Carnegie Mellon

School of Computer Science

Deep Reinforcement Learning and Control

Markov Decision Processes, Value Iteration, Policy Iteration

Spring 2020, CMU 10-403

Katerina Fragkiadaki

Supervision for learning to act

1. **Learning from expert demonstrations** (last lecture)
   Instructive feedback: the expert directly suggests correct actions, e.g., your advisor directly suggests to you ideas that are worth pursuing

2. **Learning from rewards** while interacting with the environment
   Evaluative feedback: the environment provides signal whether actions are good or bad. E.g., your advisor tells you if your research ideas are worth pursuing (but does not suggest to you other ideas).

Note: Evaluative feedback depends on the current policy the agent has: if you never suggest good ideas, you will never have the chance to know they are worthwhile. Instructive feedback is independent of the agent's policy.

Finite Markov Decision Process

A **Finite** Markov Decision Process is a tuple $(\mathcal{S}, \mathcal{A}, T, r, \gamma)$

- $\mathcal{S}$ is a finite set of states
- $\mathcal{A}$ is a finite set of actions
- p is one step dynamics function
- r is a reward function
- γ is a discount factor $\gamma \in [0,1]$

Definitions

Agent: an entity that is equipped with sensors, in order to sense the environment, and end-effectors in order to act in the environment, and goals that he wants to achieve

Policy: a mapping function from observations (sensations, inputs of the sensors) to actions of the end effectors.

Model: the mapping function from states/observations and actions to future states/observations

Planning: unrolling a model forward in time and selecting the best action sequence that satisfies a specific goal

Plan: a sequence of actions

Markovian States

- A state captures whatever information is available to the agent at step t about its environment.

- The state can include immediate "sensations," highly processed sensations, and structures built up over time from sequences of sensations, memories etc.

- A state should summarize past sensations so as to retain all "essential" information, i.e., it should have the **Markov Property**:

- $\mathbb{P}\left[R_{t+1}=r, S_{t+1}=s' \mid S_0, A_0, R_1, \ldots, A_{t-1}, R_t, S_t, A_t\right]=\mathbb{P}\left[R_{t+1}=r, S_{t+1}=s' \mid S_t, A_t\right]$

  for all $s' \in \mathcal{S}$, $r \in R$, and all histories

- We should be able to throw away the history once state is known

The agent learns a Policy

Definition: A policy is a distribution over actions given states,

$$\pi(a \mid s) = \mathbf{Pr}(A_t = a \mid S_t = s), \forall t$$

- A policy fully defines the behavior of an agent
- The policy is stationary (time-independent)
- During learning, the agent changes his policy as a result of experience

Special case: deterministic policies

$$\pi(s) = \text{the action taken with prob} = 1 \text{ when } S_t = s$$

The recycling robot MDP

- At each step, robot has to decide whether it should (1) actively search for a can, (2) wait for someone to bring it a can, or (3) go to home base and recharge.
- Searching is better but runs down the battery; if runs out of power while searching, has to be rescued (which is bad).
- Decisions made on basis of current energy level: high, low.
- Reward = number of cans collected

The recycling robot MDP

$\mathcal{S} = \{\texttt{high}, \texttt{low}\}$

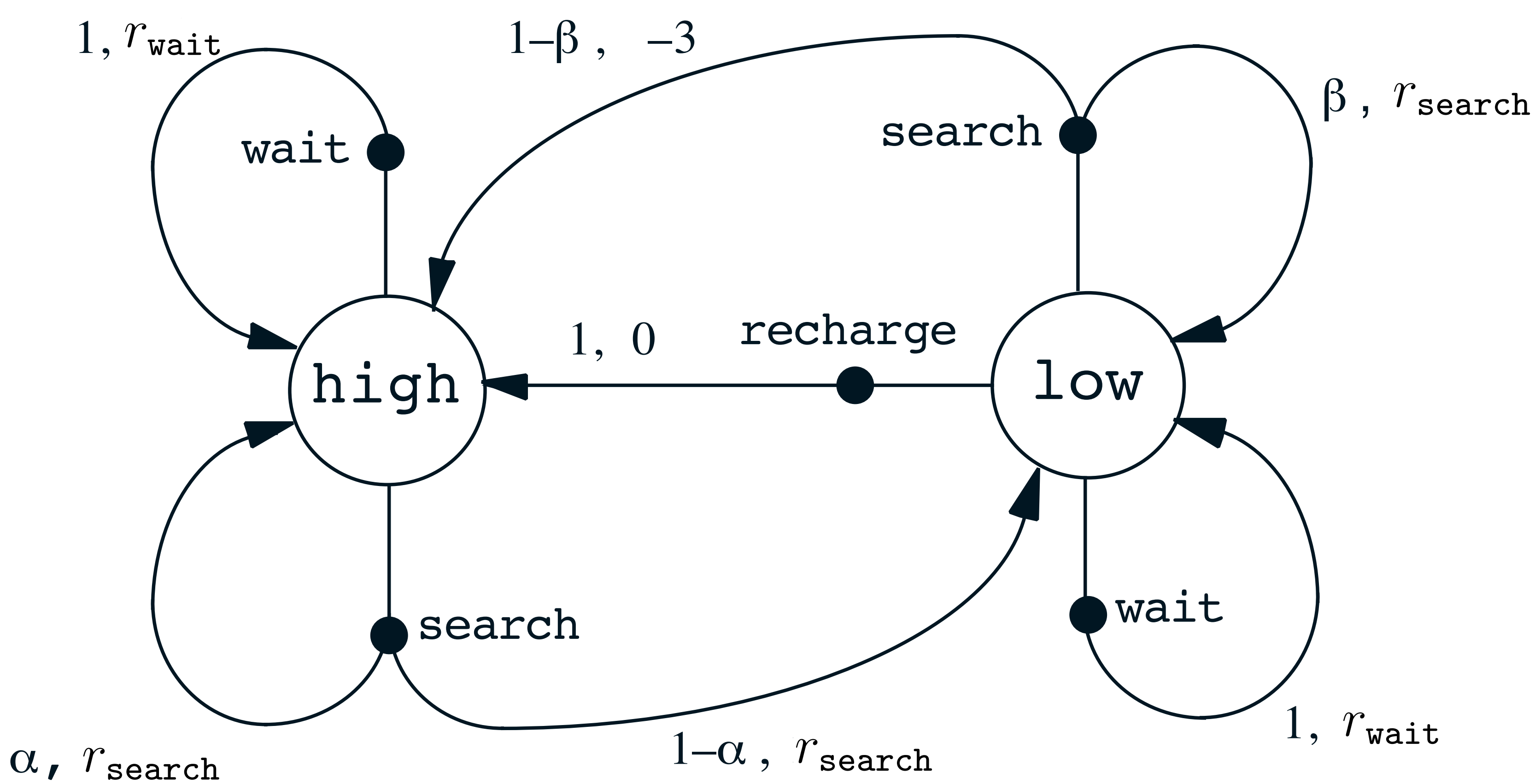
$\mathcal{A}(\texttt{high}) = \{\texttt{search}, \texttt{wait}\}$

$\mathcal{A}(\texttt{low}) = \{\texttt{search}, \texttt{wait}, \texttt{recharge}\}$

$r_{\texttt{search}}$ = expected no. of cans while searching

$r_{\texttt{wait}}$ = expected no. of cans while waiting

$r_{\texttt{search}} > r_{\texttt{wait}}$



Q: what the robot will do does it depend on the number of cans he has collected thus far?

Rewards reflect goals

Rewards are scalar values provided by the environment to the agent that indicate whether goals have been achieved, e.g., 1 if goal is achieved, 0 otherwise, or -1 for overtime step the goal is not achieved

- Goals specify what the agent needs to achieve, not how to achieve it.
- The simplest and cheapest form of supervision, and surprisingly general: All of what we mean by goals and purposes can be well thought of as the maximization of the cumulative sum of a received scalar signal (reward):

$$r(s, a) = \mathbb{E}[R_{t+1} \,|\, S_t = s, A_t = a]$$

Goal seeking behaviour, achieving purposes and expectations can be formulated mathematically as maximizing expected cumulative sum of scalar values…

Returns G_t - Episodic tasks

Episode: A sequence of interactions based on which the reward will be judged at the end.

Episodic tasks: interaction breaks naturally into episodes, e.g., plays of a game, trips through a maze.

In episodic tasks, we almost always use simple total reward:

$$G_t = R_{t+1} + R_{t+2} + \cdots + R_T$$

where T is a final time step at which a terminal state is reached, ending an episode.

Returns G_t - Continuing tasks

Continuing tasks: interaction does not have natural episodes, but just goes on and on…just like real life

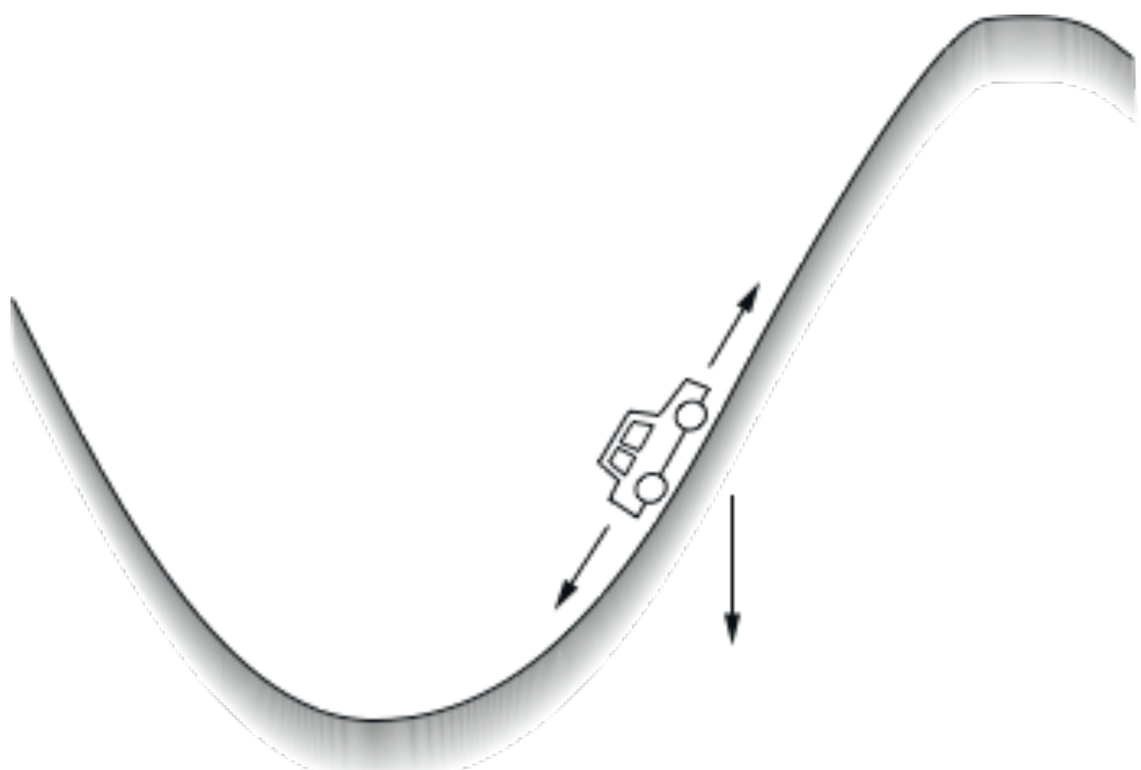
In continuing tasks, we often use simple total discounted reward:

$$G_t = R_{t+1} + \gamma R_{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Why temporal discounting?

