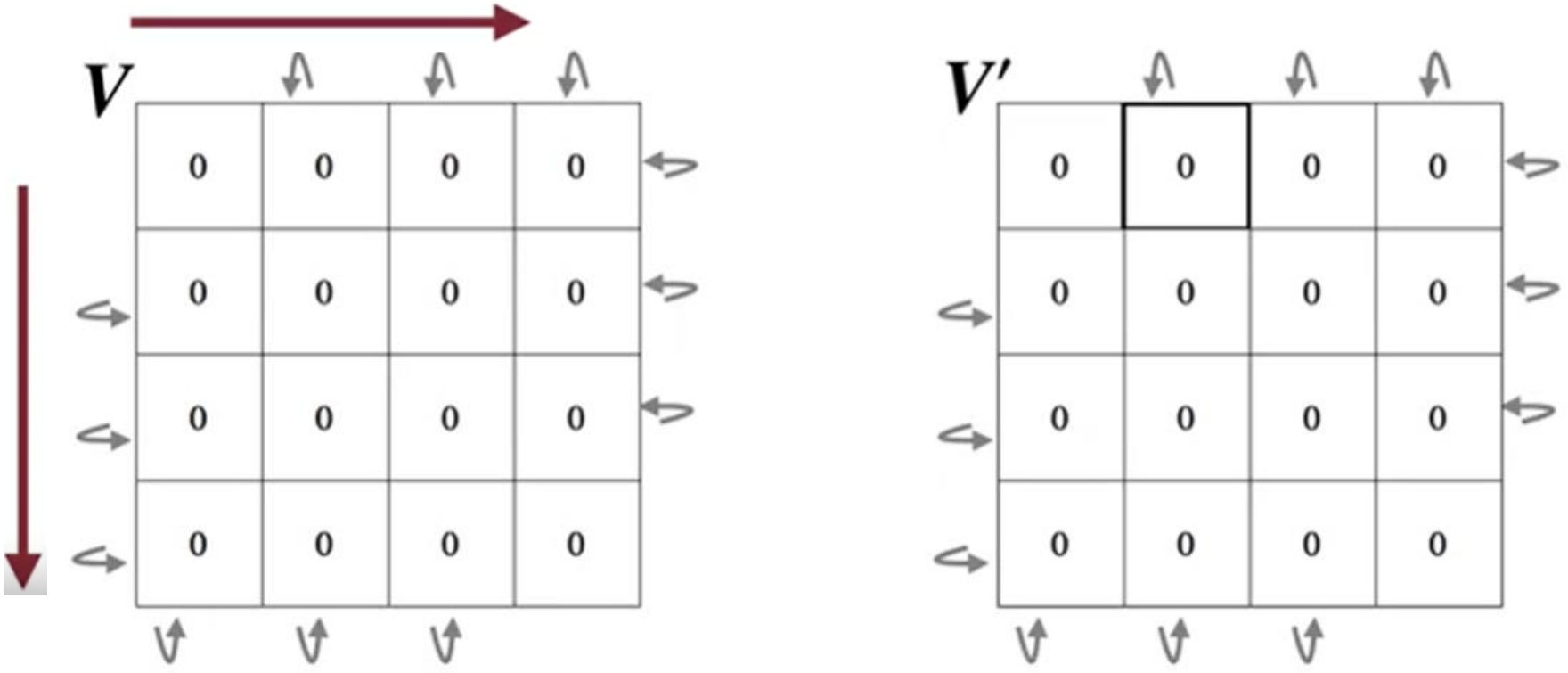
- Assume we have an episodic MDP with terminal states located in the top left and bottom right corners.
- Since the problem is episodic, let's consider the undiscounted case of gamma equals 1.
- There are four possible actions in each state up, down, left, and right.
- Each action is deterministic. If the action would move the agent off the grid, it instead leaves the agent in the same state.
- All transitions leads to a reward of -1 and when we reach a gray box the episode ends.



Example: Gridworld

- Now, let's evaluate the uniform random policy, which selects each of the four actions one-quarter of the time.
- The value function represents the expected number of steps until termination from a given state.
- Assume in this problem that initialize all the values in V to zero,

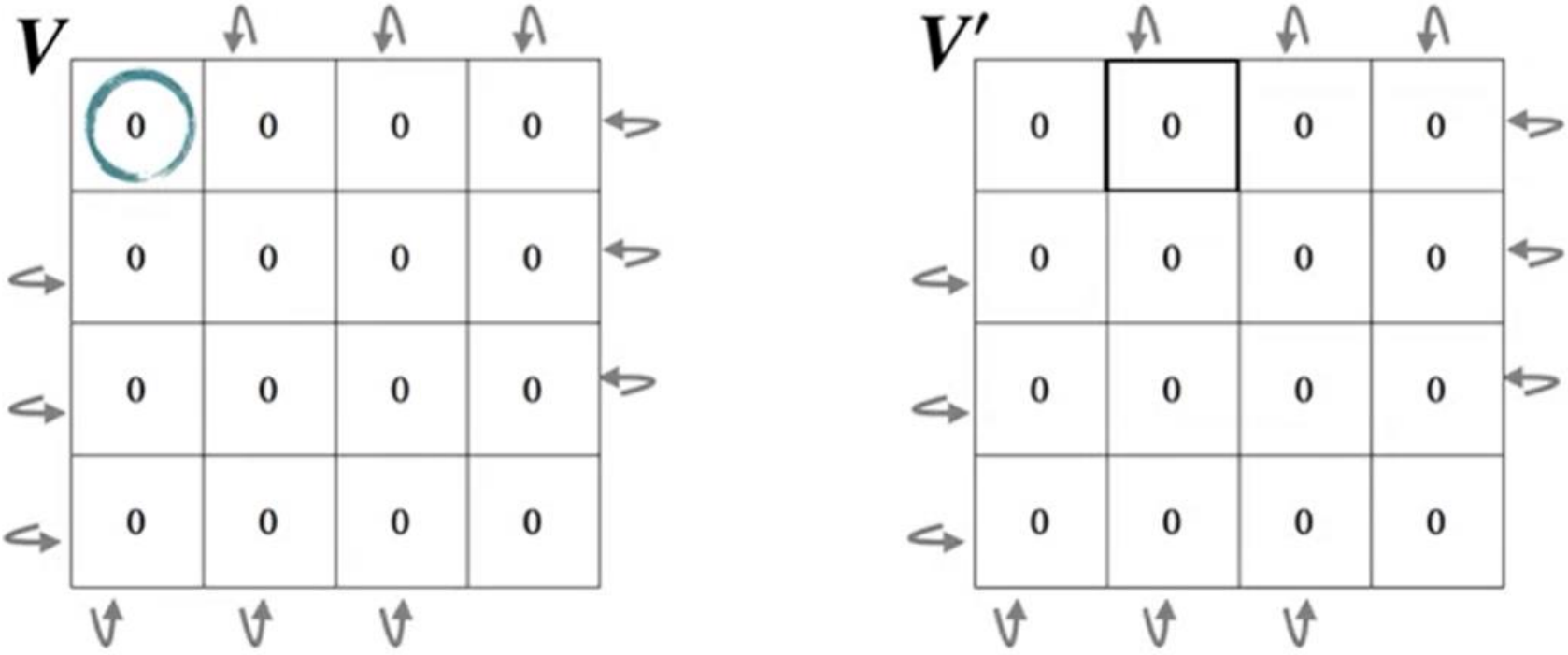
$$V'(s) \leftarrow \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)\left[r + \gamma V(s')\right]$$



Example: Gridworld

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

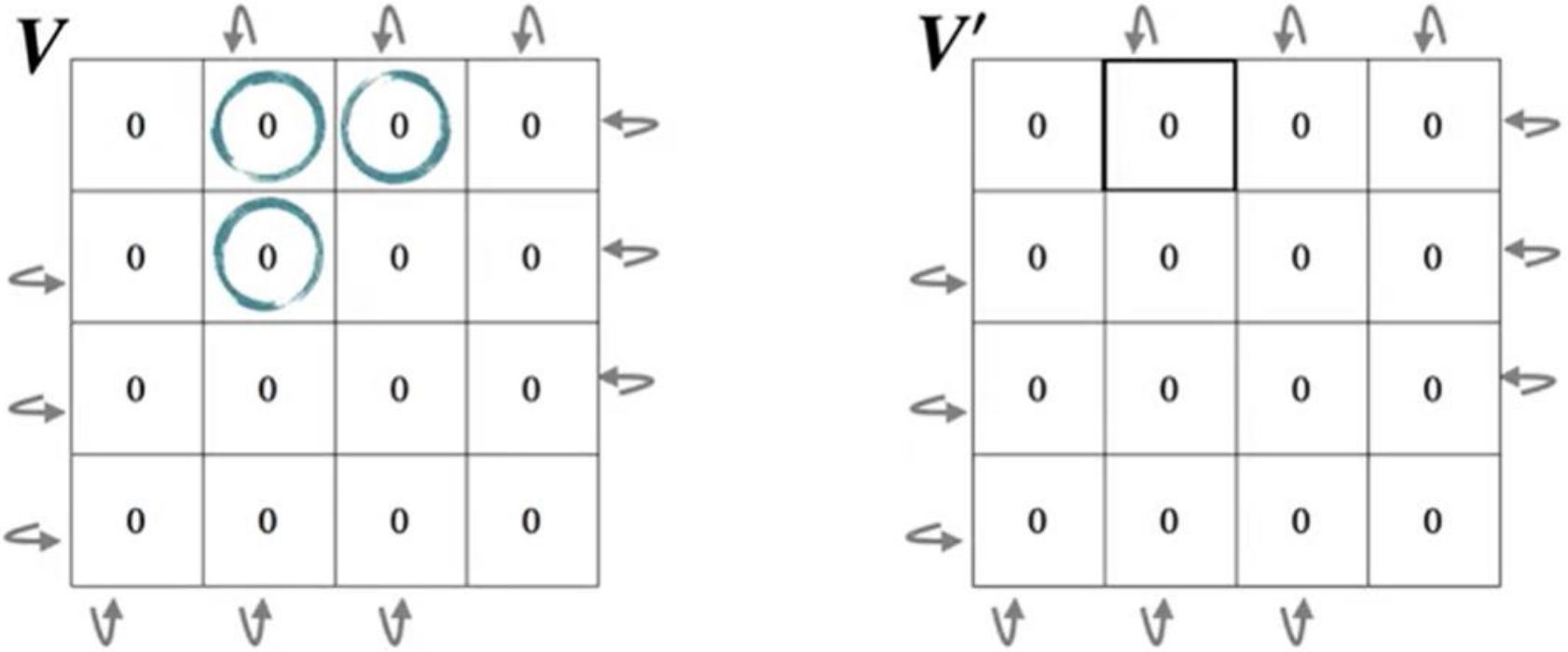
$0.25 * (-1 + 0)$



Example: Gridworld

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

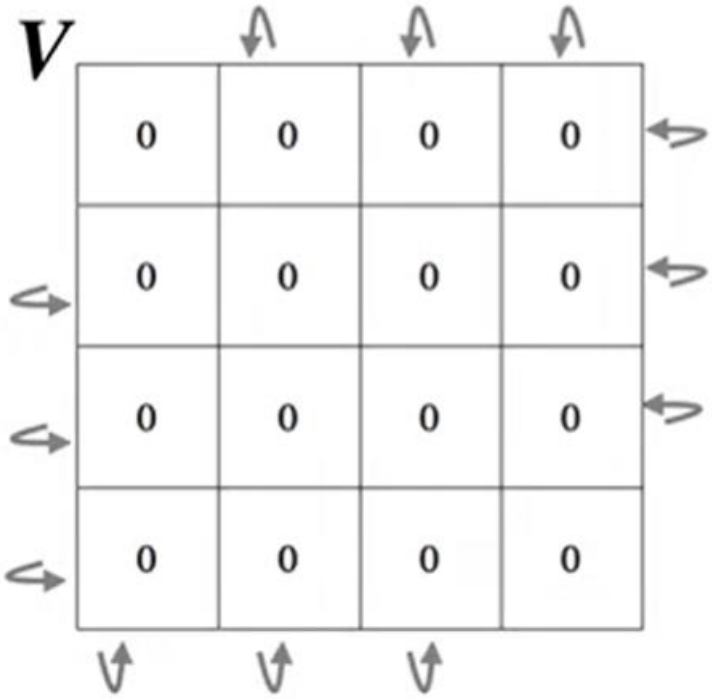
$$0.25 * (-1 + 0) + 0.25 * (-1 + 0) + 0.25 * (-1 + 0) + 0.25 * (-1 + 0) = -1$$

