10703 Deep Reinforcement Learning and Control

Russ Salakhutdinov

Slides borrowed from
Katerina Fragkiadaki

Solving known MDPs: Dynamic Programming

Markov Decision Process (MDP)

A **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

- T is a state transition probability function

$$T(s'|s, a) = \mathbb{P}[S_{t+1} = s'|S_t = s, A_t = a]$$

- r is a reward function

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

- γ is a discount factor $\gamma \in [0, 1]$

Solving MDPs

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

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

  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.

- We still consider finite MDPs (finite $\mathcal{S}$ and $\mathcal{A}$) with known dynamics.

Outline

- Policy evaluation

- Policy iteration

- Value iteration

- Asynchronous DP

Policy Evaluation

Policy evaluation: for a given policy π, compute the state value function

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

where $\mathrm{v}_\pi(s)$ is implicitly given by the **Bellman equation**

$$\mathrm{v}_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s'|s,a) \mathrm{v}_\pi(s') \right)$$

a system of $|\mathcal{S}|$ simultaneous equations.

MDPs to MRPs

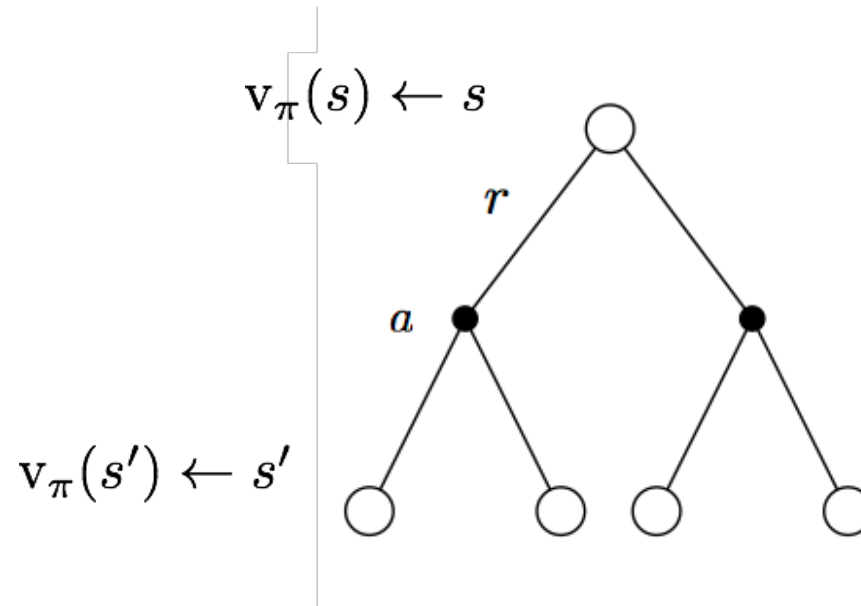MDP under a fixed policy becomes **Markov Reward Process** (MRP)

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s'|s,a) v_\pi(s') \right)$$

$$= \sum_{a \in \mathcal{A}} \pi(a|s) r(s,a) + \gamma \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}} T(s'|s,a) v_\pi(s')$$