Episodes can have finite or infinite length. For infinite length, the undercounted sum blows up, thus we add discounting to prevent this, and treat both cases in a similar manner.

Mountain Car



Get to the top of the hill
as quickly as possible.

reward $= -1$ for each step where **not** at top of hill

$\Rightarrow$ return $=$ $-$ number of steps before reaching top of hill

Return is maximized by minimizing
number of steps to reach the top of the hill.

Value Functions are Expected Returns

- **Definition**: The state-value function $v_\pi(s)$ of an MDP is the expected return starting from state s, and then following policy π:
    - $\mathrm{v}_\pi(s) = \mathbb{E}[G_t \mid S_t = s]$
- The action-value function $q_\pi(s, a)$ is the expected return starting from state s, taking action a, and then following policy:
    - $\mathrm{q}_\pi(s, a) = \mathbb{E}[G_t \mid S_t = s, A_t = a]$
- Q: What are the expectations over (what is stochastic)?

Optimal Value Functions are Best Achievable Expected Returns

- **Definition:** The *optimal state-value function* $v_*(s)$ is the maximum value function over all policies:

$$v_*(s) = \max_\pi v_\pi(s)$$

- The *optimal action-value function* $q_*(s, a)$ is the maximum action-value function over all policies:

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

Solving MDPs

- Prediction: Given an MDP $(\mathcal{S}, \mathcal{A}, T, r, \gamma)$ and a policy

$$\pi(a \mid s) = \mathbb{P}\left[A_t = a \mid S_t = s\right]$$

  find the state and action value functions.

- Optimal control: given an MDP $(\mathcal{S}, \mathcal{A}, T, r, \gamma)$, find the optimal policy (aka the *planning* problem). Compare with the *learning* problem with missing information about rewards/dynamics.

Value Functions

- Value functions measure the goodness of a particular state or state/action pair: how good is for the agent to be in a particular state or execute a particular action at a particular state, for a given policy.

- Optimal value functions measure the best possible goodness of states or state/action pairs under all possible policies.

	state values	**action values**
prediction	v_π	q_π
control	v_*	q_*

Why Value Functions are useful

Value functions capture the knowledge of the agent regarding how good is each state for the goal he is trying to achieve.

"...knowledge is represented as a large number of approximate value functions learned in parallel..."

Horde: A Scalable Real-time Architecture for Learning Knowledge from Unsupervised Sensorimotor Interaction, *Sutton et al.*

Why Value Functions are useful

An optimal policy can be found by maximizing over $q_*(s, a)$:

$$\pi_*(a \mid s) = \begin{cases} 1, & \text{if } \; a = \text{argmax}_{a \in \mathcal{A}} \, q^*(s, a) \\ 0, & \text{otherwise.} \end{cases}$$

An optimal policy can be found from the model dynamics using one step look ahead:

$$\pi_*(a \mid s) = \begin{cases} 1, & \text{if } \; a = \text{argmax}_{a \in \mathcal{A}} \left(\sum_{s', r} \mathrm{p}\left(s', r \mid s, a\right) \left(r + \gamma v_*\left(s'\right)\right) \right) \\ 0, & \text{otherwise} \end{cases}$$

- If we know $q*(s, a)$, we immediately have the optimal policy, we do not need the dynamics!
- If we know $v*(s)$, we need the dynamics to do one step lookahead, to choose the optical action.

Value Functions are Expected Returns

The value of a state, given a policy:

$$v_\pi(s) = \mathbb{E}\left\{G_t \,|\, S_t = s, A_{t:\infty} \sim \pi\right\} \qquad v_\pi : \mathcal{S} \to \mathfrak{R}$$

The value of a state-action pair, given a policy:

$$q_\pi(s,a) = \mathbb{E}\left\{G_t \,|\, S_t = s, A_t = a, A_{t+1:\infty} \sim \pi\right\} \qquad q_\pi : \mathcal{S} \times \mathcal{A} \to \mathfrak{R}$$

The optimal value of a state:

$$v_*(s) = \max_\pi v_\pi(s) \qquad v_* : \mathcal{S} \to \mathfrak{R}$$

The optimal value of a state-action pair:

$$q_*(s,a) = \max_{\bar{x}} q_\pi(s,a) \qquad q_* : \delta \times \mathcal{A} \to \mathfrak{R}$$

Optimal policy: π_* is an optimal policy if and only if

$$\pi_*(a \,|\, s) > 0 \text{ only where } q_*(s,a) = \max_b q_*(s,b) \qquad \forall s \in \mathcal{S}$$

in other words, π_* is optimal iff it is greedy w.r.t. q_*.

Roadmap

- Computing state and state-action value functions by solving linear systems of equations.
- We will then realise matrix inversion is too costly-> iterative estimation-> Bellman backup operation.
- We will then realise we cannot possibly visit every state (too many states) -> selective backups on state-actions that the agent visits as opposed to all.
- We will give up on our assumption of knowing dynamics (monte carlo learning, td learing).
- We will eventually give up on tabular representations and use functions to represent state value functions $V(s, \theta)$, $q(s, a, \phi)$ as opposed to exhaustive enumeration of $v(s)$, $q(s, a)$.

Roadmap

- Computing state and state-action value functions by solving linear systems of equations.
- We will then realise matrix inversion is too costly-> iterative estimation-> Bellman backup operation.
- We will then realise we cannot possibly visit every state (too many states) -> selective backups on state-actions that the agent visits as opposed to all.
- We will give up on our assumption of knowing dynamics (monte carlo learning, td learing).
- We will eventually give up on tabular representations and use functions to represent state value functions $V(s, \theta), q(s, a, \phi)$ as opposed to exhaustive enumeration of $v(s), q(s, a)$.

Recursive relationships for returns

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t=3} + \gamma^3 R_{t+4} + \ldots.$$

Recursive relationships for returns

$$
\begin{aligned}
G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t=3} + \gamma^3 R_{t+4} + \ldots . \\
&= R_{t+1} + \gamma \left(R_{t+2} + \gamma R_{t=3} + \gamma^2 R_{t+4} + \ldots . \right)
\end{aligned}
$$

Recursive relationships for returns

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t=3} + \gamma^3 R_{t+4} + \ldots . \\ &= R_{t+1} + \gamma \left(R_{t+2} + \gamma R_{t=3} + \gamma^2 R_{t+4} + \ldots . \right) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned}$$

Recursive relationships for returns

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t=3} + \gamma^3 R_{t+4} + \ldots . \\ &= R_{t+1} + \gamma \left(R_{t+2} + \gamma R_{t=3} + \gamma^2 R_{t+4} + \ldots . \right) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned}$$

By conditioning on a state and taking expectations:

$$\mathbb{E}[G_t \mid S_t = s] = \mathbb{E}[R_{t+1} + \gamma G_{t+1} \mid S_t = s]$$

$$\mathrm{v}_\pi(s) = \mathbb{E}[R_{t+1} + \gamma \mathrm{v}_\pi(S_{t+1})]$$

$$\mathrm{v}_\pi(s) = \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_\pi(s')]$$

Recursive relationships for returns

$$
\begin{aligned}
G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t=3} + \gamma^3 R_{t+4} + \ldots . \\
&= R_{t+1} + \gamma \left(R_{t+2} + \gamma R_{t=3} + \gamma^2 R_{t+4} + \ldots . \right) \\
&= R_{t+1} + \gamma G_{t+1}
\end{aligned}
$$

By conditioning on a state and action and taking expectations:

$$\mathbb{E}[G_t \,|\, S_t = s, A_t = a] = \mathbb{E}[R_{t+1} + \gamma G_{t+1} \,|\, S_t = s, A_t = a]$$

$$q_\pi(s, a) = \mathbb{E}[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \,|\, S_t = s, A_t = a]$$

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

Bellman Expectation Equations

$$\mathrm{v}_\pi(s) = \mathbb{E}\left[R_{t+1} + \gamma \mathrm{v}_\pi\left(S_{t+1}\right)\right]$$

$$\mathrm{v}_\pi(s) = \sum_a \pi(a \mid s) \sum_{s', r} p\left(s', r \mid s, a\right)\left[r + \gamma v_\pi(s')\right]$$

Bellman Expectation Equations

$$\mathrm{v}_\pi(s) = \mathbb{E}\left[R_{t+1} + \gamma \mathrm{v}_\pi\left(S_{t+1}\right)\right]$$

$$\mathrm{v}_\pi(s) = \sum_a \pi(a \mid s) \sum_{s',r} p\left(s', r \mid s, a\right)\left[r + \gamma v_\pi(s')\right]$$

Q: how do we compute the state values?

This is a set of linear equations, one for each state.

The state value function for π is its unique solution.

Bellman Expectation Equations

$$\mathrm{q}_\pi(s,a) = \mathbb{E}\left[R_{t+1} + \gamma \mathrm{q}_\pi\left(S_{t+1}, A_{t+1}\right) \mid S_t = s, A_t = a\right]$$

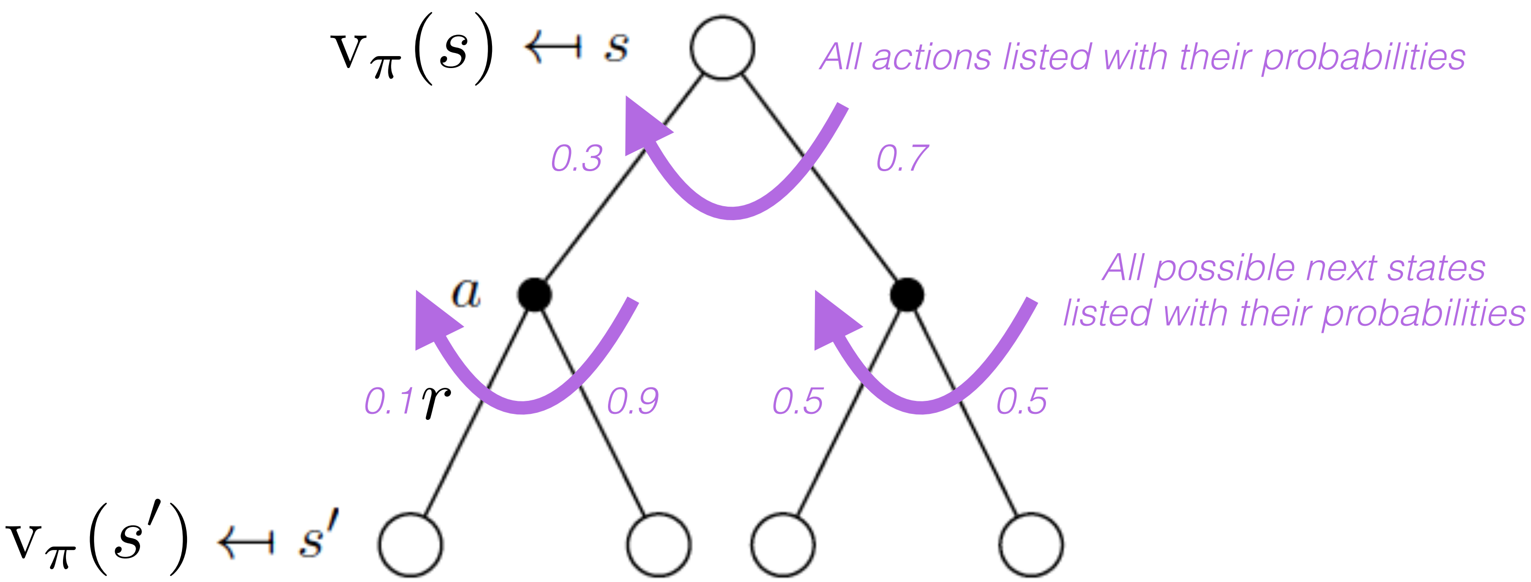
$$\mathrm{q}_\pi(s,a) = \sum_{s',r} p\left(s', r \mid s, a\right)\left[r + \gamma \sum_{a'} \pi(a \mid s) q_\pi(s', a')\right]$$

This is a set of linear equations, one for each state and action.