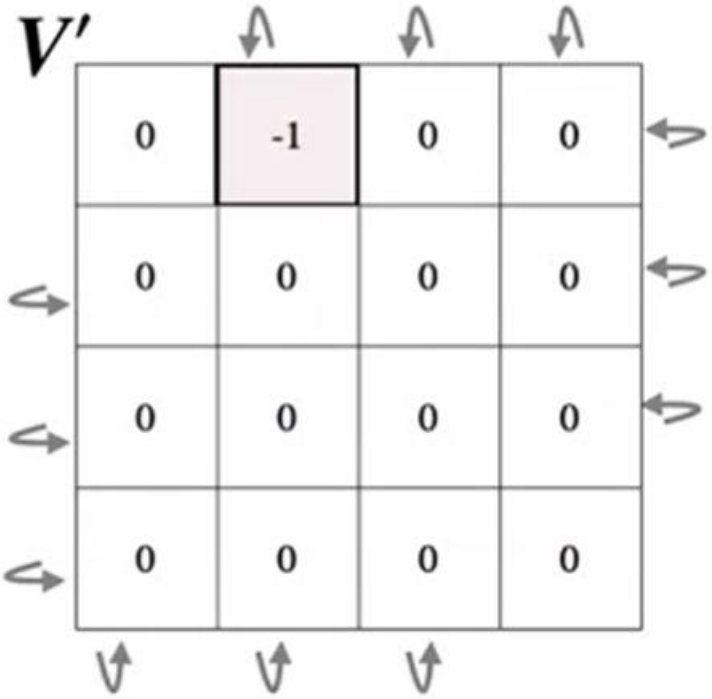


Example: Gridworld

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

$$0.25 * (-1 + 0) + 0.25 * (-1 + 0) + 0.25 * (-1 + 0) + 0.25 * (-1 + 0) = -1$$

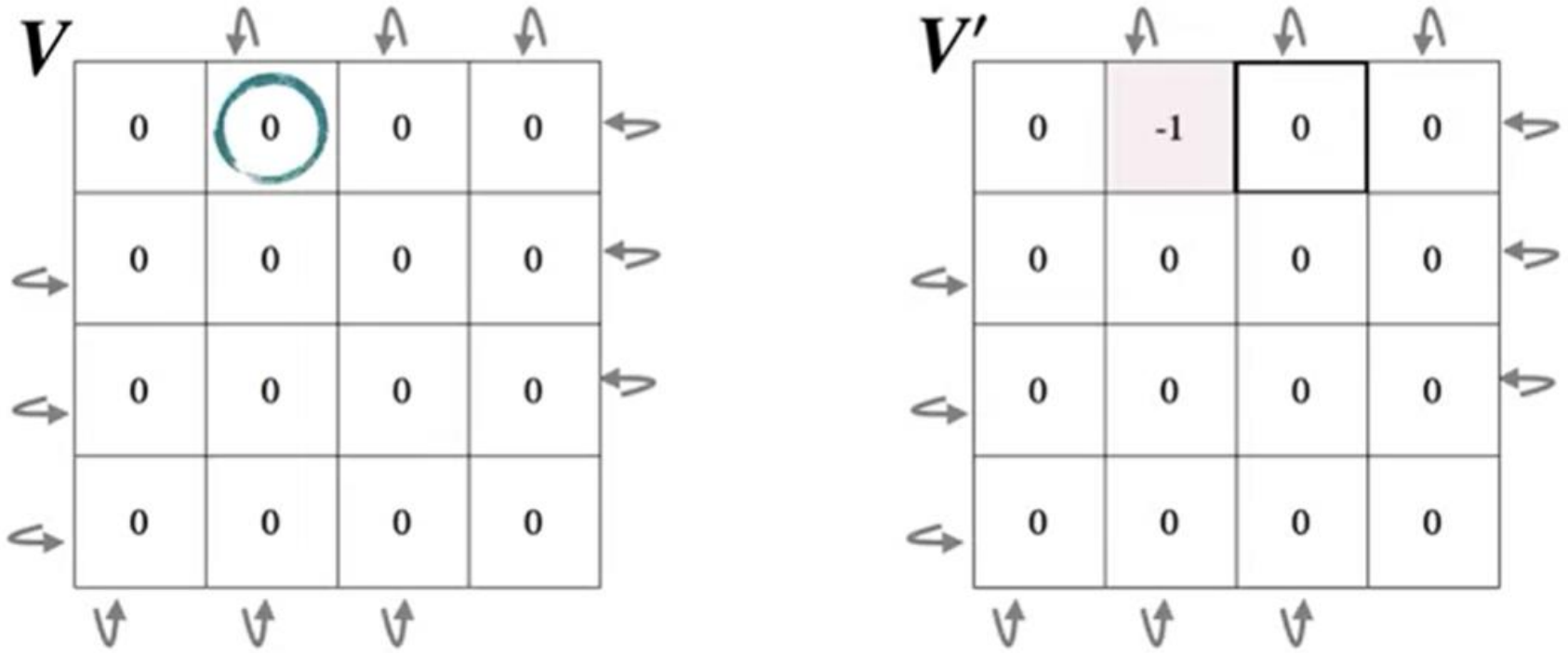




Example: Gridworld

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

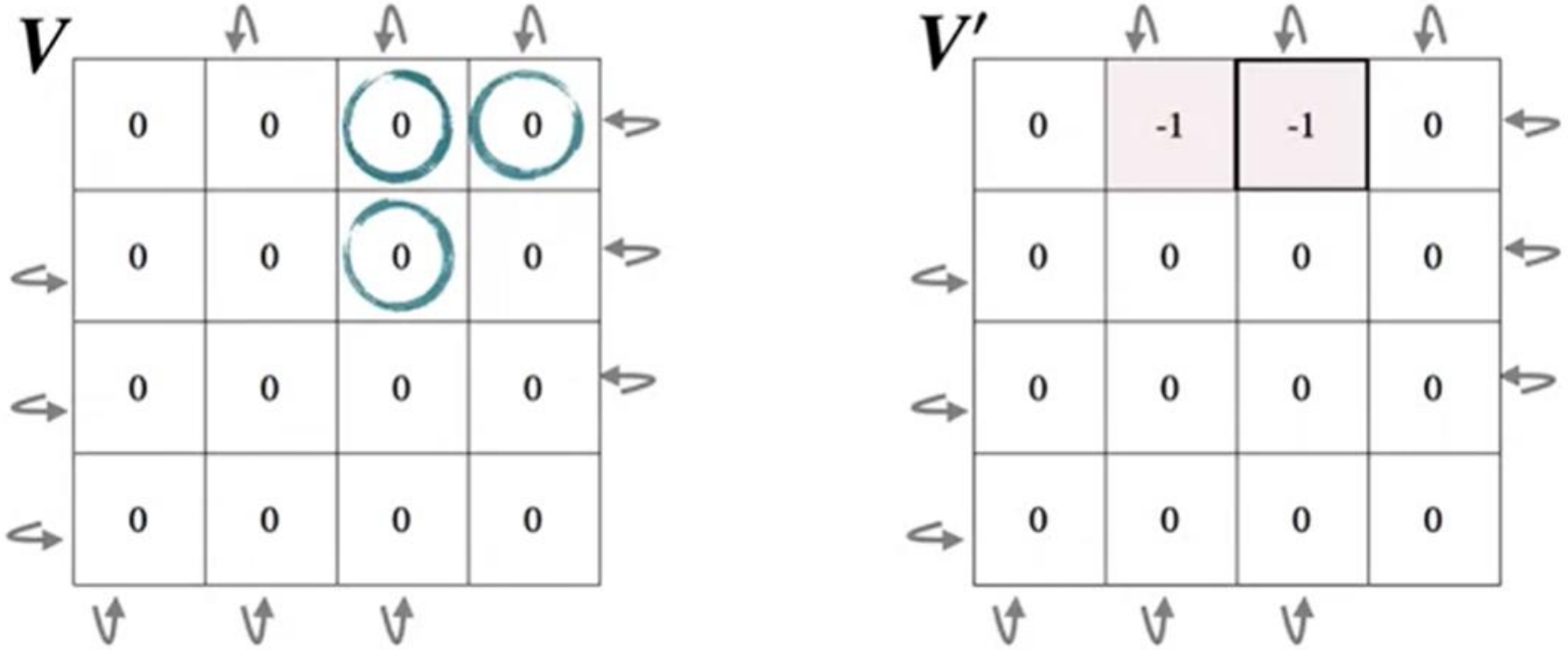
0.25



Example: Gridworld

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

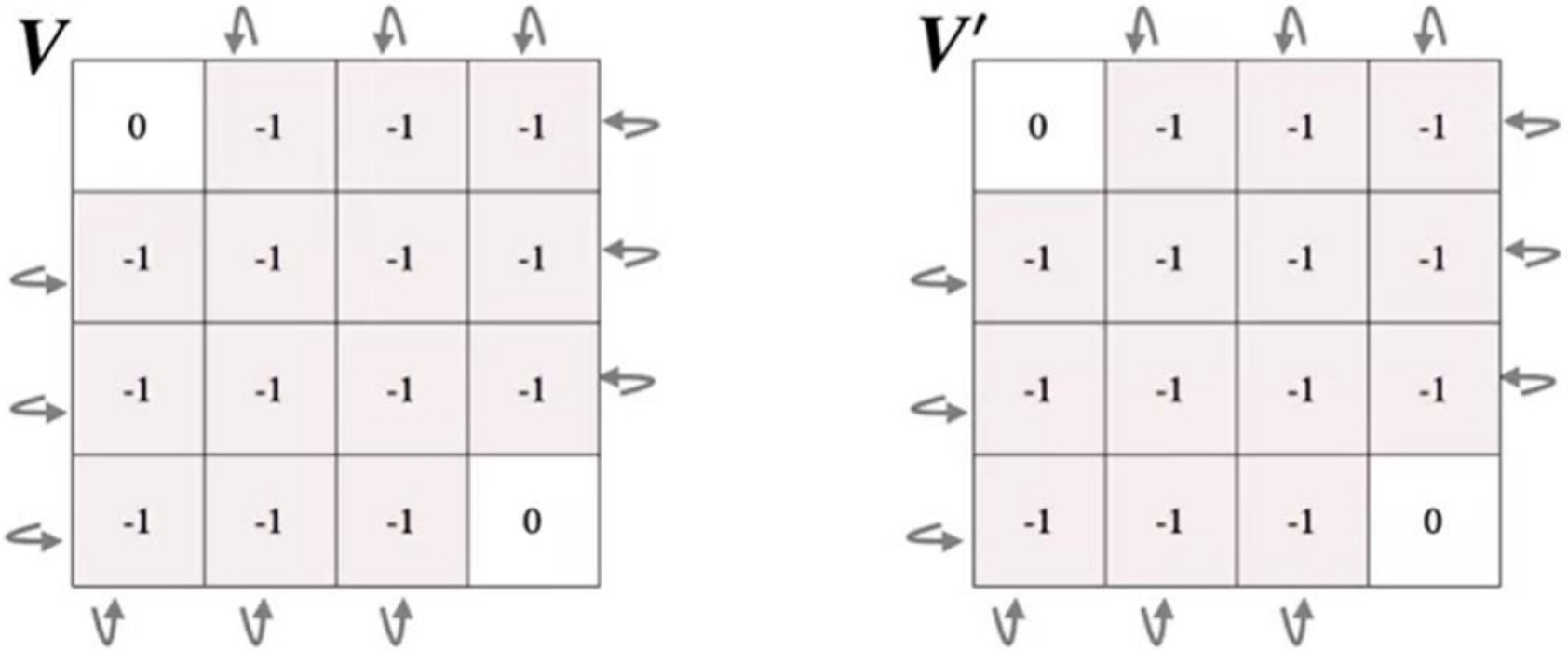
$$0.25 * (-1 + 0) + 0.25 * (-1 + 0) + 0.25 * (-1 + 0) + 0.25 * (-1 + 0) = -1$$