$$= r_s^\pi + \gamma \sum_{s' \in \mathcal{S}} T_{s's}^\pi v_\pi(s')$$

where
$$r_s^\pi = \sum_{a \in \mathcal{A}} \pi(a|s) r(s,a)$$

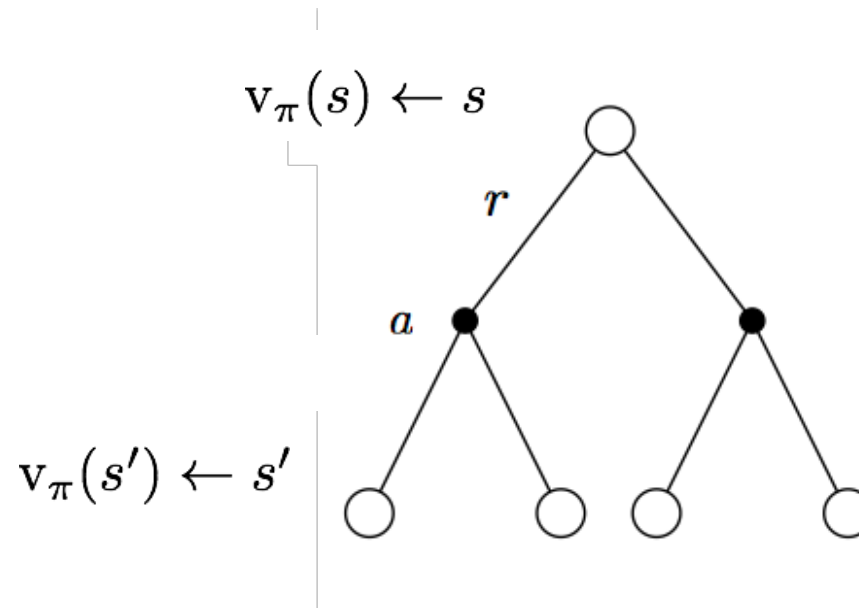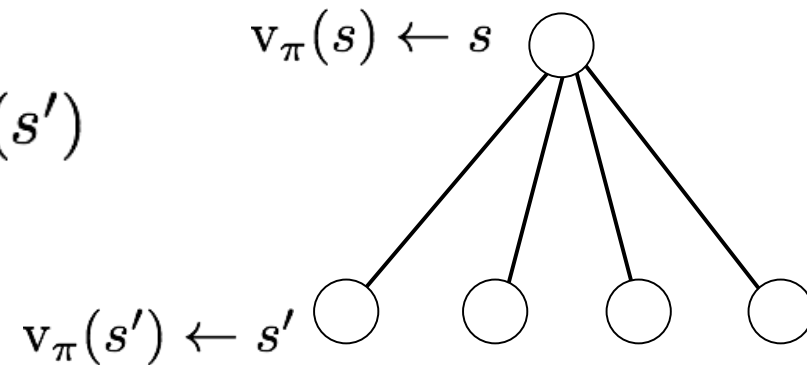$$T_{s's}^\pi = \sum_{a \in \mathcal{A}} \pi(a|s) T(s'|s,a)$$

Back Up Diagram

MDP

$$v_\pi(s) \leftarrow s$$

$$r$$

$$a$$

$$v_\pi(s') \leftarrow s'$$

Back Up Diagram

MDP

$$v_\pi(s) \leftarrow s$$

$$v_\pi(s') \leftarrow s'$$

$$v_\pi(s) = r_s^\pi + \gamma \sum_{s' \in \mathcal{S}} T_{s's}^\pi v_\pi(s')$$

$$v_\pi(s) \leftarrow s$$

$$v_\pi(s') \leftarrow s'$$

Matrix Form

The Bellman expectation equation can be written concisely using the induced form:
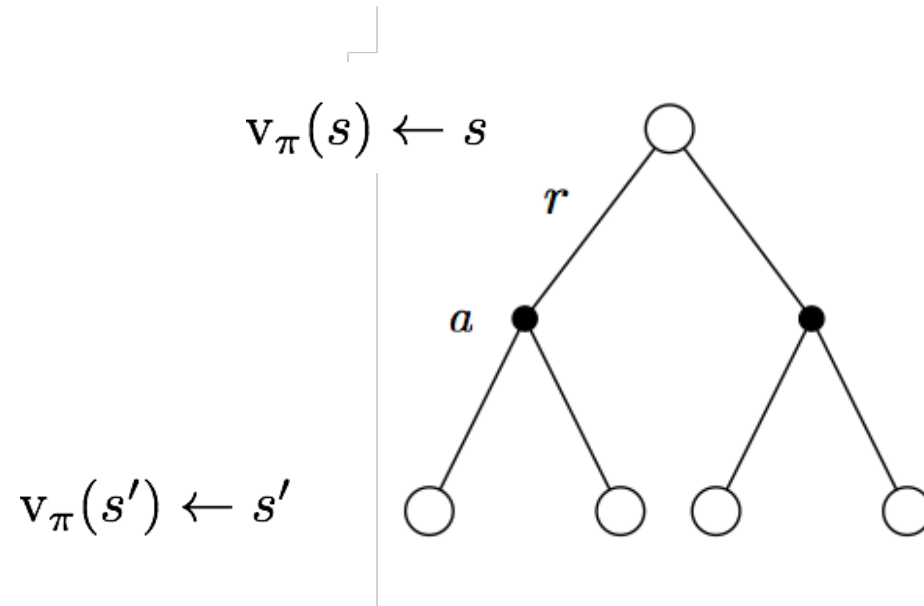
$$\mathbf{v}_\pi = r^\pi + \gamma T^\pi \mathbf{v}_\pi$$

