Week 2: Dynamic Programming

Dynamic Programming, Iterative Methods, Intro to Monte Carlo

Go to our Linktree to follow our **socials**

Week 1 Review

Goal of Reinforcement Learning

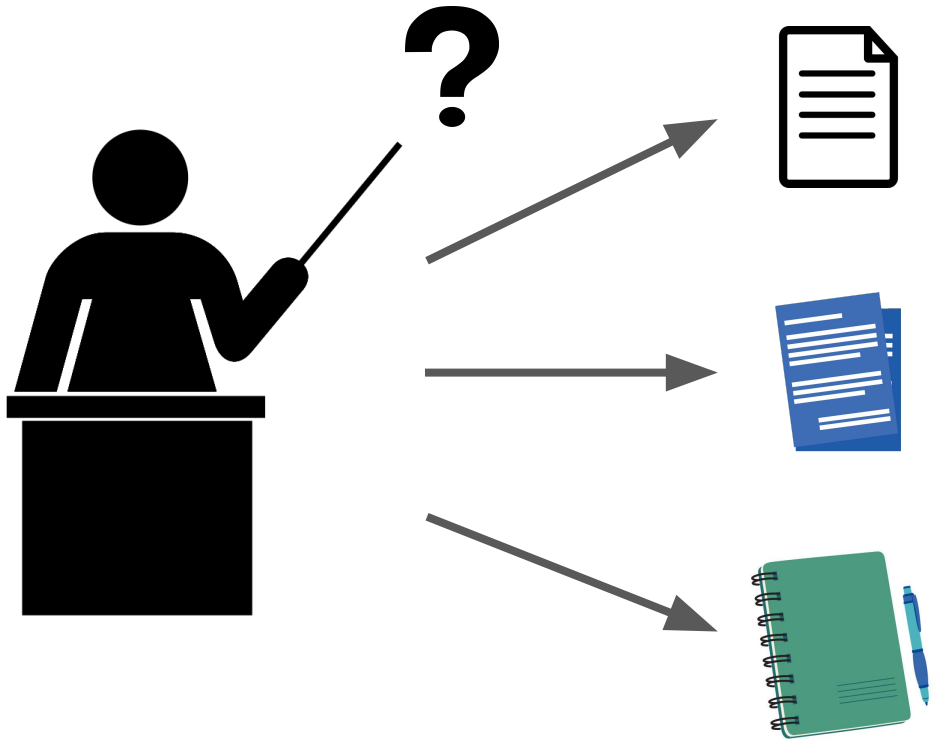
A learning agent interacts overtime with its environment in order to achieve a goal. The agent should:

- Be able to sense the state of the environment
- Have clear goals relating to the state

Maximize expected return

$$\mathbb{E}[G_t] = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots]$$

K-armed Bandits



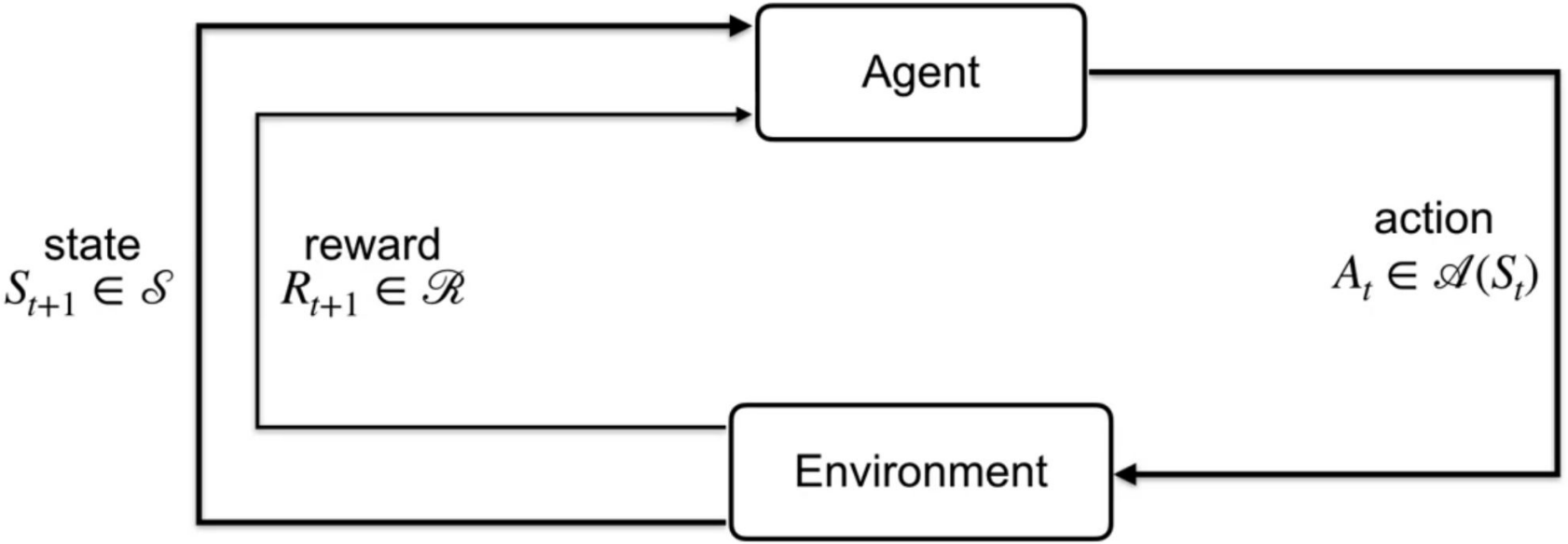
Epsilon-Greedy Action Selection

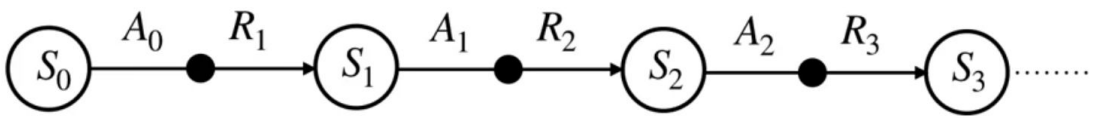
- **Exploitation:** Improve short-term benefits
- **Exploration:** Improve knowledge for long-term benefit
  - Epsilon is how often you "explore"

$$A_t \leftarrow \begin{cases} \underset{a}{\text{argmax}}\ Q_t(a) & \textbf{with probability } 1-\epsilon \\ a \sim Uniform(\{a_1 \ldots a_k\}) & \textbf{with probability } \epsilon \end{cases}$$

$$\epsilon \in [0, 1)$$

Markov Decision Process





Trajectory of experience

The RL Agent

A reinforcement learning agent consists of a combination of the following components:

- Policy: the agent's behavior function (it's brain)
- Value function: the agent's belief of how good each state and/or action is
- Model (out of scope): the agent's representation of the dynamics of the MDP

Value Functions

State Value Function

$$V_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots | S_t = s]$$

State-Action Value Function

$$Q_\pi(s, a) = \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots | S_t = s, A_t = a]$$

Classifying RL Agents

RL agents may contain a mixture of a policy, value function(s), and model:

- Value-based: Learns value function to generate optimal policy
- Policy-based: Learns policy directly (policy gradient)
- Actor-Critic: Policy and value function

Dynamic Programming and Bellman Equations

Dynamic Programming

A method for solving complex problems by breaking them down into subproblems:

- Solve the subproblems individually, and then combine to form final solution
- The optimal solution can be decomposed into subproblems

Dynamic Programming

Fibonacci Sequence: f(n) = f(n - 2) + f(n - 1), f(0) = f(1) = 1

- f(4) = f(2) + f(3) = 5
- f(3) = f(1) + f(2) = 3
- f(2) = f(1) + f(0) = 2

Bellman Equations

Recursion, basic building block to solve MDP

- Finding optimal policy and value functions

What is the **maximum reward** an agent can receive if they take the **optimal action** now and for all **future actions**?

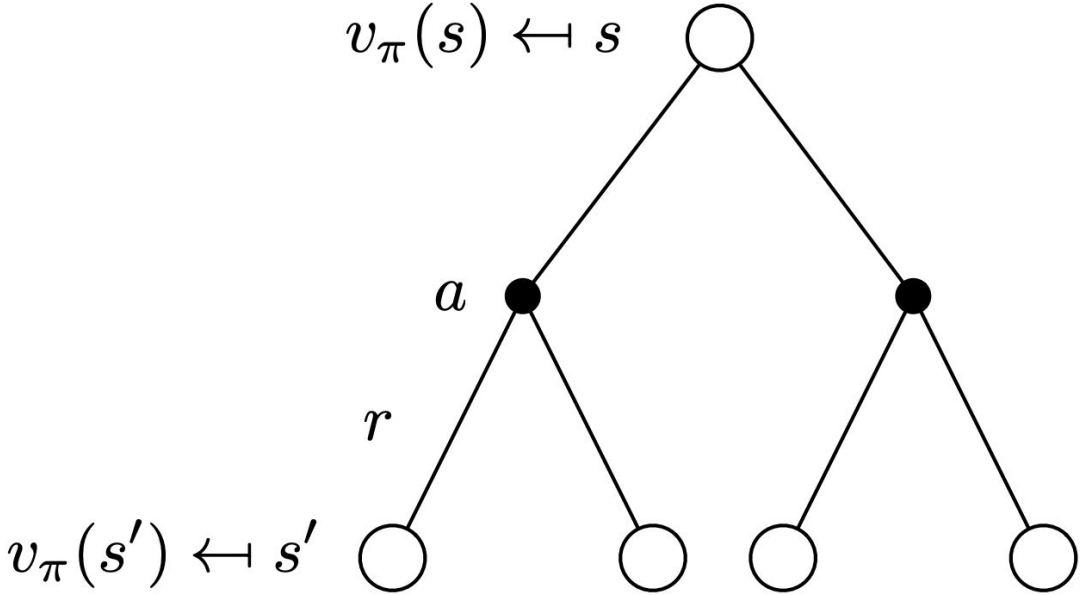
State Value Bellman Equation

Recall:

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

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

State Value Bellman Equation



State-Action Value Bellman Equation

$$Q_\pi(s,a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a]$$

$$= \sum_{s'} \sum_{r} p(s', r | s, a)[r + \gamma \mathbb{E}_\pi[G_{t+1} | S_{t+1} = s']]$$

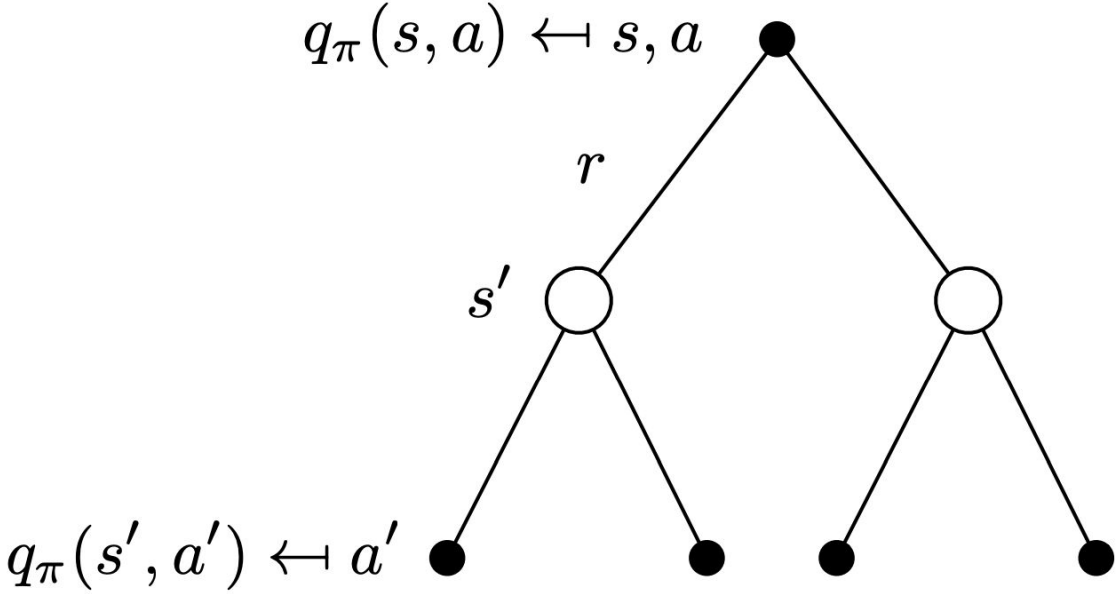
$$= \sum_{s'} \sum_{r} p(s', r | s, a)[r + \gamma \sum_{a'} \pi(a'|s') \mathbb{E}_\pi[G_{t+1} | S_{t+1} = s', A_{t+1} = a']]$$

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

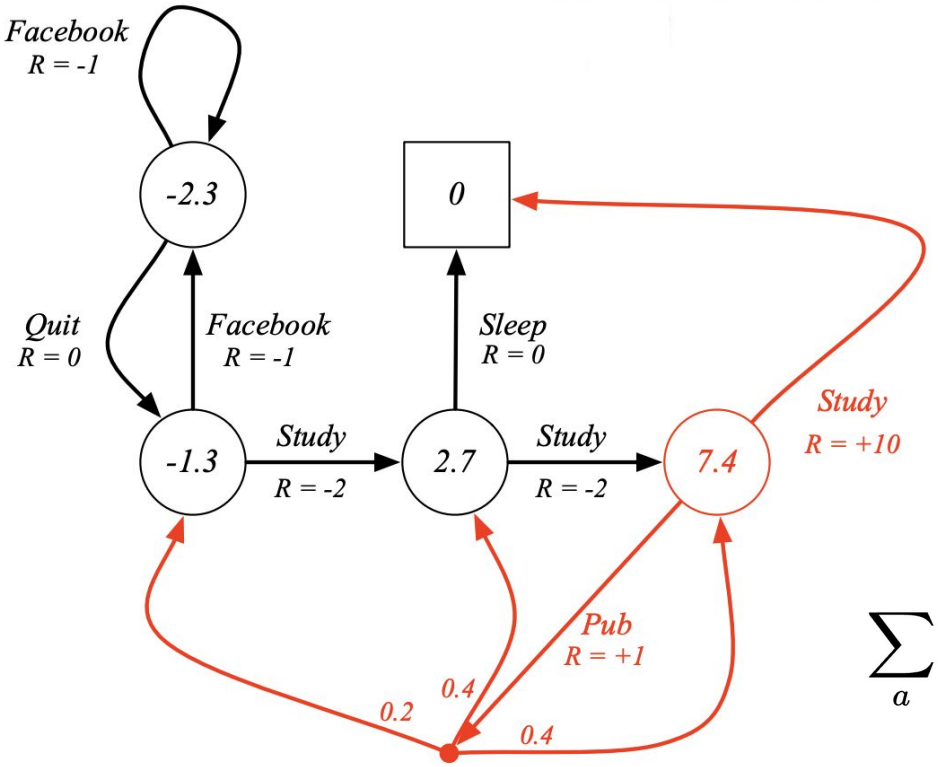
Recall:

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

State-Action Value Bellman Equation



State Value Bellman Equation Example



Assume a uniform random policy, where actions are selection 50/50

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

Optimal Policies and Optimal Value Functions

Optimal Policies

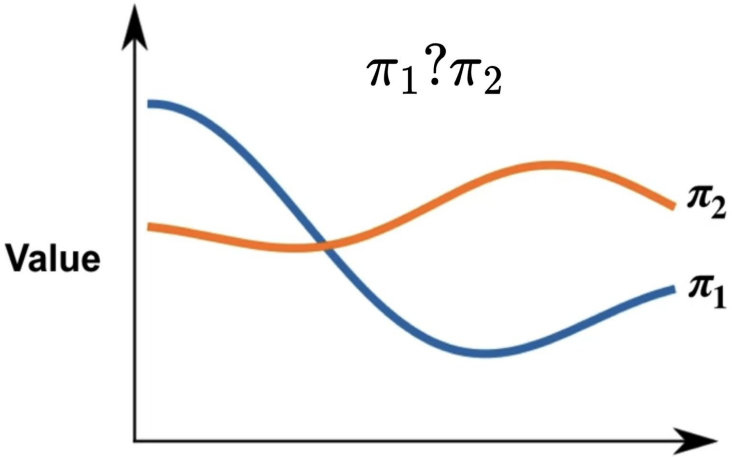
A **policy π** is better than or equal to another policy **π'** if its expected return is greater than or equal to that of π' for all states

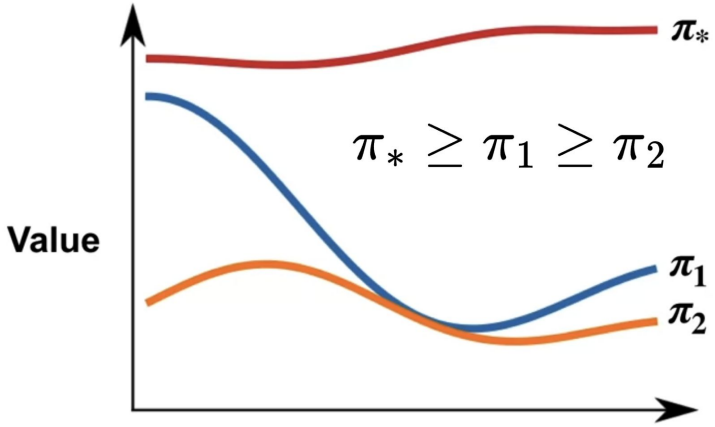
$$\pi \geq \pi' \iff V_\pi(s) \geq V_{\pi'}(s) \text{ for all } s \in \mathcal{S}$$

An optimal policy, denoted by π_* is better than or equal to all other policies

- Possibly more than one optimal

Optimal Policies





Optimal Value Functions

$$\pi \geq \pi' \iff V_\pi(s) \geq V_{\pi'}(s) \text{ for all } s \in \mathcal{S}$$