Example: Gridworld

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

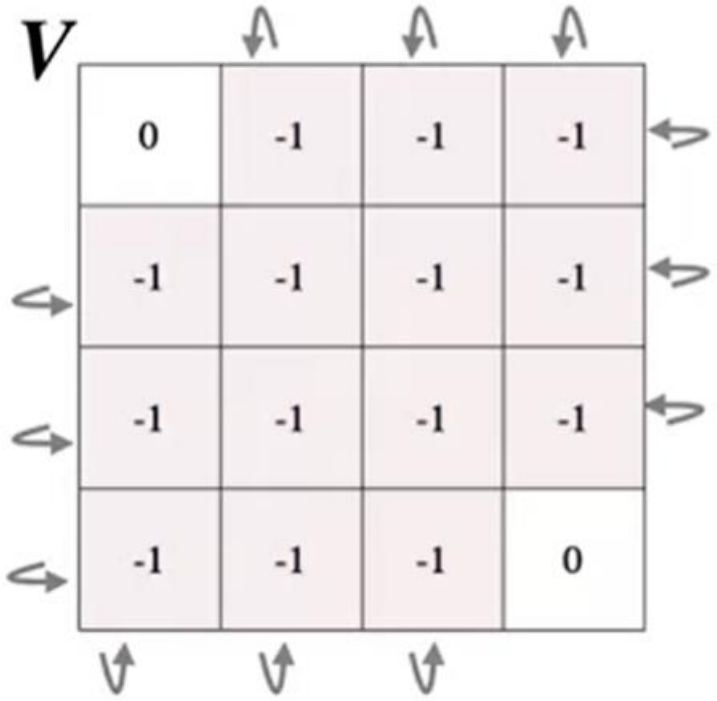
After the first full sweep…

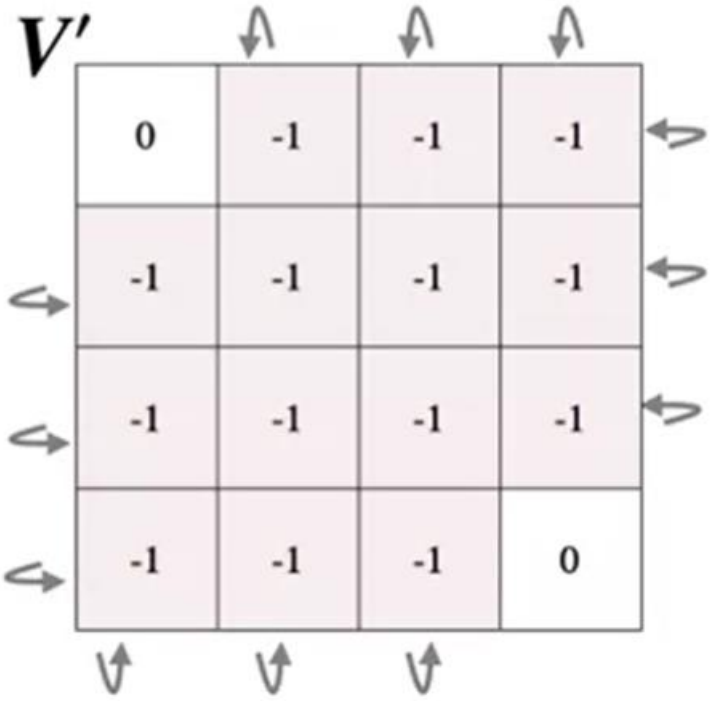


Example: Gridworld

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

$$v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_\pi$$





Iterative Policy Evaluation Pseudocode

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

Input π, the policy to be evaluated

$V \leftarrow \vec{0}, V' \leftarrow \vec{0}$

Loop:

$\Delta \leftarrow 0$

Loop for each $s \in \mathcal{S}$:

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

$$\Delta \leftarrow \max(\Delta, \mid V'(s) - V(s) \mid)$$

$V \leftarrow V'$

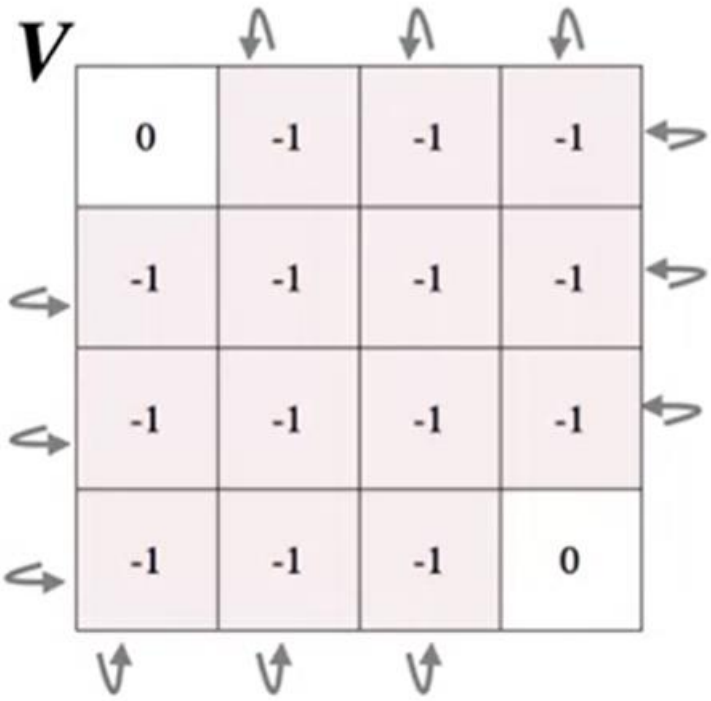
until $\Delta < \theta$ (a small positive number)