with direct solution

$$\mathbf{v}_\pi = (I - \gamma T^\pi)^{-1} r^\pi$$

of complexity $O(N^3)$

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s'|s,a) v_\pi(s') \right)$$

$v_\pi(s) \leftarrow s$

r

a

$v_\pi(s') \leftarrow s'$

Iterative Methods: Backup Operation

Given an expected value function at iteration *k*, we back up the expected value function at iteration *k+1:*
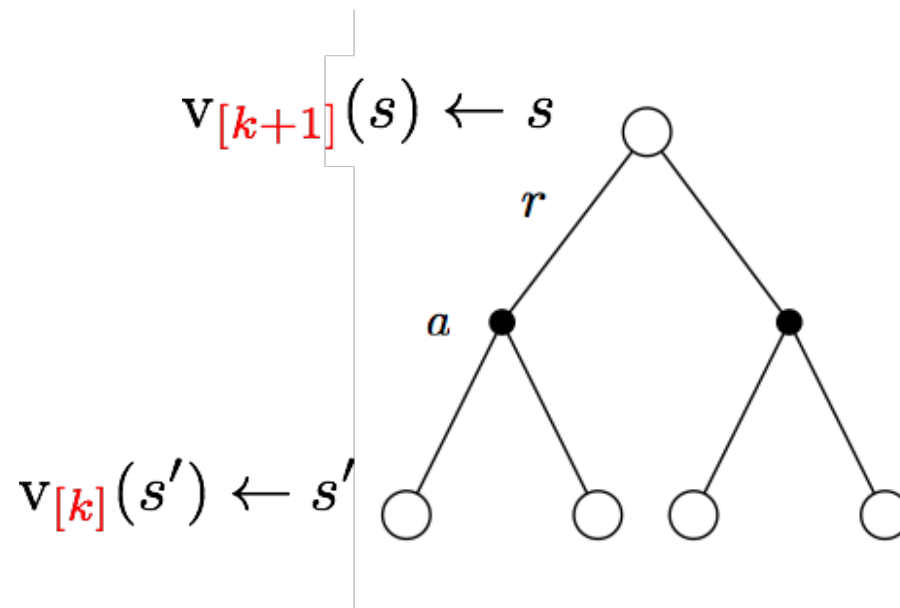
$$
v_{[k+1]}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s'|s,a) v_{[k]}(s') \right)
$$

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 Small-Grid World
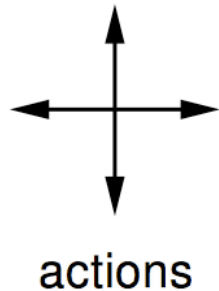
$R = -1$
on all transitions

$\gamma = 1$

- 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

$V[k]$ for the
random policy

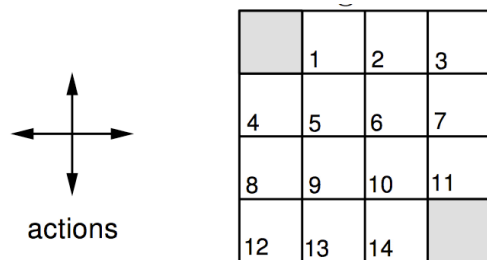| 0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