All optimal policies share the same optimal state-value function

$$V_*(s) = \mathbb{E}_{\pi_*}[G_t | S_t = s] = \max_\pi V_\pi(s)$$

All optimal policies also share the same optimal state-action value function

$$Q_*(s, a) = \max_\pi Q_\pi(s, a)$$

Bellman Optimality Equation for V_*

Recall:

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

$$V_*(s) = \sum_a \pi_*(a|s) \sum_{s'} \sum_r p(s', r|s, a)[r + \gamma V_*(s')]$$

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

Bellman Optimality Equation for State Value Function

Bellman Optimality Equation for Q_*

Recall:

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

$$Q_*(s,a) = \sum_{s'} \sum_{r} p(s', r|s, a)[r + \gamma \sum_{a'} \pi_*(a'|s') Q_*(s', a')]$$

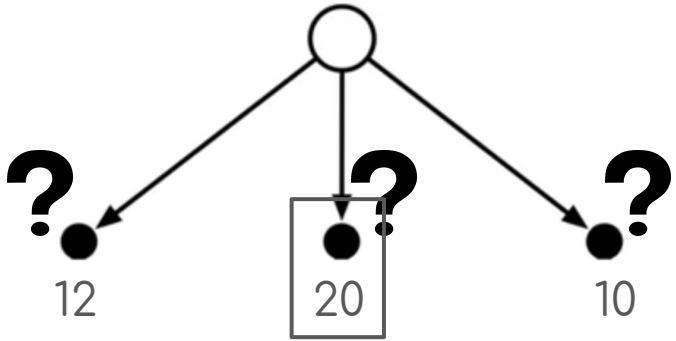
$$Q_*(s,a) = \sum_{s'} \sum_{r} p(s', r|s, a)[r + \gamma \max_{a'} Q_*(s', a')]$$

Bellman Optimality Equation for State-Action Value Function

Finding the Optimal Policy with V_*

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

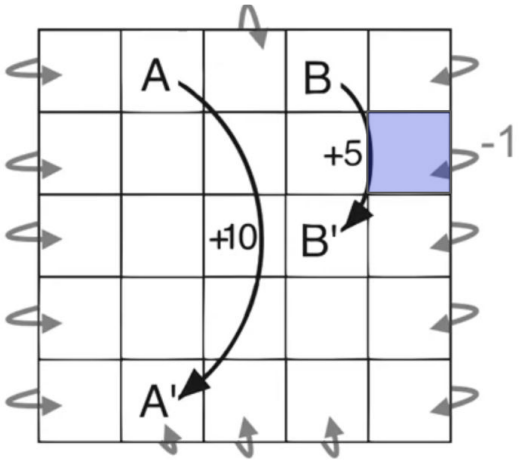
$$\pi_*(s) = \arg\max_a \boxed{\sum_{s'} \sum_r p(s', r|s, a)[r + \gamma V_*(s')]}$$



Finding Optimal Policy with Gridworld

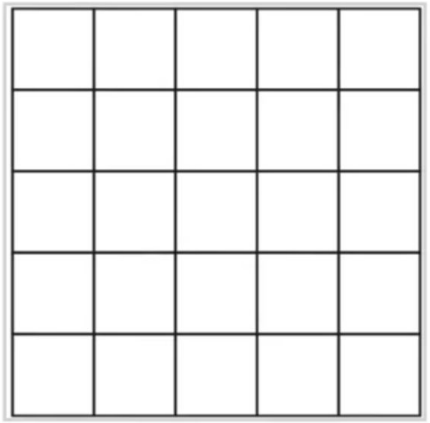
$$\pi_*(s) = \arg\max_a \sum_{s'} \sum_r p(s', r|s, a)[r + \gamma V_*(s')]$$

γ = 0.9



22.0	24.4	22.0	19.4	17.5
19.8	22.0	19.8	17.8	16.0
17.8	19.8	17.8	16.0	14.4
16.0	17.8	16.0	14.4	13.0
14.4	16.0	14.4	13.0	11.7

v_*

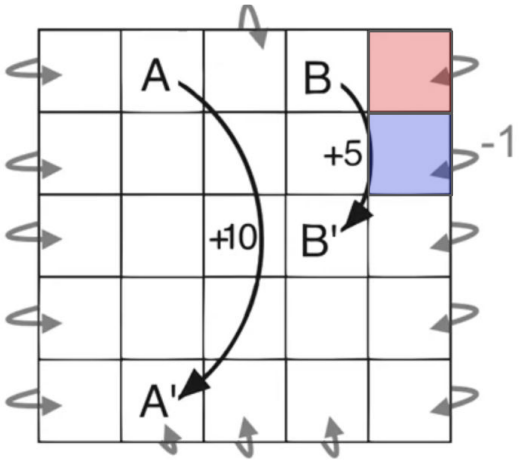


π_*

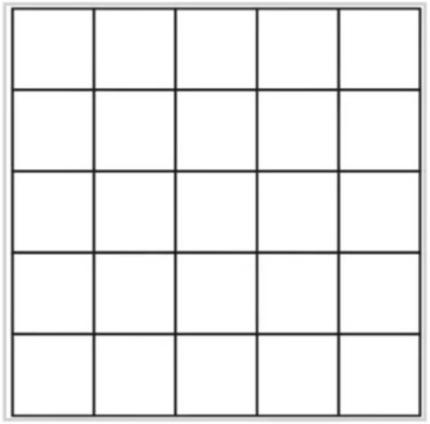
Going Up

$$\pi_*(s) = \arg\max_a \sum_{s'} \sum_r p(s', r|s, a)[r + \gamma V_*(s')]$$

γ = 0.9



22.0	24.4	22.0	19.4	17.5
19.8	22.0	19.8	17.8	16.0
17.8	19.8	17.8	16.0	14.4
16.0	17.8	16.0	14.4	13.0
14.4	16.0	14.4	13.0	11.7



$0 + 0.9(17.5) = 14.0$

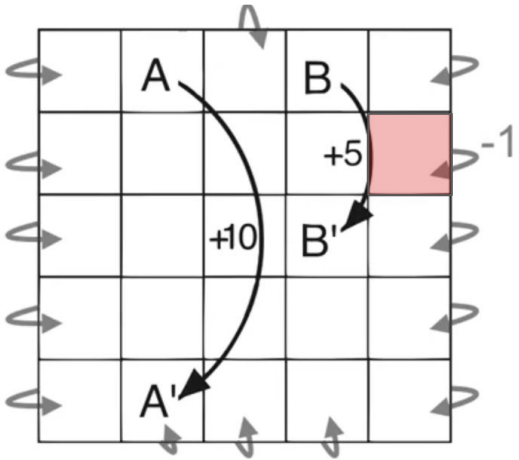
v_*

π_*

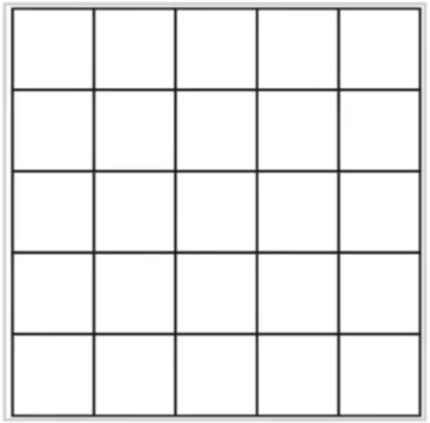
Going Right

$$\pi_*(s) = \arg\max_a \sum_{s'} \sum_r p(s', r|s, a)[r + \gamma V_*(s')]$$

γ = 0.9



22.0	24.4	22.0	19.4	17.5
19.8	22.0	19.8	17.8	16.0
17.8	19.8	17.8	16.0	14.4
16.0	17.8	16.0	14.4	13.0
14.4	16.0	14.4	13.0	11.7



$-1 + 0.9(16.0) = 13.4$

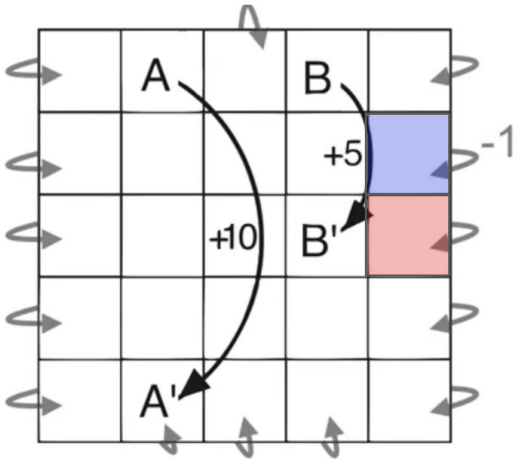
v_*

π_*

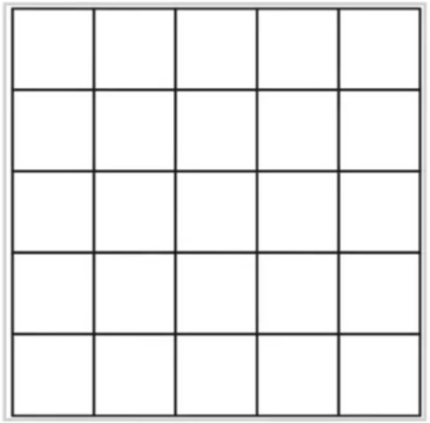
Going Down

$$\pi_*(s) = \arg\max_a \sum_{s'} \sum_r p(s', r|s, a)[r + \gamma V_*(s')]$$

γ = 0.9



22.0	24.4	22.0	19.4	17.5
19.8	22.0	19.8	17.8	16.0
17.8	19.8	17.8	16.0	14.4
16.0	17.8	16.0	14.4	13.0
14.4	16.0	14.4	13.0	11.7



$0 + 0.9(14.4) = 13.0$

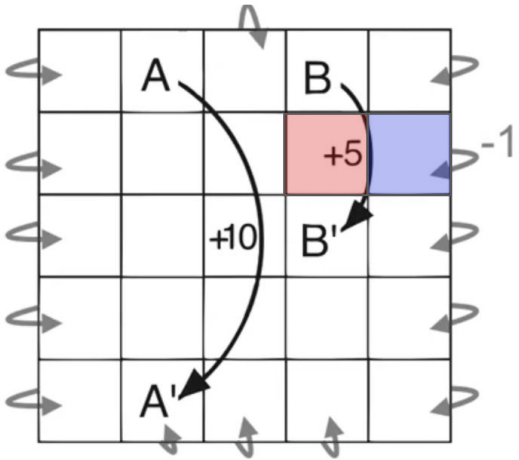
v_*

π_*

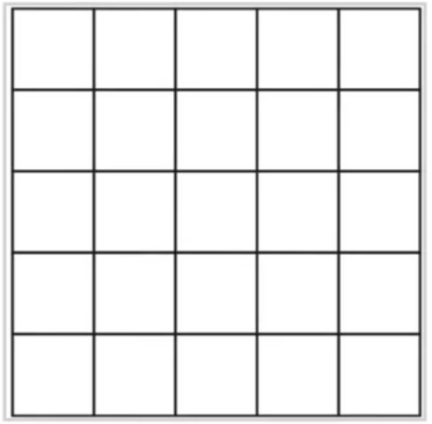
Going Left

$$\pi_*(s) = \arg\max_a \sum_{s'} \sum_r p(s', r|s, a)[r + \gamma V_*(s')]$$

γ = 0.9



22.0	24.4	22.0	19.4	17.5
19.8	22.0	19.8	17.8	16.0
17.8	19.8	17.8	16.0	14.4
16.0	17.8	16.0	14.4	13.0
14.4	16.0	14.4	13.0	11.7



$0 + 0.9(17.8) = 16.0$

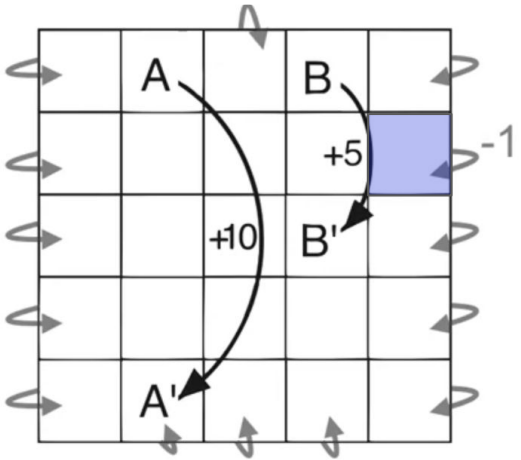
v_*

π_*

Optimal Policy for One State

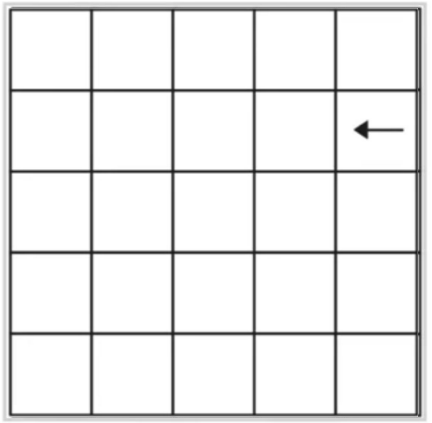
$$\pi_*(s) = \arg\max_a \sum_{s'} \sum_r p(s', r|s, a)[r + \gamma V_*(s')]$$

γ = 0.9



22.0	24.4	22.0	19.4	17.5
19.8	22.0	19.8	17.8	16.0
17.8	19.8	17.8	16.0	14.4
16.0	17.8	16.0	14.4	13.0
14.4	16.0	14.4	13.0	11.7

v_*

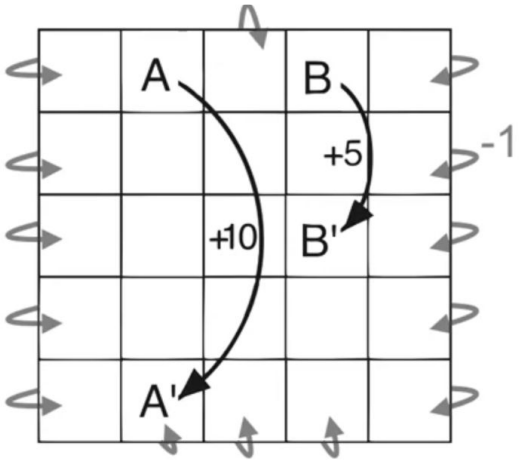


π_*

Optimal Policy for All States

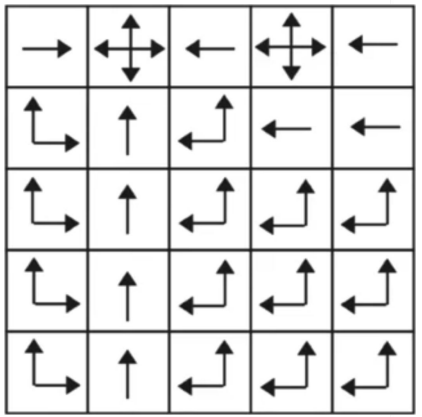
$$\pi_*(s) = \arg\max_a \sum_{s'} \sum_r p(s', r|s, a)[r + \gamma V_*(s')]$$

γ = 0.9



22.0	24.4	22.0	19.4	17.5
19.8	22.0	19.8	17.8	16.0
17.8	19.8	17.8	16.0	14.4
16.0	17.8	16.0	14.4	13.0
14.4	16.0	14.4	13.0	11.7

v_*



π_*

State-action or State Value?

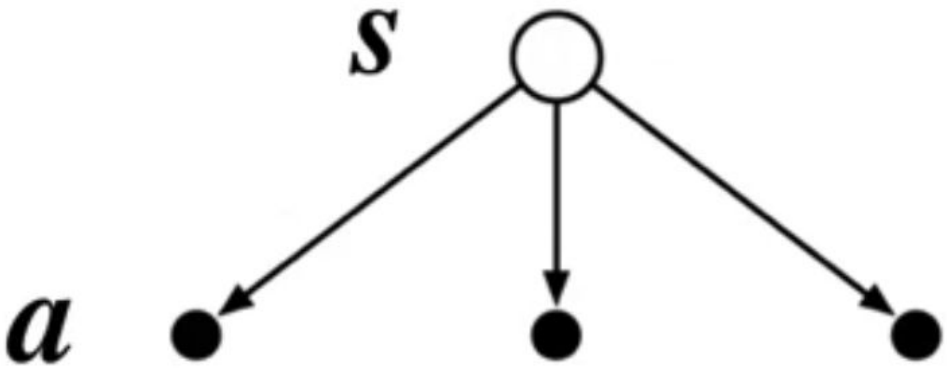
$$\pi_*(s) = \arg\max_a \sum_{s'} \sum_r p(s', r|s, a)[r + \gamma V_*(s')]$$

$$\pi_*(s) = \arg\max_a Q_*(s, a)$$

Why?