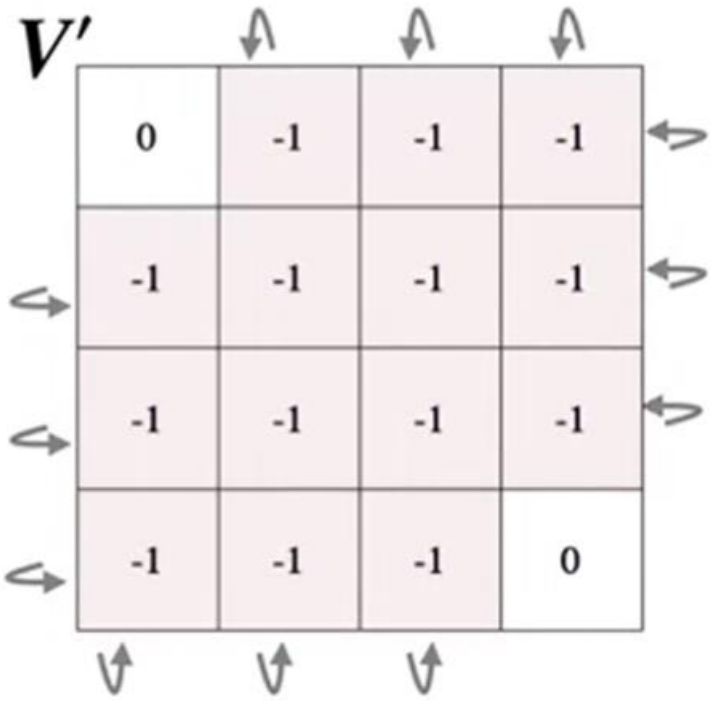
Output $V \approx v_\pi$

Example: Gridworld

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

$\theta = 0.001$

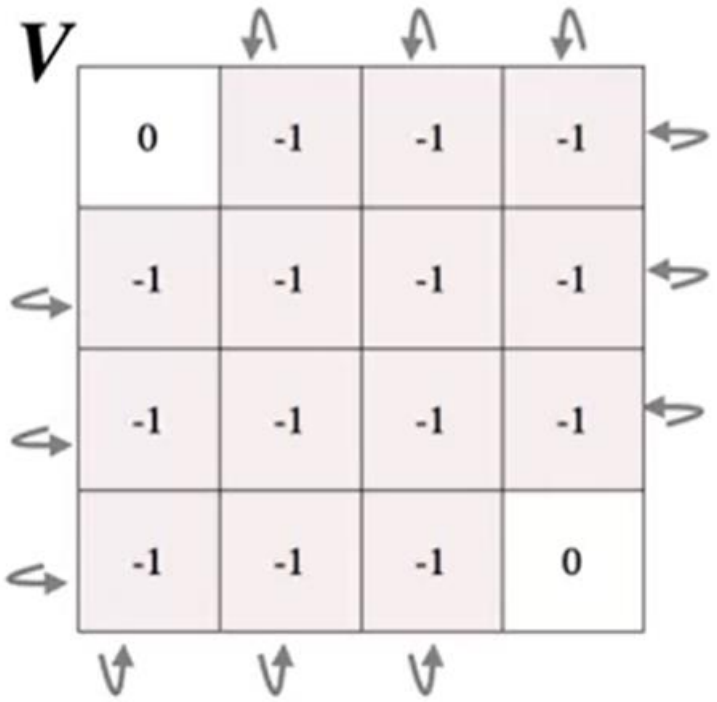


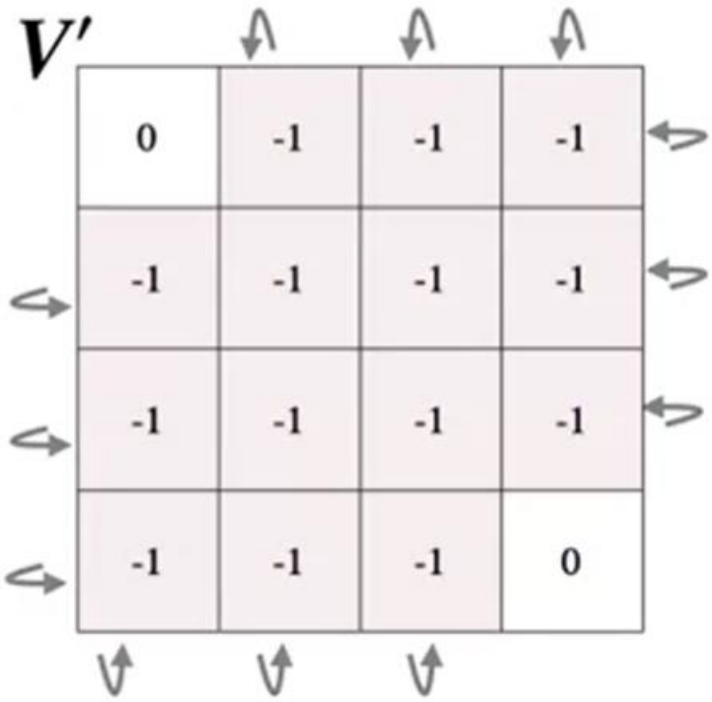


Example: Gridworld

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

$$\theta = 0.001 \quad \Delta = 1.0$$

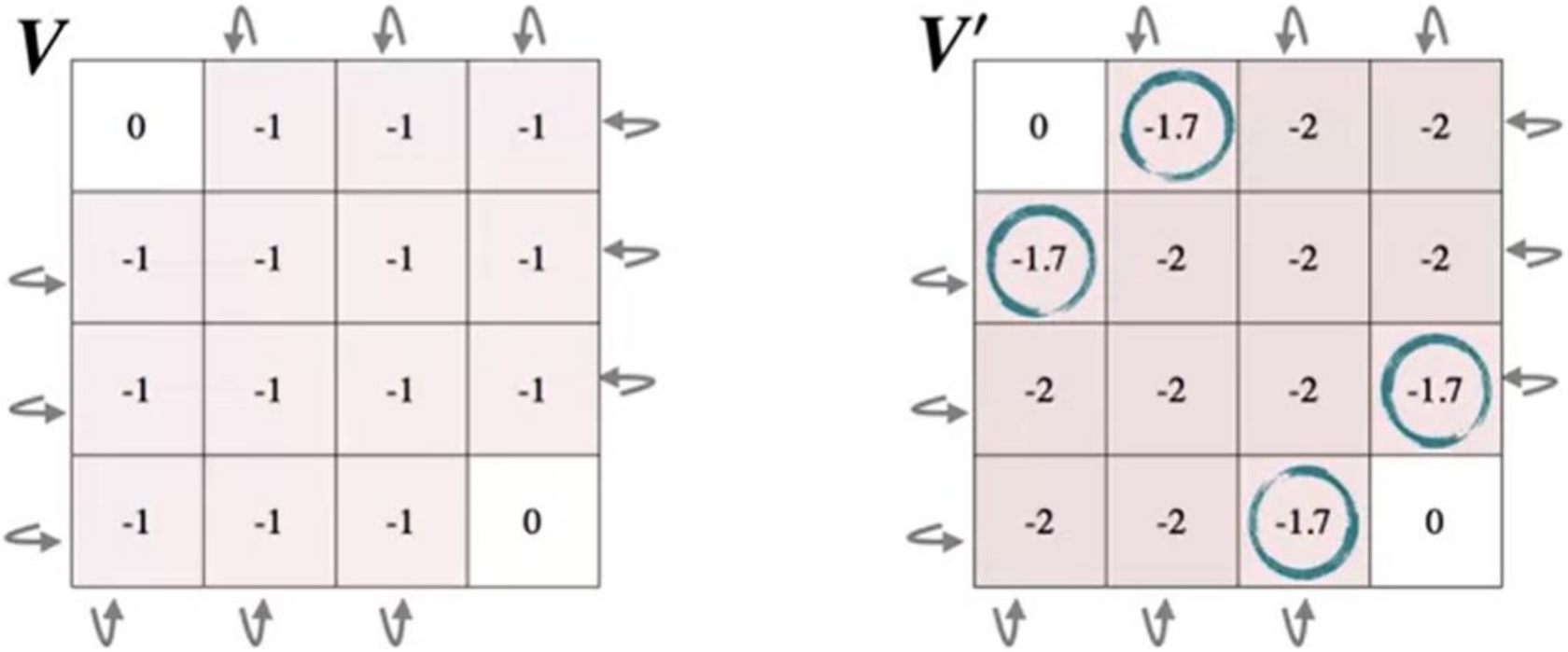




Example: Gridworld

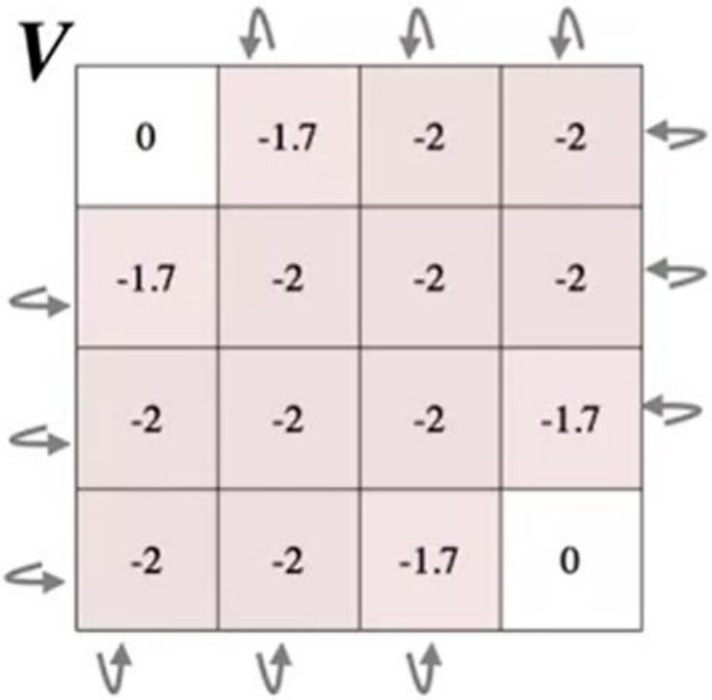
$$V'(s) \leftarrow \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a)\left[r + \gamma V(s')\right]$$

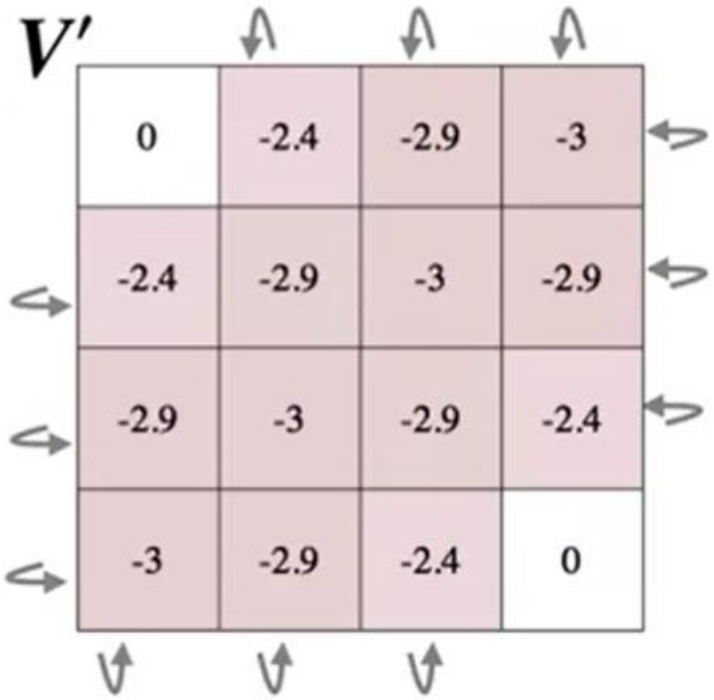
After the second full sweep…



Example: Gridworld

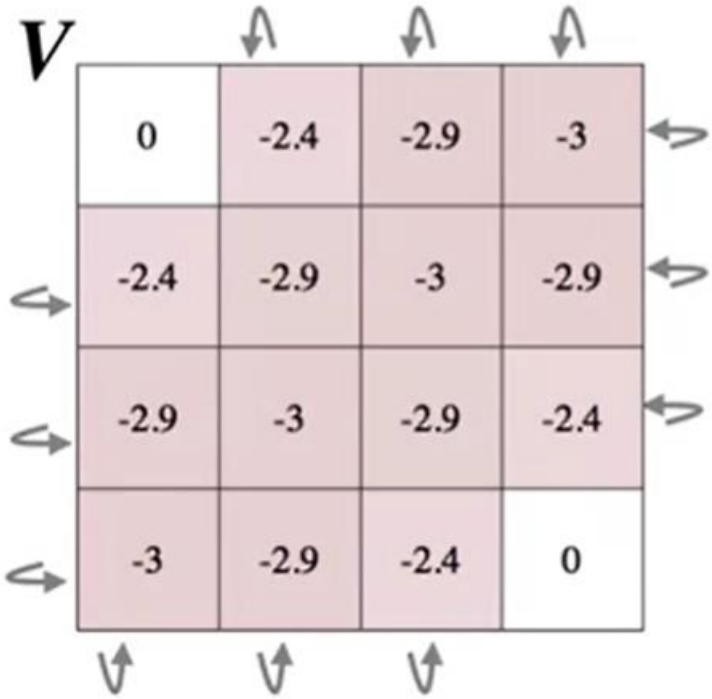
$$V'(s) \leftarrow \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a)\left[r + \gamma V(s')\right]$$

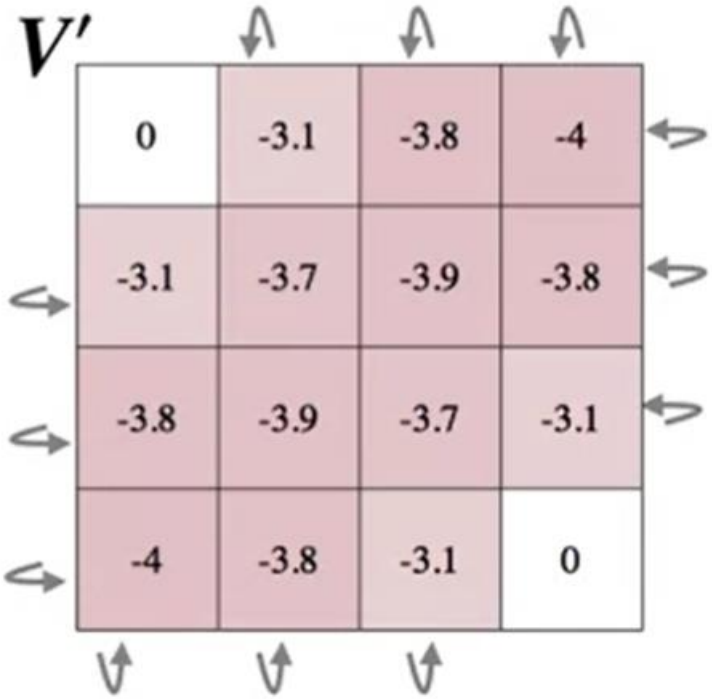




Example: Gridworld

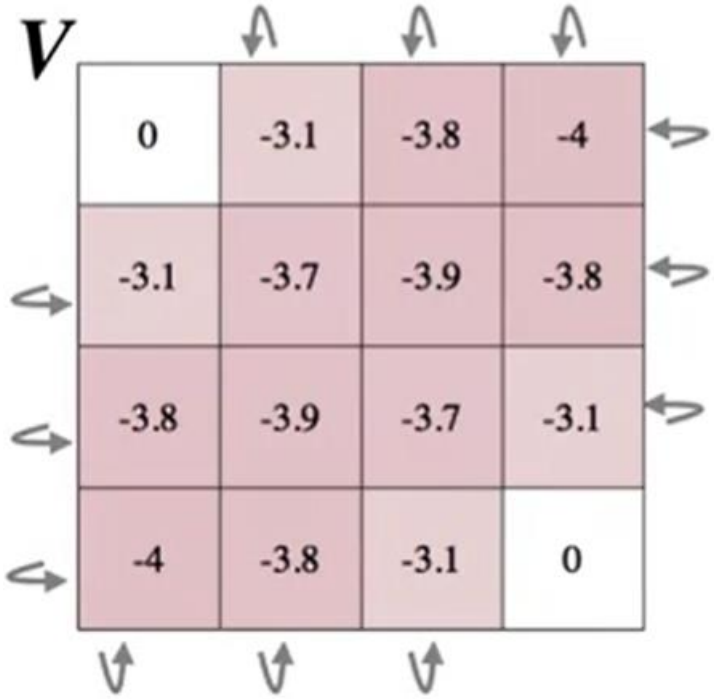
$$V'(s) \leftarrow \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a)\left[r + \gamma V(s')\right]$$