The state-action value function for π is its unique solution.

Back-up diagram for value functions

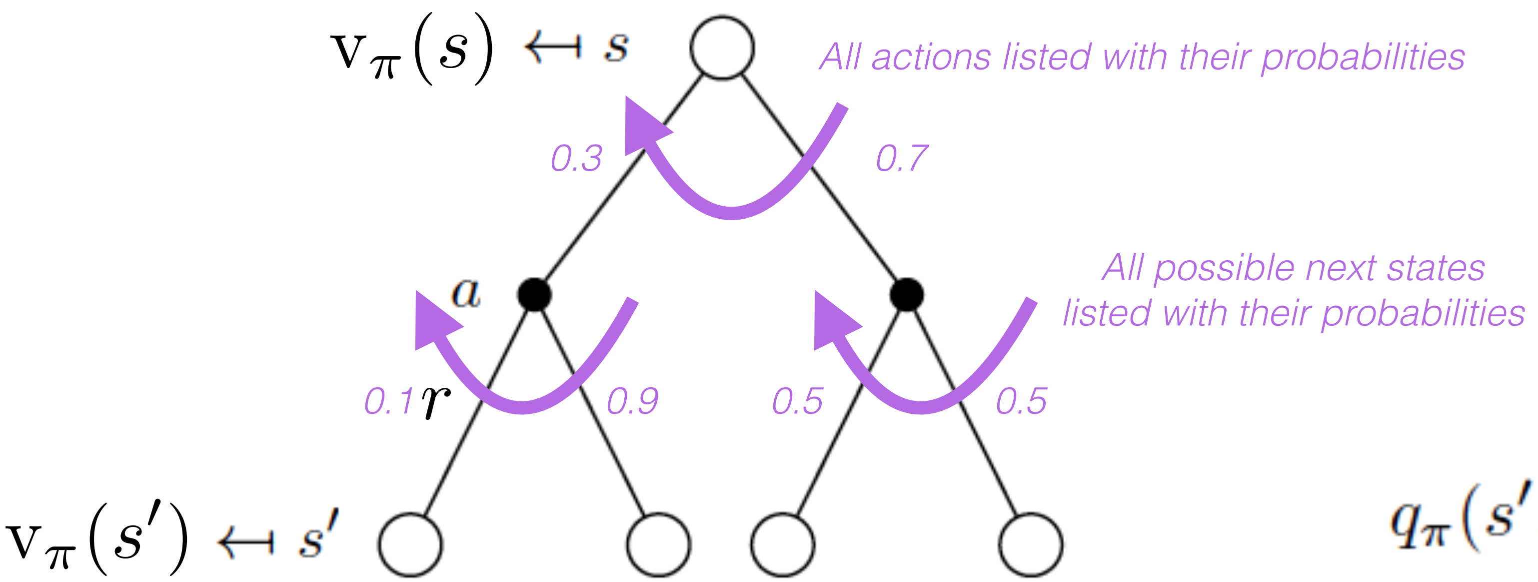
The probabilities of landing on each of the leaves sum to 1

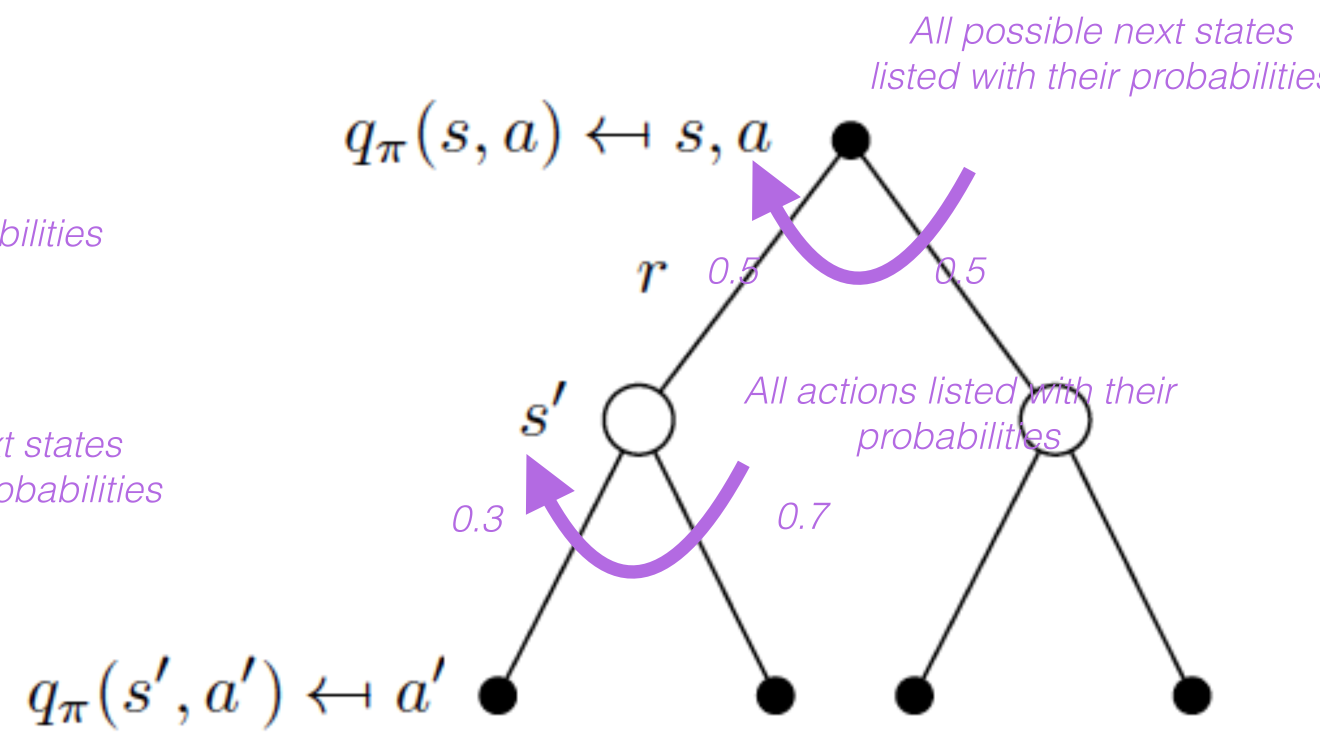


$$v_\pi(s) = \sum_a \pi(a \mid s) \sum_{s',r} p\left(s', r \mid s, a\right) \left[r + \gamma v_\pi(s')\right]$$

Back-up diagram for value functions

The probabilities of landing on each of the leaves sum to 1

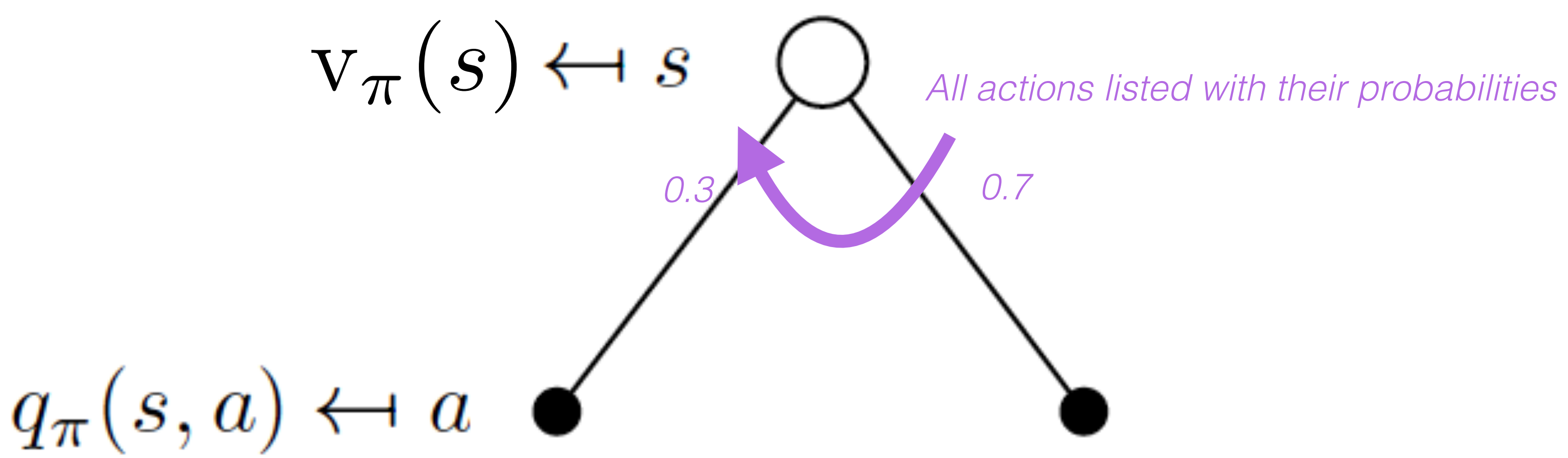




$$v_\pi(s) = \sum_a \pi(a \mid s) \sum_{s',r} p\left(s', r \mid s, a\right) \left[r + \gamma v_\pi(s')\right]$$

$$q_\pi(s,a) = \sum_{r,s'} p(s', r \mid s, a)\left(r + \gamma \sum_{a'} \pi(a' \mid s') q_\pi(s', a')\right)$$

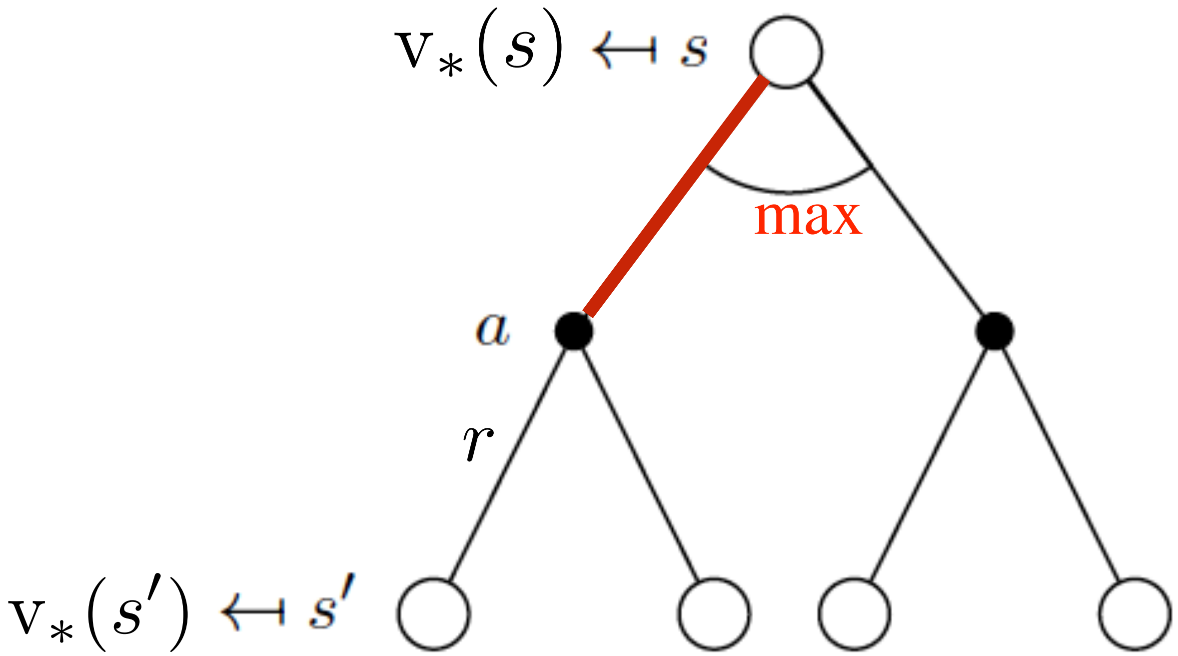
Relating state and state/action value functions