$k = 0$

Policy π, an equiprobable random action

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

$k = 1$

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

actions

$k = 2$

$k = 3$

- An undiscounted episodic task

- Nonterminal states: 1, 2, … , 14

- Terminal state: one, shown in shaded square

$k = 10$

- Actions that would take the agent off the grid leave the state unchanged
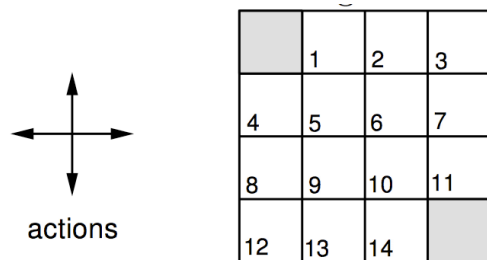
- Reward is -1 until the terminal state is reached

$k = \infty$

Iterative Policy Evaluation

Policy π, an equiprobable random action

actions

- 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

$V[k]$ 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

$V[k]$ for the
random policy

Policy π, an equiprobable random action

actions

- 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

$k = 0$

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

Contraction Mapping Theorem

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

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

Contraction Mapping Theorem

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

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

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

Remark. In general we only need metric (vs normed) space

Value Function Sapce

- Consider the vector space V over value functions

- There are $|\mathcal{S}|$ dimensions

- Each point in this space fully specifies a value function $\mathbf{v}(s)$

- 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 $\mathbf{u}$ and $\mathbf{v}$ by the ∞-norm

- i.e. the largest difference between state values:

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

Bellman Expectation Backup is a Contraction

- Define the Bellman expectation backup operator

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

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

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

Convergence of Iter. Policy Evaluation and Policy Iteration

- The Bellman expectation operator F^π has a unique fixed point

- V_π is a fixed point of F^π (by Bellman expectation equation)

- By contraction mapping theorem

- Iterative policy evaluation converges on V_π

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:

$$q_\pi(s, a) = \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s, A_t = a]$$
$$= r(s, a) + \gamma \sum_{s' \in \mathcal{S}} T(s'|s, a) v_\pi(s')$$

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

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:

$$
\begin{aligned}
q_\pi(s, a) &= \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = a] \\
&= r(s, a) + \gamma \sum_{s' \in \mathcal{S}} T(s'|s, a) v_\pi(s')
\end{aligned}
$$

q_\pi(s, a) & = \mathbb{E}[R_{t+1} + \gamma \text{v}_\pi(S_{t+1})|S_t=s,A_t=a] \\
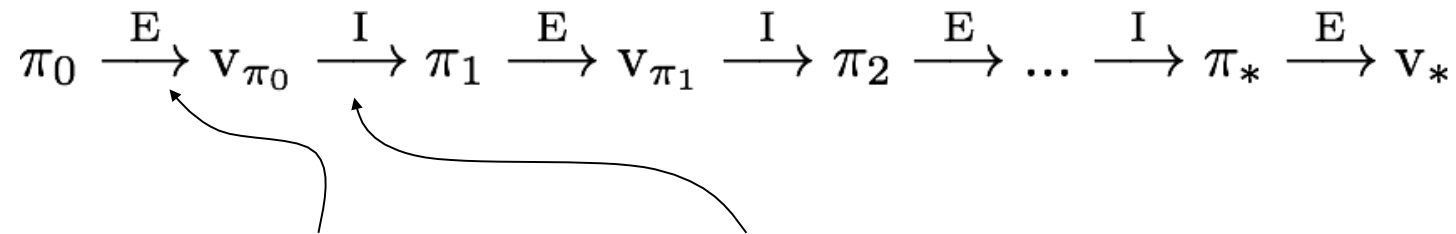
Policy Improvement Cont.

- Do this for all states to get a new policy $\pi' \geq \pi$ that is greedy with respect to v_π:

$$
\begin{aligned}
\pi'(s) &= \arg\max_a q_\pi(s, a) \\
&= \arg\max_a \mathbb{E}[R_{t+1} + \gamma v_\pi(s')|S_t = s, A_t = a] \\
&= \arg\max r(s, a) + \gamma \sum_{s' \in \mathcal{S}} T(s'|s, a)v_\pi(s')
\end{aligned}
$$

- What if the policy is unchanged by this?

  - Then the policy must be optimal.