Finding the Optimal Policy with Q_*

$$\pi_*(s) = \arg\max_a Q_*(s, a)$$



Dynamic Programming Methods

Recall Incremental Sample Average

$$Q_{n+1} = \frac{R_1 + \cdots + R_n}{n}$$

$$= \ldots$$

$$= Q_n + \frac{1}{n}(R_n - Q_n)$$

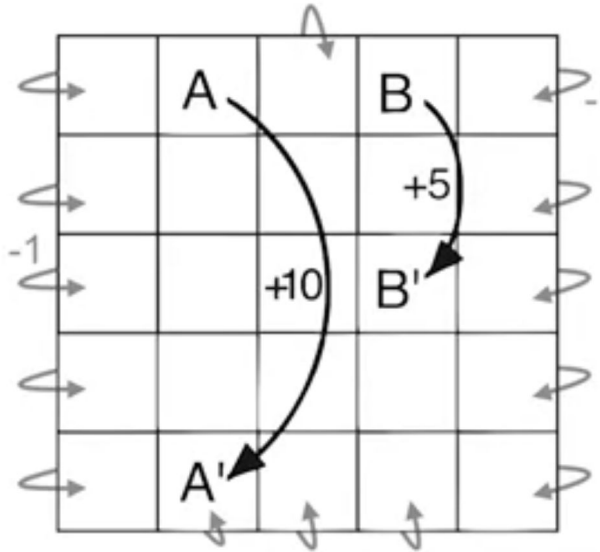
Recall:

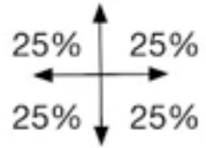
$$Q_n = \frac{1}{n-1}\sum_{i=1}^{n-1} R_i$$

NewEstimate ← OldEstimate + StepSize (Target - OldEstimate)

Policy Evaluation

Given a policy, find its state value function $\pi \rightarrow V_\pi(s)$

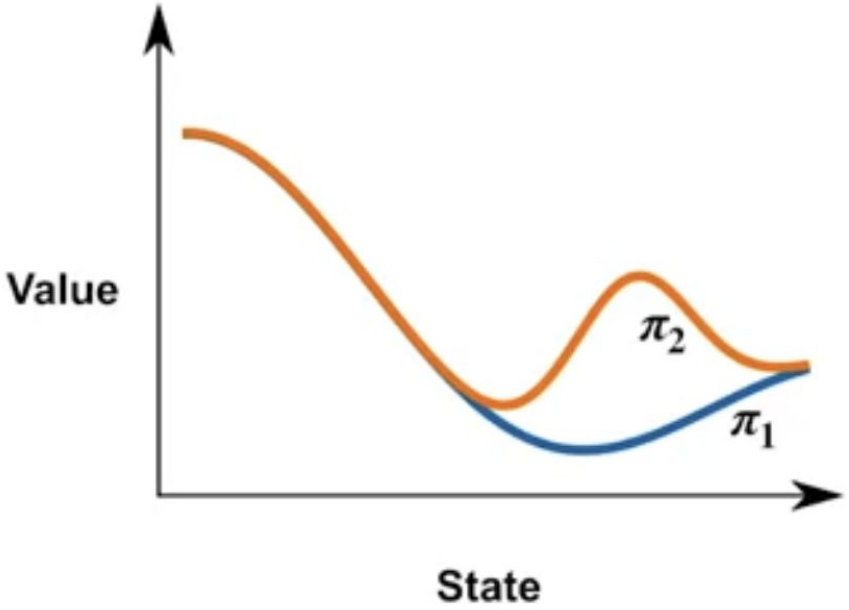




3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

Control (Approximating Optimal Policies)

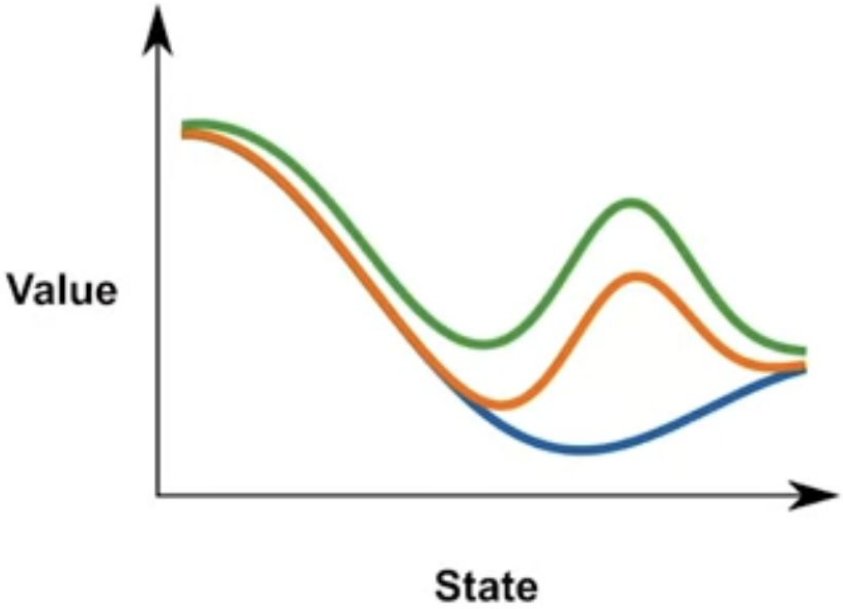
The task of improving a policy



Credit: University of Alberta

Control

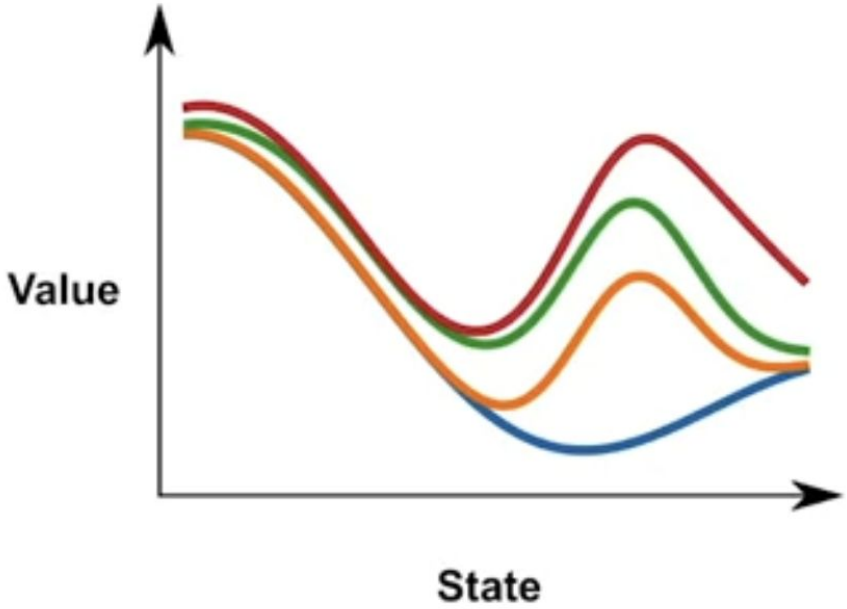
The task of improving a policy



Credit: University of Alberta

Control

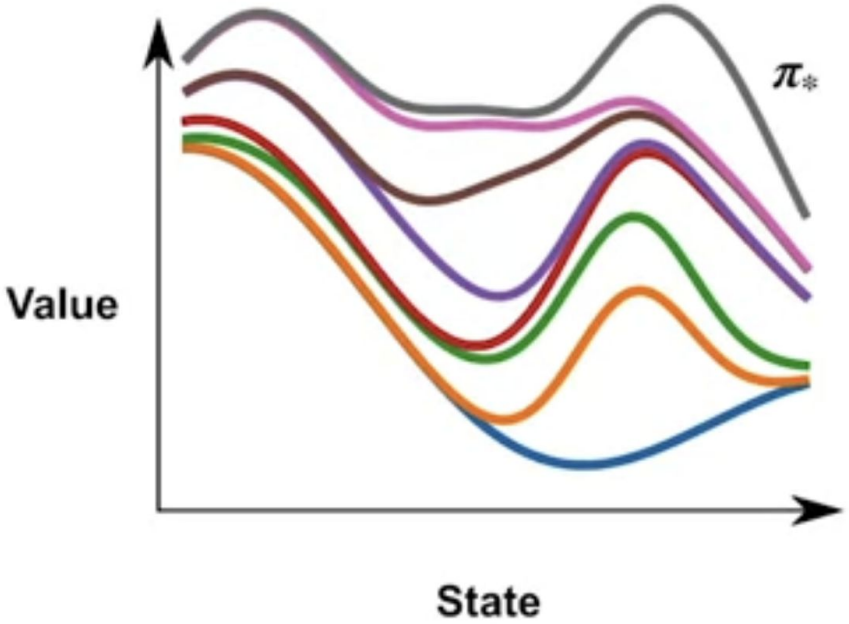
The task of improving a policy



Credit: University of Alberta

Control

Eventually, the hope is that the result is the optimal policy



Credit: University of Alberta

Iterative Policy Evaluation

Iterative Policy Evaluation

Uses the Bellman expectation equation as an update rule for the value function

- Given: a policy π
- Continuously apply the Bellman equation update rule. At each iteration *k* :
  - For **all** states *s*
  - Update $V_{k+1}(s)$ from $V_k(s')$ where *s'* follows *s*

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

Iterative Policy Evaluation

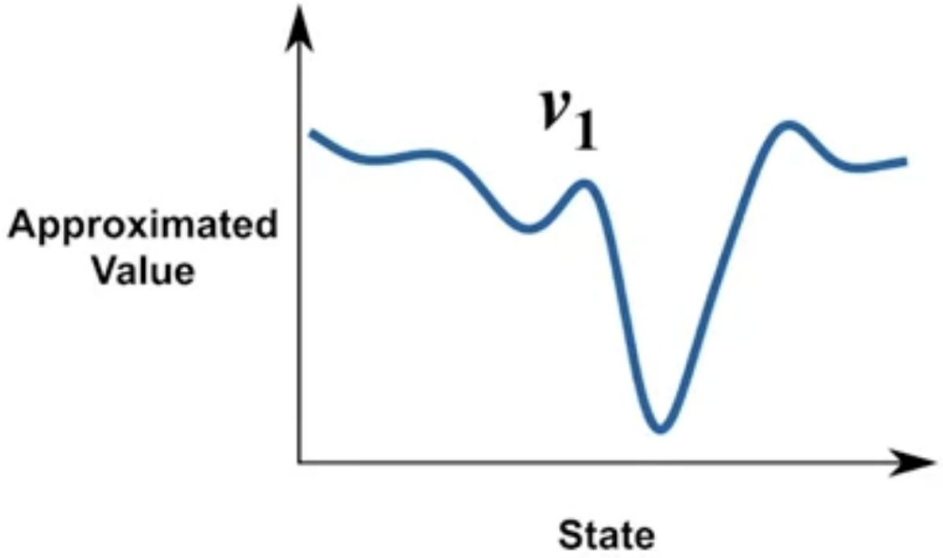
Uses the Bellman expectation equation as an update rule for the value function

- Given: a policy π
- Continuously apply the Bellman equation update rule: At each iteration *k* :
  - For **all** states *s*
  - Update $V_{k+1}(s)$ from $V_k(s')$ where *s'* follows *s*

$$V_1 \rightarrow V_2 \rightarrow \ldots \rightarrow V_\pi$$

Gradual Value Approximations

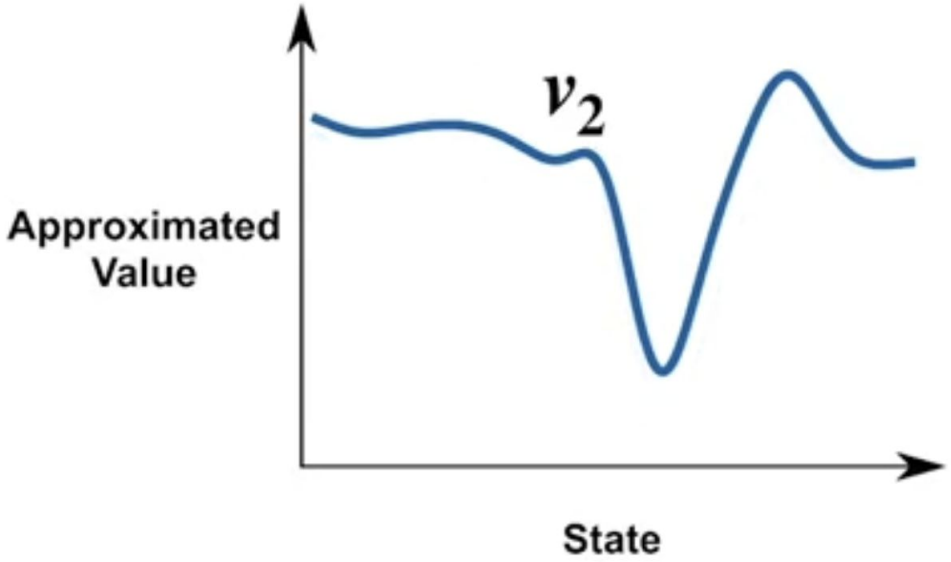
$$V_{k+1}(s) \leftarrow \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)[r + \gamma V_k(s')]$$



Credit: University of Alberta

Gradual Value Approximations

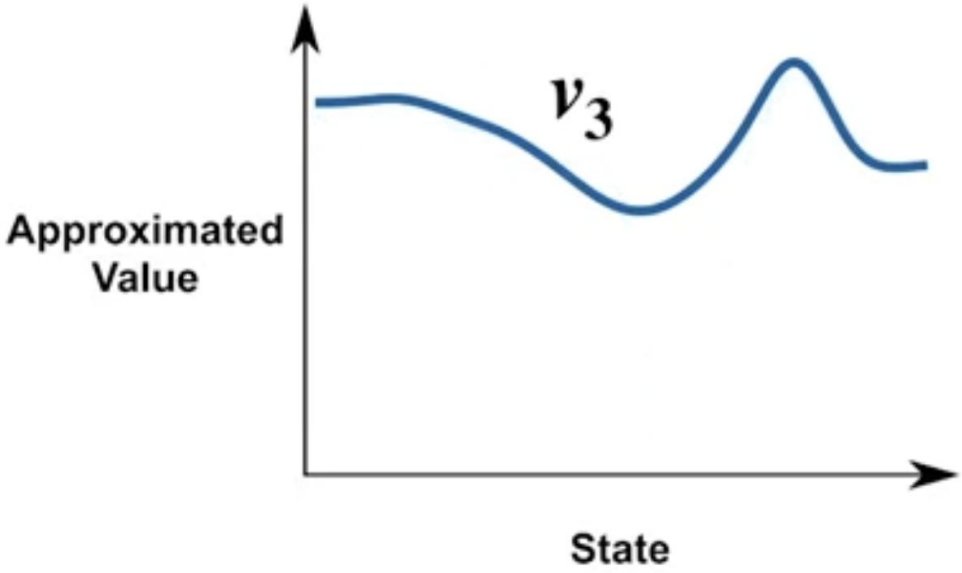
$$V_{k+1}(s) \leftarrow \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)[r + \gamma V_k(s')]$$



Credit: University of Alberta

Gradual Value Approximations

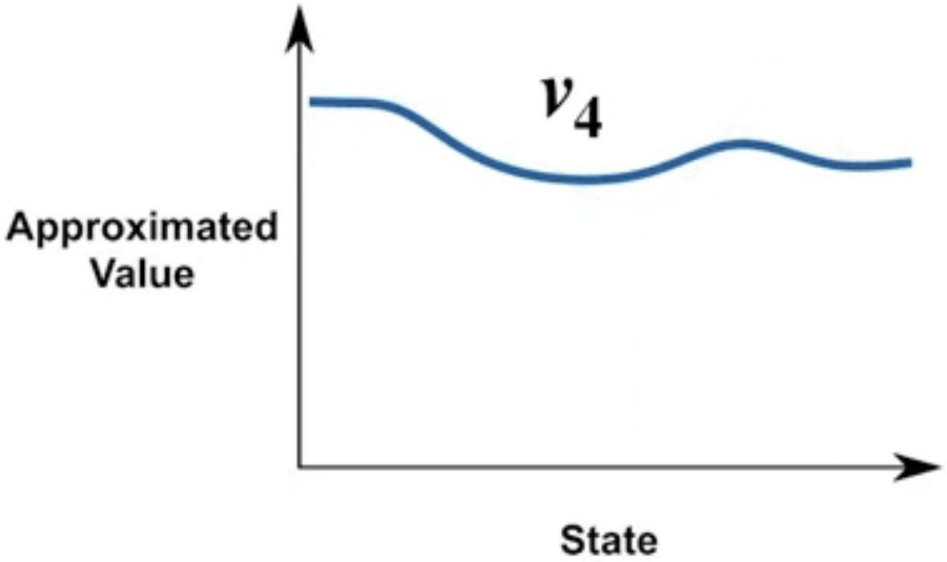
$$V_{k+1}(s) \leftarrow \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)[r + \gamma V_k(s')]$$



Credit: University of Alberta

Gradual Value Approximations

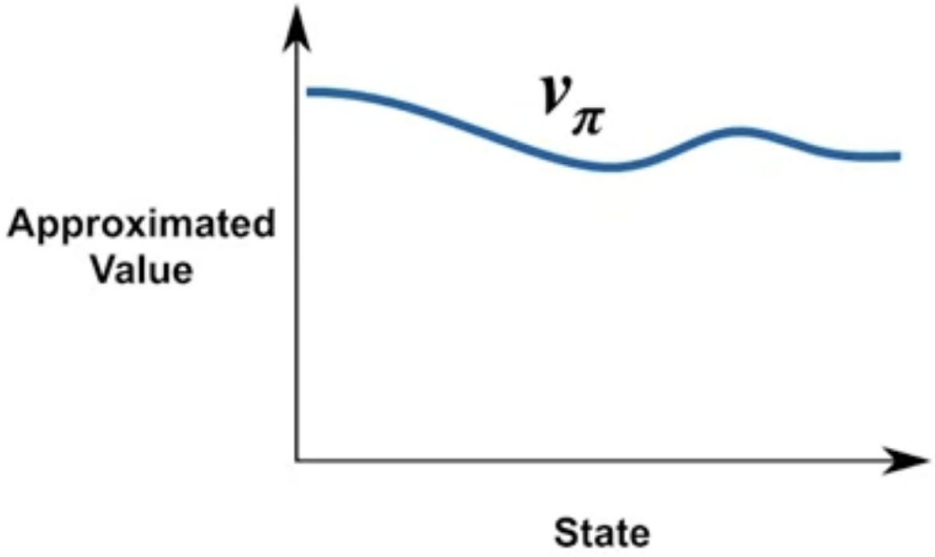
$$V_{k+1}(s) \leftarrow \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)[r + \gamma V_k(s')]$$