$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a \mid s) q_\pi(s,a)$$

Bellman Optimality Equations for

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

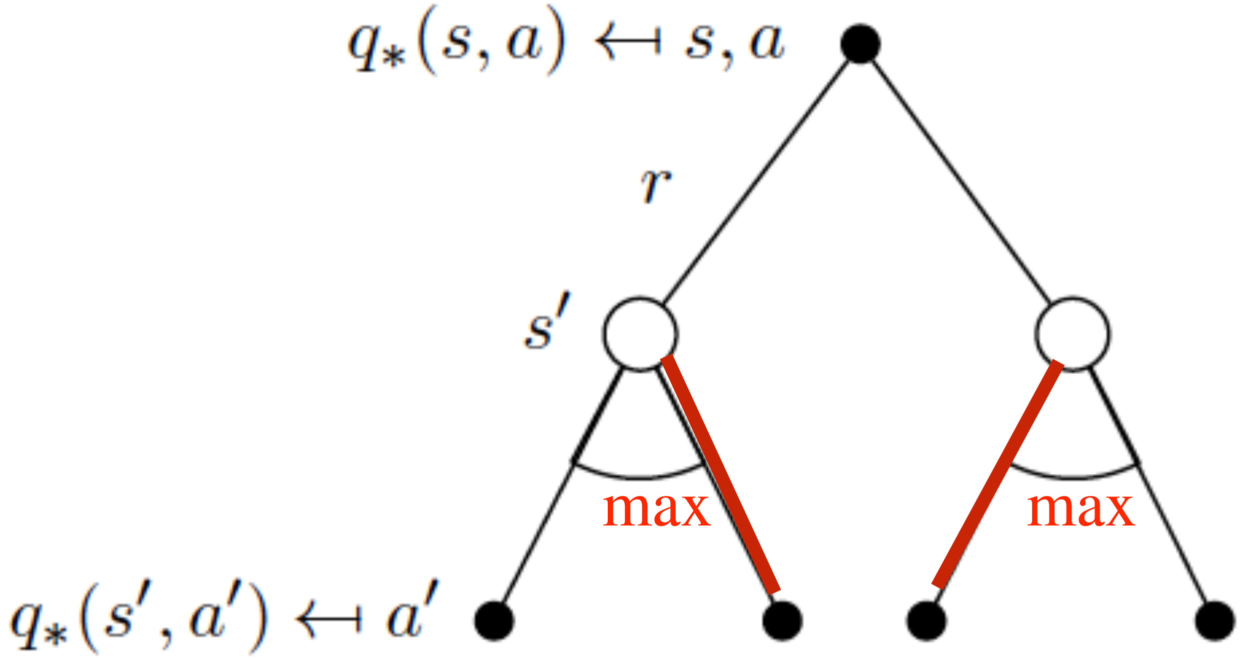


For the Bellman expectation equations we sum over all the leaves, here we choose only the best action branch!

$$v_*(s) = \max_{a \in \mathcal{A}} \left(\sum_{s',r} p(s', r \mid s, a)(r + \gamma v_*(s')) \right)$$

v* is the unique solution of this system of nonlinear equations

Bellman Optimality Equations for

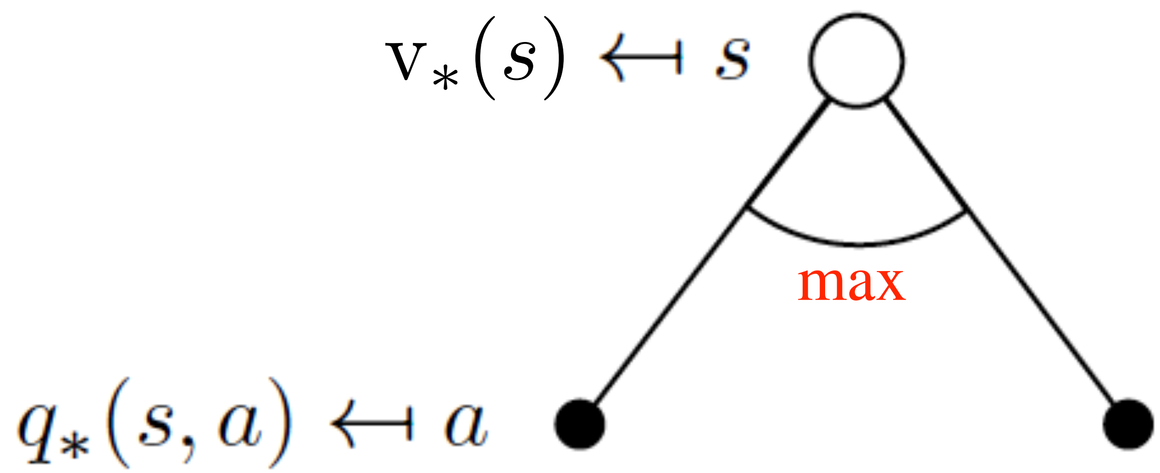


$$q_*(s,a) = \mathbb{E}[R_{t+1} + \gamma \max_{a' \in \mathcal{A}} q_*(S_{t+1}, a') \,|\, S_t = s, A_t = a]$$

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

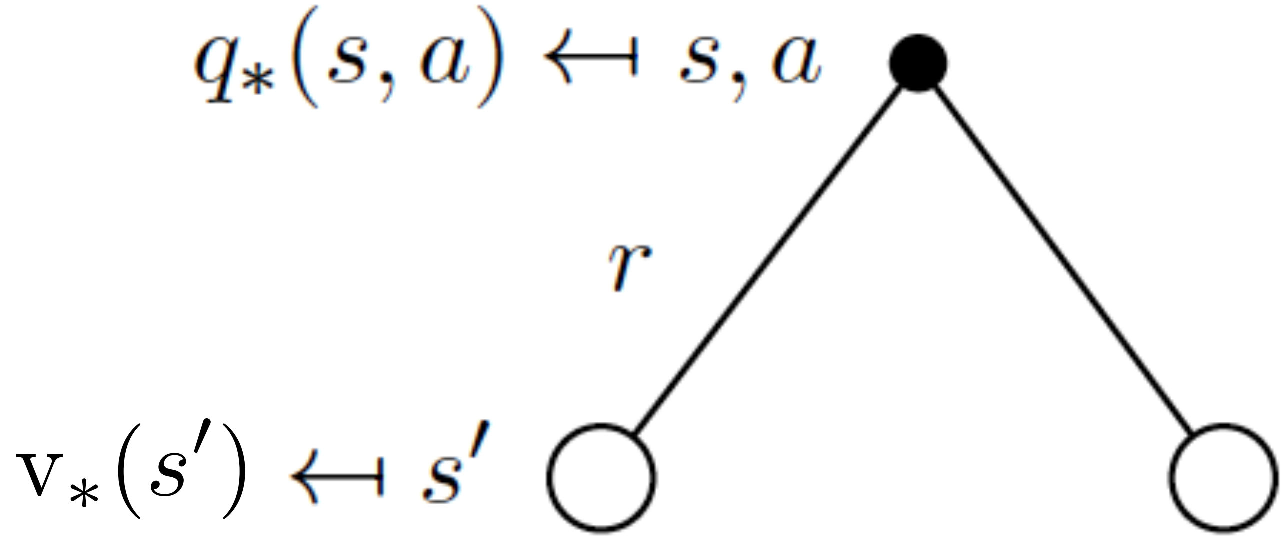
q* is the unique solution of this system of nonlinear equations

Relating Optimal State and Action Value Functions



$$v_*(s') = \max_a q_*(s,a)$$

Relating Optimal State and Action Value Functions

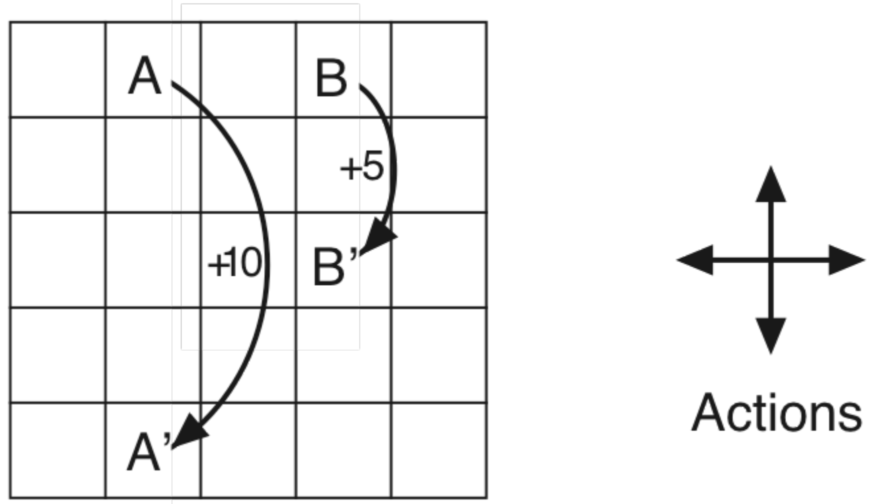


$$q_*(s,a) = \sum_{s',r} p(s', r \mid s, a)(r + \gamma v_*(s'))$$

Gridworld-value function

- Actions: `north`, `south`, `east`, `west`; deterministic.
- If would take agent off the grid: no move but reward = −1
- Other actions produce reward = 0, except actions that move agent out of special states A and B as shown.

$$8.83 = 10 + 0.9 * (-1.3)$$



(a)

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

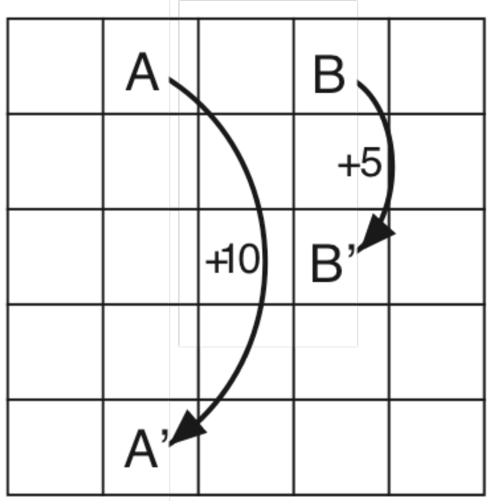
(b)

State-value function for equiprobable random policy; $\gamma = 0.9$

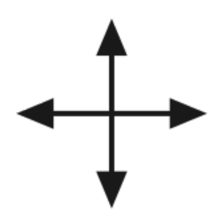
Gridworld-value function

- Actions: `north`, `south`, `east`, `west`; deterministic.
- If would take agent off the grid: no move but reward = −1
- Other actions produce reward = 0, except actions that move agent out of special states A and B as shown.

$$
\begin{aligned}
4.43 = 0.25 * (&0+0.9 * 5.3+\\
&0+0.9 * 2.3+\\
&0+0.9 * 8.8+\\
&-1+0.9 * 4.4)
\end{aligned}
$$



(a)



Actions

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

(b)

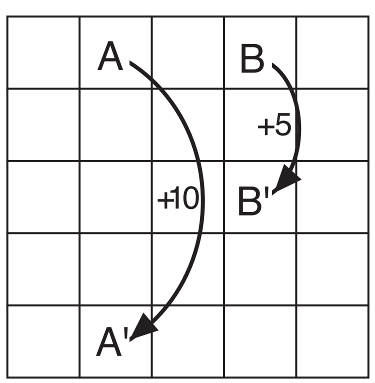
State-value function for equiprobable random policy; $\gamma = 0.9$

Gridworld - optimal value function

Any policy that is greedy with respect to v_* is an optimal policy.

