Reinforcement Learning IV

Lecture 23

Learning strategy

Knowledge of **Environment**	Model-based (planning)	Model-free
No knowledge Must learn from experience		**Monte Carlo Learning**
Perfect knowledge Known MDP	**Dynamic Programming** Policy iteration Value iteration	

Reinforcement Learning (top row)

From Dynamic Programming to true RL

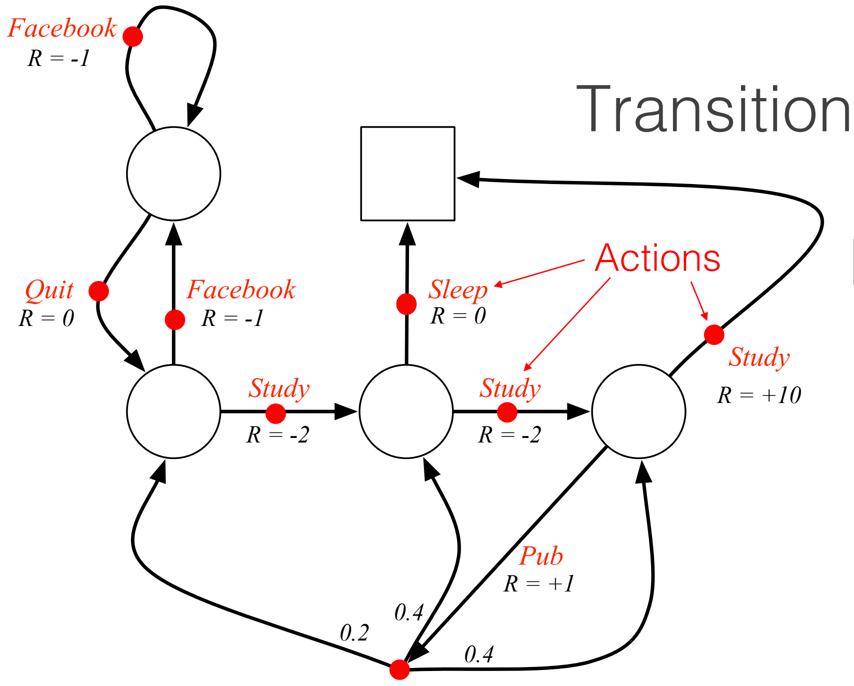
We assume a **fully known MPD environment**
(Markov Decision Process)

Dynamic Programming

1. How well will a policy work? — **Policy evaluation**
2. How can we find a better policy? — **Policy improvement**
3. How do we find the best policy? — **Policy iteration**
4. How do we find the best policy faster? — **Value iteration**
5. Are there other approaches? — **Generalized Policy Iteration**

What if we don't have a fully known MDP? — **Monte Carlo Methods**

Markov Decision Process



Components:

State space S

Transition probabilities, P

Rewards, R

Discount rate, γ

Actions, A

Returns (Expected future rewards)

(discount factor weights the the future)

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} \dots$$

Policy (how we choose actions)

(can be stochastic or deterministic)

$$\pi(a|s) = P(a|s)$$

State value function

(expected return from state s, and following policy π)

$$v_\pi(s) = E[G_t|s]$$

$$v_\pi(s) = E[R_s^a + \gamma v_\pi(s')|s]$$

Action value function

(expected return from state s, taking action a, and following policy π)

$$q_\pi(s,a) = E[G_t|s,a]$$

$$q_\pi(s,a) = E[R_s^a + \gamma q_\pi(s',a')|s,a]$$

David Silver, UCL, 2015

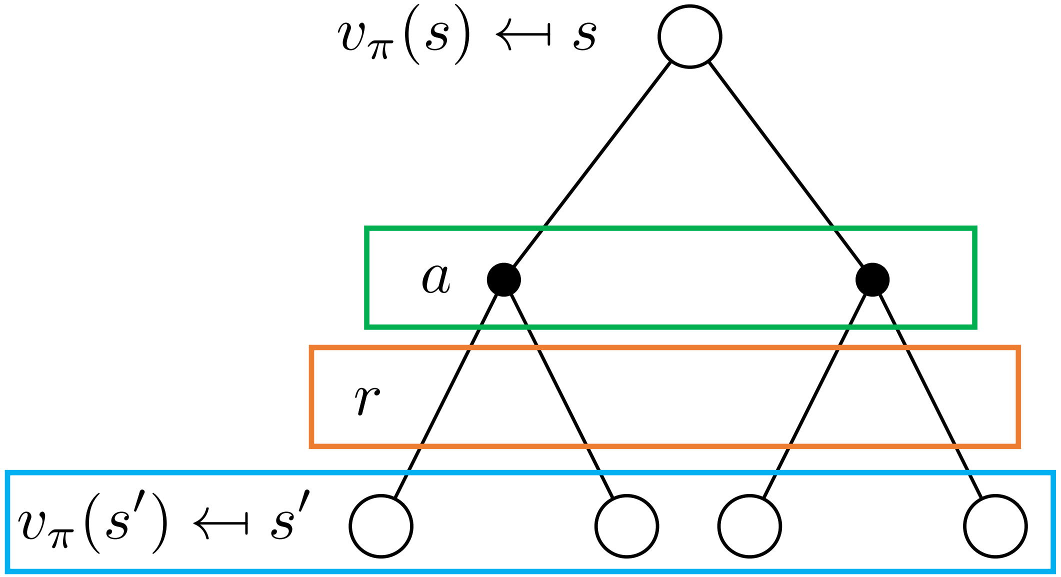
Bellman Expectation Equations for the state value function

(expected return from state s, and following policy π)

$$v_\pi(s) = E[G_t|s]$$

$$v_\pi(s) = E[R_s^a + \gamma v_\pi(s')|s]$$

$$R_s^a = E[r_{t+1}|S_t = s, A_t = a]$$



Expectation over the possible actions

Expectation over the rewards

(based on state and choice of action)