Credit: University of Alberta

Gradual Value Approximations

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

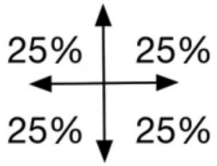


Credit: University of Alberta

Gridworld Example

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

- Terminal states in gray squares
- Actions: North, South, East, West, and agent cannot move off the grid
- Reward: -1 per timestep
- Episodic task ($\gamma = 1$)

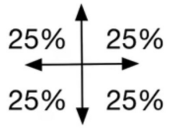


Policy

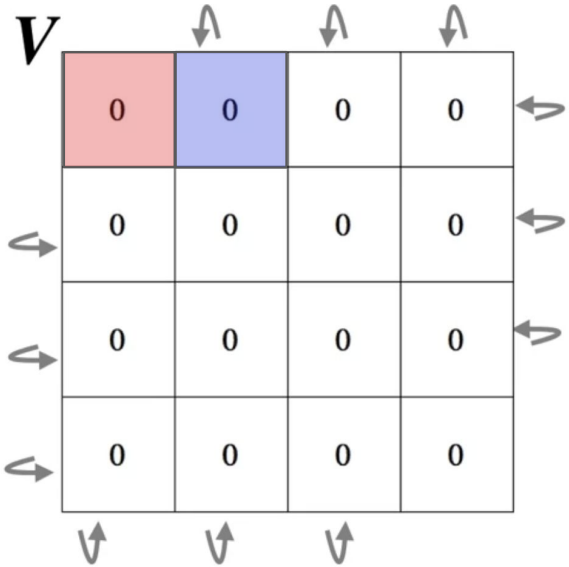
$$V_{k+1}(s) \leftarrow \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)[r + \gamma V_k(s')]$$

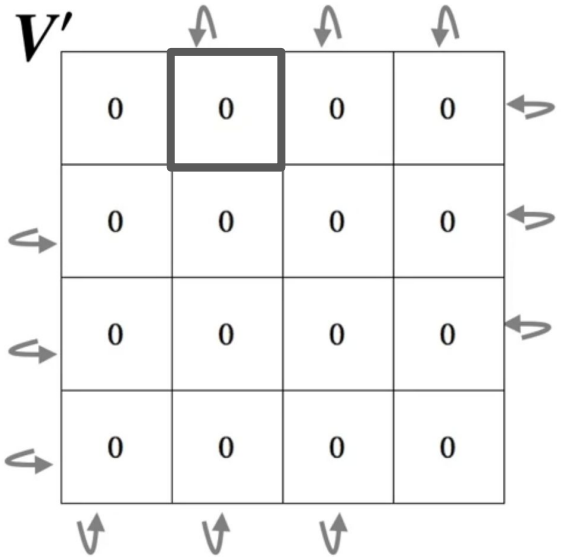
Left

$$0.25(-1 + 0)$$



Policy



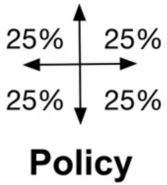


$$V_{k+1}(s) \leftarrow \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)[r + \gamma V_k(s')]$$

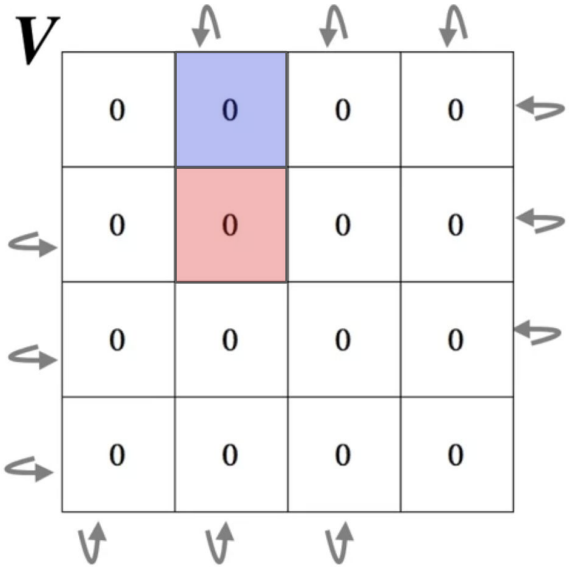
Left

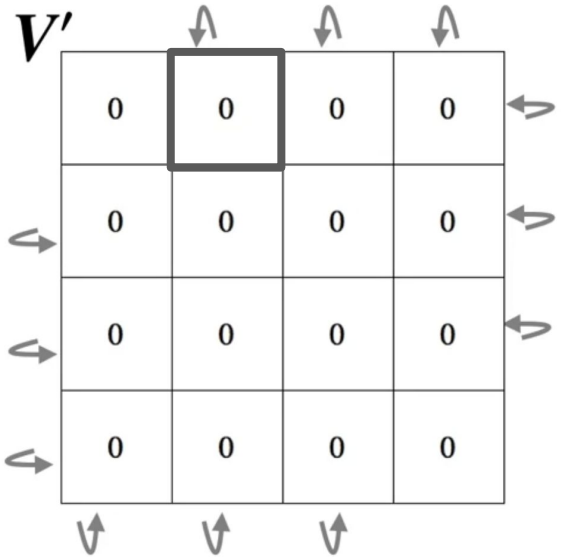
Down

$$0.25(-1 + 0)$$



Policy

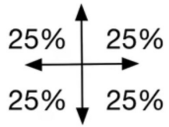




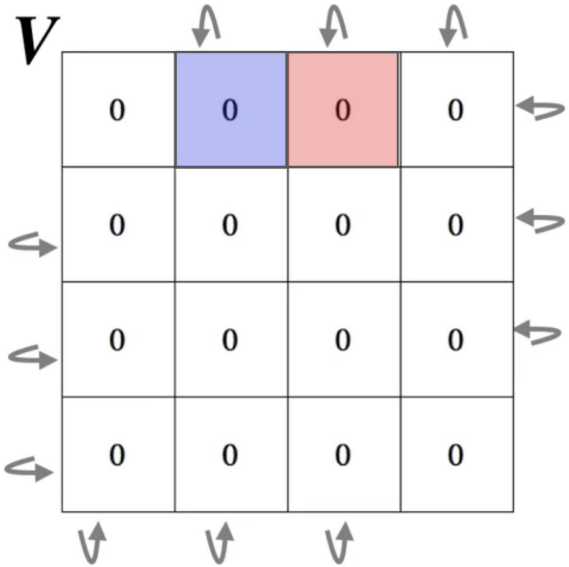
$$V_{k+1}(s) \leftarrow \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)[r + \gamma V_k(s')]$$

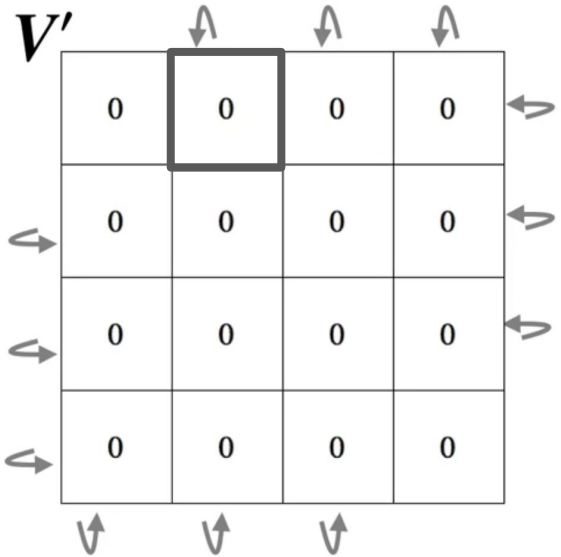
Left Down Right

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



Policy



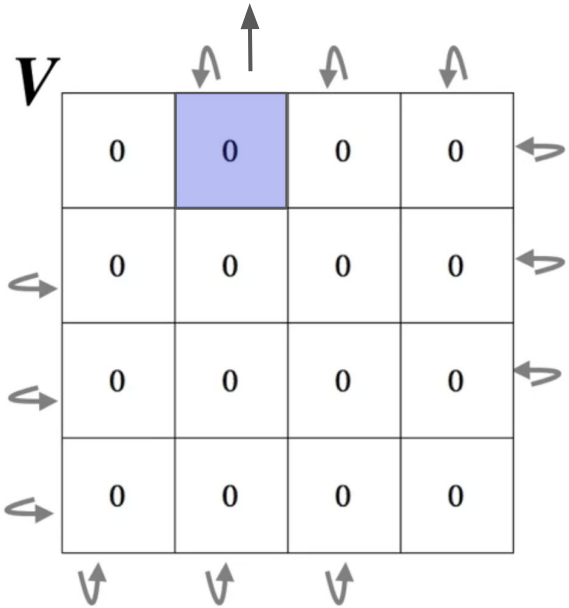


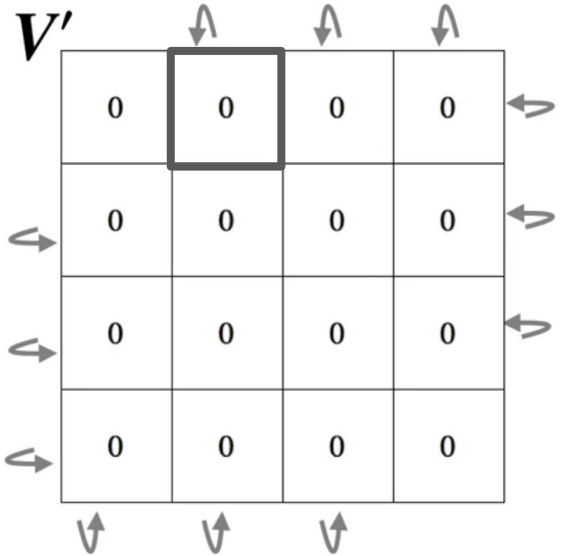
$$V_{k+1}(s) \leftarrow \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)[r + \gamma V_k(s')]$$

Left Down Right Up

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

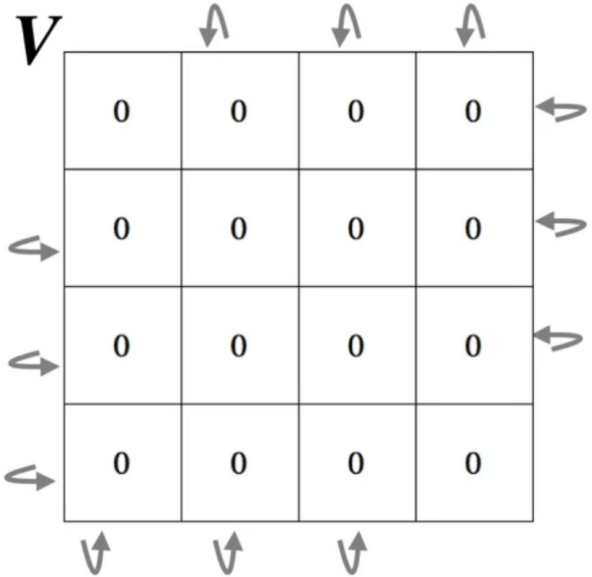
Policy

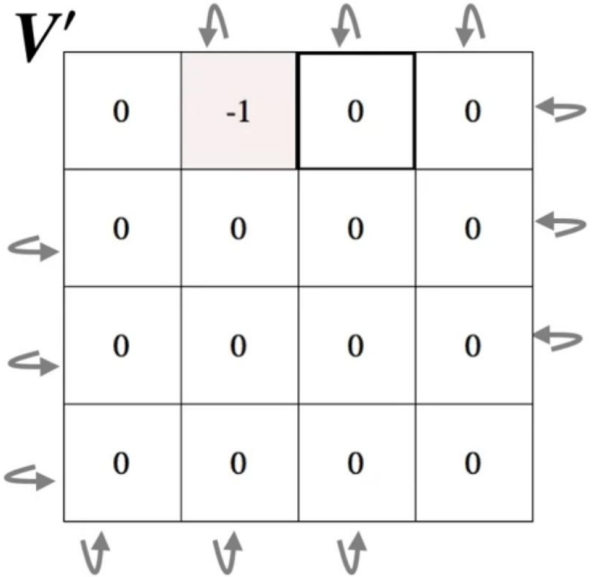




Example: Gridworld

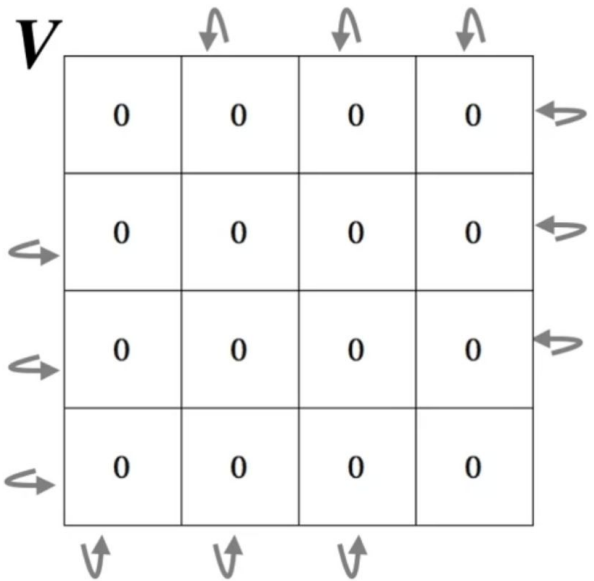
Update the value of the calculated state and repeat for ALL other states in the world

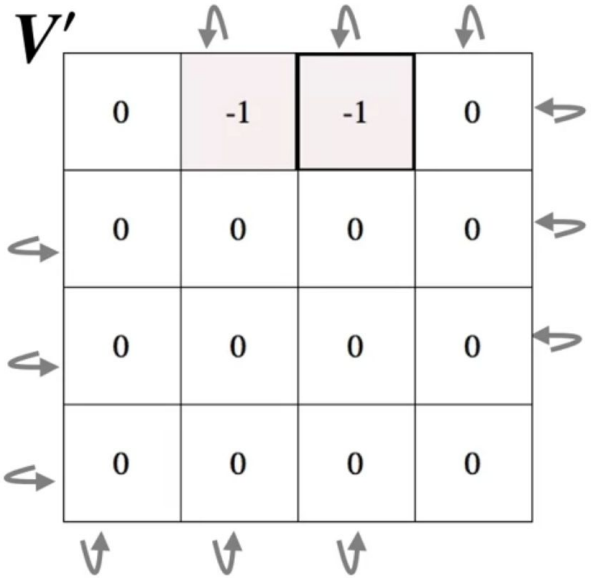




Gridworld Example

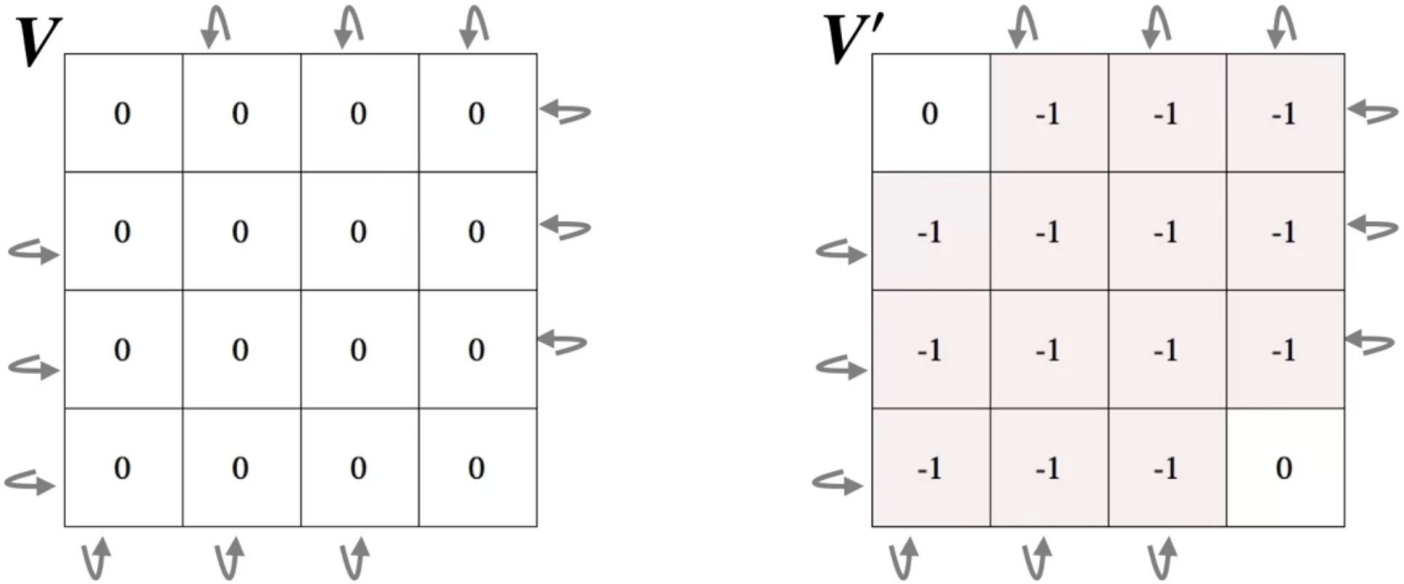
$$0.25(-1+0)+0.25(-1+0)+0.25(-1+0)+0.25(-1+0)=-1$$





Gridworld Example

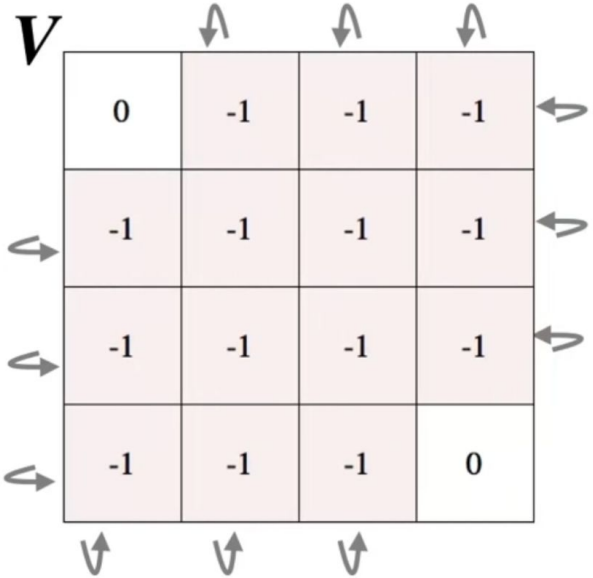
All the values of the other states will be set to -1, since all the initial values are 0

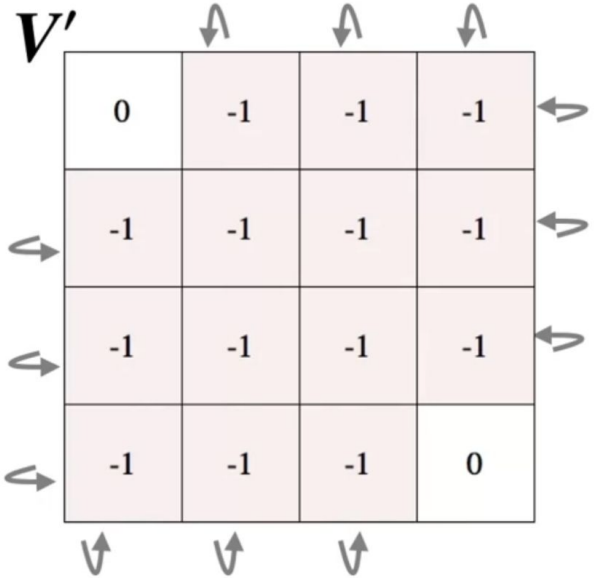


Gridworld Example

Copy the calculated next values over to update the current state values.