Therefore, given v_*, one-step-ahead search produces the long-term optimal actions.

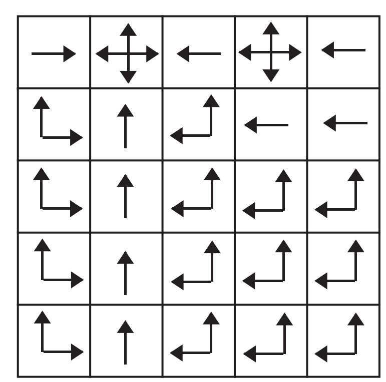
$$24.4 = 10 + 0.9 * (16.0)$$



a) gridworld

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

b) v_*



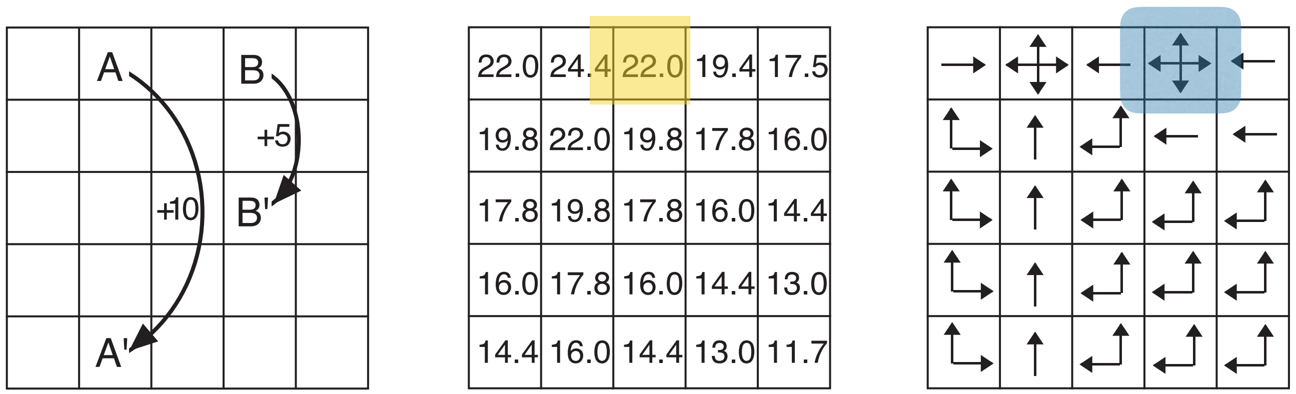
c) π_*

Gridworld - optimal value function

Any policy that is greedy with respect to v_* is an optimal policy.

Therefore, given v_*, one-step-ahead search produces the long-term optimal actions.

$$
\begin{aligned}
22.0 = \max(&0+0.9 * 19.4,\\
&0+0.9 * 19.8,\\
&\boxed{0+0.9 * 24.4},\\
&-1+0.9 * 22.0)
\end{aligned}
$$



a) gridworld b) v_* c) π_*

Solving the Bellman Equations

MDPs to MRPs

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

MDP under a fixed policy becomes **Markov Reward Process** (MRP):

$$\begin{aligned} v_\pi(s) &= \sum_{a \in \mathcal{A}} \pi(a \mid s) \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} T\left(s' \mid s, a\right) v_\pi(s') \right) \\ &= \sum_{a \in \mathcal{A}} \pi(a \mid s) r(s,a) + \gamma \sum_{a \in \mathcal{A}} \pi(a \mid s) \sum_{s' \in \mathcal{S}} T\left(s' \mid s, a\right) v_\pi(s') \\ &= r_s^\pi + \gamma \sum_{s' \in \mathcal{S}} T_{s's}^\pi v_\pi(s') \end{aligned}$$

Expected reward at state s: $r_s^\pi = \sum_{a \in \mathcal{A}} \pi(a \mid s) r(s,a)$

State transition dynamics: $T_{s's}^\pi = \sum_{a \in \mathcal{A}} \pi(a \mid s) T\left(s' \mid s, a\right)$

Matrix Form

The Bellman expectation equation can be written concisely as a system of linear equations

$$v_\pi = r^\pi + \gamma T^\pi v_\pi$$

with direct solution

$$v_\pi = \left(I - \gamma T^\pi\right)^{-1} r^\pi$$

of complexity $\mathcal{O}(|S|^3)$

here T^π is an |S|x|S| matrix, whose (j,k) entry gives $P(s_k \mid s_j, a=\pi(s_j))$
r^π is an |S|-dim vector whose j^{th} entry gives $E[r \mid s_j, a=\pi(s_j)]$
v_π is an |S|-dim vector whose j^{th} entry gives $V_\pi(s_j)$

where |S| is the number of distinct states

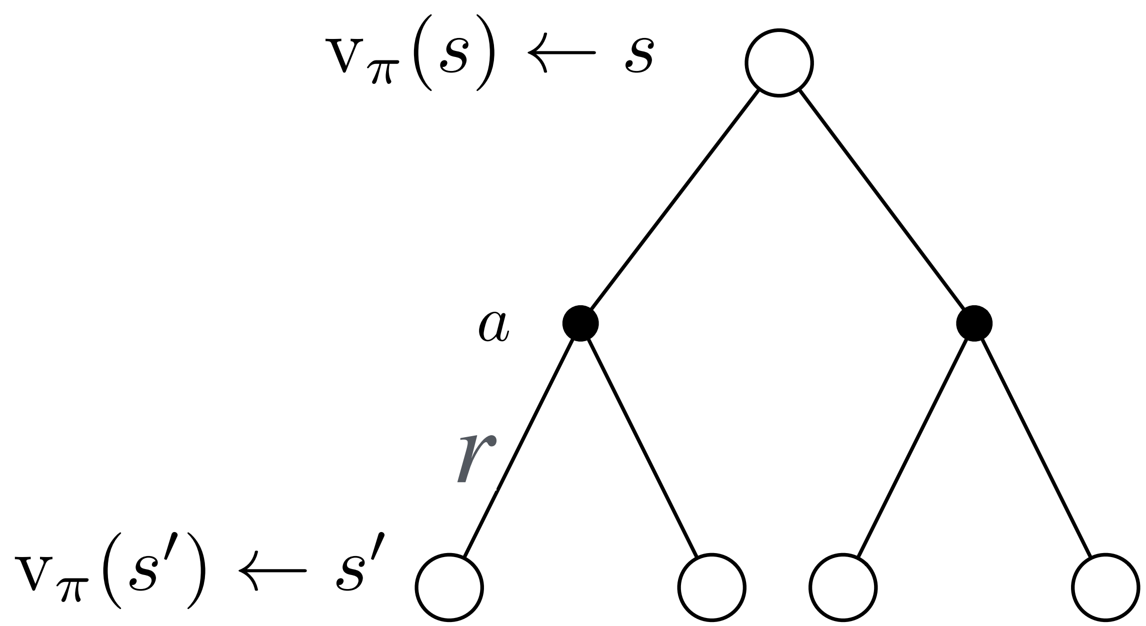
Roadmap

- Computing state and state-action value functions by solving linear systems of equations.
- We will then realise matrix inversion is too costly-> iterative estimation-> Bellman backup operation.
- We will then realise we cannot possibly visit every state (too many states) -> selective backups on state-actions that the agent visits as opposed to all.
- We will give up on our assumption of knowing dynamics (monte carlo learning, td learing).
- We will eventually give up on tabular representations and use functions to represent state value functions $V(s, \theta)$, $q(s, a, \phi)$ as opposed to exhaustive enumeration of $v(s)$, $q(s, a)$.

Iterative Methods: Recall the Bellman Equation

$$v_\pi(s) = \sum_a \pi(a \mid s) \sum_{r,s'} p(s', r \mid s, a)\big(r + \gamma v_\pi(s')\big)$$

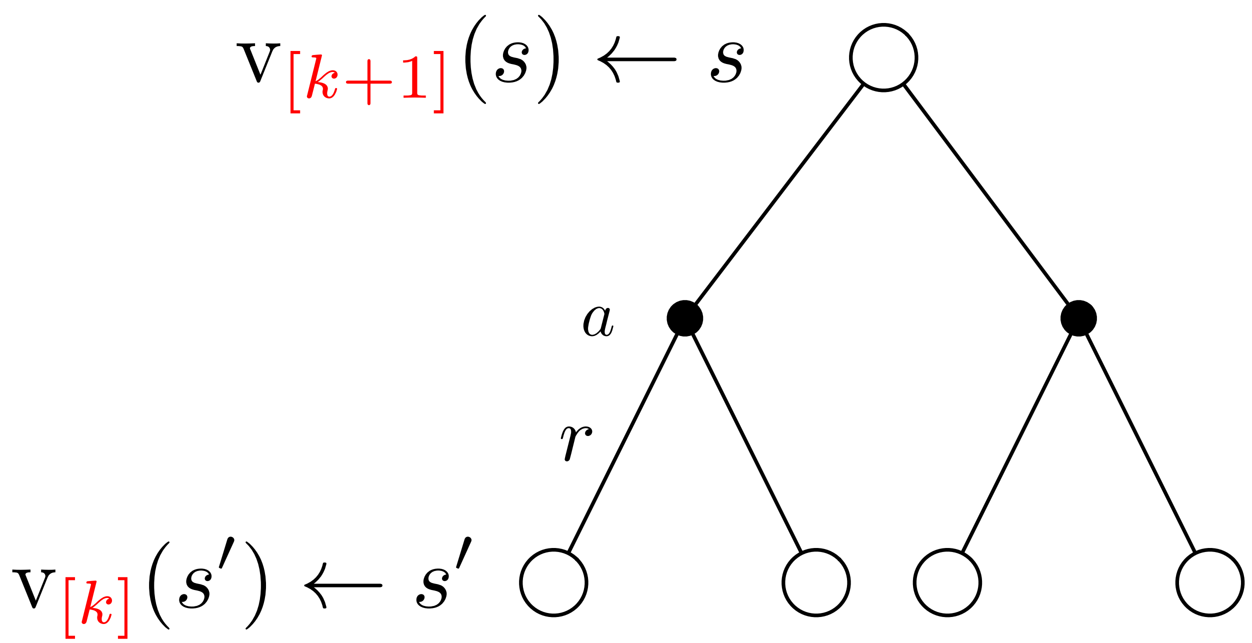
$$v_\pi(s) = \sum_a \pi(a \mid s)\left(r(s,a) + \gamma \sum_{r,s'} p(s' \mid s, a) v_\pi(s') \right)$$