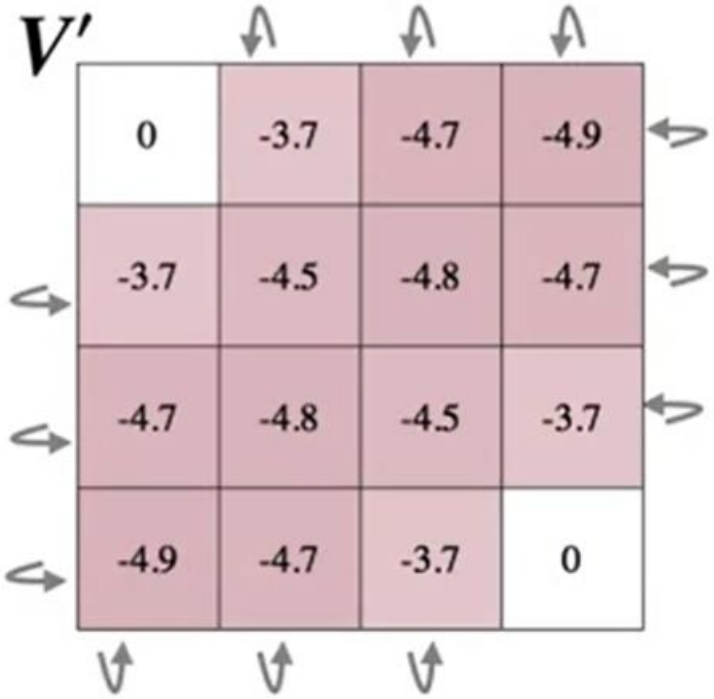




Example: Gridworld

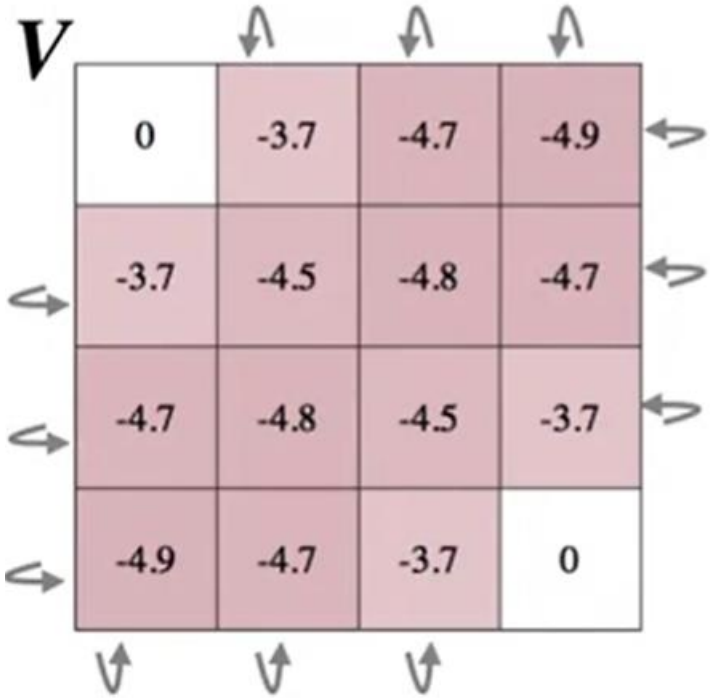
$$V'(s) \leftarrow \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a)\left[r + \gamma V(s')\right]$$

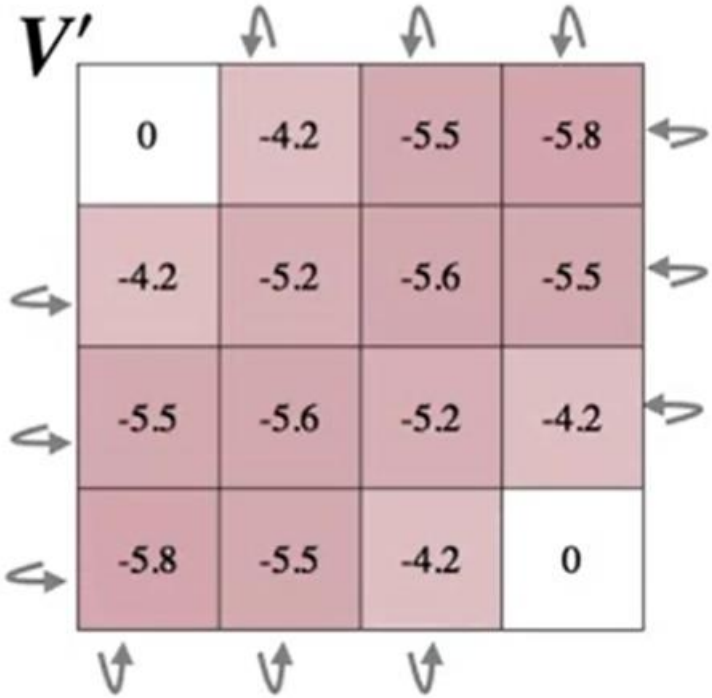




Example: Gridworld

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

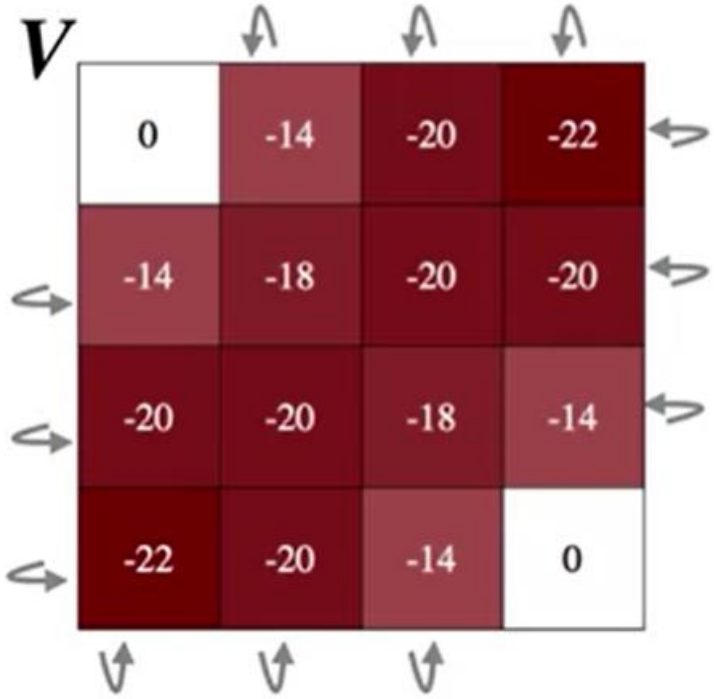


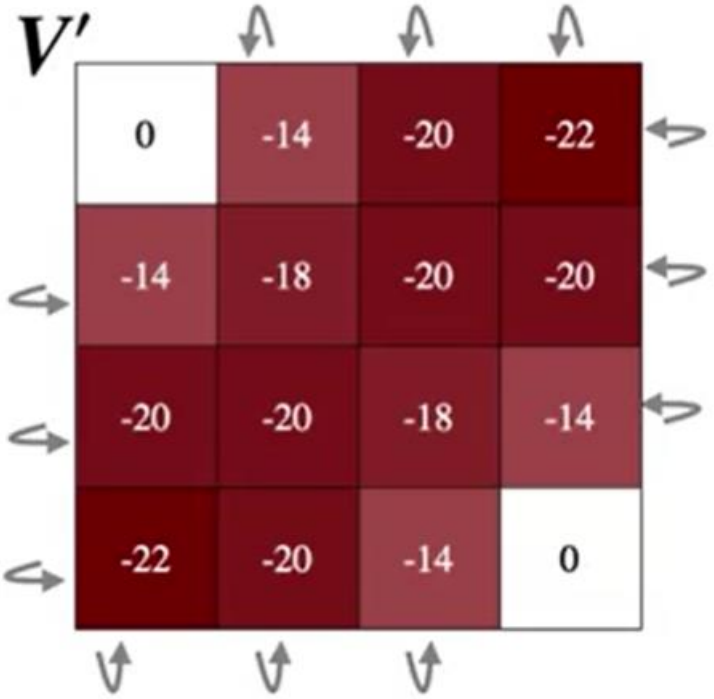


Example: Gridworld

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

$\Delta < 0.001$





Policy Improvement

- Last lecture we have seen that given v_*, we can find the optimal policy by choosing the Greedy action.

Recall that Greedy action

$$\pi_*(s) = \underset{a}{\text{argmax}} \sum_{s'} \sum_{r} p(s', r \mid s, a)\left[r + \gamma v_*(s')\right]$$

- The Greedy action maximizes the Bellman's optimally equation in each state.

- Imagine instead of the optimal value function, we select an action which is greedy with respect to the value function v_π of an arbitrary policy π.

? $\underset{a}{\text{argmax}} \sum_{s'} \sum_{r} p(s', r \mid s, a)\left[r + \gamma v_\pi(s')\right]$

$$\pi(s) = \underset{a}{\text{argmax}} \sum_{s'} \sum_{r} p(s', r \mid s, a)\left[r + \gamma v_\pi(s')\right] \text{ for all } s \in \mathcal{S}$$

→ v_π obeys the Bellman optimality equation

Policy Improvement Theorem

- Recall the definition of q_π. It tells you the value of a state if you take action A, and then follow policy π.

- Imagine we take action A according to π', and then follow policy π. If this action has higher value than the action under π, then π' must be better.

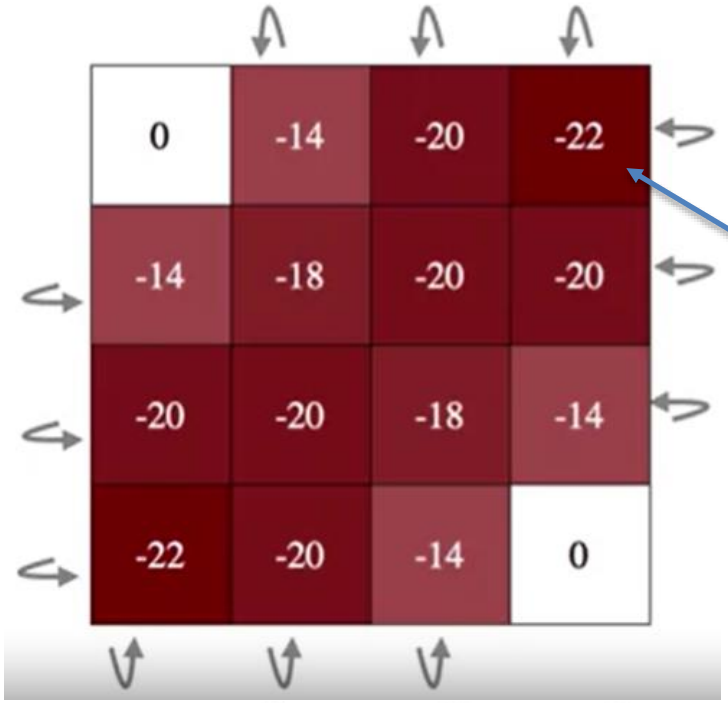
- The policy improvement theorem is as follows:

$$q_\pi(s, \pi'(s)) \geq q_\pi(s, \pi(s)) \forall s \in S \rightarrow \pi' \geq \pi$$

$$q_\pi(s, \pi'(s)) \geq q_\pi(s, \pi(s)) \text{For at least one } s \in S \rightarrow \pi' \geq \pi$$