Expectation over the next possible states

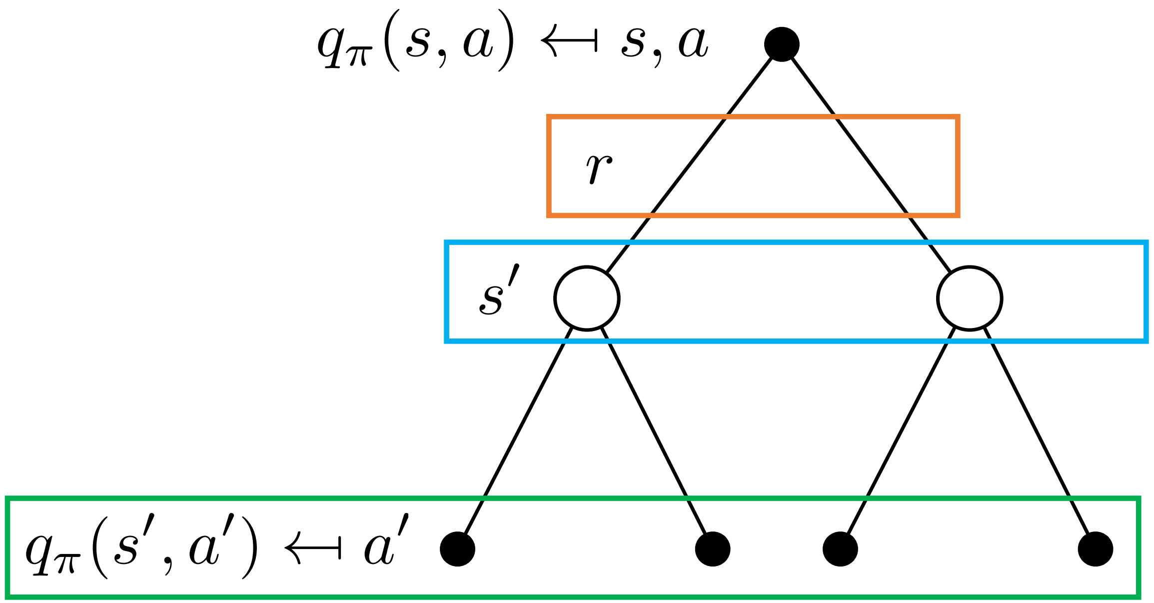
$$v_\pi(s) = \underbrace{\sum_a \pi(a|s)}\left(\underbrace{R_s^a} + \gamma \underbrace{\sum_{s'} P_{ss'}^a v_\pi(s')}\right)$$

Bellman Expectation Equations for the action value function

(expected return from state s, taking action a, then following policy π)

$$q_\pi(s,a) = E[G_t|s,a]$$

$$q_\pi(s,a) = E[R_s^a + \gamma q_\pi(s',a')|s,a]$$



$R_s^a = E[r_{t+1}|S_t = s, A_t = a]$

Expectation over the rewards

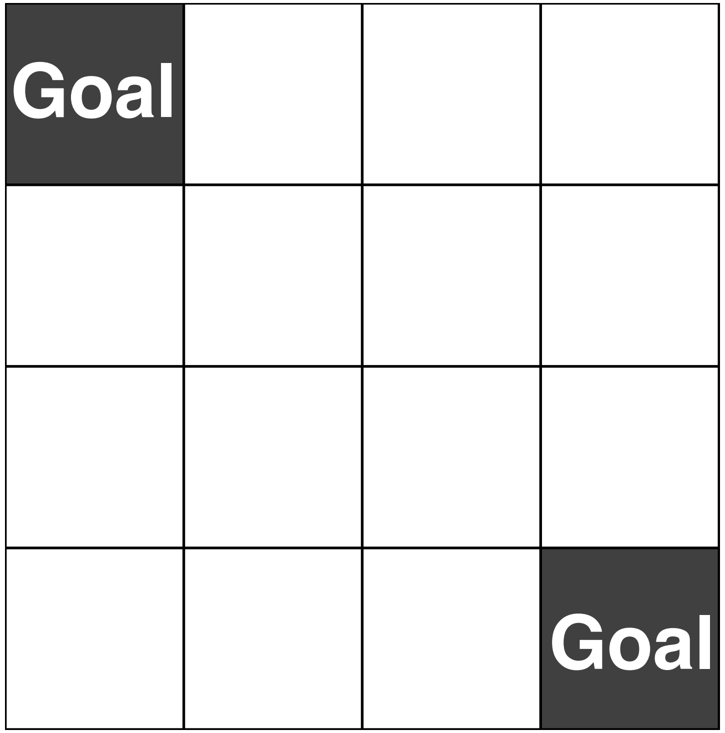
(based on state and choice of action)

Expectation over the next possible states

Expectation over the possible actions

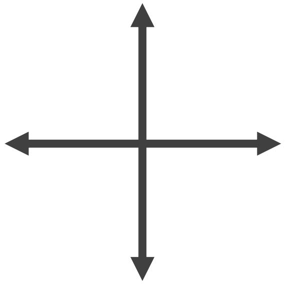
$$q_\pi(s,a) = R_s^a + \gamma \sum_{s'} P_{ss'}^a \sum_{a'} \pi(a'|s') q_\pi(s',a')$$

Running example: Gridworld



14 states and 2 ways to the terminal state labeled "goal"

Valid actions:



Reward:

-1 for all transitions

(until the terminal state has been reached)

Note: actions that would take the agent off the board leave the position unchanged, but result in a reward of -1

Sutton and Barto, 2018

1. Policy Evaluation

Input: policy $\pi(a|s)$

Output: value function $v_\pi(s)$
(unknown)

1. Select a policy function to evaluate (find the value function of)
2. Start with a guess of the value function, v_0 (often all zeros)
3. **Iteratively** apply the Bellman Expectation Equation to "backup" the values until they converge on the actual value function for the policy, v_π

$$v_0 \rightarrow v_1 \rightarrow \cdots \rightarrow v_\pi$$

Adapted from David Silver, 2015

1. Policy Evaluation
in Gridworld

Policy: $\pi(a|s) = 0.25$
(randomly go in any direction!)

Value function initialization:

$v_0(s) = 0$ (all zeros)

$v_0(s)$
(initialization)

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

1. Policy Evaluation in Gridworld

Policy: $\pi(a|s) = \frac{1}{4}$

(randomly go in any direction!)