Iterative Methods: Backup Operation

Given an expected value function at iteration *k*, we back up the expected value function at iteration *k+1*:

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



Iterative Methods: Sweep

A **sweep** consists of applying the backup operation $v \rightarrow v'$ for all the states in $\mathcal{S}$

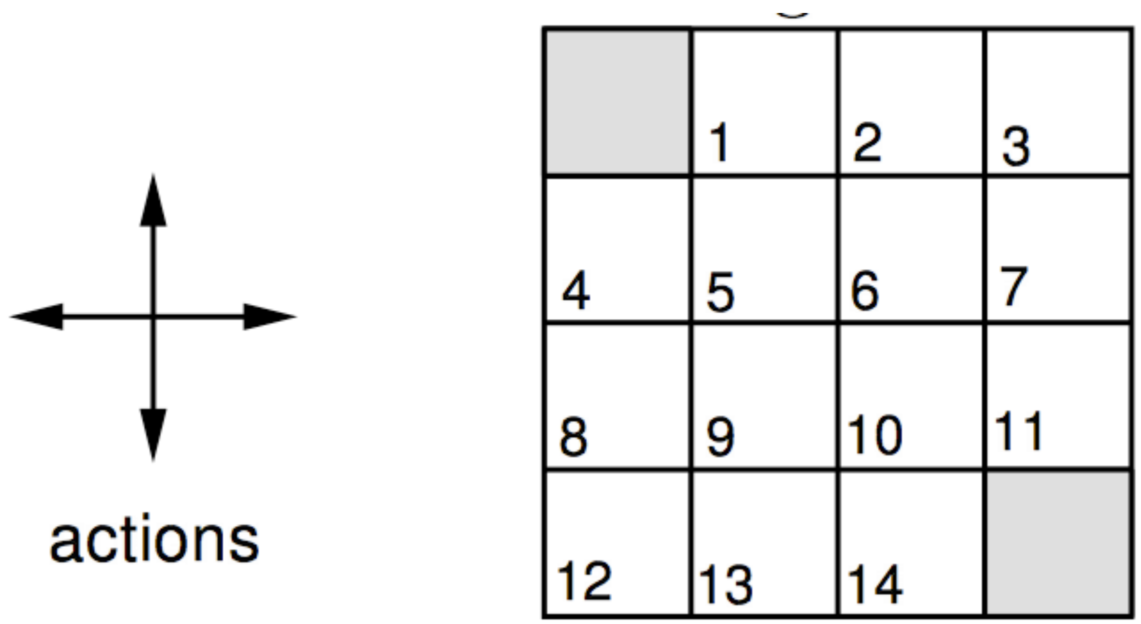
Applying the back up operator iteratively

$$v_{[0]} \rightarrow v_{[1]} \rightarrow v_{[2]} \rightarrow \ldots v_{\pi}$$

A full policy evaluation backup:

$$v_{[k+1]}(s) = \sum_{a} \pi(a \mid s) \left(r(s, a) + \gamma \sum_{r,s'} p(s' \mid s, a) v_{[k]}(s') \right), \forall s$$

A Small-Grid World



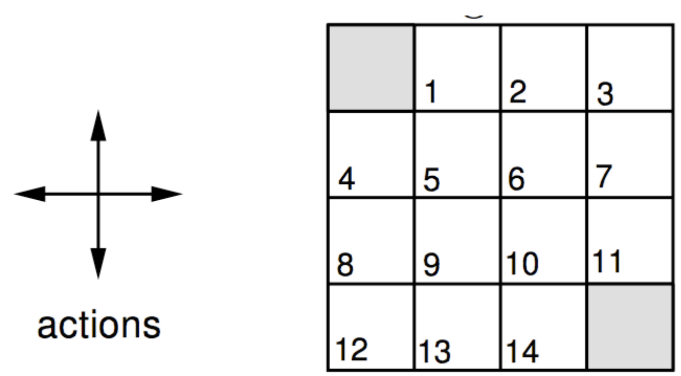
$R = -1$
on all transitions

$\gamma = 1$

- An undiscounted episodic task
- Nonterminal states: 1, 2, … , 14
- Terminal state: one, shown in shaded square
- Actions that would take the agent off the grid leave the state unchanged
- Reward is -1 until the terminal state is reached

Iterative Policy Evaluation

Policy π, an equiprobable random action



- An undiscounted episodic task
- Nonterminal states: 1, 2, … , 14
- Terminal state: one, shown in shaded square
- Actions that would take the agent off the grid leave the state unchanged
- Reward is -1 until the terminal state is reached

$\mathrm{V}[k]$ for the random policy

$k = 0$

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

$k = 1$

$k = 2$

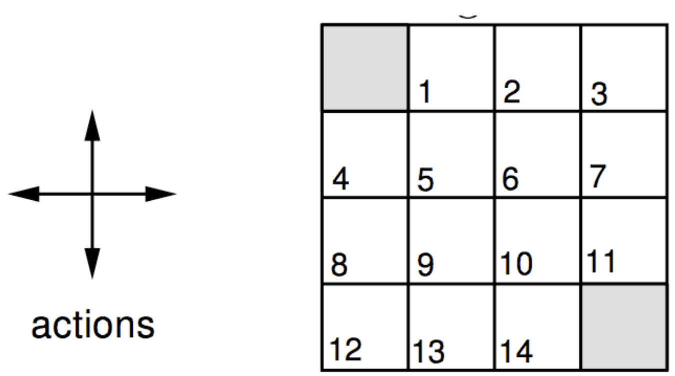
$k = 3$

$k = 10$

$k = \infty$

Iterative Policy Evaluation

Policy π, an equiprobable random action



- An undiscounted episodic task
- Nonterminal states: 1, 2, … , 14
- Terminal state: one, shown in shaded square
- Actions that would take the agent off the grid leave the state unchanged
- Reward is -1 until the terminal state is reached

$\mathrm{V}[k]$ for the random policy

$k = 0$

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

$k = 2$

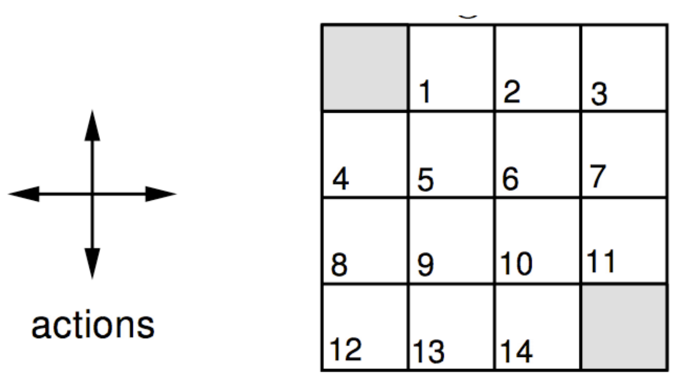
$k = 3$

$k = 10$

$k = \infty$

Iterative Policy Evaluation

Policy π, an equiprobable random action



- An undiscounted episodic task
- Nonterminal states: 1, 2, … , 14
- Terminal state: one, shown in shaded square
- Actions that would take the agent off the grid leave the state unchanged
- Reward is -1 until the terminal state is reached

$\mathrm{V}[k]$ for the random policy

$k = 0$

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

$k = 2$

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

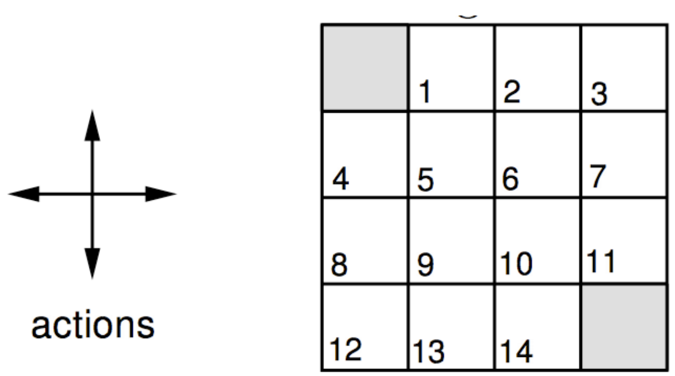
$k = 3$

$k = 10$

$k = \infty$

Iterative Policy Evaluation

Policy π, an equiprobable random action



- An undiscounted episodic task
- Nonterminal states: 1, 2, … , 14
- Terminal state: one, shown in shaded square
- Actions that would take the agent off the grid leave the state unchanged
- Reward is -1 until the terminal state is reached

$\mathrm{V}[k]$ for the random policy

$k = 0$

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

$k = 2$

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

$k = 3$

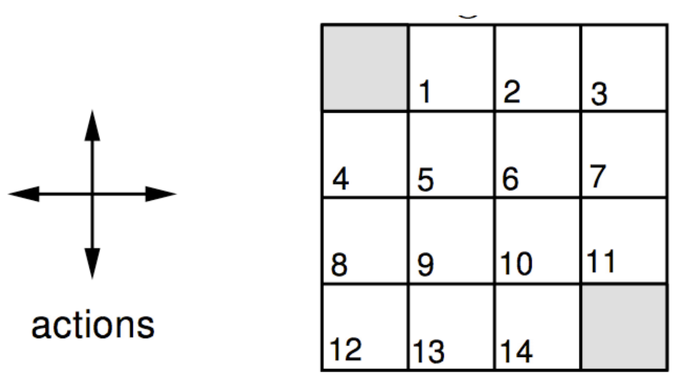
0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

$k = 10$

$k = \infty$

Iterative Policy Evaluation

Policy π, an equiprobable random action



- An undiscounted episodic task
- Nonterminal states: 1, 2, … , 14
- Terminal state: one, shown in shaded square
- Actions that would take the agent off the grid leave the state unchanged
- Reward is -1 until the terminal state is reached

$\mathrm{V}[k]$ for the random policy

$k = 0$

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

$k = 2$

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

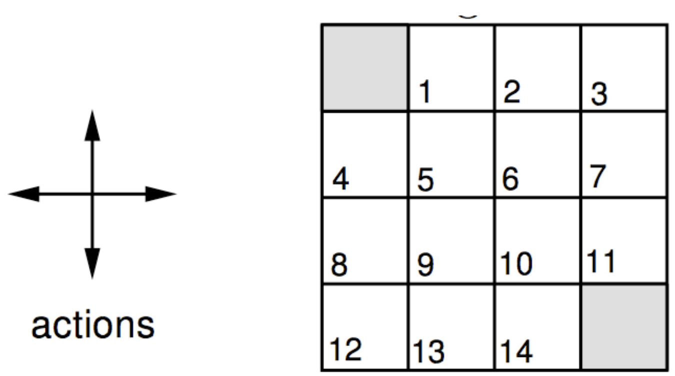
$k = 10$

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

$k = \infty$

Iterative Policy Evaluation

Policy π, an equiprobable random action



- An undiscounted episodic task
- Nonterminal states: 1, 2, … , 14
- Terminal state: one, shown in shaded square
- Actions that would take the agent off the grid leave the state unchanged
- Reward is -1 until the terminal state is reached