The new policy is a strict improvement over π unless π is already optimal

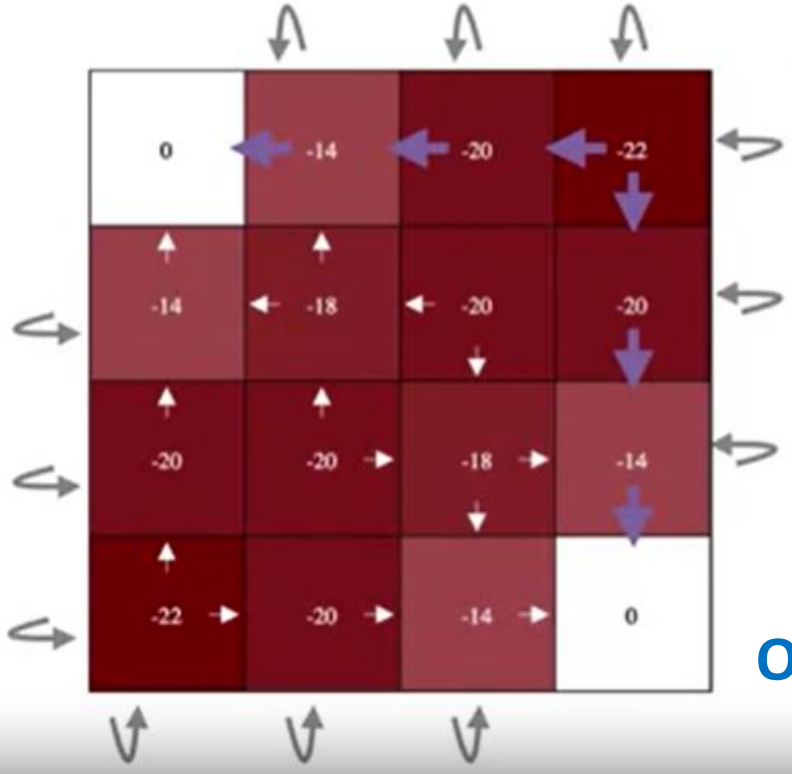
Example



v_π

$R = -1$

π 25% 25% 25% 25%



for all $s \in \mathcal{S}$

$$\pi'(s) = \underset{a}{\text{argmax}} \sum_{s'} \sum_{r} p(s', r \mid s, a)\left[r + \gamma v_\pi(s')\right]$$

$\pi' > \pi$

Optimal

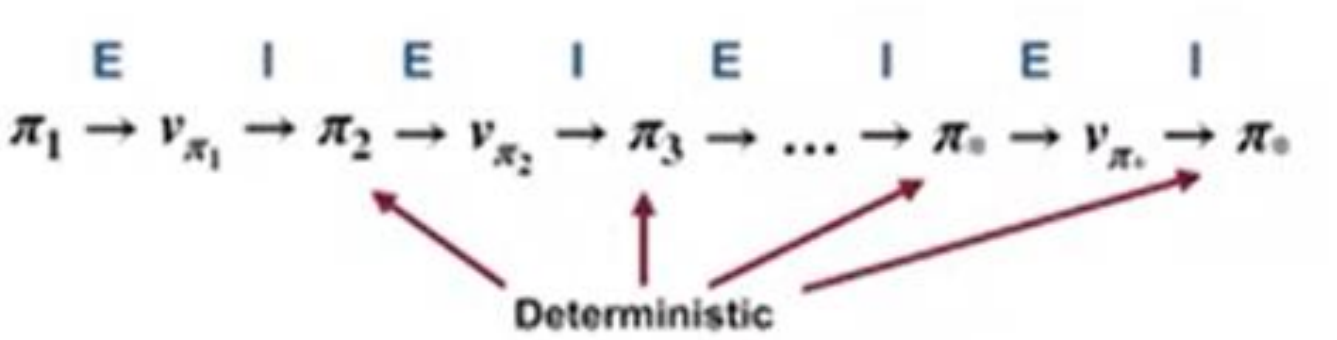
$R = -1$

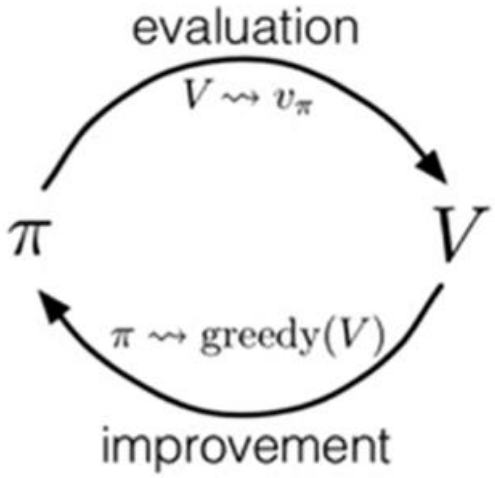
π 25% 25% 25% 25%

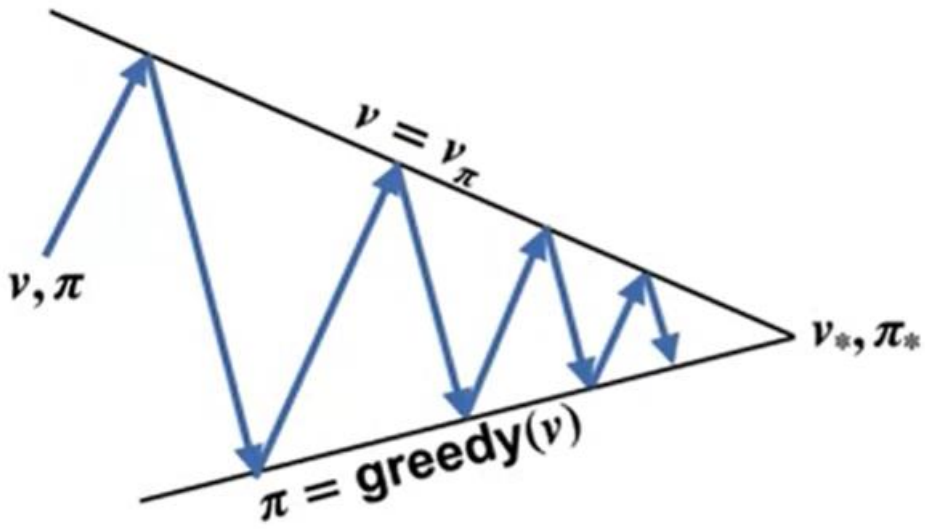
Policy Iteration

- We will show now how we can use the policy improvement theorem to find the optimal policy by iteratively evaluating a sequence of policies.

- Policy iteration consists of two distinct steps repeated over and over, evaluation and improvement.







Policy Iteration Pseudocode

- We initialize V and π in any way we like for each state s. Next, we call iterative policy evaluation to make V reflect the value of π. Then, in each state, we set π to select the maximizing action under the value function. If this procedure changes the selected action in any state, we note that the policy is still changing, and set policy stable to force. After completing step 3, we check if the policy is stable. If not, we carry on and evaluate the new policy.

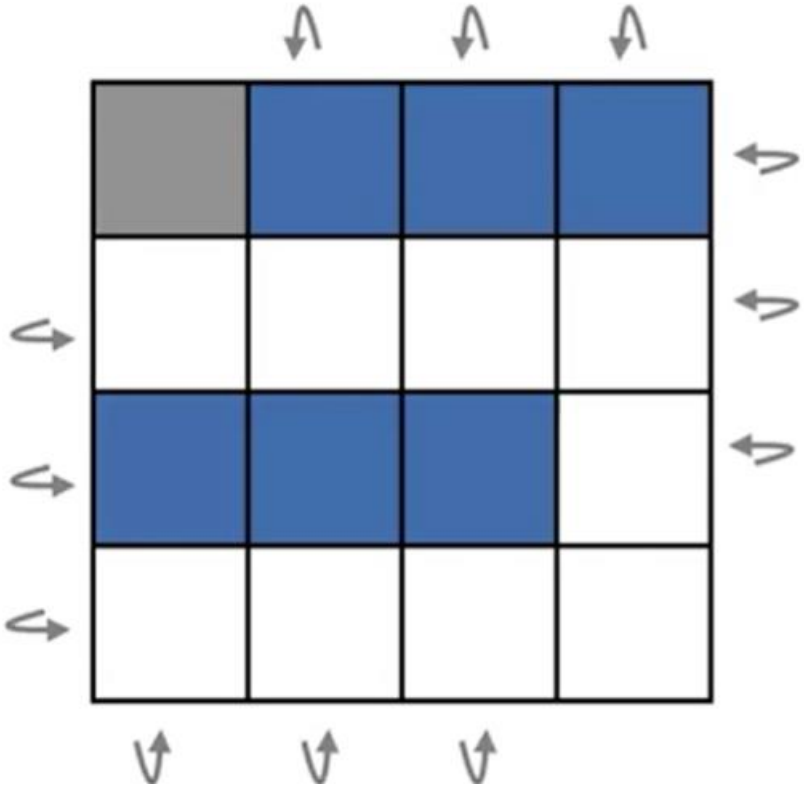
Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

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 $s \in \mathcal{S}$:
         $v \leftarrow V(s)$
         $V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))[r + \gamma V(s')]$
         $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   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 $\leftarrow \pi(s)$
      $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$
      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

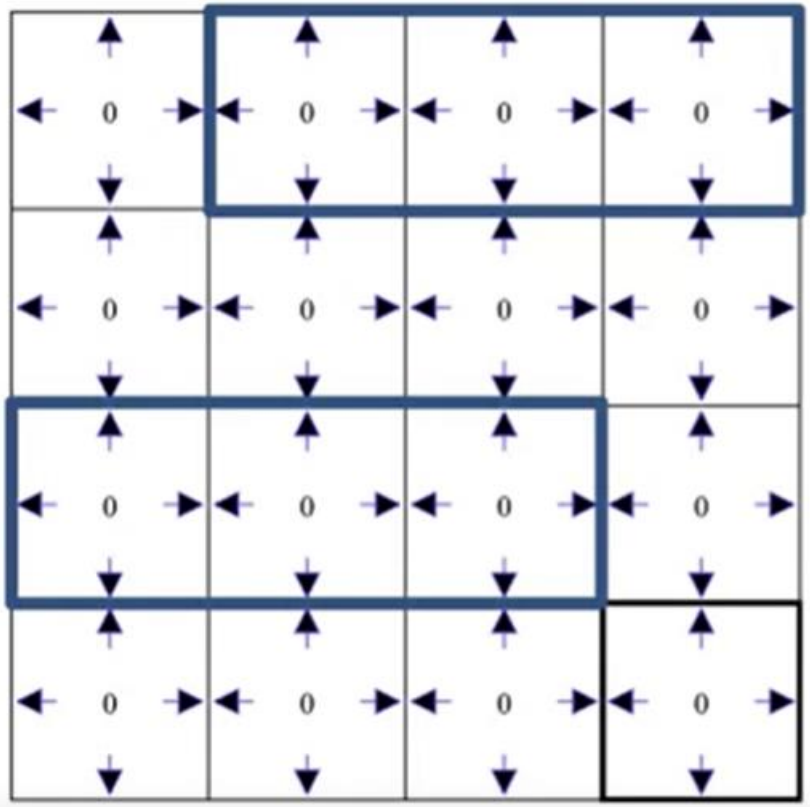
Example: Gridworld



$$R = \begin{cases} -10 & \text{Blue states} \\ -1 & \text{Other states} \end{cases}$$