Bellman Expectation Equation:

$$v_{k+1}(s) = \sum_{a} \pi(a|s)\left(R_s^a + \gamma \sum_{s'} P_{ss'}^a v_k(s')\right)$$

In Gridworld: 0.25 $\quad -1 \quad$ 1 (once you pick an action there's no uncertainty as to which state you'll transition to)

$v_0(s)$

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

$$v_{k+1}(s) = \sum_{a} \frac{1}{4}\left(-1 + \sum_{s'} v_k(s')\right) = -1 + \sum_{a} \frac{1}{4} \sum_{s'} v_k(s') = -1 + \sum_{a} \frac{1}{4} v_k(s')$$

Each action leads to only one state, so the sum over states is not needed

Average of the value of the 4 neighboring states

1. Policy Evaluation
in Gridworld

$$v_{k+1}(s) = -1 + \sum_a \frac{1}{4} v_k(s')$$

$$v_1 = -1 + \sum_a \frac{1}{4} v_k(s') \quad = -1$$

$v_0(s)$

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

One neighborhood
in $v_0(s)$

	0		
0		0	
	0		

$v_1(s)$

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

$v_0(s)$

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

$v_1(s)$

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

$v_2(s)$

0	-1.7	-2	-2
-1.7	-2	-2	-2
-2	-2	-2	-1.7
-2	-2	-1.7	0

$v_3(s)$

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

$v_{10}(s)$

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

$v_\infty(s) = v_\pi(s)$

0	-14	-20	-22
-14	-18	-20	-20
-20	-20	-18	-14
-22	-20	-14	

We've found the value function (expected returns) from our random movement policy

1. Policy Evaluation

in Gridworld

2. Policy Improvement

Input: policy $\pi(a|s)$
Output: better policy $\pi'(a|s)$

Definition of better: has greater or equal expected return in all states:
$v_{\pi'}(s) \geq v_\pi(s)$ for all states

1. Select a policy function to improve
2. Evaluate the value function (our last discussion)
3. **Greedily** select a new policy, π', that chooses actions that maximize value

$$\pi'(s) = \text{greedy}(s)$$

(i.e. pick the action that brings us to the state with highest value)

Adapted from David Silver, 2015

2. Policy Improvement

Input: policy $\pi(a|s)$

Output: better policy $\pi'(a|s)$

How do we do this: $\pi'(s) = \text{greedy}(s)$

i.e. pick the **action** that brings us to the state with **highest value**

We can use the state value function to help us choose the right action:

$$\pi'(s) = \arg\max_{a} q_{\pi}(s,a)$$

Reminder:

Action value function

(expected return from state s, taking action a, and following policy π)

$$q_{\pi}(s,a) = E[G_t|s,a]$$

Adapted from David Silver, 2015

2. Policy Improvement
in Gridworld

Value function:

In this case, $q_\pi(s, \pi(s)) = v_\pi(s)$ since each action leads to only one state

$v_0(s)$

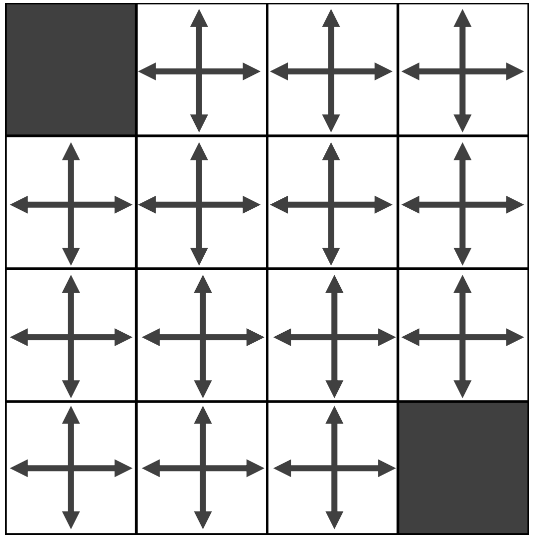
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

Initial policy:

$$\pi(a|s) = \frac{1}{4}$$

(randomly go in any direction!)

$\pi(s)$



2. Policy Improvement
in Gridworld

Value function:

In this case, $q_\pi(s, \pi(s)) = v_\pi(s)$ since each action leads to only one state

$v_0(s)$

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

$v_\infty(s) = v_\pi(s)$

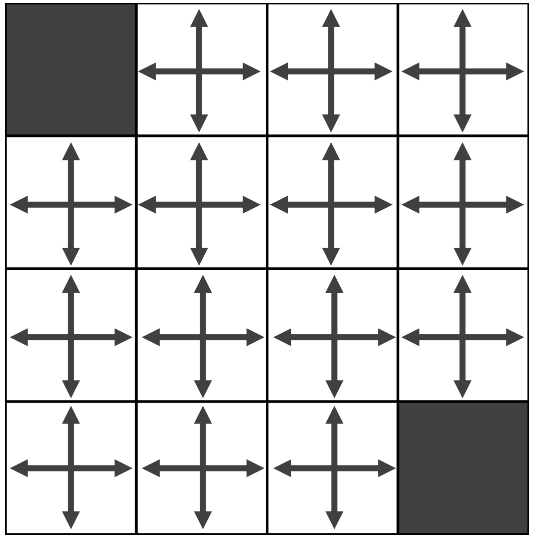
0	-14	-20	-22
-14	-18	-20	-20
-20	-20	-18	-14
-22	-20	-14	

Initial policy:

$$\pi(a|s) = \frac{1}{4}$$

(randomly go in any direction!)

$\pi(s)$



2. Policy Improvement
in Gridworld

$v_0(s)$

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

$v_\infty(s) = v_\pi(s)$

0	-14	-20	-22
-14	-18	-20	-20
-20	-20	-18	-14
-22	-20	-14	

Value function:

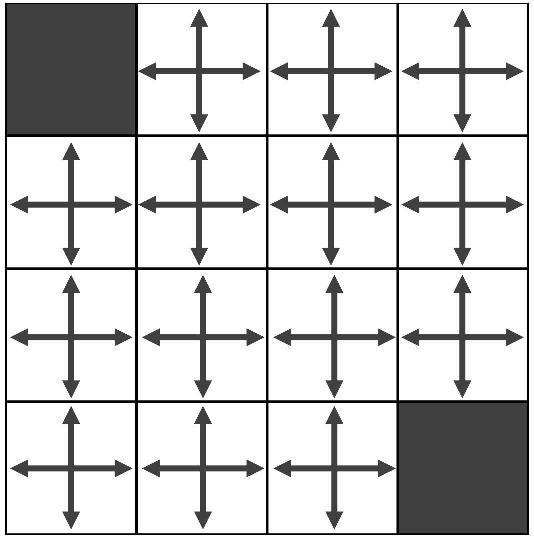
In this case, $q_\pi(s, \pi(s)) = v_\pi(s)$ since each action leads to only one state

Initial policy:

$$\pi(a|s) = \frac{1}{4}$$

(randomly go in any direction!)

