Sequential decisions under uncertainty
Policy iteration

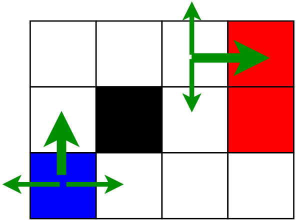
Tomáš Svoboda & Matej Hoffmann

Vision for Robots and Autonomous Systems, Center for Machine Perception
Department of Cybernetics
Faculty of Electrical Engineering, Czech Technical University in Prague

March 29, 2021

Unreliable actions in observable grid world

- Walls block movement – agent/robot stays in place.
- Actions do not always go as planned.
- Agent receives rewards each time step:
  - Small "living" reward/penalty.
  - Big rewards/penalties at the end.
- Goal: maximize sum of (discounted) rewards



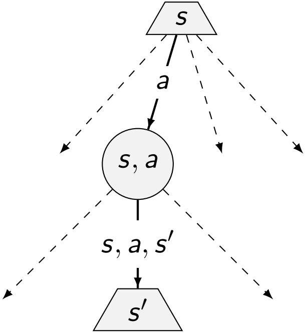
MDPs recap

Markov decision processes (MDPs):

- Set of states $\mathcal{S}$
- Set of actions $\mathcal{A}$
- Transitions $p(s'|s,a)$ or $T(s,a,s')$
- Rewards $r(s,a,s')$; and discount γ

MDP quantities:

- Policy $\pi(s) : \mathcal{S} \rightarrow \mathcal{A}$
- Utility – sum of (discounted) rewards.
- Values – expected future utility from a state (max-node), $v(s)$
- *Q*-Values – expected future utility from a *q*-state (chance-node), $q(s,a)$



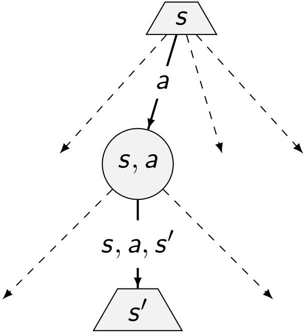
MDPs recap

Markov decision processes (MDPs):

- Set of states $\mathcal{S}$
- Set of actions $\mathcal{A}$
- Transitions $p(s'|s,a)$ or $T(s,a,s')$
- Rewards $r(s,a,s')$; and discount γ

MDP quantities:

- Policy $\pi(s) : \mathcal{S} \rightarrow \mathcal{A}$
- Utility – sum of (discounted) rewards.
- Values – expected future utility from a state (max-node), $v(s)$
- Q-Values – expected future utility from a q-state (chance-node), $q(s,a)$



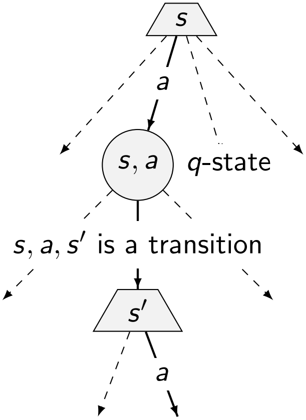
Optimal quantities

- The optimal policy: $\pi^*(s)$ – optimal action from state s
- Expected utility/return of a policy.

$$U^\pi(S_t) = \mathrm{E}^\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}\right]$$

Best policy π^* maximizes above.

- The value of a state s: $v^*(s)$ – expected utility starting in s and acting optimally.
- The value of a q-state (s, a): $q^*(s, a)$ - expected utility having taken a from state s and acting optimally thereafter.



s, a, s' is a transition

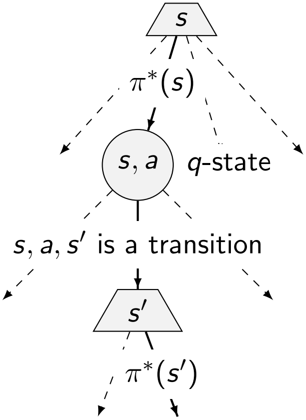
Optimal quantities

- The optimal policy: $\pi^*(s)$ – optimal action from state s
- Expected utility/return of a policy.

$$U^\pi(S_t) = \mathrm{E}^\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}\right]$$

Best policy π^* maximizes above.