$$\gamma = 1$$

Example: Gridworld

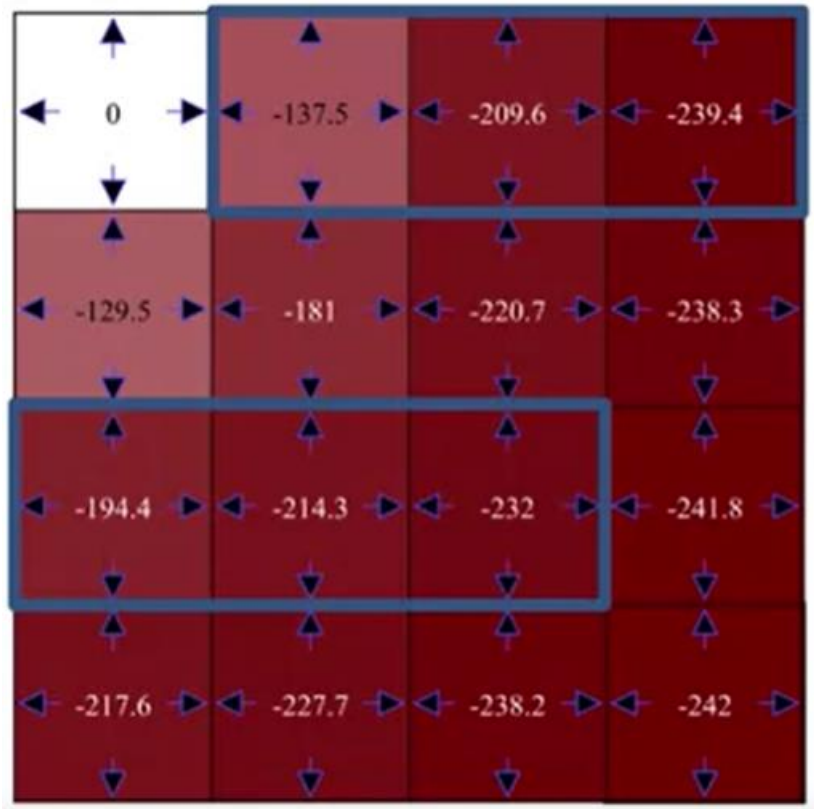


Policy Iteration

Evaluation

Improvement

Example: Gridworld

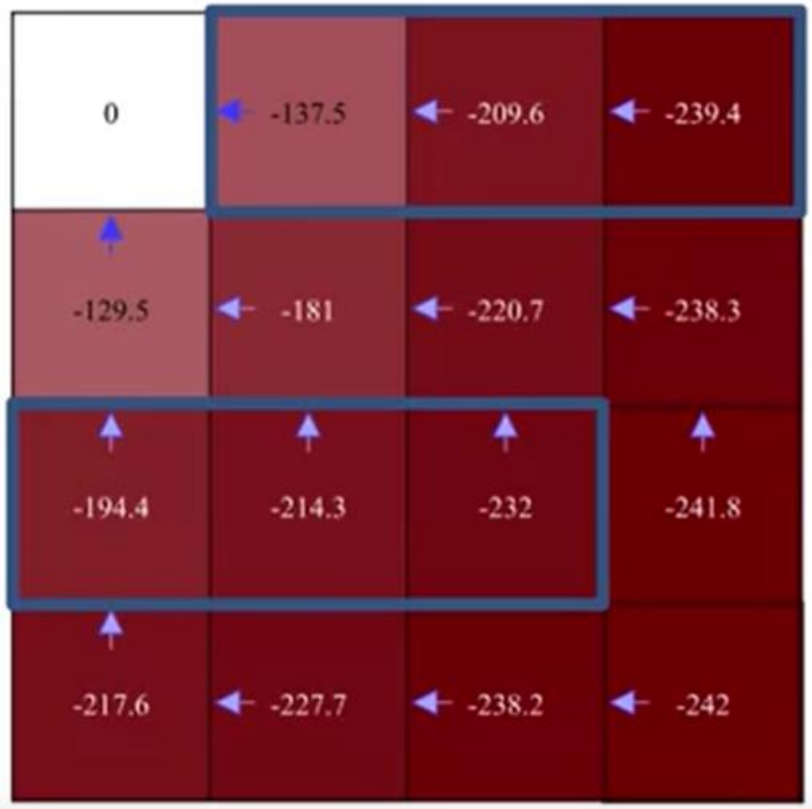


Policy Iteration

Evaluation

Improvement

Example: Gridworld

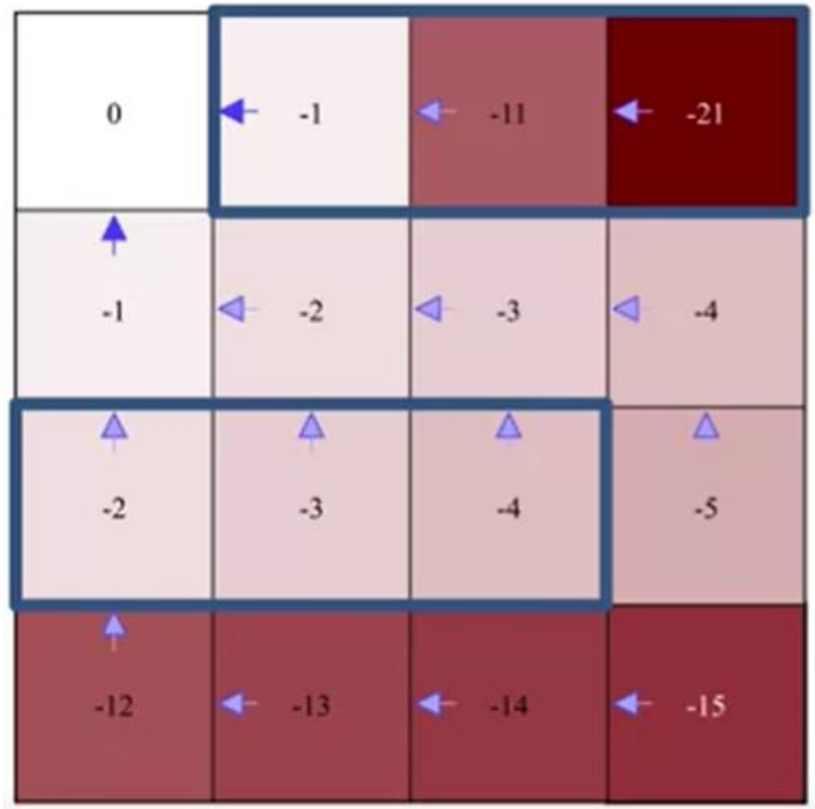


Policy Iteration

Evaluation

Improvement

Example: Gridworld

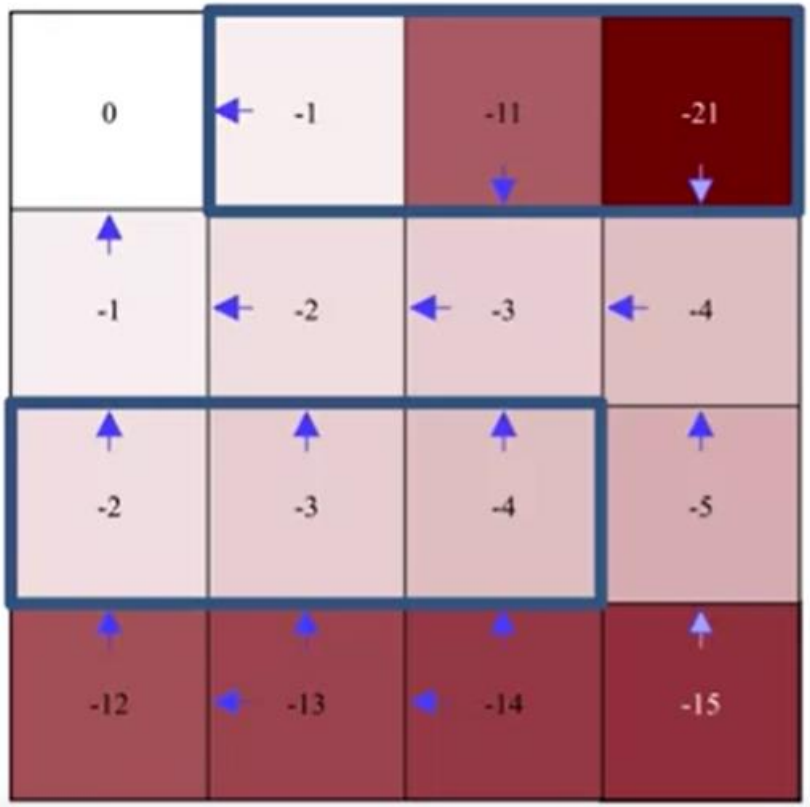


Policy Iteration

Evaluation

Improvement

Example: Gridworld

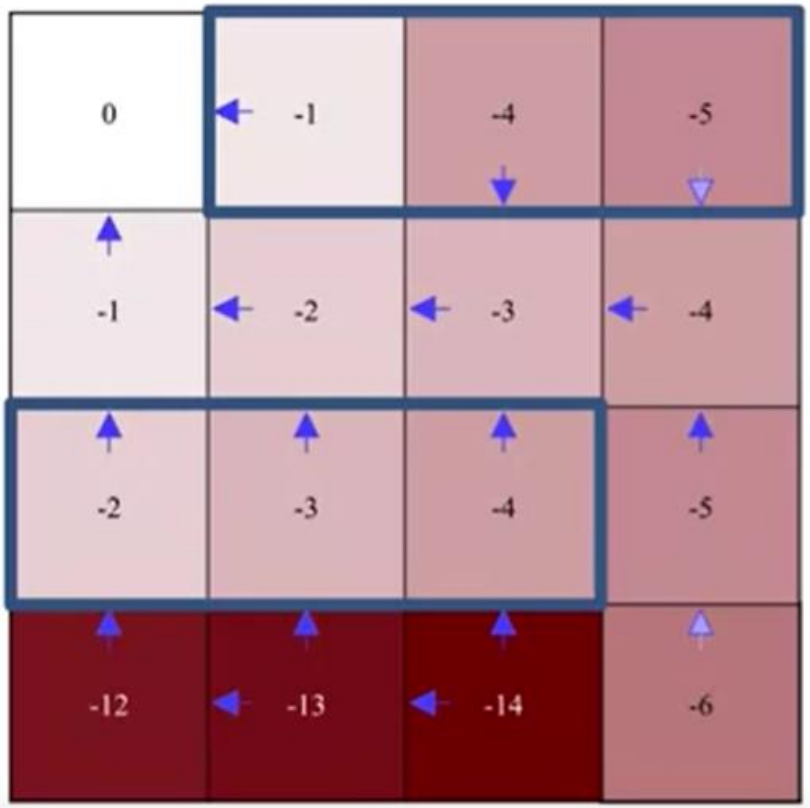


Policy Iteration

Evaluation

Improvement

Example: Gridworld

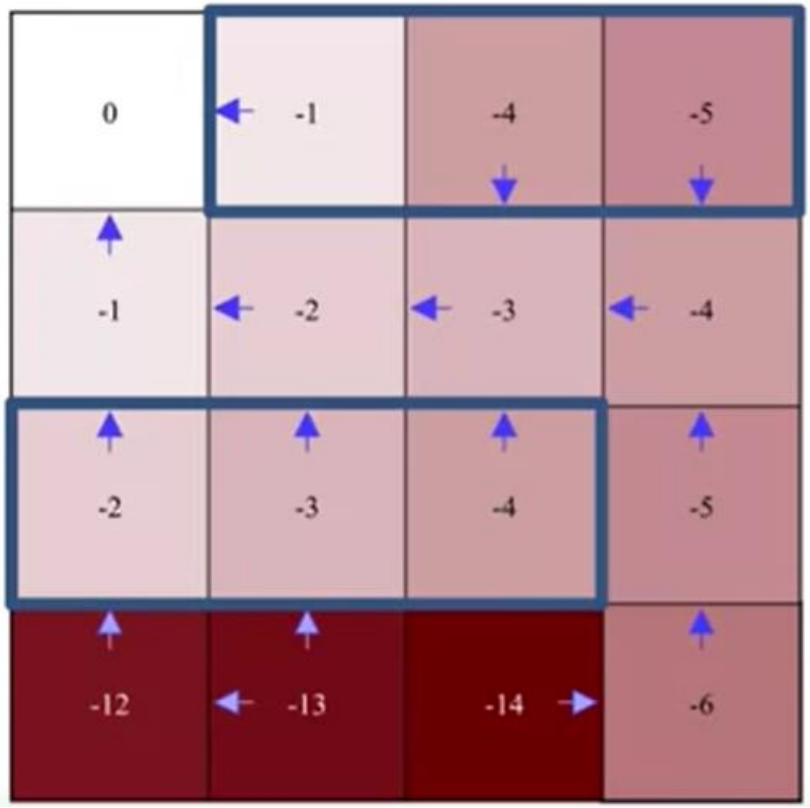


Policy Iteration

Evaluation

Improvement

Example: Gridworld

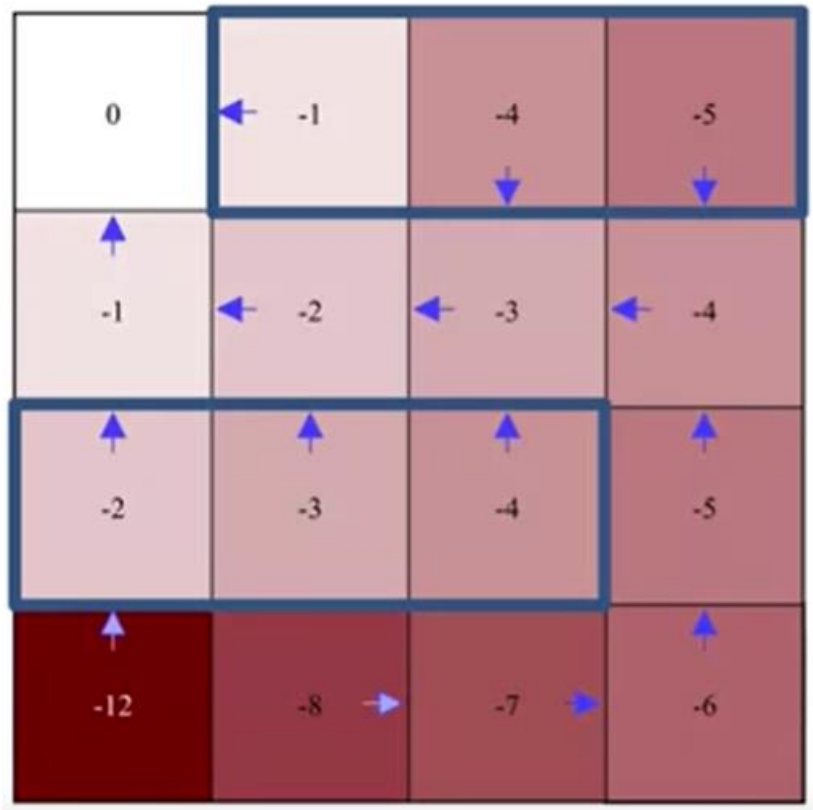


Policy Iteration

Evaluation

Improvement

Example: Gridworld

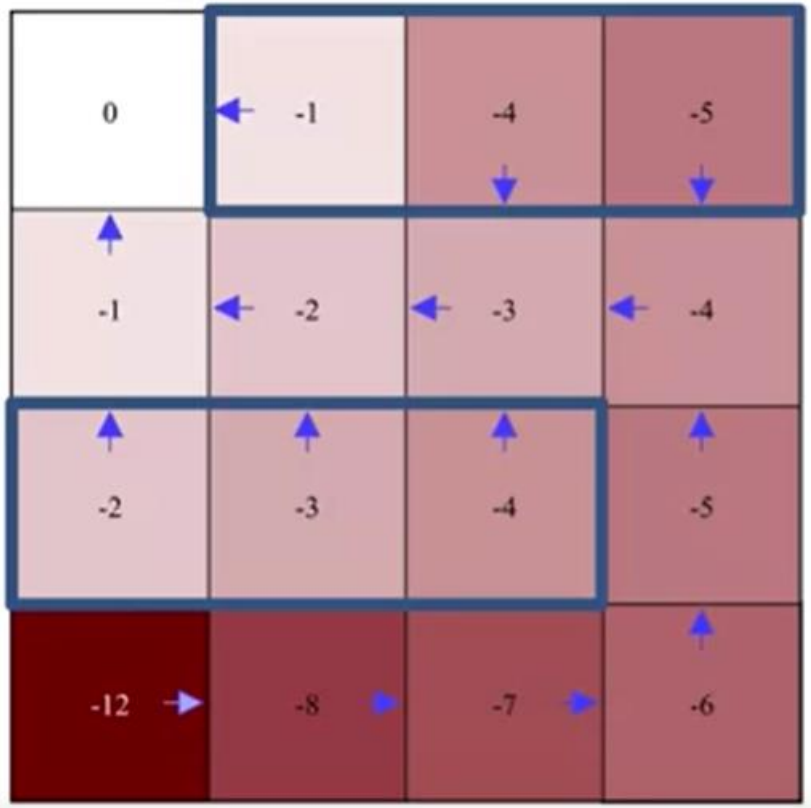


Policy Iteration

Evaluation

Improvement

Example: Gridworld

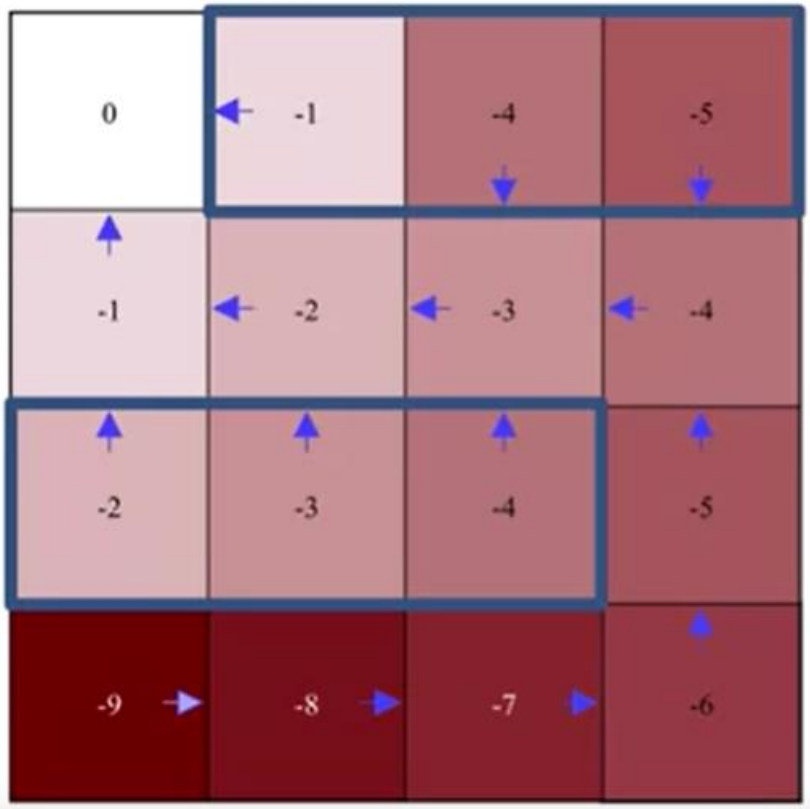


Policy Iteration

Evaluation

Improvement

Example: Gridworld

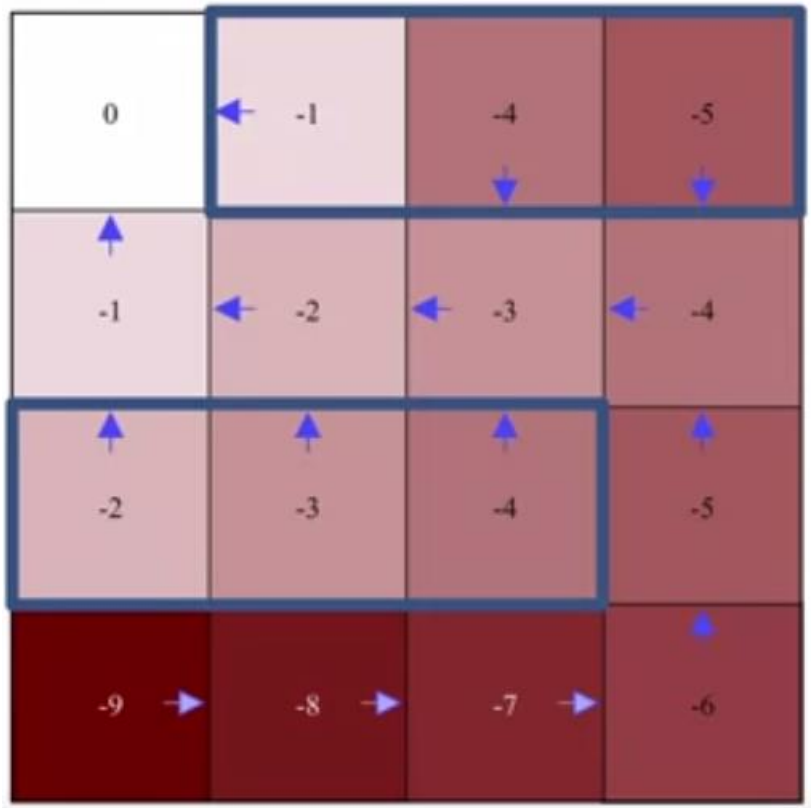


Policy Iteration

Evaluation

Improvement

Example: Gridworld



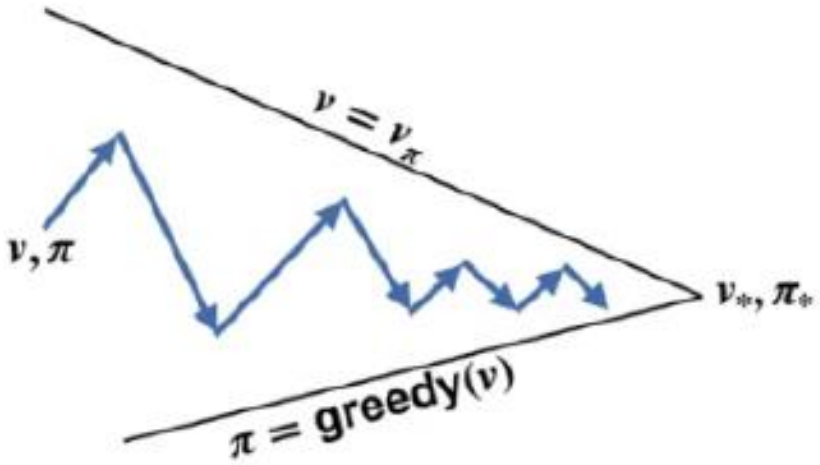
Policy Iteration

Evaluation

Improvement

Flexibility of the Policy Iteration Framework

- The policy iteration algorithm runs each step all the way to completion.
- Imagine instead, we follow a trajectory like this.



- Each evaluation step brings our estimate a little closer to the value of the current policy but not all the way.
- Each policy improvement step makes our policy a little more greedy, but not totally greedy.
- Intuitively, this process should still make progress towards the optimal policy and value function.
- We use the term generalized policy iteration to refer to all the ways we can interleave policy evaluation and policy improvement.

Generalized Policy Iteration

- The term Generalized Policy Iteration refers to all the ways we can interleave policy evaluation and policy improvement.

- Value Iteration is generalized policy iteration algorithm that combines both policy evaluation and improvement into a single update.
- In value iteration, we do not run policy evaluation to completion, we sweep over all the states and greedify with respect to the current value function.

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

Output a deterministic policy, $\pi \approx \pi_*$, such that
$\pi(s) = \text{argmax}_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$

Generalized Policy Iteration

- The term Generalized Policy Iteration refers to all the ways we can interleave policy evaluation and policy improvement.

- Value Iteration is generalized policy iteration algorithm that combines both policy evaluation and improvement into a single update.
- In value iteration, we do not run policy evaluation to completion, we sweep over all the states and greedify with respect to the current value function.

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

Output a deterministic policy, $\pi \approx \pi_*$, such that
$\pi(s) = \text{argmax}_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$