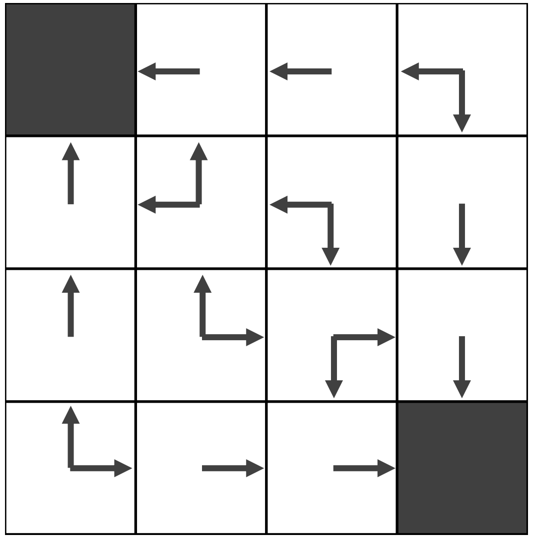
$\pi(s)$



Improved policy:

$\pi'(s)$

(in this case this is the optimal policy)



3. Policy Iteration

Input: policy $\pi(a|s)$

Output: **best** policy $\pi^*(a|s)$

Best in the sense that: $v_{\pi^*}(s) \geq v_\pi(s)$ for all states and for all **policies**

Policy **Evaluation**

$$\begin{tikzcd} \pi \arrow[r, bend left, "v \to v_\pi"] & v \arrow[l, bend left, "\text{greedy}(v_\pi) \to \pi'"] \end{tikzcd}$$

Policy **Improvement**

⋮

This process will converge onto the optimal functions

$$\pi^* \rightleftarrows v^*$$

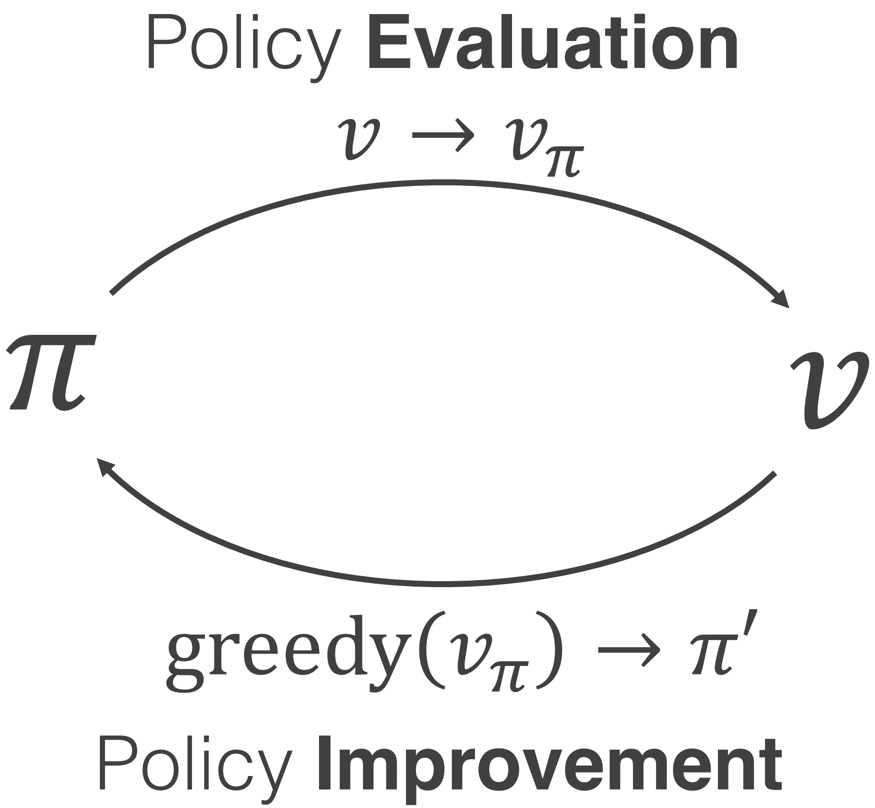
$$\pi_0 \xrightarrow{\text{E}} v_{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} v_{\pi_1} \to \cdots \xrightarrow{\text{I}} \pi^* \xrightarrow{\text{E}} v^*$$

Adapted from David Silver, 2015 and Sutton and Barto, 1998

3. Policy Iteration

Input: policy $\pi(a|s)$

Output: **best** policy $\pi^*(a|s)$



1. **Policy Evaluation**: estimate v_π
   Iterative policy evaluation
   Note: This is VERY slow
2. **Policy Improvement**: generate $\pi' \geq \pi$
   Greedy policy improvement
3. Iterate 1 and 2 until convergence

Adapted from David Silver, 2015 and Sutton and Barto, 1998

$v_0(s)$

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

$v_1(s)$

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

$v_2(s)$

0	-1.7	-2	-2
-1.7	-2	-2	-2
-2	-2	-2	-1.7
-2	-2	-1.7	0

$v_3(s)$

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

$v_{10}(s)$

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

$v_\infty(s) = v_\pi(s)$

0	-14	-20	-22
-14	-18	-20	-20
-20	-20	-18	-14
-22	-20	-14	

So far, we've run policy evaluation all the way to convergence (**this is slow**)

$v_0(s)$

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

$v_1(s)$

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

What if we stopped after one sweep. This is…

4. Value Iteration

4. Value Iteration

Input: policy $\pi(a|s)$

Output: **best** policy $\pi^*(a|s)$

Policy **Evaluation**

$v \to v_\pi$

π v

$\text{greedy}(v_\pi) \to \pi'$

Policy **Improvement**

1. **Policy Evaluation**: estimate v_π
   One-sweep of policy evaluation
2. **Policy Improvement**: generate $\pi' \geq \pi$
   Greedy policy improvement
3. Iterate 1 and 2 until convergence

Adapted from David Silver, 2015 and Sutton and Barto, 1998

Demo

https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld_dp.html

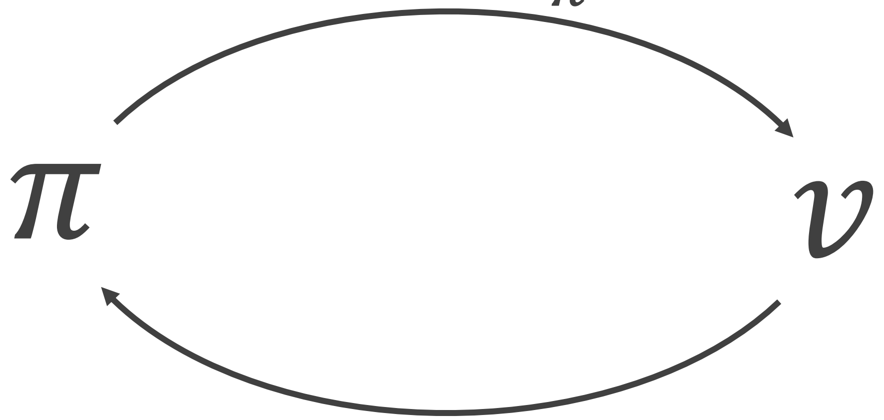
5. Generalized Policy Iteration

Input: policy $\pi(a|s)$

Output: **best** policy $\pi^*(a|s)$

Policy **Evaluation**

$v \rightarrow v_\pi$



$\text{improvement}(v_\pi) \rightarrow \pi'$

Policy **Improvement**

1. **Policy Evaluation**: estimate v_π
   Any policy evaluation algorithm
2. **Policy Improvement**: generate $\pi' \geq \pi$
   Any policy improvement algorithm
3. Iterate 1 and 2 until convergence