$\mathrm{V}[k]$ for the random policy

$k = 0$

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

$k = 2$

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

$k = 10$

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

$k = \infty$

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

Iterative Policy Evaluation

Input π, the policy to be evaluated

Initialize an array $V(s) = 0$, for all $s \in \mathcal{S}^+$

Repeat

$\quad \Delta \leftarrow 0$

$\quad$ For each $s \in \mathcal{S}$:

$\qquad v \leftarrow V(s)$

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

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

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

Output $V \approx v_\pi$

Contraction Mapping Theorem

An operator F on a normed vector space $\mathcal{X}$ is a γ**-contraction**, for $0 < \gamma < 1$ provided for all $x, y \in \mathcal{X}$

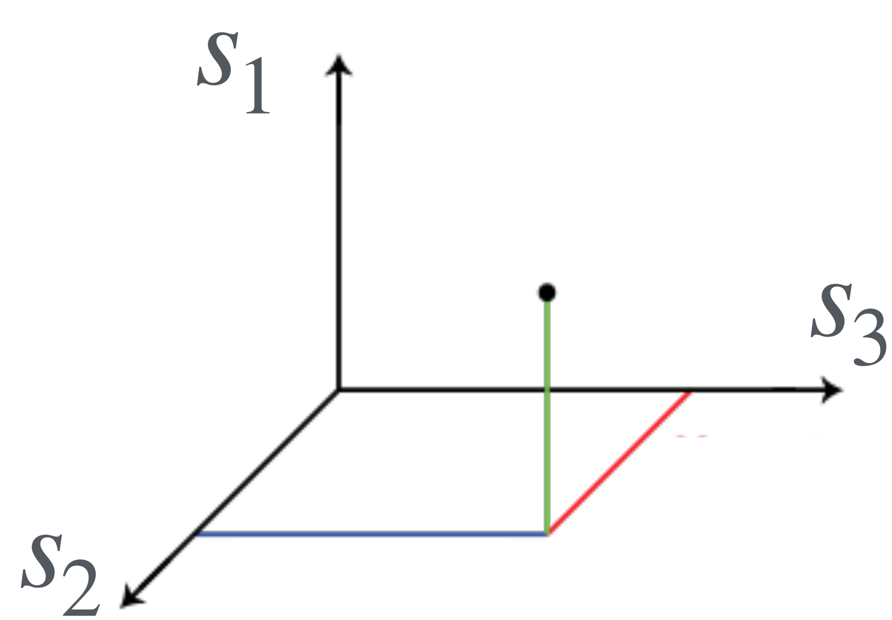
$$\|F(x) - F(y)\| \leq \gamma \|x - y\|$$

Theorem (Contraction mapping)
For a γ-contraction F in a complete normed vector space $\mathcal{X}$

- F converges to a unique fixed point in $\mathcal{X}$
- at a linear convergence rate γ

Value Function Space

- Consider the vector space V over value functions
- There are $|\mathcal{S}|$ dimensions
- Each point in this space fully specifies a value function $v(s)$
- Bellman backup brings value functions closer in this space
- And therefore the backup must converge to a unique solution



Value Function ∞-Norm

- We will measure distance between state-value functions u and v by the ∞-norm
- i.e. the largest difference between state values,

$$\|\mathrm{u} - \mathrm{v}\|_\infty = \max_{s \in \mathcal{S}} |\mathrm{u}(s) - \mathrm{v}(s)|$$

$$\|u\|_\infty = \max_{s \in \mathcal{S}} |u(s)|$$

Bellman Expectation Backup is a Contraction

- Define the Bellman expectation backup operator

$$F^{\pi}(\mathrm{v}) = r^{\pi} + \gamma T^{\pi}\mathrm{v}$$

- This operator is a γ-contraction, i.e. it makes value functions closer by at least γ,

$$\begin{aligned}
\|F^{\pi}(u) - F^{\pi}(v)\|_{\infty} &= \|(r^{\pi} + \gamma T^{\pi}u) - (r^{\pi} + \gamma T^{\pi}v)\|_{\infty} \\
&= \|\gamma T^{\pi}(u - v)\|_{\infty} \\
&\leq \|\gamma T^{\pi}(\mathbf{1}\|(u - v)\|_{\infty})\|_{\infty} \\
&= \|\gamma(T^{\pi}\mathbf{1})\|u - v\|_{\infty}\|_{\infty} \\
&= \|\gamma\mathbf{1}\|u - v\|_{\infty}\|_{\infty} \\
&= \gamma\|u - v\|_{\infty}
\end{aligned}$$

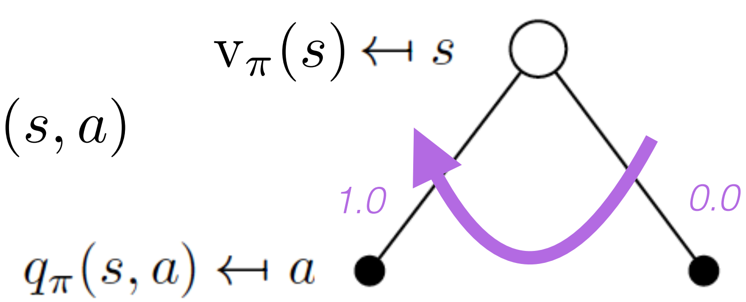
Finding Optimal Policies

Policy Improvement

- Suppose we have computed v_π for a deterministic policy π.
- For a given state s, would it be better to do an action $a \neq \pi(s)$?
- It is better to switch to action a for state s if and only if $q_\pi(s, a) > v_\pi(s)$.
- And we can compute $q_\pi(s, a)$ from v_π by:

$$q_\pi(s, a) = r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s, a) v_\pi(s')$$

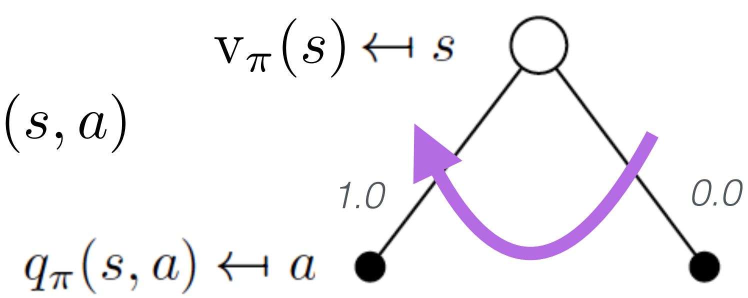
$$\mathrm{v}_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s, a)$$



Policy Improvement (greedification)

- Suppose we have computed v_π for a deterministic policy π.
- For a given state s, would it be better to do an action $a \neq \pi(s)$?
- It is better to switch to action a for state s if and only if $q_\pi(s, a) > v_\pi(s)$.
- And we can compute $q_\pi(s, a)$ from v_π: $q_\pi(s, a) = r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s, a) v_\pi(s')$

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s, a)$$



- Do this for all states to get a new policy π' that is greedy with respect to $q_\pi(s, a)$: $\pi'(s) = \underset{a}{\operatorname{argmax}}\, q_\pi(s, a)$
- After policy update it holds that: $v_\pi(s) \leq q_\pi(s, \pi'(s))\ \forall s$

Policy Improvement Cont.

- Trivial proof:
$$\begin{aligned} q_\pi(s, \pi'(s)) &= q_\pi(s, \operatorname*{argmax}_a q_\pi(s, a)) \\ &= \max_a q_\pi(s, a) \\ &\geq q_\pi(s, \pi(s)) \\ &\geq v_\pi(s) \end{aligned}$$

- We have indeed improved the policy (or ended up on an equally good policy)
$$\begin{aligned} v_\pi(s) &\leq q_\pi(s, \pi'(s)) \\ &= \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = \pi'(s)] \\ &= \mathbb{E}_{\pi'}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] \\ &\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) \mid S_t = s] \\ &= \mathbb{E}_{\pi'}[R_{t+1} + \gamma \mathbb{E}_{\pi'}[R_{t+2} + \gamma v_\pi(S_{t+2}) | S_{t+1}] \mid S_t = s] \\ &= \mathbb{E}_{\pi'}\big[R_{t+1} + \gamma R_{t+2} + \gamma^2 v_\pi(S_{t+2}) \bigm| S_t = s\big] \\ &\leq \mathbb{E}_{\pi'}\big[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 v_\pi(S_{t+3}) \bigm| S_t = s\big] \\ &\vdots \\ &\leq \mathbb{E}_{\pi'}\big[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \bigm| S_t = s\big] \\ &= v_{\pi'}(s). \end{aligned}$$

Policy Improvement Cont.

- If policy is unchanged after the greedification step, this means that:
  $v_\pi(s) = \max_a q_\pi(s, a) \ \forall s$
- What does this mean?
- This is the Bellman optimality equation. So $v_\pi(s) = v_*(s)$ and π is optimal.

Optimal Policy

Define a partial ordering over policies: $\pi \geq \pi', \ if \quad v_\pi(s) \geq v_{\pi'}(s) \forall s.$

Theorem: For any Markov Decision Process

- There exists an optimal policy π^* that is better than or equal to all other policies, $\pi^* \geq \pi, \forall \pi.$
- All optimal policies achieve the optimal value function, $v_{\pi^*}(s) = v_*(s) \forall s$.
- All optimal policies achieve the optimal action-value function, $q_{\pi^*}(s, a) = q_*(s, a).$

Policy Iteration

$$\pi_0 \xrightarrow{\mathrm{E}} \mathrm{v}_{\pi_0} \xrightarrow{\mathrm{I}} \pi_1 \xrightarrow{\mathrm{E}} \mathrm{v}_{\pi_1} \xrightarrow{\mathrm{I}} \pi_2 \xrightarrow{\mathrm{E}} \cdots \xrightarrow{\mathrm{I}} \pi_* \xrightarrow{\mathrm{E}} \mathrm{v}_*$$

policy evaluation

policy improvement

“greedification”

Policy Iteration

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

2. Policy Evaluation (Till convergence)
   Repeat
   $\quad \Delta \leftarrow 0$
   $\quad$ For each $s \in \mathcal{S}$:
   $\quad\quad \mathrm{v} \leftarrow V(s)$
   $\quad\quad V(s) \leftarrow \Sigma_{a \in \mathcal{A}} \pi(a|s)\left(r(s,a) + \gamma \Sigma_{s' \in \mathcal{S}} p(s'|s,a) V(s')\right)$
   $\quad\quad \Delta \leftarrow \max(\Delta, |\mathrm{v} - V(s)|)$
   until $\Delta < \theta$ (a small positive number)

