Policy and Value Iteration

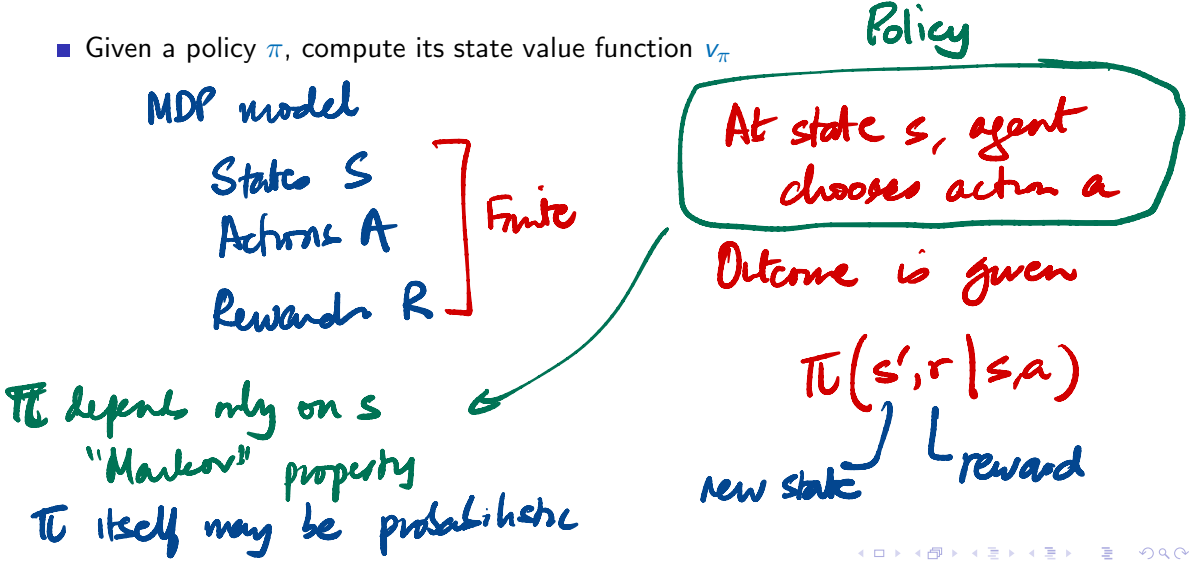
Madhavan Mukund

https://www.cmi.ac.in/~madhavan

Advanced Machine Learning
September–December 2021

Policy evaluation

- Given a policy π, compute its state value function v_π



Policy evaluation

- Given a policy π, compute its state value function v_π
- Bellman equations: $v_\pi(s) = \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a) \left[r + \gamma v_\pi(s') \right]$
  - For MDP with n states, n equations in n unknowns
  - Can solve to get v_π, but computationally infeasible for large n

$\pi_1 \geq \pi_2$ if $v_{\pi_1}(s) \geq v_{\pi_2}(s)$ for all s

There exists an optimal π_*, $\pi_* \geq \pi \ \forall \pi$

Policy evaluation

- Given a policy π, compute its state value function v_π
- Bellman equations: $v_\pi(s) = \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a) \left[r + \gamma v_\pi(s') \right]$
  - For MDP with n states, n equations in n unknowns
  - Can solve to get v_π, but computationally infeasible for large n
- Instead, use the Bellman equations as update rules

Given π $\longrightarrow$ compute v_π

$(x, y) = (y, x)$

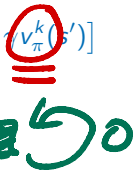
Policy evaluation

- Given a policy π, compute its state value function v_π
- Bellman equations: $v_\pi(s) = \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a) \left[r + \gamma v_\pi(s') \right]$
  - For MDP with n states, n equations in n unknowns
  - Can solve to get v_π, but computationally infeasible for large n
- Instead, use the Bellman equations as update rules
  - Initialize $v_\pi^0(s)$: set $v_\pi^0(\text{term}) = 0$ for terminal state term, arbitrary values for other s