Adapted from David Silver, 2015 and Sutton and Barto, 1998

So far, we've assumed full knowledge of the environment (MDP)

What if we DO NOT assume full knowledge of the environment (MDP)

This means we have to learn by experience:
true reinforcement learning

6. Monte Carlo Policy Evaluation

For **state** values

Input: policy $\pi(a|s)$

Output: state value $v_\pi(s)$

1. Select a policy function to evaluate (find the value function of)
2. Start with a guess of the value function, v_0 (often all zeros)
3. Repeat forever:
   - A. Generate an episode (takes actions until a terminal state)
   - B. Save the returns following the first occurrence of each state
   - C. Assign $\mathrm{AVG}(\mathrm{Returns}(s)) \rightarrow \hat{v}_\pi(s)$

Sutton and Barto, 1998

6. Monte Carlo Policy Evaluation

For **state** values

$v_0(s)$

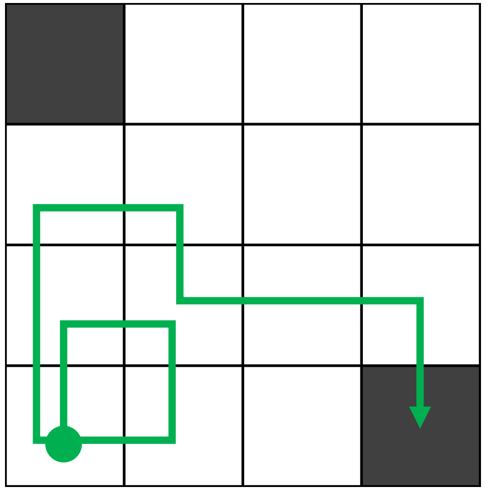
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

6. Monte Carlo Policy Evaluation

For **state** values

Episode 1

Total Reward: -11



For each state, we store the running returns seen **after** the first visit to that state

$v_0(s)$

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

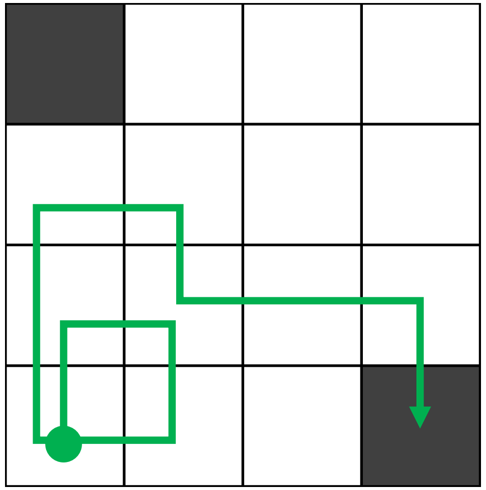
6. Monte Carlo Policy Evaluation

For **state** values

For each state, we store the running returns seen **after** the first visit to that state

Episode 1

Total Reward: -11



Episode 1 **returns** after the first visit of each state

$R^{(1)}$

-5	-4		
-10	-9	-2	-1
-11	-8		

Discount rate: $\gamma = 1$

$v_0(s)$

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

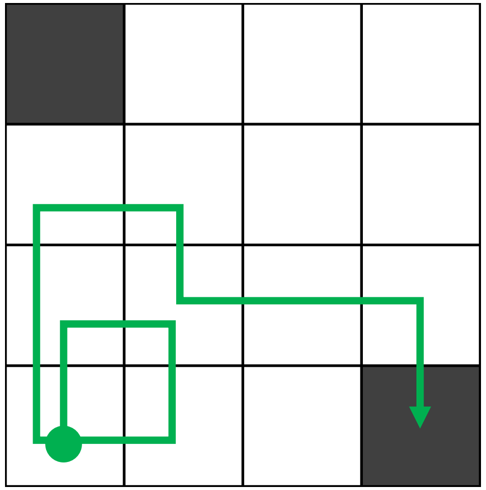
6. Monte Carlo Policy Evaluation

For **state** values

For each state, we store the running returns seen **after** the first visit to that state

Episode 1

Total Reward: -11



Episode 1 **returns** after the first visit of each state

$R^{(1)}$

-5	-4		
-10	-9	-2	-1
-11	-8		

Discount rate: $\gamma = 1$

$v_0(s)$

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

The value function is the **running average** of the returns after the visit to that state, averaged over episodes

(or zero if the state has not been visited)

$v_1(s)$

0	0	0	0
-5	-4	0	0
-10	-9	-2	-1
-11	-8	0	0

v_1 is just the first visit returns, $R^{(1)}$

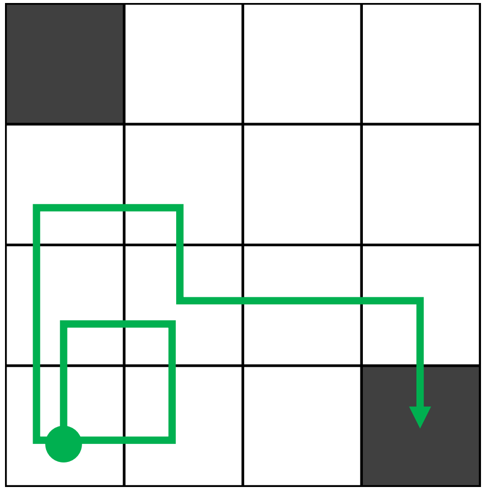
6. Monte Carlo Policy Evaluation

For **state** values

For each state, we store the running returns seen **after** the first visit to that state

Episode 1

Total Reward: -11



Episode 1 **returns** after the first visit of each state

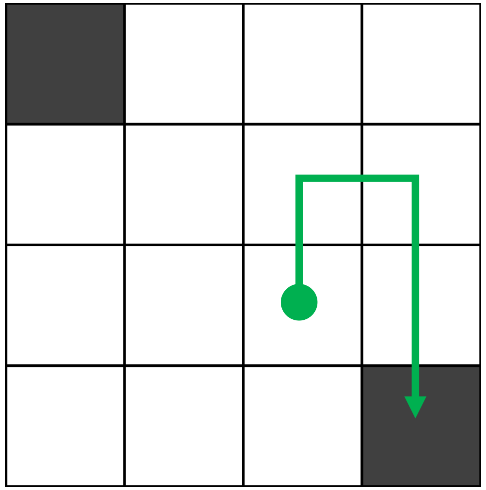
$R^{(1)}$

-5	-4		
-10	-9	-2	-1
-11	-8		

Discount rate: $\gamma = 1$

Episode 2

Total Reward: -4



$v_0(s)$

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

The value function is the **running average** of the returns after the visit to that state, averaged over episodes