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

Policy Iteration

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

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
   Repeat
   $\quad \Delta \leftarrow 0$
   $\quad$ For each $s \in \mathcal{S}$:
   $\quad\quad v \leftarrow V(s)$
   $\quad\quad V(s) \leftarrow \boxed{\Sigma_{a \in \mathcal{A}} \pi(a|s) \left(r(s,a) + \gamma \Sigma_{s' \in \mathcal{S}} T(s'|s,a) V(s')\right)}$
   $\quad\quad \Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number)

3. Policy Improvement
   $policy\text{-}stable \leftarrow true$
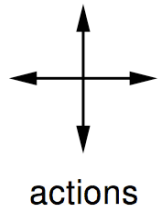   For each $s \in \mathcal{S}$:
   $\quad a \leftarrow \pi(s)$
   $\quad \pi(s) \leftarrow \boxed{\arg\max r(s,a) + \gamma \Sigma_{s' \in \mathcal{S}} T(s'|s,a) v_\pi(s')}$
   $\quad$ If $a \neq \pi(s)$, then $policy\text{-}stable \leftarrow false$
   If $policy\text{-}stable$, then stop and return V and π; else go to 2

Iterative Policy Eval for the Small Gridworld

Policy π, an equiprobable random action
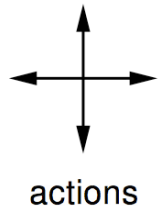
$R = -1$ on all transitions

$\gamma = 1$

actions

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

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

V_k for the Random Policy

Greedy Policy w.r.t. V_k

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

Policy π, an equiprobable random action

$R = -1$
on all transitions

$\gamma = 1$

actions

- An undiscounted episodic task

- Nonterminal states: 1, 2, … , 14

- Terminal state: one, shown in shaded square

- Actions that take the agent off the grid leave the state unchanged

- Reward is -1 until the terminal state is reached

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?

- For example, in the small grid world $k = 3$ was sufficient to achieve optimal policy

- Why not update policy every iteration? i.e. stop after $k = 1$

  - This is equivalent to value iteration (next section)

Generalized Policy Iteration

Generalized Policy Iteration (GPI): any interleaving of policy evaluation and policy improvement, independent of their granularity.
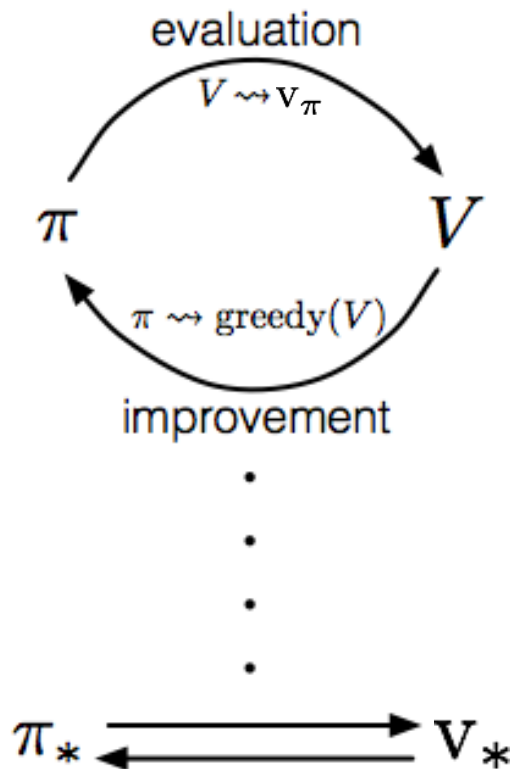
A geometric metaphor for convergence of GPI:
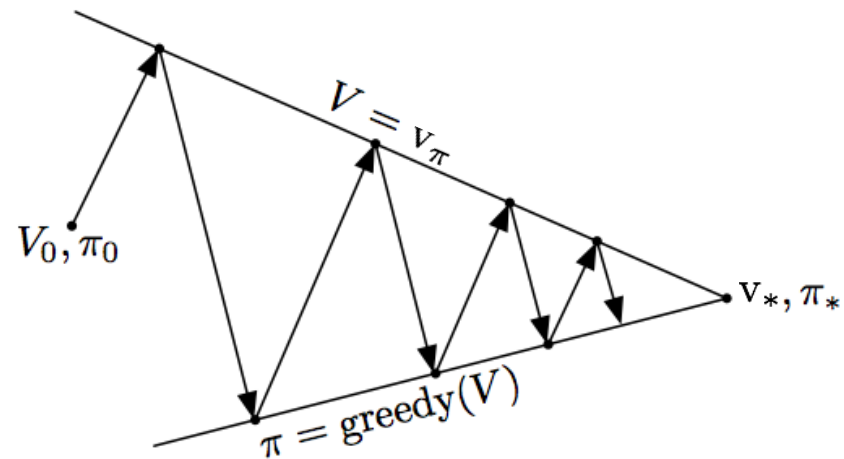
Principle of Optimality