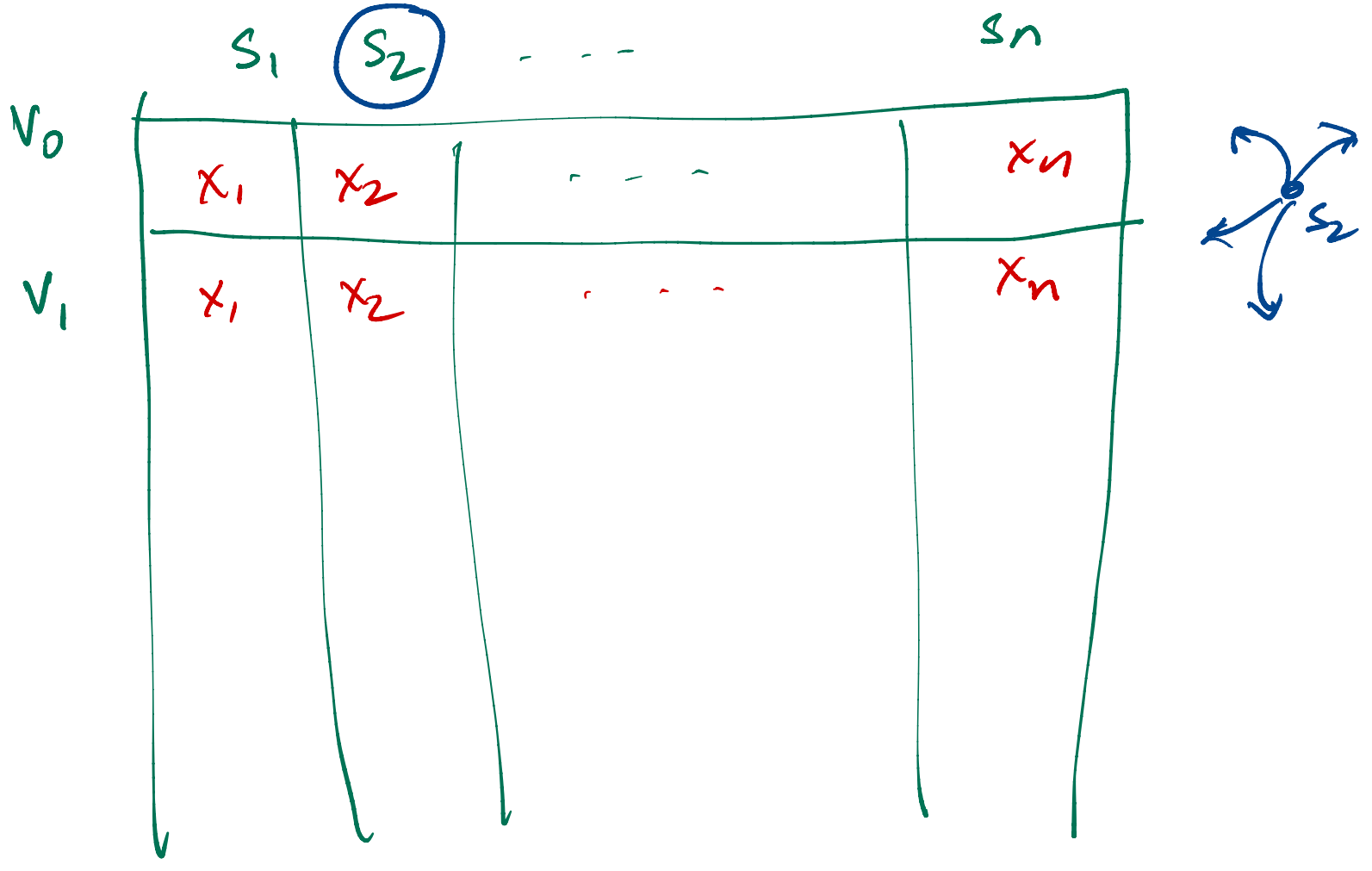
Policy evaluation

- Given a policy π, compute its state value function v_π
- Bellman equations: $v_\pi(s) = \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a) \left[r + \gamma v_\pi(s') \right]$
  - For MDP with n states, n equations in n unknowns
  - Can solve to get v_π, but computationally infeasible for large n

- Instead, use the Bellman equations as update rules
  - Initialize $v_\pi^0(s)$: set $v_\pi^0(\text{term}) = 0$ for terminal state term, arbitrary values for other s
  - Update v_π^k to v_π^{k+1} using: $v_\pi^{k+1}(s) = \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a) \left[r + \gamma v_\pi^k(s') \right]$