(or zero if the state has not been visited)

$v_1(s)$

0	0	0	0
-5	-4	0	0
-10	-9	-2	-1
-11	-8	0	0

v_1 is just the first visit returns, $R^{(1)}$

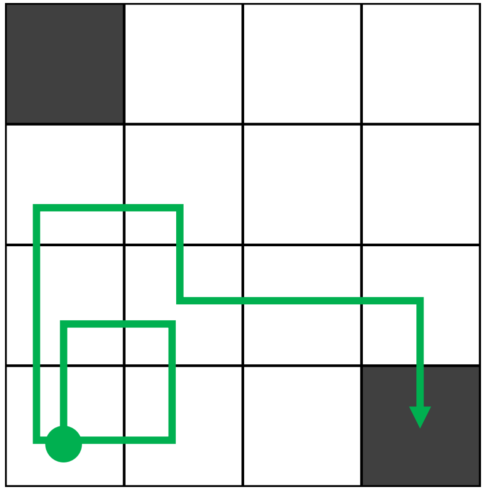
6. Monte Carlo Policy Evaluation

For **state** values

For each state, we store the running returns seen **after** the first visit to that state

Episode 1

Total Reward: -11



Episode 1 **returns** after the first visit of each state

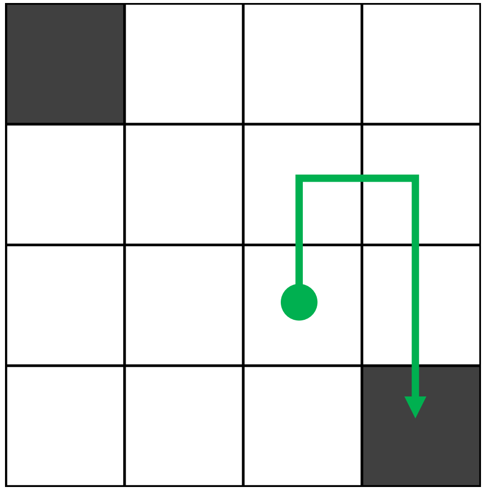
$R^{(1)}$

-5	-4		
-10	-9	-2	-1
-11	-8		

Discount rate: $\gamma = 1$

Episode 2

Total Reward: -4



Episode 2 **returns** from the first visit of each state

$R^{(2)}$

		-3	-2
		-4	-1

Discount rate: $\gamma = 1$

$v_0(s)$

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

The value function is the **running average** of the returns after the visit to that state, averaged over episodes (or zero if the state has not been visited)

$v_1(s)$

0	0	0	0
-5	-4	0	0
-10	-9	-2	-1
-11	-8	0	0

v_1 is just the first visit returns, $R^{(1)}$

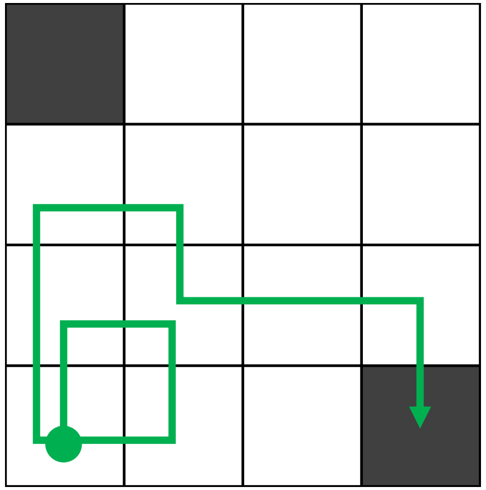
6. Monte Carlo Policy Evaluation

For **state** values

For each state, we store the running returns seen **after** the first visit to that state

Episode 1

Total Reward: -11



Episode 1 **returns** after the first visit of each state

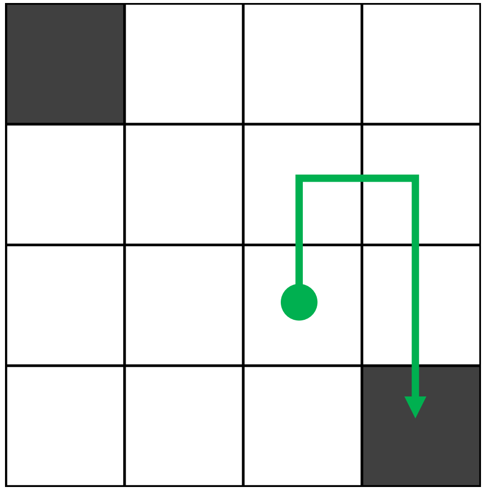
$R^{(1)}$

-5	-4		
-10	-9	-2	-1
-11	-8		

Discount rate: $\gamma = 1$

Episode 2

Total Reward: -4



Episode 2 **returns** from the first visit of each state

$R^{(2)}$

		-3	-2
		-4	-1

Discount rate: $\gamma = 1$

$v_0(s)$

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

The value function is the **running average** of the returns after the visit to that state, averaged over episodes (or zero if the state has not been visited)

$v_1(s)$

0	0	0	0
-5	-4	0	0
-10	-9	-2	-1
-11	-8	0	0

v_1 is just the first visit returns, $R^{(1)}$

$v_2(s)$

0	0	0	0
-5	-4	-3	-2
-10	-9	-3	-1
-11	-8	0	0

v_2 is the average first visit returns, $R^{(1)}$ and $R^{(2)}$

The **state value function** doesn't tell us about actions

If we don't have a model, to pick a policy we need **action values**

6. Monte Carlo Policy Evaluation

For **action** values

Input: policy $\pi(a|s)$

Output: action value $q_\pi(s,a)$

1. Select a policy function to evaluate (find the value function of)
2. Start with a guess of the value function, v_0 (often all zeros)
3. Repeat forever:
   - A. Generate an episode (takes actions until a terminal state)
   - B. Save returns following first occurrence of each state **& action**
   - C. Assign $\mathrm{AVG}(\mathrm{Returns}(\mathrm{s},\mathrm{a})) \rightarrow \hat{q}_\pi(s,a)$