- Any optimal policy can be subdivided into two components:

  - An optimal first action $\mathcal{A}_*$

  - Followed by an optimal policy from successor state $\mathcal{S}'$

- Theorem (Principle of Optimality)

  - A policy $\pi(a|s)$ achieves the optimal value from state s, $v_\pi(s) = v_*(s)$, if and only if

  - For any state s' reachable from s, π achieves the optimal value from state s', $v_\pi(s') = v_*(s')$

Value Iteration

- If we know the solution to subproblems $\mathbf{v}_*(s')$

- Then solution $\mathbf{v}_*(s')$ can be found by one-step lookahead

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

- The idea of value iteration is to apply these updates iteratively

- Intuition: start with final rewards and work backwards

- Still works with loopy, stochastic MDPs

`\text{v}_*(s) \leftarrow \max_{a \in \mathcal{A}}{ r(s,a) + \gamma \sum_{s'\in \mathcal{S}} T(s'|s,a) {\text{v}_*(s')} }`

Example: Shortest Path

g			

Problem

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

V_1

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

V_2

0	-1	-2	-2
-1	-2	-2	-2
-2	-2	-2	-2
-2	-2	-2	-2

V_3

0	-1	-2	-3
-1	-2	-3	-3
-2	-3	-3	-3
-3	-3	-3	-3

V_4

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-4
-3	-4	-4	-4

V_5

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-5
-3	-4	-5	-5

V_6

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-5
-3	-4	-5	-6

V_7

Value Iteration

- **Problem**: find optimal policy π

- **Solution**: iterative application of Bellman optimality backup

- $v_1 \rightarrow v_2 \rightarrow ... \rightarrow v_*$

- Using synchronous backups

  - At each iteration $k + 1$

  - For all states $s \in \mathcal{S}$

  - Update $v_{k+1}(s)$ from $v_k(s')$

\text{v}_1 \rightarrow \text{v}_2 \rightarrow ... \rightarrow \text{v}_*

Value Iteration (2)

$$v_{k+1}(s) \leftarrow s$$

$$v_k(s') \leftarrow s'$$
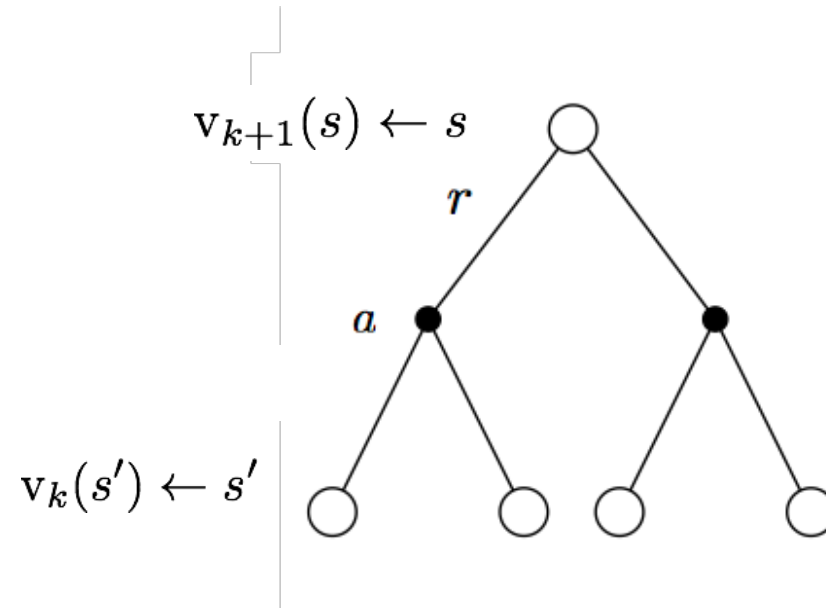
$$v_{k+1}(s) = \max_{a \in \mathcal{A}} \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s'|s,a) v_k(s') \right)$$