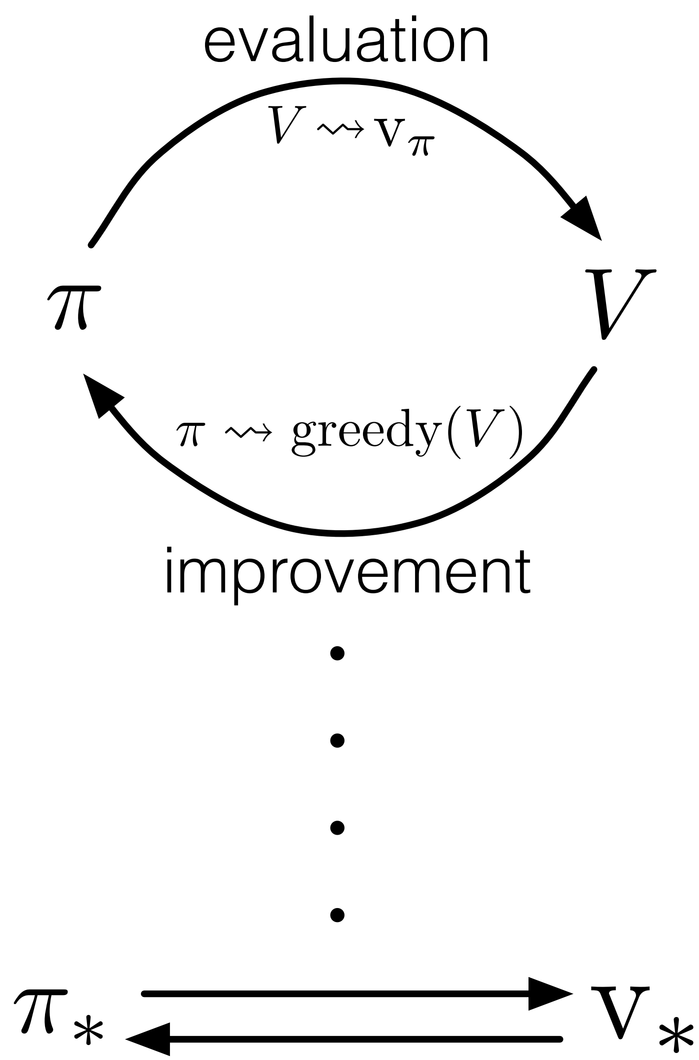
3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   $\quad a \leftarrow \pi(s)$
   $\quad \pi(s) \leftarrow \arg\max r(s,a) + \gamma \Sigma_{s' \in \mathcal{S}} p(s'|s,a)\, \mathrm{v}_{\pi}(s')$
   $\quad$ If $a \neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return V and π; else go to 2

Generalized Policy Iteration

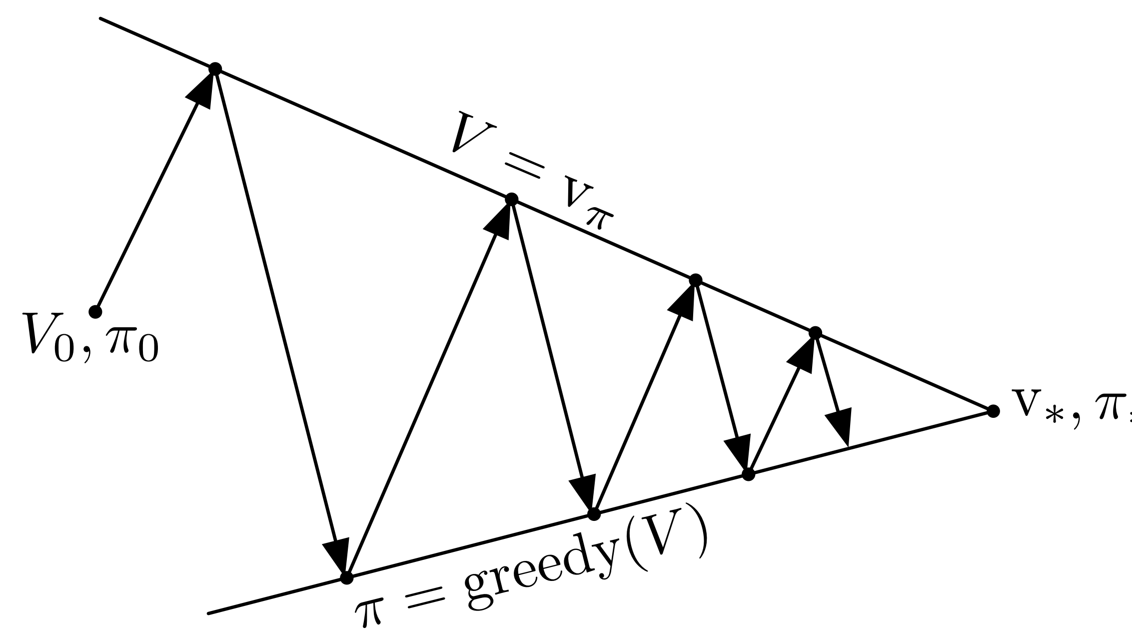
- Does policy evaluation need to converge to v_π?
- Or should we introduce a stopping condition
  - e.g. ϵ-convergence of value function
- Or simply stop after k iterations of iterative policy evaluation?

Generalized Policy Iteration

Generalized Policy Iteration (GPI): any interleaving of policy evaluation and policy improvement, independent of their granularity.



A geometric metaphor for convergence of GPI:



- All RL methods are a form of GPI
- GPI converges. Why?
- Why when it converges it converges to optimum?

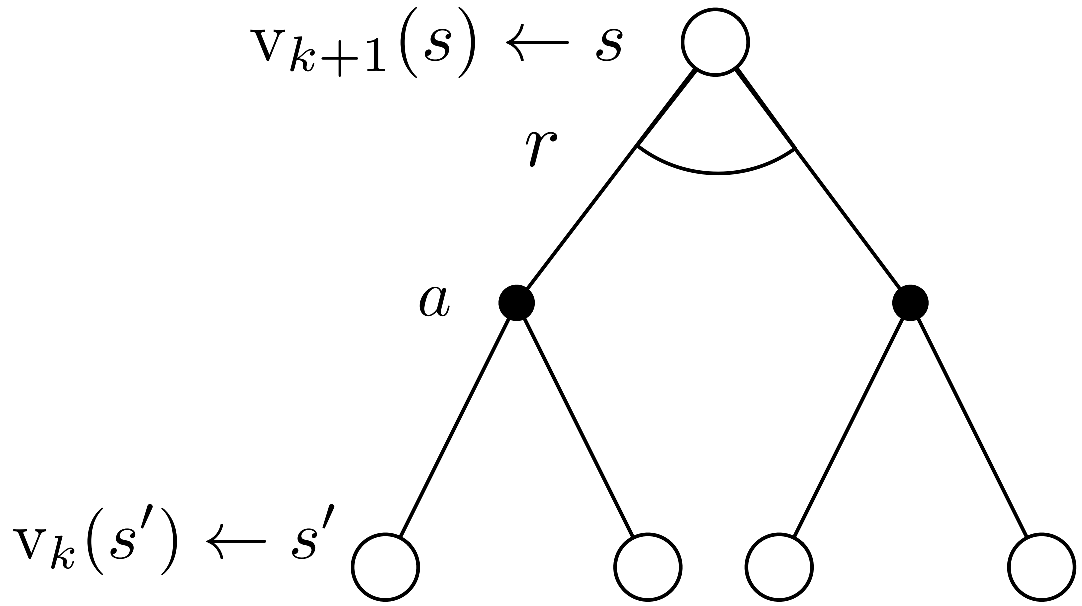
Generalized Policy Iteration

- Does policy evaluation need to converge to v_π?
- Or should we introduce a stopping condition
  - e.g. ϵ-convergence of value function
- Or simply stop after k iterations of iterative policy evaluation?
- Why not update the policy after every iteration? i.e. stop after k = 1
  - This is equivalent to value iteration (next section)

Value Iteration

- Problem: find optimal policy π
- Solution: iterative application of Bellman optimality backup
- $v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_*$
- Using synchronous backups
  - At each iteration k + 1
  - For all states $s \in \mathcal{S}$
  - Update $v_{k+1}(s)$ from $v_k(s')$

Value Iteration (2)



$$v_{[k+1]}(s) = \max_{a \in \mathcal{A}} \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \,|\, s,a) v_{[k]}(s') \right), \forall s$$

$$v_{k+1} = \max_{a \in \mathcal{A}} r(a) + \gamma p(a) v_k$$

Bellman Optimality Backup is a Contraction

- Define the Bellman optimality backup operator $F*$,

$$F^*(v) = \max_{a \in \mathscr{A}} r(a) + \gamma p(a) v$$

- This operator is a γ-contraction, i.e. it makes value functions closer by at least γ (similar to previous proof)

$$\left\| F^*(\mathrm{u}) - F^*(\mathrm{v}) \right\|_\infty \leq \gamma \|\mathrm{u} - \mathrm{v}\|_\infty$$

Convergence of Value Iteration

- The Bellman optimality operator F^* has a unique fixed point
- v_* is a fixed point of F^* (by Bellman optimality equation)
- By contraction mapping theorem
- Value iteration converges on v_*

Synchronous Dynamic Programming Algorithms

Problem	Bellman Equation	Algorithm
Prediction	Bellman Expectation Equation	Iterative Policy Evaluation
Control	Bellman Expectation Equation + Greedy Policy Improvement	Policy Iteration
Control	Bellman Optimality Equation	Value Iteration

- Algorithms are based on state-value function $v_\pi(s)$ or $v_*(s)$
- Complexity $O\left(mn^2\right)$ per iteration, for m actions and n states
- Could also apply to action-value function $q_\pi(s, a)$ or $q_*(s, a)$
- Complexity $O\left(m^2n\right)$ per iteration

Efficiency of DP

- To find an optimal policy is polynomial in the number of states…
- BUT, the number of states is often astronomical, e.g., often growing exponentially with the number of state variables (what Bellman called “the curse of dimensionality”).
- In practice, classical DP can be applied to problems with a few millions of states.

Asynchronous DP

- All the DP methods described so far require exhaustive sweeps of the entire state set.
- Asynchronous DP does not use sweeps. Instead it works like this:
    - Repeat until convergence criterion is met:
        - **Sample** a state at random and apply the appropriate backup
- Still need lots of computation, but does not get locked into hopelessly long sweeps
- Guaranteed to converge if *all* states continue to be selected

Asynchronous Dynamic Programming

- Three simple ideas for asynchronous dynamic programming:
  - In-place dynamic programming
  - Prioritized sweeping
  - Real-time dynamic programming

In-Place Dynamic Programming

- Synchronous value iteration stores two copies of value function
  - for all s in $\mathcal{S}$

$$v_{new}(s) \leftarrow \max_{a \in \mathcal{A}} \left(r(s,a) + \gamma \sum_{s' \in \mathcal{D}} p\left(s' | s, a\right) v_{old}\left(s'\right) \right)$$

$$v_{old} \leftarrow v_{new}$$

- In-place value iteration only stores one copy of value function
  - for all s in $\mathcal{S}$

$$v(s) \leftarrow \max_{a \in \mathcal{A}} \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} T\left(s' | s, a\right) v\left(s'\right) \right)$$

Prioritized Sweeping

- Use magnitude of Bellman error to guide state selection, e.g.

$$\left| \max_{a \in \mathcal{A}} \left(r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p\left(\quad s, a\right) v\left(s'\right) \right) - v(s) \right|$$

- Backup the state with the largest remaining Bellman error
- Update Bellman poor of affected states after each backup
- Requires knowledge of reverse dynamics (predecessor states)
- Can be implemented efficiently by maintaining a priority queue

Real-time Dynamic Programming

- Idea: **only states that are relevant to agent**
- Use agent's experience to guide the selection of states
- After each time-step $\mathcal{S}_t, \mathcal{A}_t, r_{t+1}$
- Backup the state $\mathcal{S}$

$$v\left(\mathcal{S}_t\right) \leftarrow \max_{a \in \mathcal{A}} \left(r\left(\mathcal{S}_t, a\right) + \gamma \sum_{s' \in \mathcal{S}} p\left(s' \mid \mathcal{S}_t, a\right) v\left(s'\right) \right)$$