- The value of a state s: $v^*(s)$ – expected utility starting in s and acting optimally.
- The value of a q-state (s,a): $q^*(s,a)$ - expected utility having taken a from state s and acting optimally thereafter.



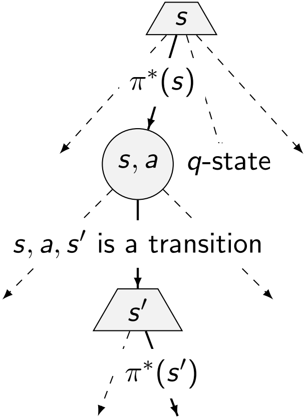
Optimal quantities

- The optimal policy: $\pi^*(s)$ – optimal action from state s
- Expected utility/return of a policy.

$$U^\pi(S_t) = \mathrm{E}^\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}\right]$$

Best policy π^* maximizes above.

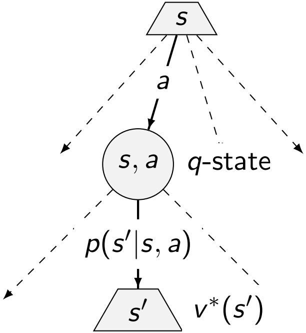
- The value of a state s: $v^*(s)$ – expected utility starting in s and acting optimally.
- The value of a q-state (s, a): $q^*(s, a)$ - expected utility having taken a from state s and acting optimally thereafter.



v^* and q^*

The value of a q-state (s, a):

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



The value of a state s:

$$v^*(s) = \max_a q^*(s, a)$$

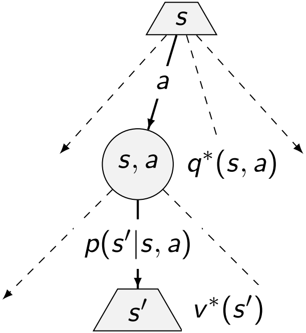
v^* and q^*

The value of a q-state (s, a):

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

The value of a state s:

$$v^*(s) = \max_a q^*(s,a)$$



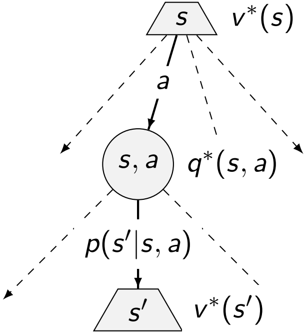
v^* and q^*

The value of a q-state (s, a):

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

The value of a state s:

$$v^*(s) = \max_a q^*(s,a)$$

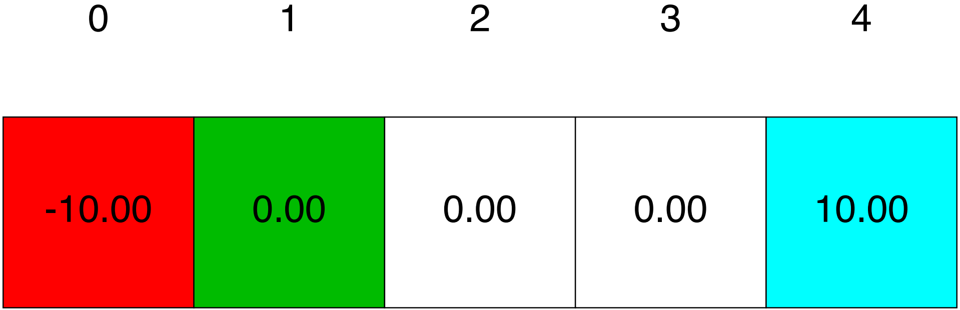


Maze: $V_0 = [0, 0, 0]^\top$, $r(s) = -1$, deterministic robot, $\mathcal{A} = \{\leftarrow, \uparrow, \downarrow, \rightarrow\}$, $\gamma = 1$

0	1	2	3	4
-10.00	0.00	0.00	0.00	10.00

$$\begin{aligned} q^*(s, a) &= \sum_{s'} p(s'|a, s)\left[r(s, a, s') + \gamma\, v^*(s')\right] \\ v^*(s) &= \max_a\, q^*(s, a) \end{aligned}$$

What will be V^* after first sweep? $V_1^* = [v_1^*(1), v_1^*(2), v_1^*(3)]^\top$?