Terminal state has "no future"
V(s) — future reward from s


S_1
S_2
S_n
V_0
V_1
x_1
x_2
x_n
x_1
x_2
x_n
S_2

Policy evaluation

- Given a policy π, compute its state value function v_π
- Bellman equations: $v_\pi(s) = \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a) \left[r + \gamma v_\pi(s') \right]$
  - For MDP with n states, n equations in n unknowns
  - Can solve to get v_π, but computationally infeasible for large n
- Instead, use the Bellman equations as update rules
  - Initialize $v_\pi^0(s)$: set $v_\pi^0(\text{term}) = 0$ for terminal state ~~term~~, arbitrary values for other s
  - Update v_π^k to v_π^{k+1} using: $v_\pi^{k+1}(s) = \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a) \left[r + \gamma v_\pi^k(s') \right]$

Policy evaluation

- Given a policy π, compute its state value function v_π
- Bellman equations: $v_\pi(s) = \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a) \left[r + \gamma v_\pi(s') \right]$
  - For MDP with n states, n equations in n unknowns
  - Can solve to get v_π, but computationally infeasible for large n
- Instead, use the Bellman equations as update rules
  - Initialize $v_\pi^0(s)$: set $v_\pi^0(\text{term}) = 0$ for terminal state term, arbitrary values for other s
  - Update v_π^k to v_π^{k+1} using: $v_\pi^{k+1}(s) = \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a) \left[r + \gamma v_\pi^k(s') \right]$
  - Stop when incremental change $\Delta = |v_\pi^{k+1} - v_\pi^k|$ is below threshold θ

Policy evaluation

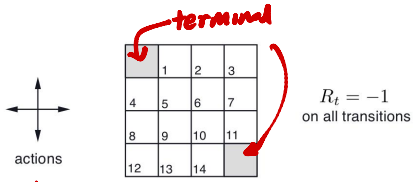
Iterative Policy Evaluation, for estimating $V \approx v_\pi$

Input π, the policy to be evaluated
Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:
  $\Delta \leftarrow 0$ — max difference (absolute) observed in this iteration
  Loop for each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\left[r + \gamma V(s')\right]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Absolute value

Policy evaluation example



outcome is deterministic, stay in place if you try to move off-grid

π = uniformly random

0.25 in all directions

v_π = expected no of steps to terminate

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

$4 \times (0.25 \times -1 + 0)$

$k = 2$

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 improvement

- Assume a deterministic policy π
- Using v_π, can we find a better policy π'?

Policy improvement

- Assume a deterministic policy π
- Using v_π, can we find a better policy π'?
- Is there a state s where we can substitute $\pi(s)$ by a better choice a?

$v_\pi(s)$

$q_\pi(s,a)$

$\pi(a|s)$ $= 1$ for chosen a
0 otherwise

$p(s', r|s, a)$ – probability

Policy improvement

- Assume a deterministic policy π
- Using v_π, can we find a better policy π'?
- Is there a state s where we can substitute $\pi(s)$ by a better choice a?
- $q_\pi(s, a) = \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a]$

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

Policy improvement

- Assume a deterministic policy π
- Using v_π, can we find a better policy π'?
- Is there a state s where we can substitute $\pi(s)$ by a better choice a?
- $q_\pi(s, a) = \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a]$

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

- If $q_\pi(s, a) > v_\pi(s)$, modify π so that $\pi(s) = a$

Policy improvement

- Assume a deterministic policy π
- Using v_π, can we find a better policy π'?
- Is there a state s where we can substitute $\pi(s)$ by a better choice a?
- $q_\pi(s,a) = \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a]$
$$= \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma v_\pi(s') \right]$$
- If $q_\pi(s,a) > v_\pi(s)$, modify π so that $\pi(s) = a$
- The new policy π' is strictly better

$\pi' = \pi$ except at s

Policy improvement

Policy Improvement Theorem

For deterministic policies π, π':

- If $q_\pi(s, \pi'(s)) \geq v_\pi(s)$ for all s, then $\pi' \geq \pi$,
- If $\pi' \geq \pi$ and $q_\pi(s, \pi'(s)) > v_\pi(s)$ for some s, then $v_{\pi'}(s) > v_\pi(s)$.

Policy improvement

Policy Improvement Theorem

For deterministic policies π, π':

- If $q_\pi(s, \pi'(s)) \geq v_\pi(s)$ for all s, then $\pi' \geq \pi$,
- If $\pi' \geq \pi$ and $q_\pi(s, \pi'(s)) > v_\pi(s)$ for some s, then $v_{\pi'}(s) > v_\pi(s)$.