One Iteration Done!

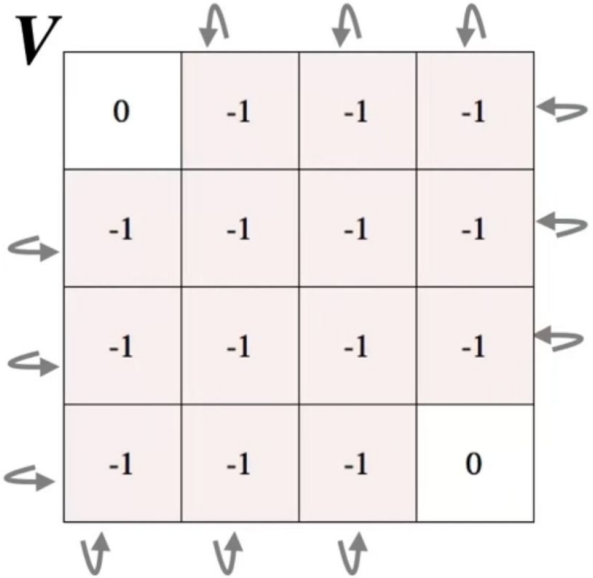


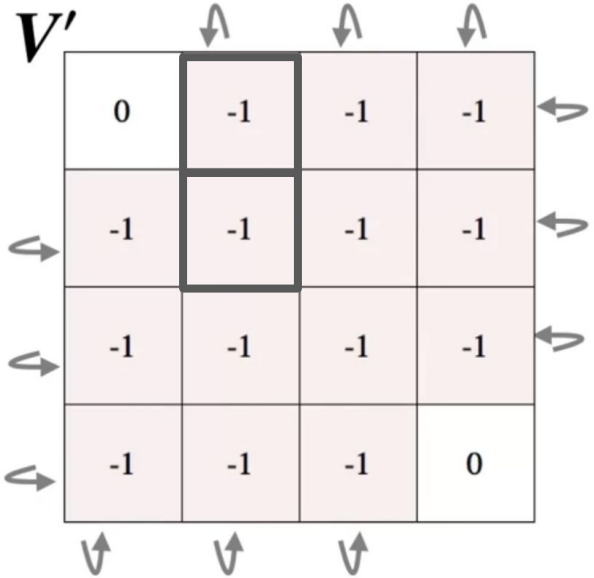


Gridworld Example

$$0.25(-1+0)+0.25(-1+(-1))+0.25(-1+(-1))+0.25(-1+(-1))=-1.75$$

$$0.25(-1+(-1))+0.25(-1+(-1))+0.25(-1+(-1))+0.25(-1+(-1))=-2$$





Gridworld Example

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

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

Gridworld Example

k = 10

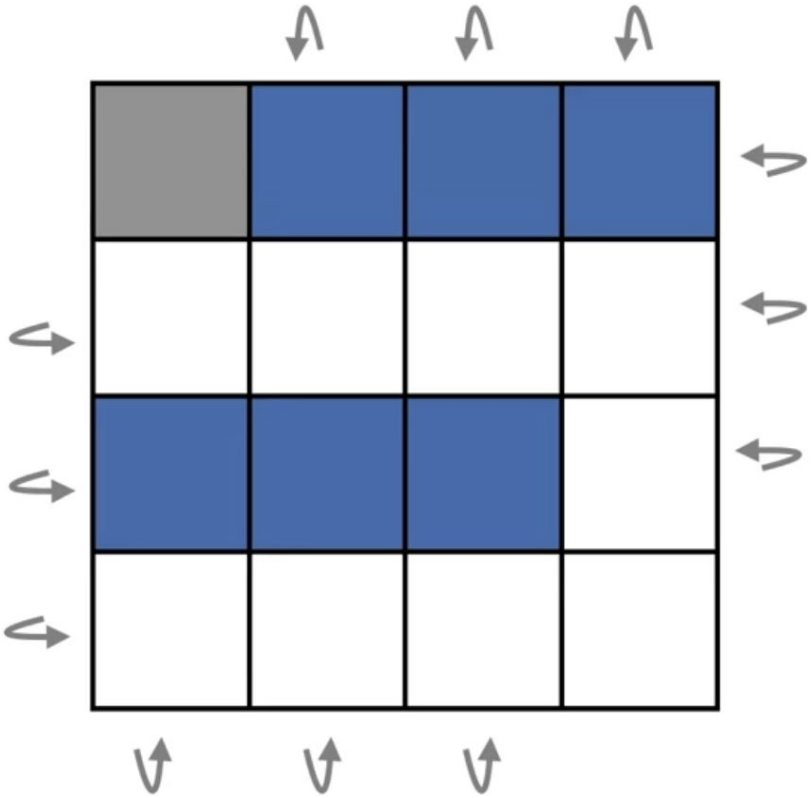
0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

k = ∞

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

Notebook:
Iterative Policy Evaluation

(New) Maze



- Singular terminal state at gray square
- Actions: North, South, East, West, and agent cannot move off the grid
- Reward: -1 per timestep, but if agent transitions into **blue state, -10 reward**
- Start with **random policy**

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 $V(s)$ arbitrarily, for $s \in \mathcal{S}$, and $V(terminal)$ to 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 $\Delta < \theta$

Need to know environment dynamics!

Policy Iteration

Policy Improvement

Recall:

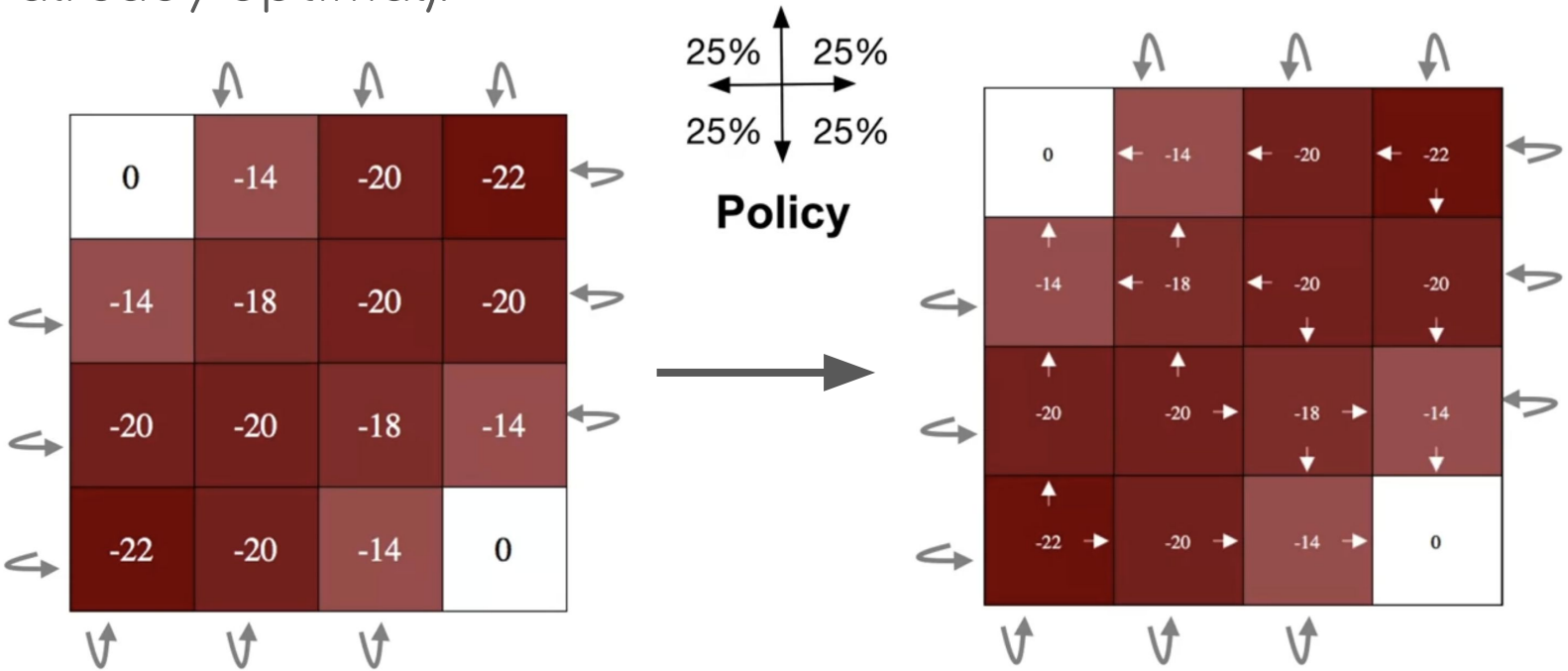
$$\pi_*(s) = \arg\max_a \sum_{s'} \sum_r p(s', r|s, a)[r + \gamma V_*(s')]$$

What happens if instead of using V_*, we **acted greedily w.r.t.** V_π instead?

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

Policy Improvement

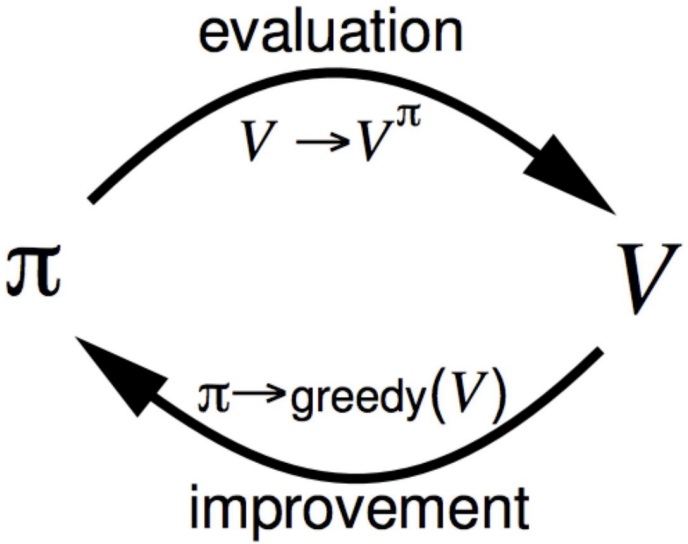
The new policy will be **a strict improvement** (unless the policy is already optimal)!



Policy Improvement Theorem

You can **always** obtain a strictly better policy by **acting greedily with respect to the value function of the current policy**, unless the current policy is already optimal

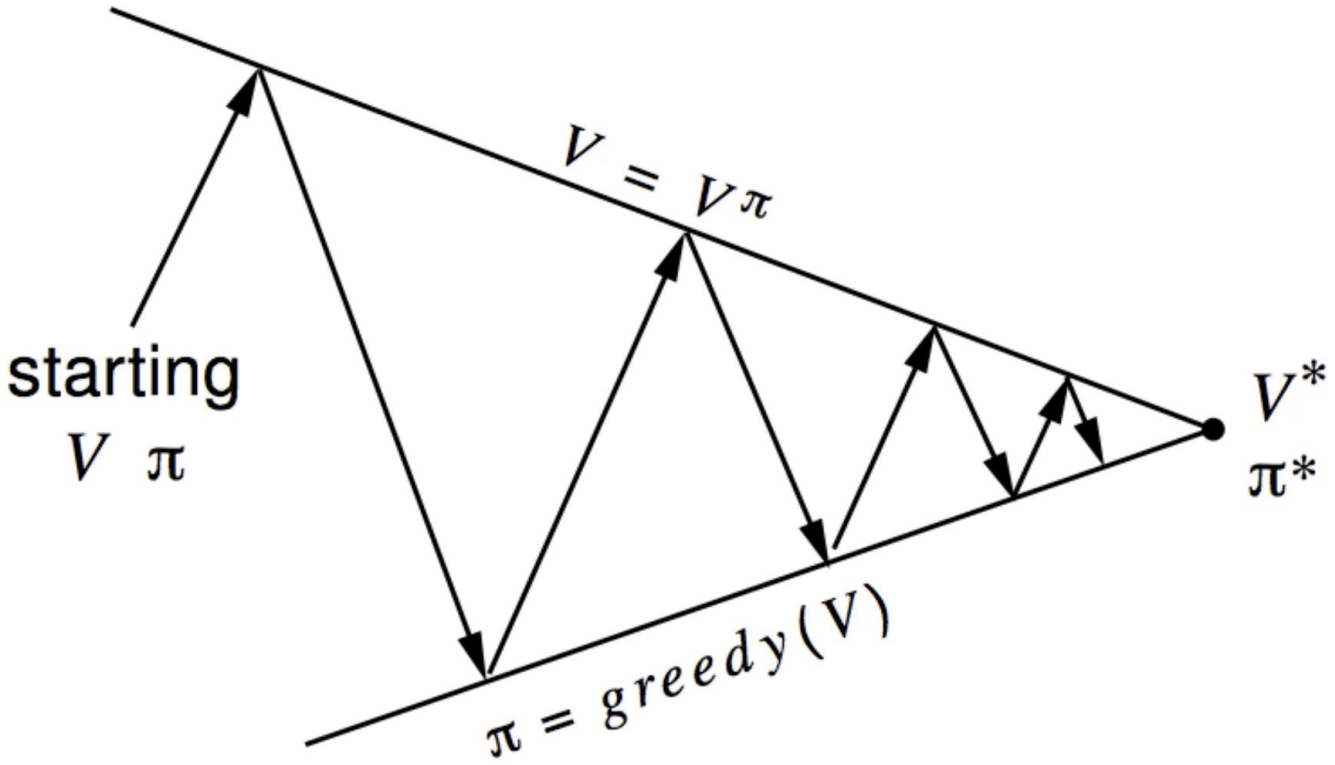
Policy Iteration



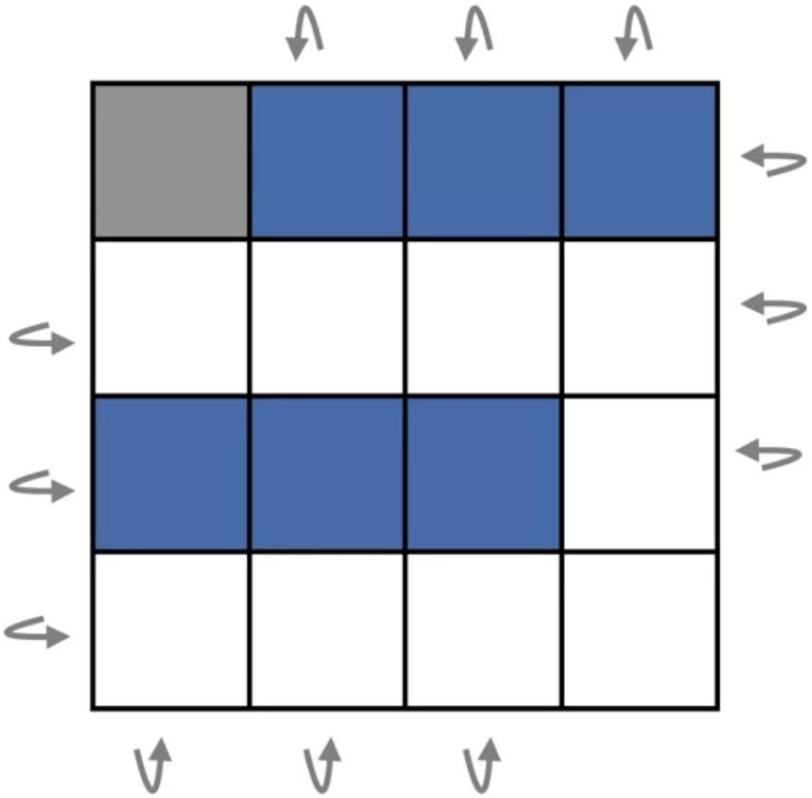
1. Initialize a random policy
2. Policy Evaluation: find the value function for the current policy
3. Policy Improvement: act greedily w.r.t. the found value function
4. Repeat until optimal policy

$$\pi_1 \rightarrow V_{\pi_1} \rightarrow \pi_2 \rightarrow V_{\pi_2} \rightarrow \pi_3 \rightarrow V_{\pi_3} \rightarrow \ldots \rightarrow \pi_*$$