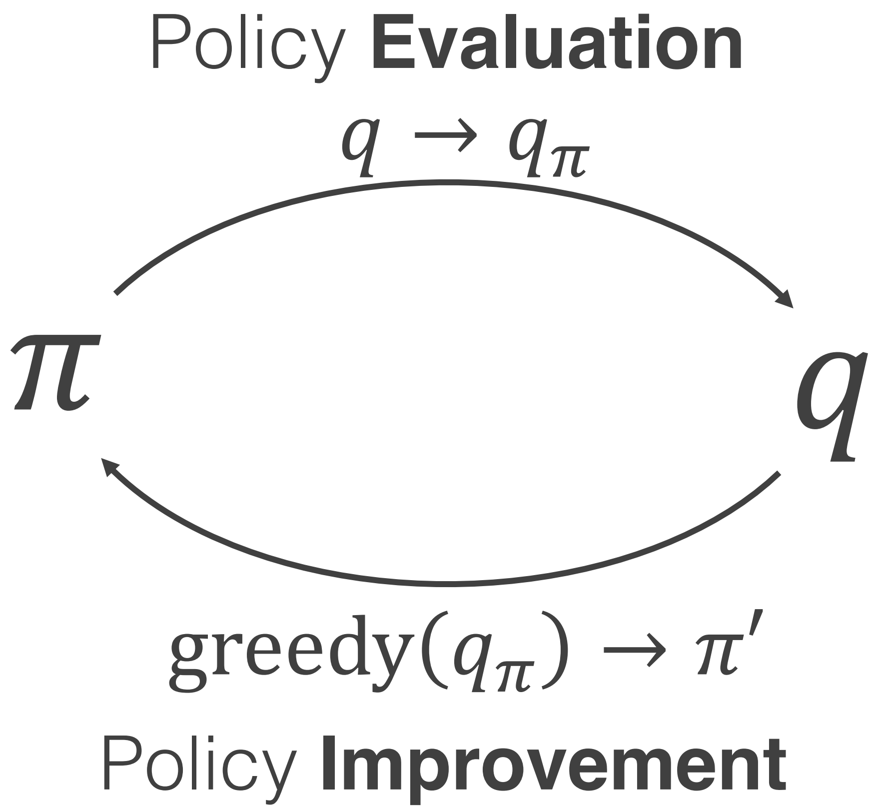
Sutton and Barto, 1998

7. Monte Carlo Control

(policy iteration)

Input: policy $\pi(a|s)$

Output: **best policy** $\pi^*(a|s)$



1. **Policy Evaluation**: estimate q_π
   Monte Carlo action policy evaluation
2. **Policy Improvement**: generate $\pi' \geq \pi$
   Greedy policy improvement
3. Iterate 1 and 2 until convergence

Sutton and Barto, 1998

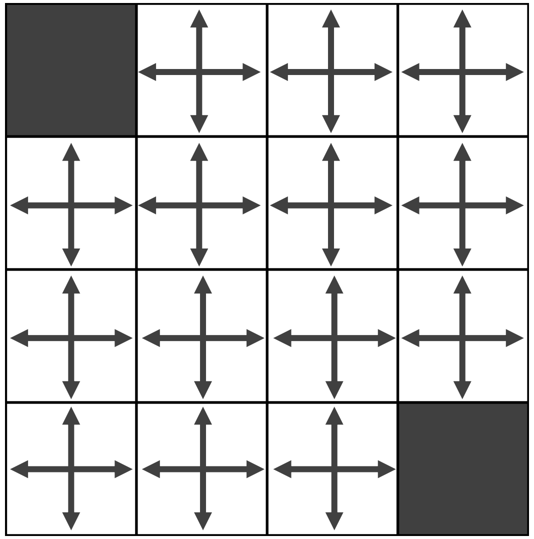
7. Monte Carlo Control

In this case, $q_\pi(s, \pi(s)) = v_\pi(s)$ since each action leads to only one state. In this way the optimal $q_\pi(s, \pi(s))$ value is easily read off the $v_\pi(s)$ grid

$v_0(s)$

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

$\pi_0(s)$



Discount rate: $\gamma = 1$

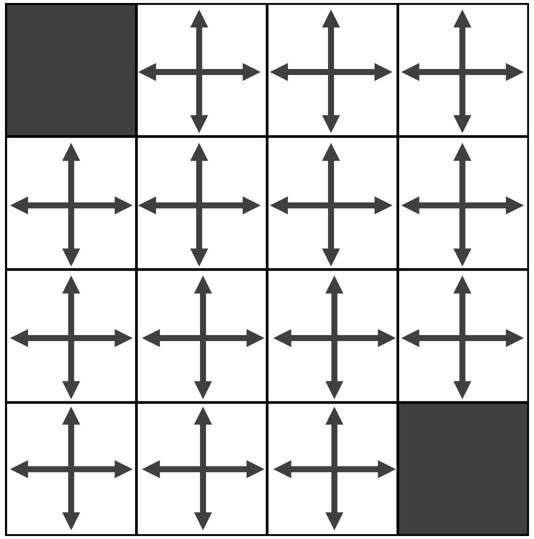
7. Monte Carlo Control

In this case, $q_\pi(s, \pi(s)) = v_\pi(s)$ since each action leads to only one state. In this way the optimal $q_\pi(s, \pi(s))$ value is easily read off the $v_\pi(s)$ grid

$v_0(s)$

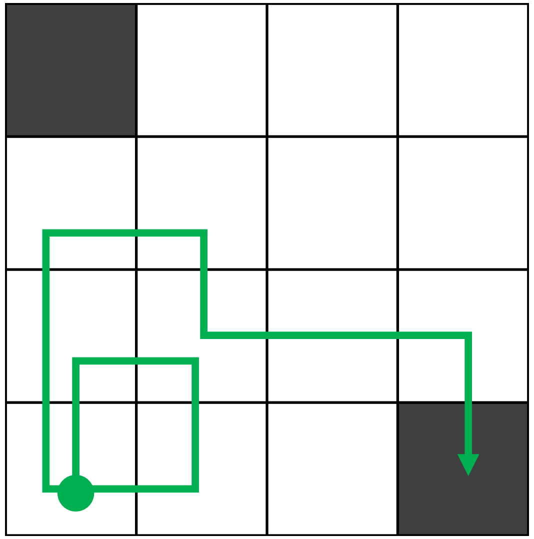
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