- Proof of the theorem is not difficult for deterministic policies

Policy improvement

Policy Improvement Theorem

For deterministic policies π, π':

- If $q_\pi(s, \pi'(s)) \geq v_\pi(s)$ for all s, then $\pi' \geq \pi$,
- If $\pi' \geq \pi$ and $q_\pi(s, \pi'(s)) > v_\pi(s)$ for some s, then $v_{\pi'}(s) > v_\pi(s)$.

- Proof of the theorem is not difficult for deterministic policies
- The theorem extends to probabilistic policies also

Policy improvement

Policy Improvement Theorem

For deterministic policies π, π':

- If $q_\pi(s, \pi'(s)) \geq v_\pi(s)$ for all s, then $\pi' \geq \pi$,
- If $\pi' \geq \pi$ and $q_\pi(s, \pi'(s)) > v_\pi(s)$ for some s, then $v_{\pi'}(s) > v_\pi(s)$.

- Proof of the theorem is not difficult for deterministic policies
- The theorem extends to probabilistic policies also
- Provides a basis to iteratively improve the policy

Policy iteration

- Start with a random policy π_0

Policy iteration

- Start with a random policy π_0
- Use policy evaluation to compute v_{π_0}

Policy iteration

- Start with a random policy π_0
- Use policy evaluation to compute v_{π_0}
- Use policy improvement to construct a better policy π_1

Policy iteration

- Start with a random policy π_0
- Use policy evaluation to compute v_{π_0}
- Use policy improvement to construct a better policy π_1
- Policy iteration: Alternate between policy evaluation and policy improvement

$$\pi_0 \xrightarrow{\text{evaluate}} v_{\pi_0} \xrightarrow{\text{improve}} \pi_1 \xrightarrow{\text{evaluate}} v_{\pi_1} \xrightarrow{\text{improve}} \pi_2 \xrightarrow{\text{evaluate}} \cdots$$

Policy iteration

- Start with a random policy π_0
- Use policy evaluation to compute v_{π_0}
- Use policy improvement to construct a better policy π_1
- Policy iteration: Alternate between policy evaluation and policy improvement

$$\pi_0 \xrightarrow{\text{evaluate}} v_{\pi_0} \xrightarrow{\text{improve}} \pi_1 \xrightarrow{\text{evaluate}} v_{\pi_1} \xrightarrow{\text{improve}} \pi_2 \xrightarrow{\text{evaluate}} \cdots \xrightarrow{\text{improve}} \pi_* \xrightarrow{\text{evaluate}} v_{\pi_*}$$

- Finite MDPs — can improve π only finitely many times,
  - Must converge to optimal policy

Policy iteration

- Start with a random policy π_0
- Use policy evaluation to compute v_{π_0}
- Use policy improvement to construct a better policy π_1
- Policy iteration: Alternate between policy evaluation and policy improvement

$$\pi_0 \xrightarrow{\text{evaluate}} v_{\pi_0} \xrightarrow{\text{improve}} \pi_1 \xrightarrow{\text{evaluate}} v_{\pi_1} \xrightarrow{\text{improve}} \pi_2 \xrightarrow{\text{evaluate}} \cdots \xrightarrow{\text{improve}} \pi_* \xrightarrow{\text{evaluate}} v_{\pi_*}$$

- Finite MDPs — can improve π only finitely many times,
  - Must converge to optimal policy
- Nested iteration — each policy evaluation is itself an iteration
  - Speed up by using v_{π_i} as initial state to compute $v_{\pi_{i+1}}$