Policy Iteration

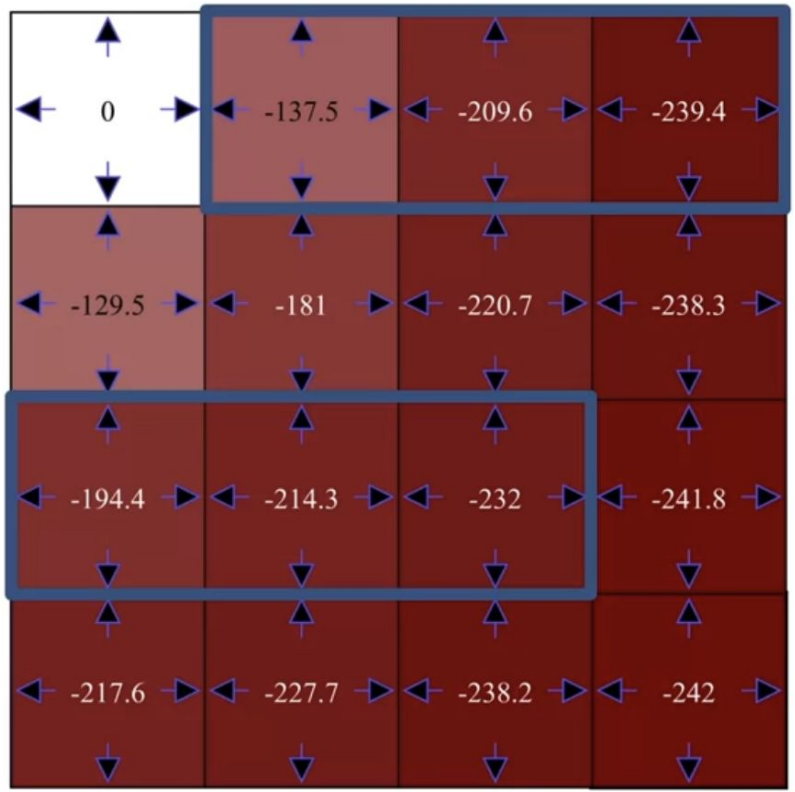


Maze

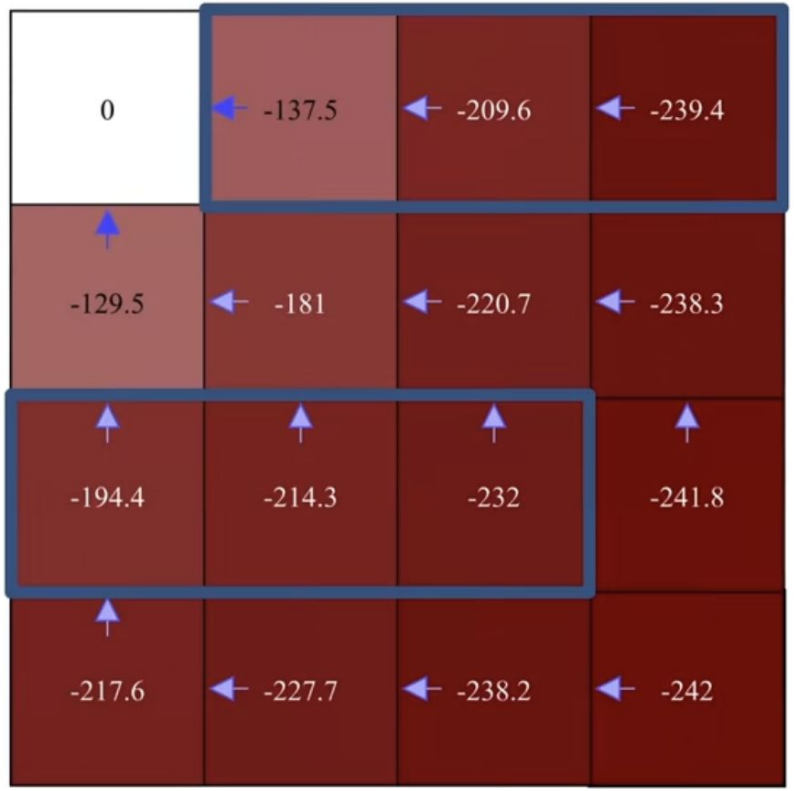


- Singular terminal state at gray square
- Actions: North, South, East, West, and agent cannot move off the grid
- Reward: -1 per timestep, but if agent transitions into **blue state, -10 reward**
- Episodic task with discount factor $\gamma = 1$
- Start with **random policy**