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

\text{v}_{k+1}(s) = \max_{a \in \mathcal{A}}{ \left(r(s,a) + \gamma \sum_{s'\in \mathcal{S}} T(s'|s,a) {\text{v}_k(s')} \right) }

Bellman Optimality Backup is a Contraction

- Define the Bellman optimality backup operator F^*,

$$F^*(\mathbf{v}) = \max_{a \in \mathcal{A}} r(a) + \gamma T(a)\mathbf{v}$$

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

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

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 $O(mn^2)$ per iteration, for m actions and n states
- Could also apply to action-value function $q_\pi(s, a)$ or $q_*(s, a)$

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:

    - Pick 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

- Can you select states to backup intelligently? YES: an agent's experience can act as a guide.

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 $\mathcal{S}$

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

$$\mathbf{v}_{old} \leftarrow \mathbf{v}_{new}$$

- In-place value iteration only stores one copy of value function

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

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

Prioritized Sweeping

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

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

- Backup the state with the largest remaining Bellman error

- Requires knowledge of reverse dynamics (predecessor states)

- Can be implemented efficiently by maintaining a priority queue

\left\lvert \max_{a \in \mathcal{A}} {\left(r(s,a) + \gamma \sum_{s'\in \mathcal{S}} T(s'|s,a) \textcolo\r{red}{\text{v}(s')} \righ

Real-time Dynamic Programming

- **Idea**: update 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 $\mathcal{S}_t$

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

Sample Backups

- In subsequent lectures we will consider sample backups

- Using sample rewards and sample transitions $(\mathcal{S}, \mathcal{A}, r, \mathcal{S}')$

- Instead of reward function and transition dynamics

- Advantages:

  - Model-free: no advance knowledge of MDP required

  - Breaks the curse of dimensionality through sampling

  - Cost of backup is constant, independent of $n = |\mathcal{S}|$
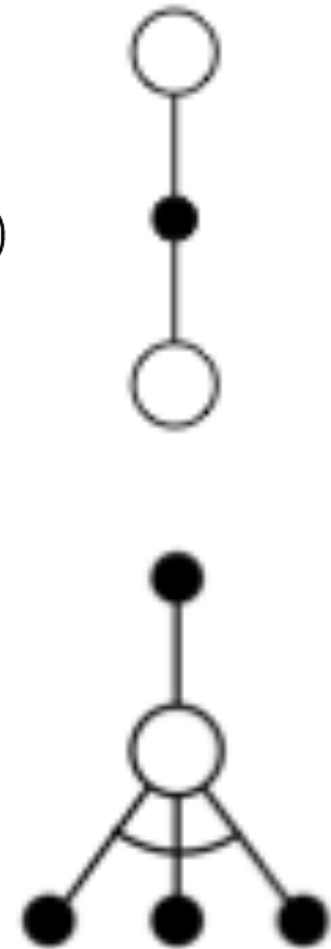
Approximate Dynamic Programming

- Approximate the value function

- Using a function approximate $\hat{v}(s, \mathbf{w})$

- Apply dynamic programming to $\hat{v}(\cdot, \mathbf{w})$

- e.g. Fitted Value Iteration repeats at each iteration *k*,

  - Sample states $\tilde{\mathcal{S}} \subseteq \mathcal{S}$

  - For each state $s \in \tilde{\mathcal{S}}$, estimate target value using Bellman optimality equation,

$$\tilde{v}_k(s) = \max_{a \in \mathcal{A}} \left(r(s, a) + \gamma \sum_{s' \in \mathcal{S}} T(s'|s, a) \hat{v}(s', \mathbf{w}_k) \right)$$

  - Train next value function $\hat{v}(\cdot, \mathbf{w}_{k+1})$ using targets $\{\langle s, \tilde{v}_k(s) \rangle\}$