$\pi_0(s)$



Episode 1

Total Reward: -11



Discount rate: $\gamma = 1$

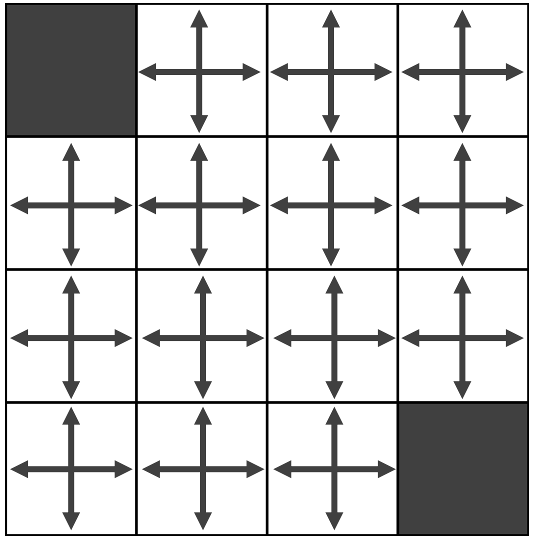
7. Monte Carlo Control

In this case, $q_{\pi}(s, \pi(s)) = v_{\pi}(s)$ since each action leads to only one state. In this way the optimal $q_{\pi}(s, \pi(s))$ value is easily read off the $v_{\pi}(s)$ grid

$v_0(s)$

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

$\pi_0(s)$

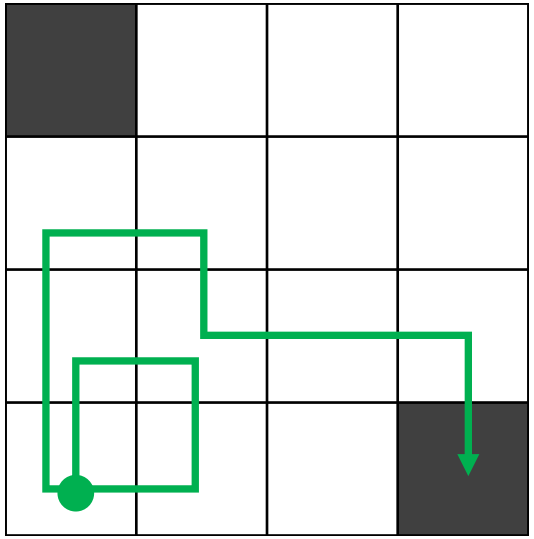


$v_1(s)$

0	0	0	0
-5	-4	0	0
-10	-9	-2	-1
-11	-8	0	0

Episode 1

Total Reward: -11



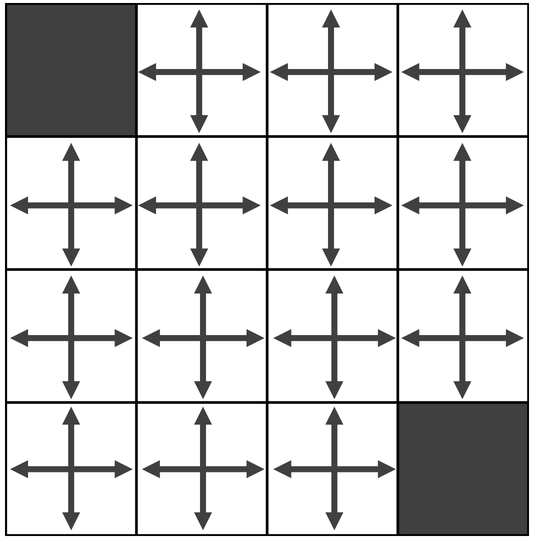
Discount rate: $\gamma = 1$

7. Monte Carlo Control

$v_0(s)$

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

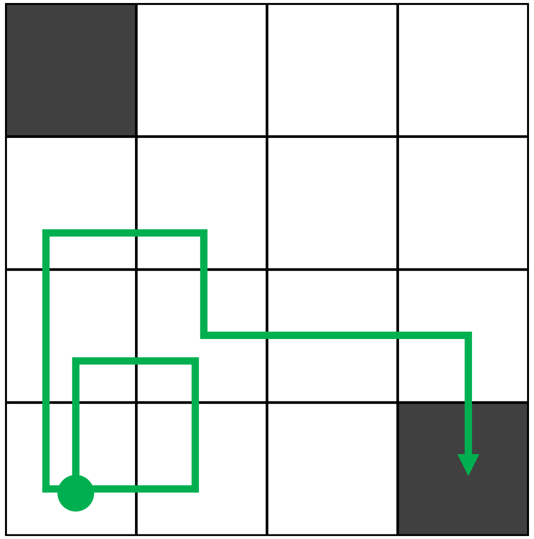
$\pi_0(s)$



In this case, $q_\pi(s, \pi(s)) = v_\pi(s)$ since each action leads to only one state. In this way the optimal $q_\pi(s, \pi(s))$ value is easily read off the $v_\pi(s)$ grid

Episode 1

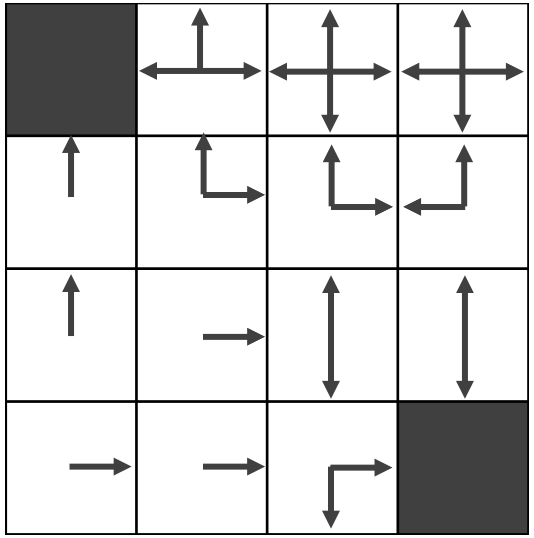
Total Reward: -11



$v_1(s)$

0	0	0	0
-5	-4	0	0
-10	-9	-2	-1
-11	-8	0	0

$\pi_1(s)$



Discount rate: $\gamma = 1$

Extensions

Monte Carlo methods require that we finish each episode before updating
Solution: **Temporal Difference** (TD) methods

What if we want to learn about one policy while following or observing another?
Solution: **Off-policy learning** instead of on-policy learning

What if our state space has too many states that we can't build a table of values?
Solution: **Value function approximation** (involving supervised learning techniques)

How can we simulate what the environment might output for next states and rewards?
Solution: **Model-based learning**: simulate the environment and plan ahead

From Dynamic Programming to true RL

We assume a **fully known MPD environment**

(Markov Decision Process) **Sutton & Barto, Chapter 3**

1. How well will a policy work?	**Policy evaluation**
2. How can we find a better policy?	**Policy improvement**
3. How do we find the best policy?	**Policy iteration**
4. How do we find the best policy faster?	**Value iteration**
5. Are there other approaches?	**Generalized Policy Iteration**

Sutton & Barto, Chapter 4

What if we don't have a fully known MDP? **Monte Carlo Methods**

Sutton & Barto, Chapter 5