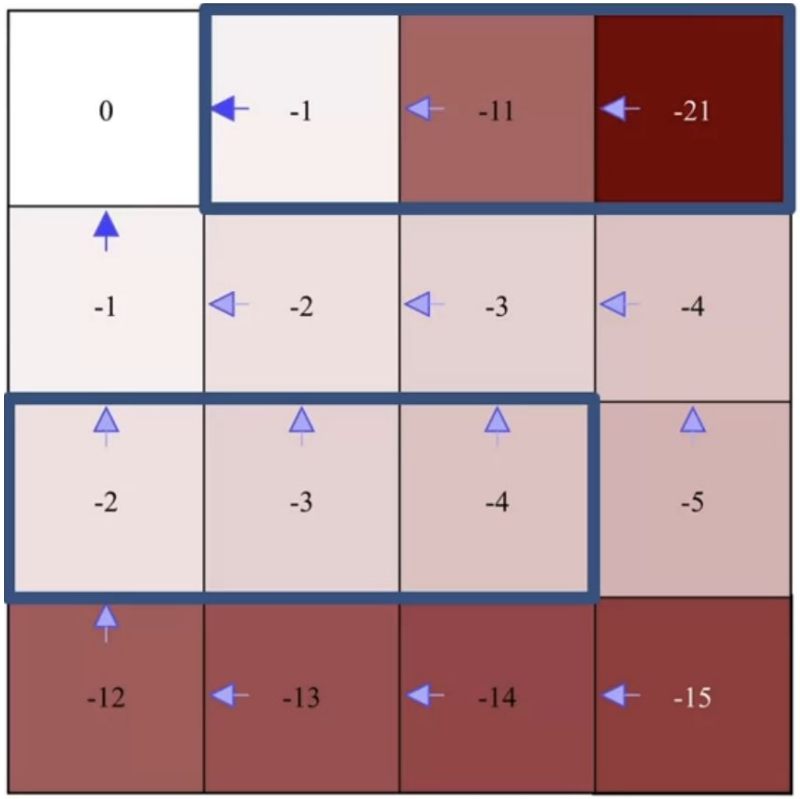
Evaluation



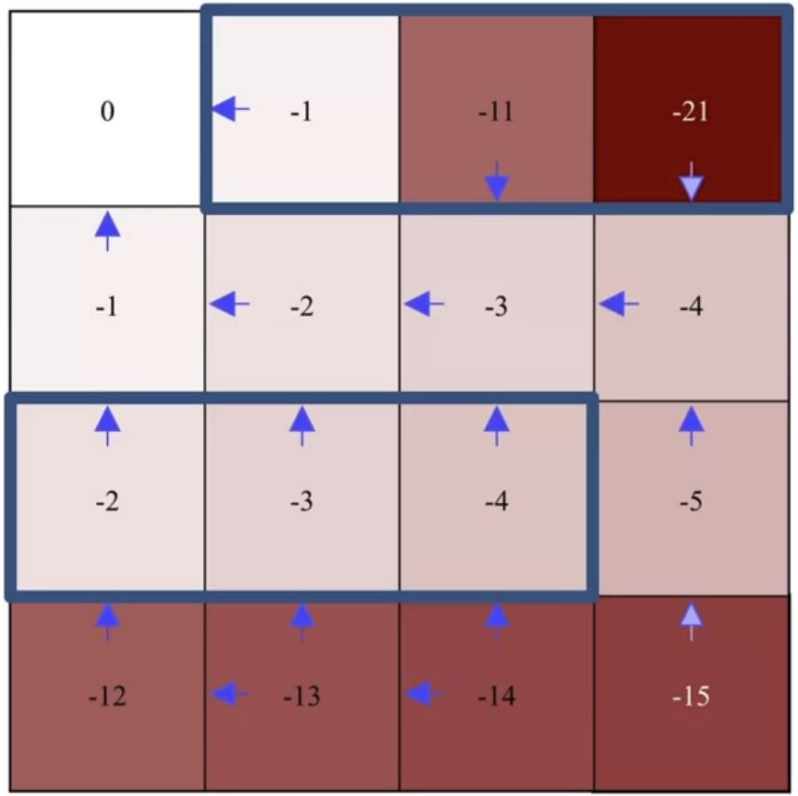
Improvement



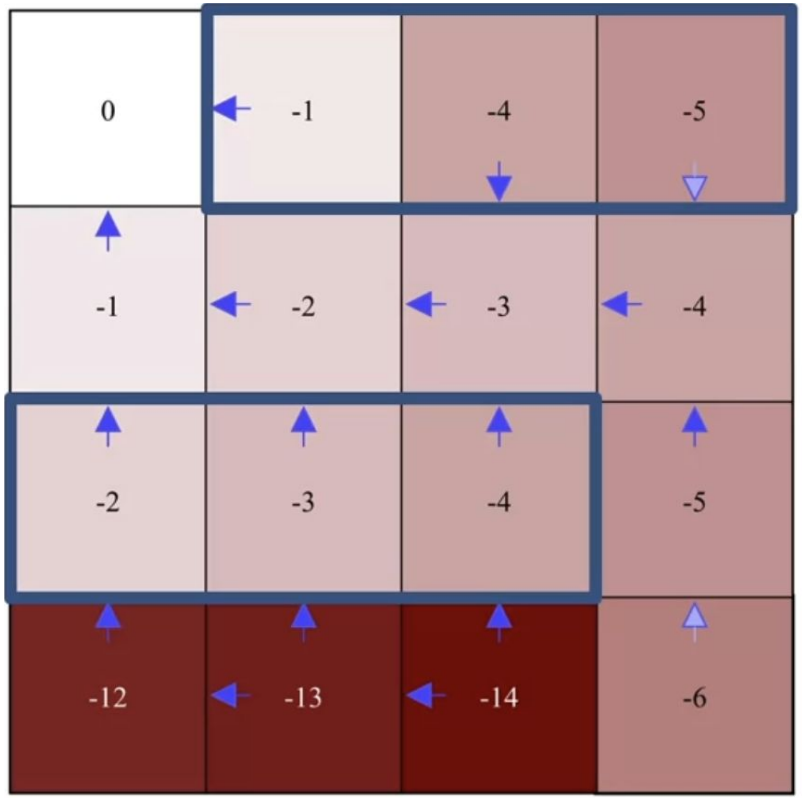
Evaluation



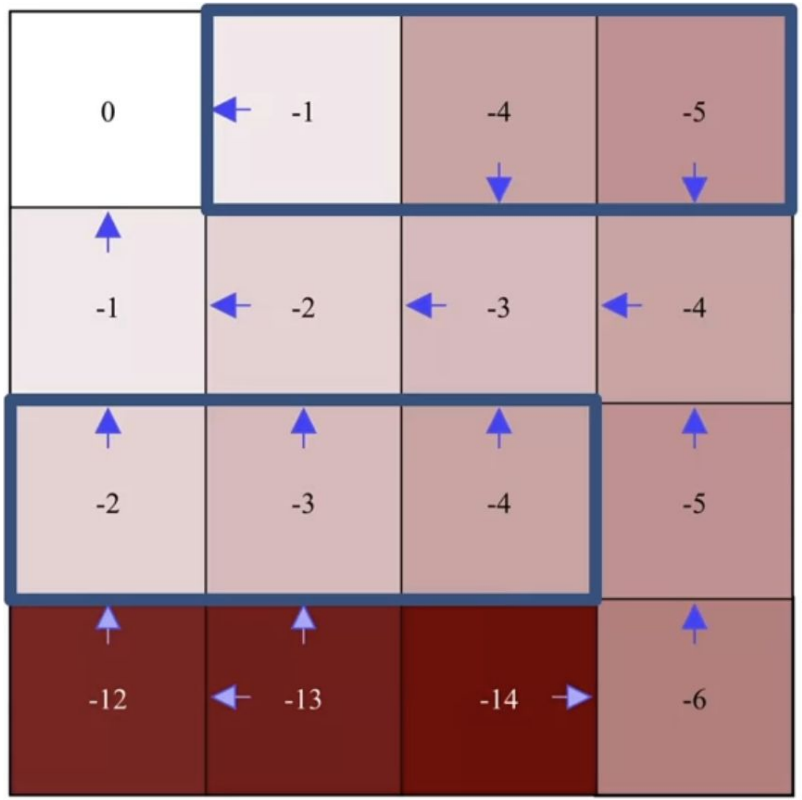
Improvement



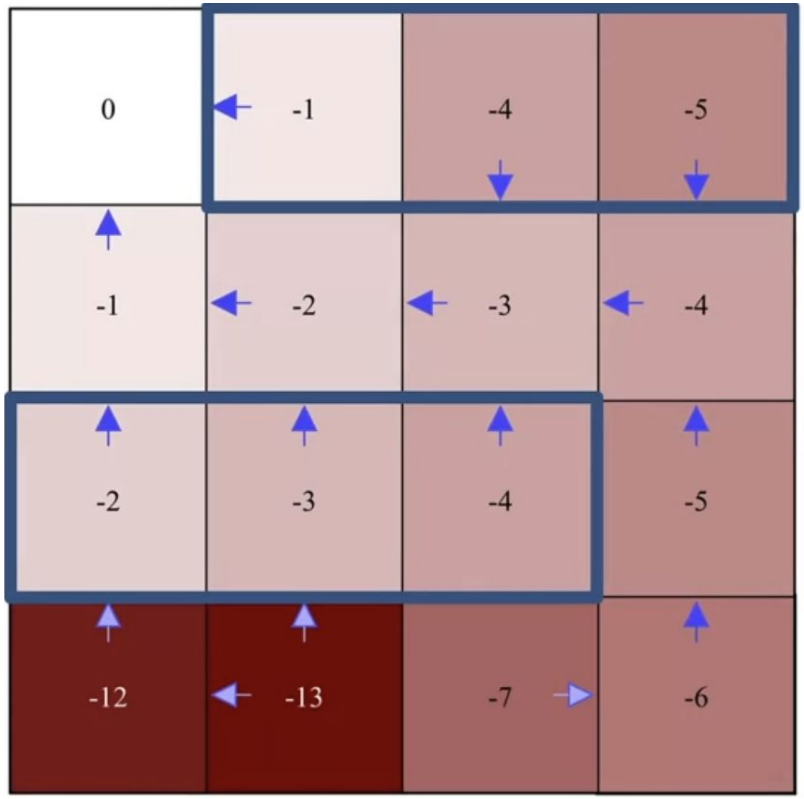
Evaluation



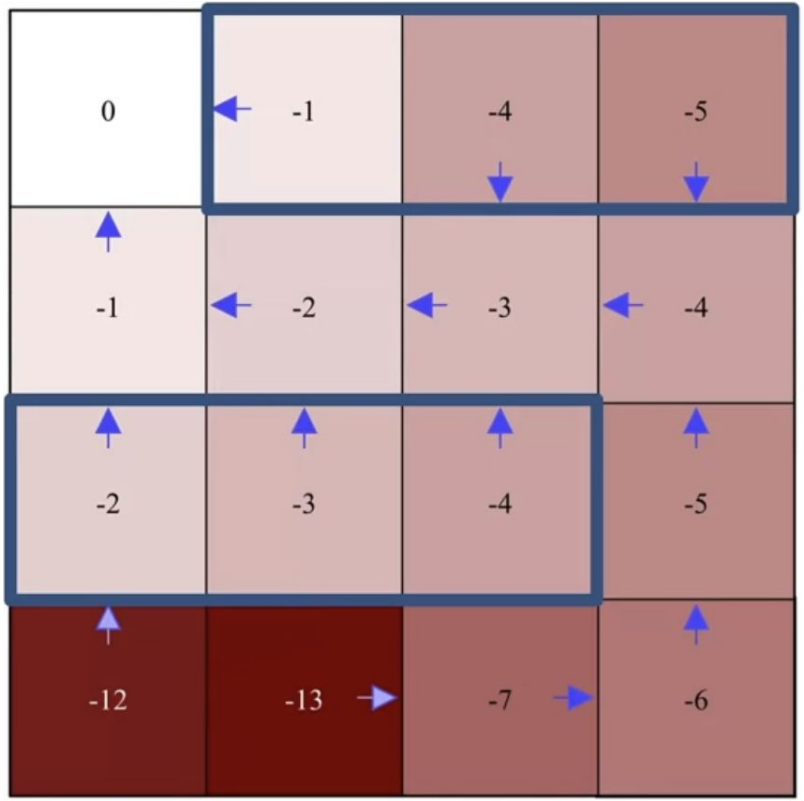
Improvement



Evaluation

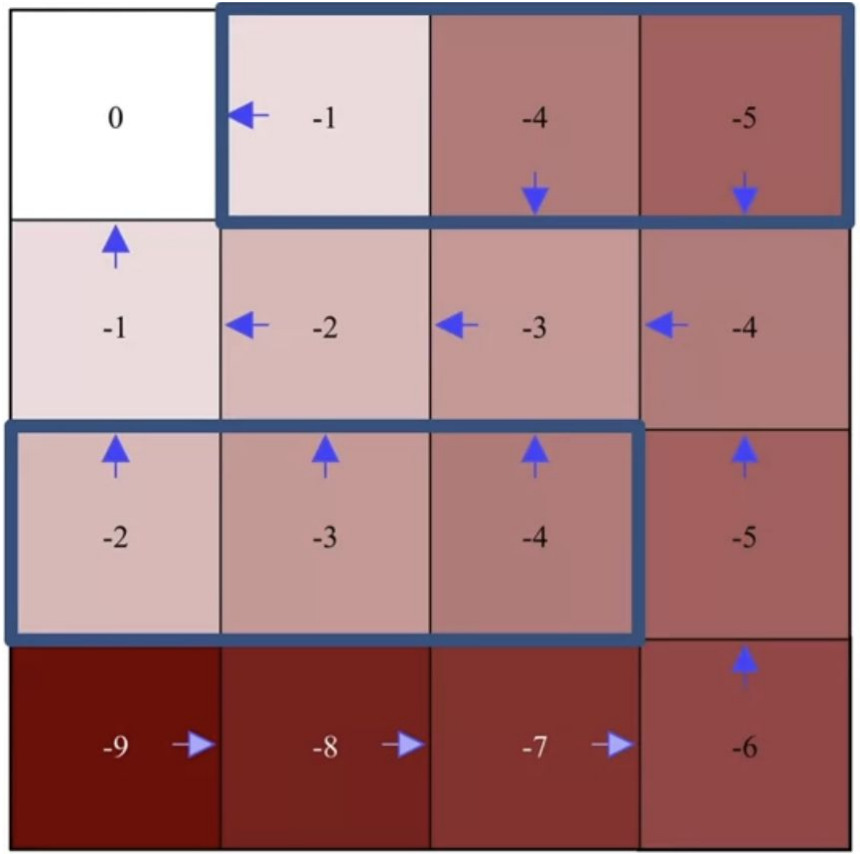


Improvement



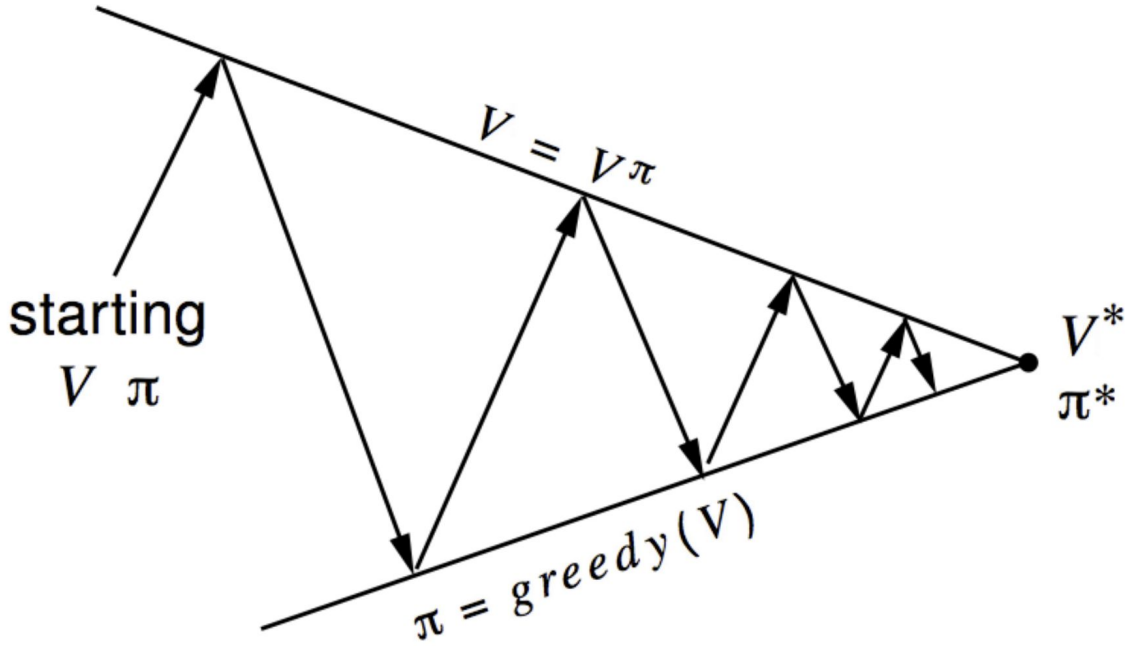
Eventually, we will reach the optimal policy and optimal value function, as shown to the left

(don't forget about reward)



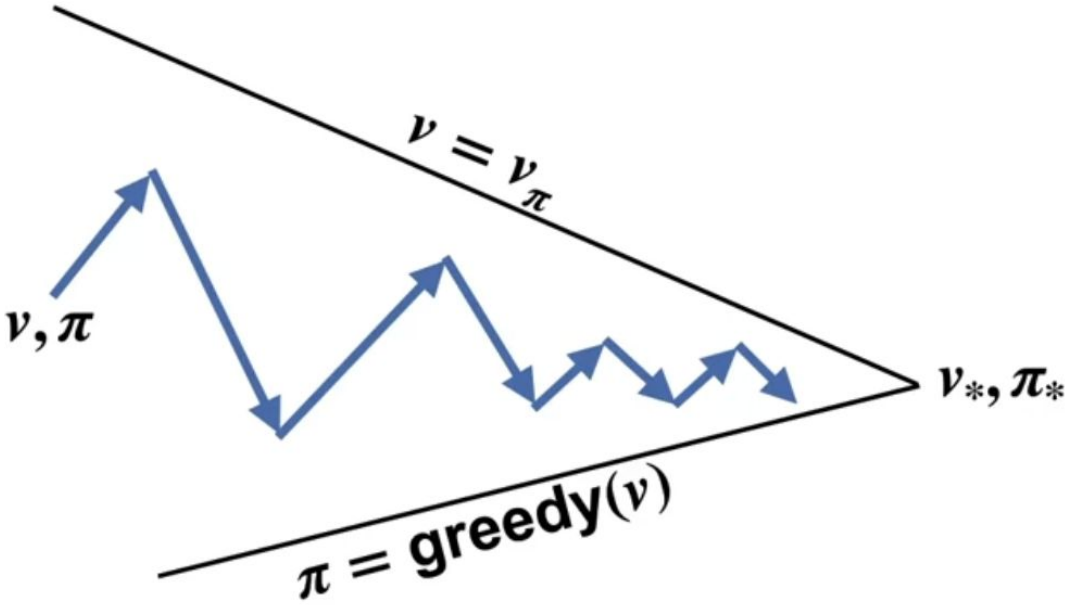
Generalized Policy Iteration

We can relax the framework of policy iteration.



Generalized Policy Iteration

We can relax the framework of policy iteration.



Notebook: Policy Improvement

Policy Improvement

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 $s \in \mathcal{S}$; $V(terminal) \doteq 0$

2. Policy Evaluation
   Loop:
      $\Delta \leftarrow 0$
      Loop for each $s \in \mathcal{S}$:
         $v \leftarrow V(s)$
         $V(s) \leftarrow \sum_{s',r} p(s', r \,|\, s, \pi(s)) \big[r + \gamma V(s')\big]$
         $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   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 $\leftarrow \pi(s)$
      $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r \,|\, s, a) \big[r + \gamma V(s')\big]$
      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

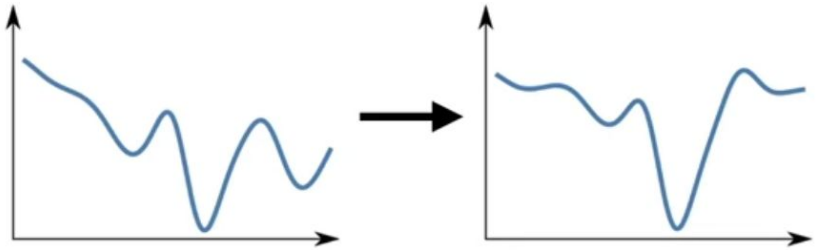
Value Iteration

Value Iteration

Special case of policy iteration where policy evaluation stops after one sweep

Uses the Bellman optimality equation instead!

Evaluation



Improvement

$$\pi \leftarrow \textbf{greedy}(v)$$

Doesn't make reference to a specific policy!

Recall: Policy Improvement

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 $s \in \mathcal{S}$; $V(terminal) \doteq 0$

2. Policy Evaluation
   Loop:
   $\Delta \leftarrow 0$
   Loop for each $s \in \mathcal{S}$:
   $v \leftarrow V(s)$
   $V(s) \leftarrow \sum_{s',r} p(s', r \mid s, \pi(s)) \big[r + \gamma V(s')\big]$
   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   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 $\leftarrow \pi(s)$
   $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r \mid s, a) \big[r + \gamma V(s')\big]$
   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

Value Iteration

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 $V(\mathit{terminal}) = 0$

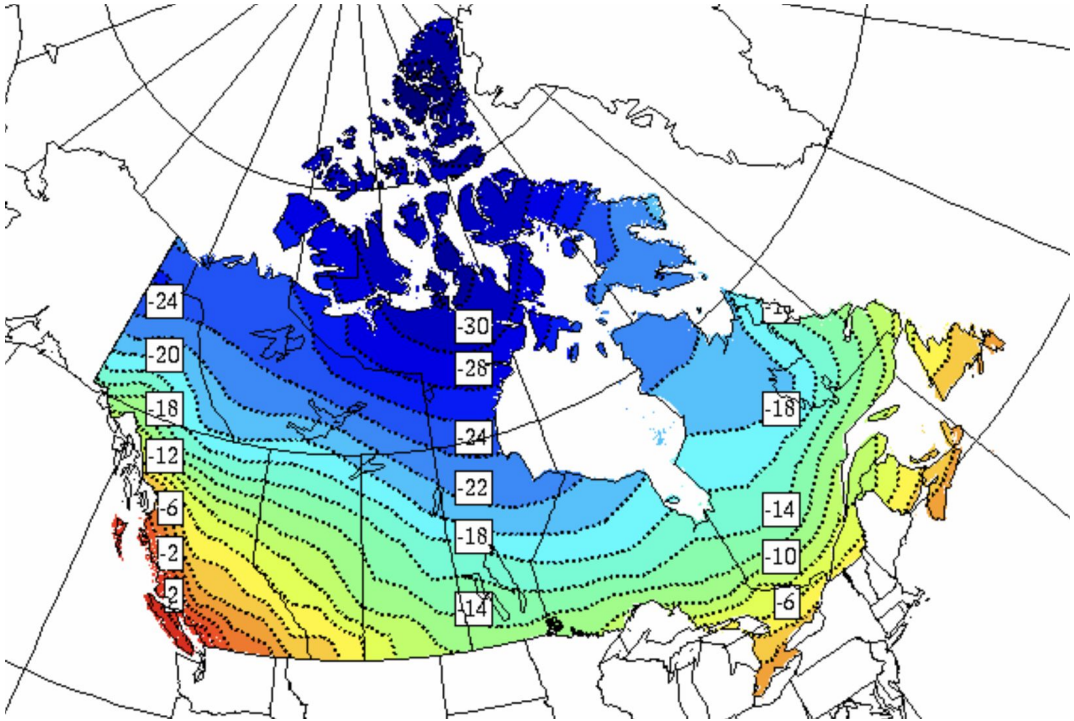
Loop:
| $\Delta \leftarrow 0$
| Loop for each $s \in \mathcal{S}$:
| $\quad v \leftarrow V(s)$
| $\quad V(s) \leftarrow \max_a \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma V(s')\big]$
| $\quad \Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

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

Take-Home Notebook: Value Iteration

Monte Carlo Methods

Problems with Dynamic Programming?



Don't know the environment transition dynamics!!!

p(s', r | s, a) is unknown

Recall: K-armed Bandits

We estimated expected reward of an arm by **averaging over a large number of random samples**

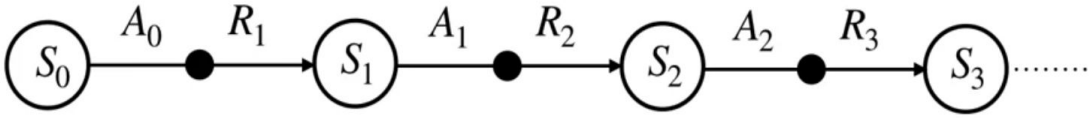
True Average: 84

t_1 → 90 t_2 → 90 t_3 → 70

$$q_t(a) = \frac{90 + 90 + 70}{3} = 83.3$$

What are Monte Carlo methods?

Given only rollouts of samples (episodes),



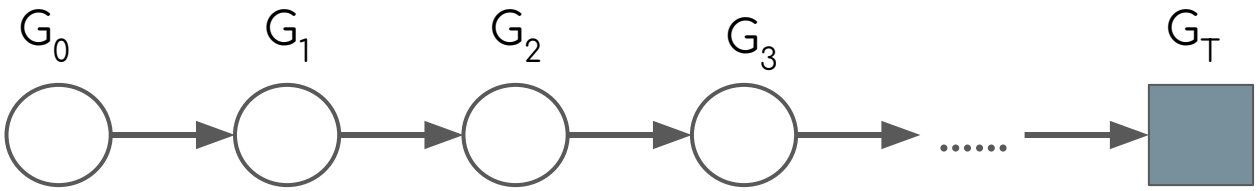
Monte Carlo methods:

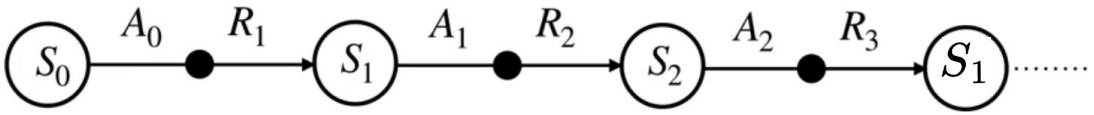
- Learn values or a policy by averaging a large number of random samples
- Learn from complete episodes and their returns
- Do not make any assumptions about the environment

Monte Carlo Prediction

Monte Carlo Prediction

This averaging can be used to calculate the value function for a policy for **episodic tasks**





Trajectory of experience

For each episode, count the number of state visitations and average the returns!

Incremental Updates for Monte Carlo

Update V(s) after every episode

NewEstimate ← OldEstimate + StepSize (Target - OldEstimate)

For every state S_n with return G_n:

$$C(S_n) = C(S_n) + 1$$

$$V(S_n) = V(S_n) + \frac{1}{C(S_n)}(G_n - V(S_n))$$

First-Visit MC Prediction

The first time S_n is visited in an episode:

1. Increment a counter C_n for that state: $C_n \leftarrow C_n + 1$
2. After episode terminates, calculate return G, and add it to the total return for that state: $G_n \leftarrow G_n + G$
3. The value is estimated by $V_n = G_n / C_n$
4. As we take more samples, the estimate V_n converges to the actual value function

First-Visit MC Prediction

First-visit MC prediction, for estimating $V \approx v_\pi$

Input: a policy π to be evaluated

Initialize:

$V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$

$Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Loop forever (for each episode):

Generate an episode following π: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:

$G \leftarrow \gamma G + R_{t+1}$

Unless S_t appears in $S_0, S_1, \ldots, S_{t-1}$:

Append G to $Returns(S_t)$

$V(S_t) \leftarrow \text{average}(Returns(S_t))$

Every-Visit MC Prediction

Every time S_n is visited in an episode:

1. Increment a counter C_n for that state: $C_n \leftarrow C_n + 1$
2. After episode terminates, calculate return G, and add it to the total return for that state: $G_n \leftarrow G_n + G$
3. The value is estimated by $V_n = G_n / C_n$
4. As we take more samples, the estimate V_n converges to the actual value function

Incremental Updates for Monte Carlo

In general, we may want to forget past episodes. So, we replace the count with another step size

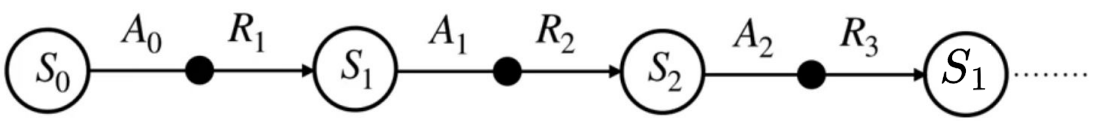
$$V(S_t) = V(S_t) + \alpha[G_t - V(S_t)]$$

Almost all the algorithms we will investigate later on will have an equation of this form, but with different quantities!

Monte Carlo for State-Action Values

Monte Carlo for State-Action Values

Exactly the same as MC for state values! Except this time with state-action pairs

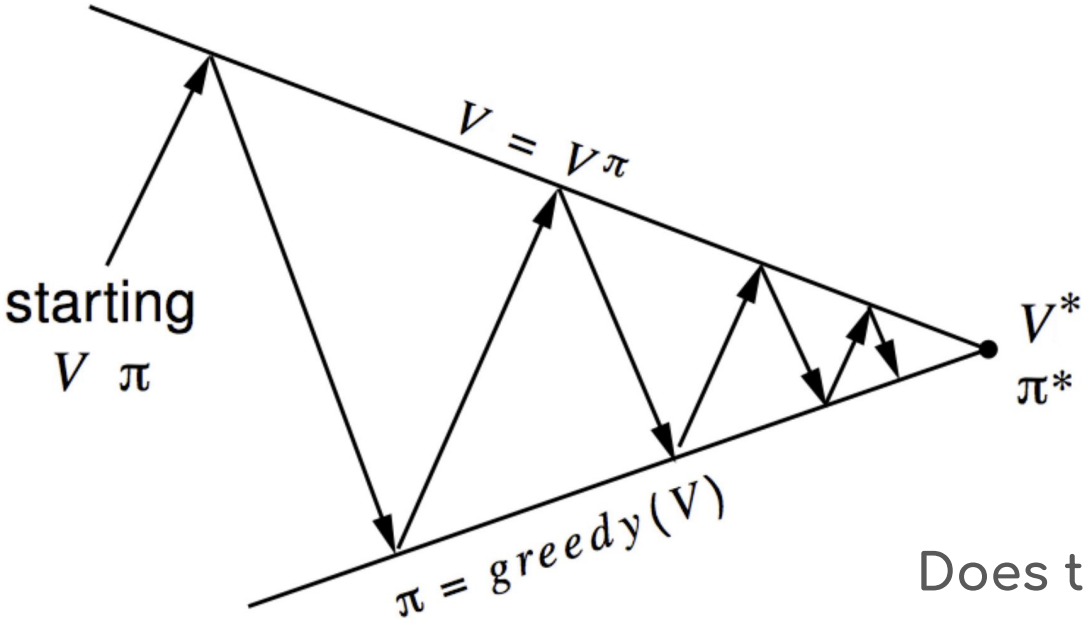


Trajectory of experience

Count state-action visitations and average the returns

Why do we want to learn action values?