Policy iteration

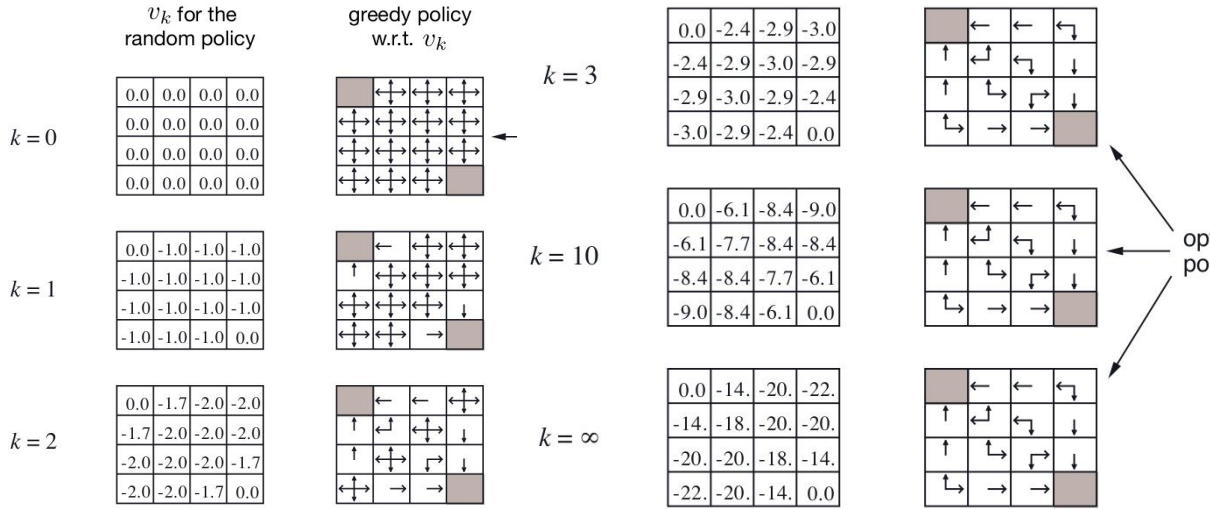
Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

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

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

Optimizing Policy Iteration



Value iteration

- Policy iteration — policy evaluation requires a nested iteration

Value iteration

- Policy iteration — policy evaluation requires a nested iteration
- A partial computation of v_{π_k} is sufficent to proceed towards π_*, v_*

Value iteration

- Policy iteration — policy evaluation requires a nested iteration
- A partial computation of v_{π_k} is sufficent to proceed towards π_*, v_*
- Even a single iteration in the computation of v_{π_k} will do

Value iteration

- Policy iteration — policy evaluation requires a nested iteration
- A partial computation of v_{π_k} is sufficent to proceed towards π_*, v_*
- Even a single iteration in the computation of v_{π_k} will do
- Combine policy improvement and one step update at each state

Value iteration

- Policy iteration — policy evaluation requires a nested iteration
- A partial computation of v_{π_k} is sufficent to proceed towards π_*, v_*
- Even a single iteration in the computation of v_{π_k} will do
- Combine policy improvement and one step update at each state
- Value iteration

$$v_{\pi_{k+1}}(s,a) = \max_a \mathbb{E}[R_{t+1} + \gamma v_{\pi_k}(S_{t+1}) \mid S_t = s, A_t = a]$$

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

greedy policy update

1 step value update

Value iteration

- Policy iteration — policy evaluation requires a nested iteration
- A partial computation of v_{π_k} is sufficent to proceed towards π_*, v_*
- Even a single iteration in the computation of v_{π_k} will do
- Combine policy improvement and one step update at each state
- Value iteration

$$v_{\pi_{k+1}}(s,a) = \max_a \mathbb{E}[R_{t+1} + \gamma v_{\pi_k}(S_{t+1}) \mid S_t = s, A_t = a]$$

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

- Again, stop when incremental change $\Delta = |v_{\pi_{k+1}} - v_{\pi_k}|$ is below threshold θ

Dynamic programming

- In the literature, policy iteration and value iteration are referred to as dynamic programming methods

Jeff Ericlson
Algorithms
(Purdue)

Dynamic programming

- In the literature, policy iteration and value iteration are referred to as dynamic programming methods
- Requires knowledge of the model — $p(s', r \mid s, a)$

Dynamic programming

- In the literature, policy iteration and value iteration are referred to as dynamic programming methods
- Requires knowledge of the model — $p(s', r \mid s, a)$
- How to combine policy evaluation and policy improvement is flexible
  - Value iteration is policy iteration with policy evaluation truncated to a single step
  - Generalized policy iteration — simultaneously maintain and update approximations of π_* and v_*

Dynamic programming

- In the literature, policy iteration and value iteration are referred to as dynamic programming methods
- Requires knowledge of the model — $p(s', r \mid s, a)$
- How to combine policy evaluation and policy improvement is flexible
  - Value iteration is policy iteration with policy evaluation truncated to a single step
  - Generalized policy iteration — simultaneously maintain and update approximations of π_* and v_*
- Asynchronous dynamic programming for large state spaces

Fair exploration of entire space