Why Learn Action Values?



Policy Evaluation:
Monte Carlo

Policy Improvement:
Greedy Selection (V)

Does this work?

Why Learn Action Values?

No, it does not work.

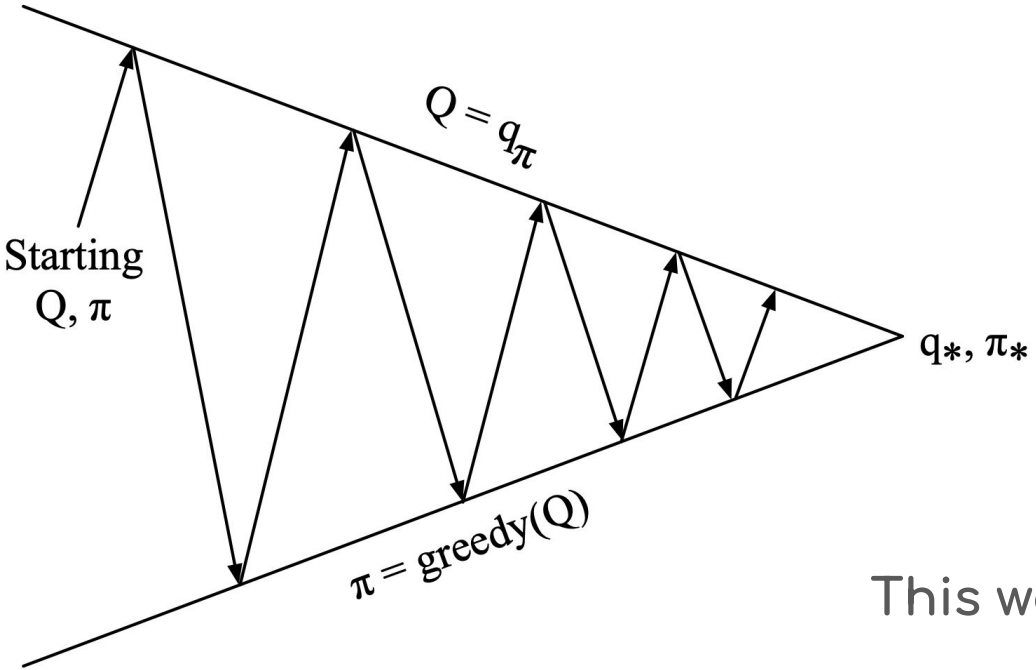
Using the state value function for greedy improvement requires knowledge of MDP dynamics. Recall:

$$\pi_*(s) = \arg\max_a \sum_{s'} \sum_r p(s', r|s, a)[r + \gamma V_*(s')]$$

VS

$$\pi_*(s) = \arg\max_a Q_*(s, a)$$

Why Learn Action Values?

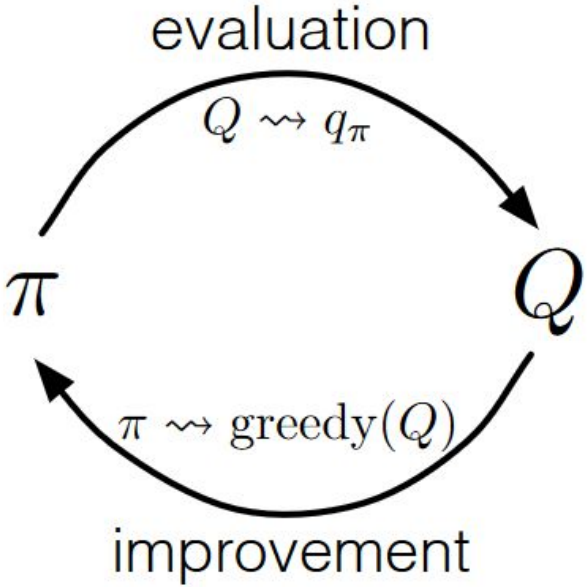


Policy Evaluation:
Monte Carlo

Policy Improvement:
Greedy Selection (Q)

This works!

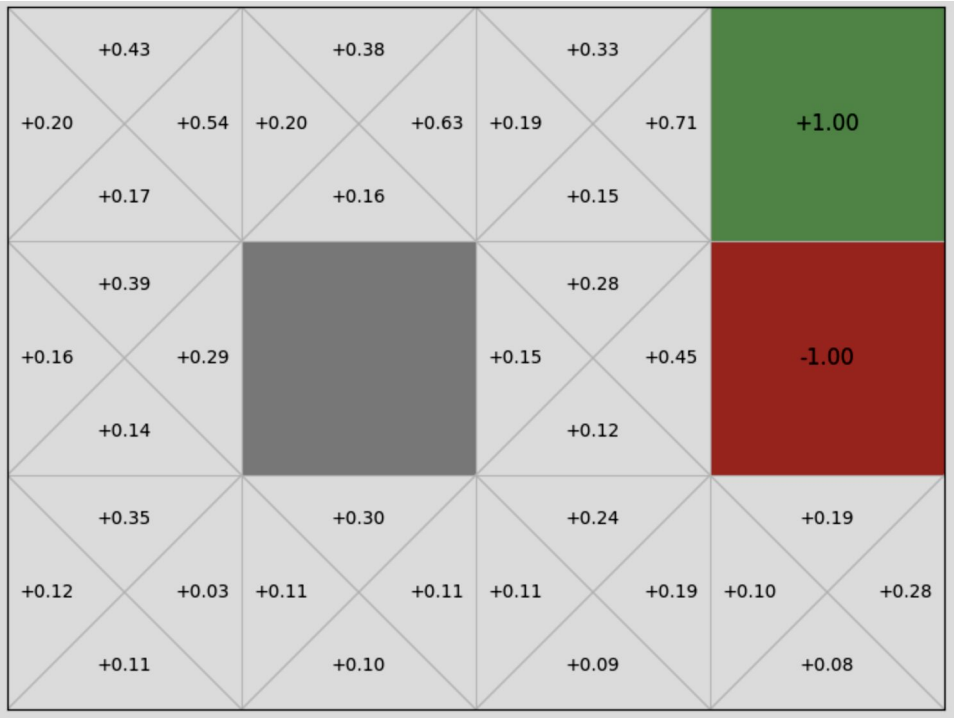
Monte Carlo Control



1. Initialize a random policy
2. Policy Evaluation: find the value function for the current policy
3. Policy Improvement: act greedily w.r.t. the found value function
4. Repeat until optimal policy

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

Greedy Selection with Q-Function Example

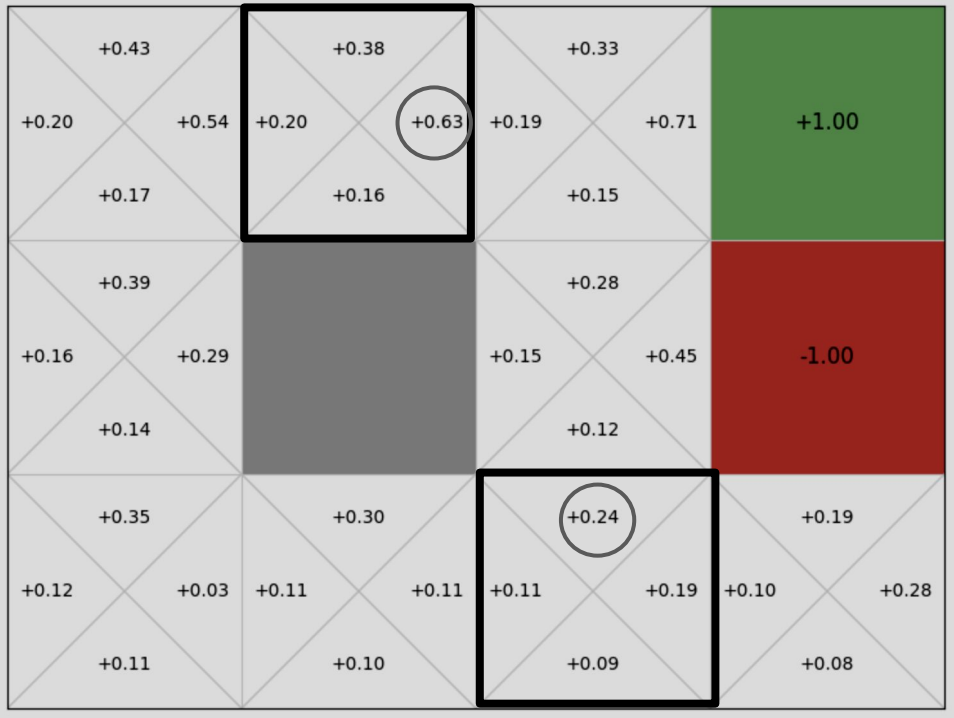


Actions: Up, Down, Left, Right

Green and Red states are terminating

How to select the best actions (greedify policy)?

Greedy Selection with Q Function Example



$$\pi'(s) = \arg\max_a Q(s, a)$$

Just pick the largest state-action value!

No need for dynamics model, all future transitions "encoded" in Q value already

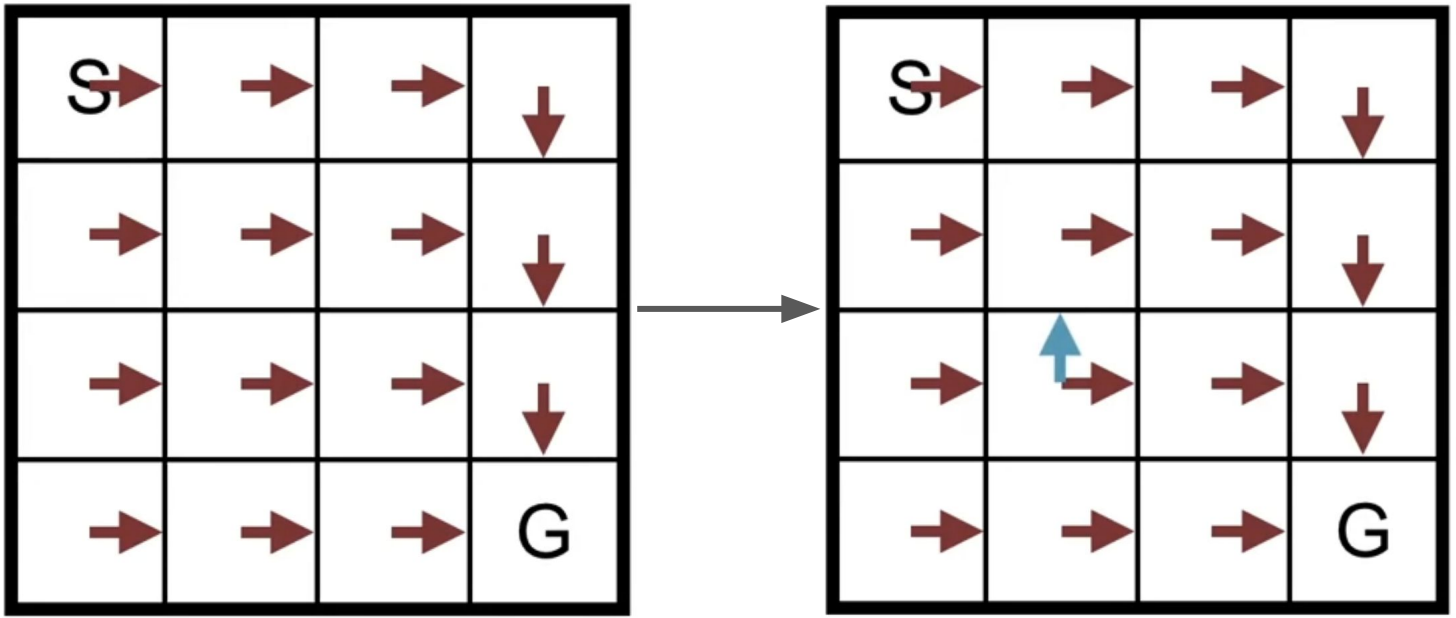
Exploration Methods for Monte Carlo

Why is exploration especially important in Monte Carlo methods?

- Monte Carlo is a sample-based method, so what if certain actions are never taken?
  - **Won't know the state-action value!** Must explore
- Two methods of exploration:
  - Monte Carlo with Exploring Starts
  - Epsilon-soft policies

Monte Carlo with Exploring Starts

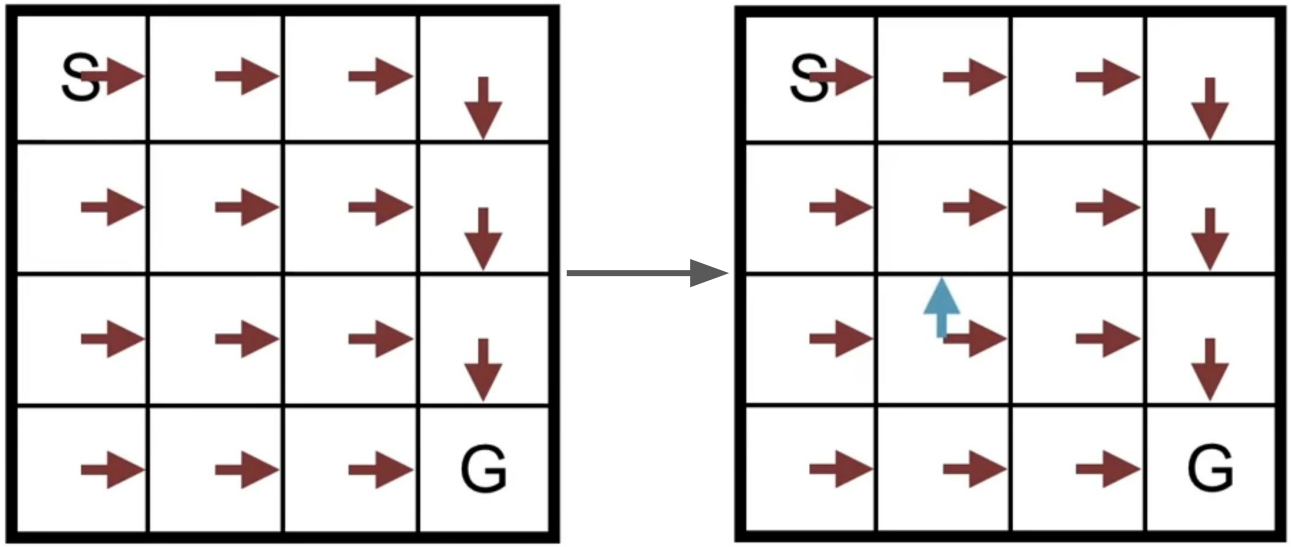
Randomly sample a starting state action pair, such that ALL such pairs are eventually sampled



Monte Carlo with Exploring Starts

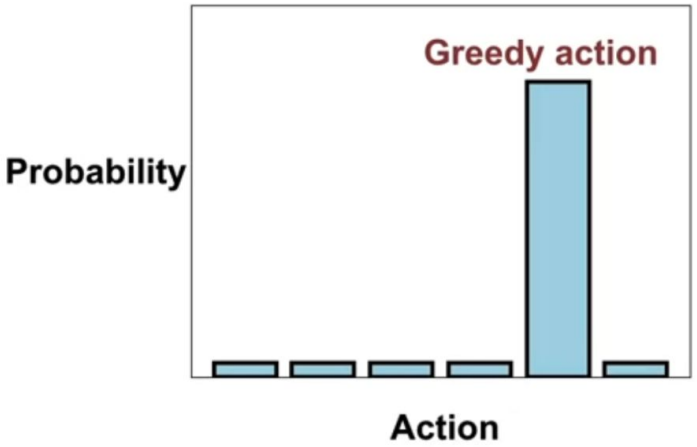
$$s_0, a_0, s_1, a_1, s_2, a_2, \cdots$$

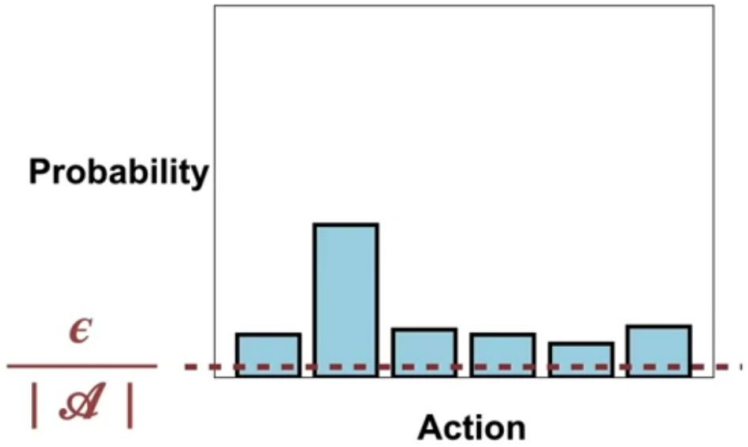
Random From π and p



Epsilon Soft Policies

Sometimes, not possible to use exploring starts. Hence, use epsilon-soft policies instead





First-Visit MC Prediction

First-visit MC prediction, for estimating $V \approx v_\pi$

Input: a policy π to be evaluated

Initialize:

$V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$

$Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Loop forever (for each episode):

Generate an episode following π: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:

$G \leftarrow \gamma G + R_{t+1}$

Unless S_t appears in $S_0, S_1, \ldots, S_{t-1}$:

Append G to $Returns(S_t)$

$V(S_t) \leftarrow \text{average}(Returns(S_t))$

End of Week 2 Questions?

Please Fill Out the Feedback Survey