0	1	2	3	4
-10.00	0.00	0.00	0.00	10.00

Sweep is meant as the Bellmann update for all states: $V_1^* = BV_0^*$. $r(s) = -1$. Assume sync version of the algorithm.

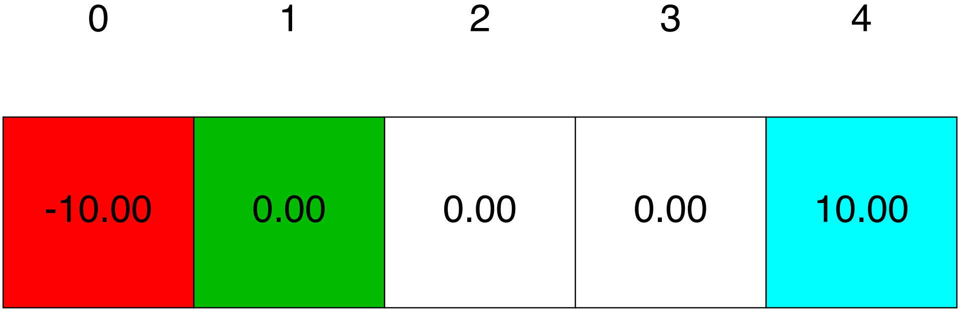
A: $V_1^* = [-1, -1, 9]^\top$

B: $V_1^* = [0, 8, 9]^\top$

C: $V_1^* = [-1, 0, 0]^\top$

D: $V_1^* = [-11, 8, 9]^\top$

What will be V^* after second sweep? $V_2^* = [v_2^*(1), v_2^*(2), v_2^*(3)]^\top$?



0	1	2	3	4
🟥 -10.00	🟩 0.00	0.00	0.00	10.00

Sweep is meant as the Bellmann update for all states: $V_2^* = B(BV_0^*)$. $r(s) = -1$. Assume sync version of the algorithm.

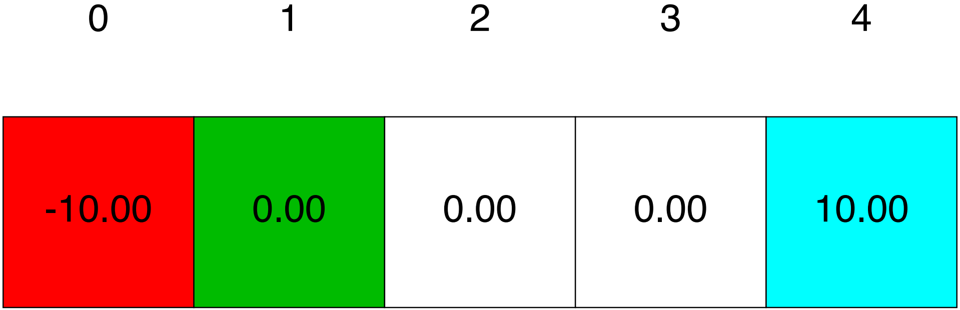
A: $V_2^* = [-1, -1, 9]^\top$

B: $V_2^* = [-1, 8, 9]^\top$

C: $V_2^* = [-2, 8, 9]^\top$

D: $V_2^* = [7, 8, 9]^\top$

What will be the $q^*(s, a)$ values after second value-iter sweep?



0	1	2	3	4
-10.00	0.00	0.00	0.00	10.00

Pick the option which is *wrong*:

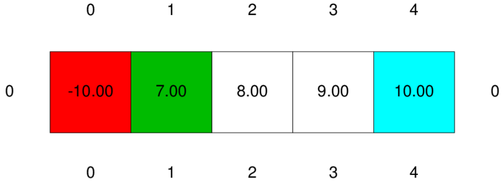
A: $q^*(1, \uparrow) = -2$

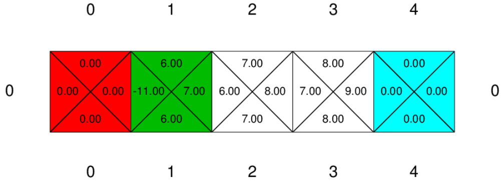
B: $q^*(1, \rightarrow) = -2$

C: $q^*(2, \rightarrow) = -2$

D: $q^*(3, \leftarrow) = -2$

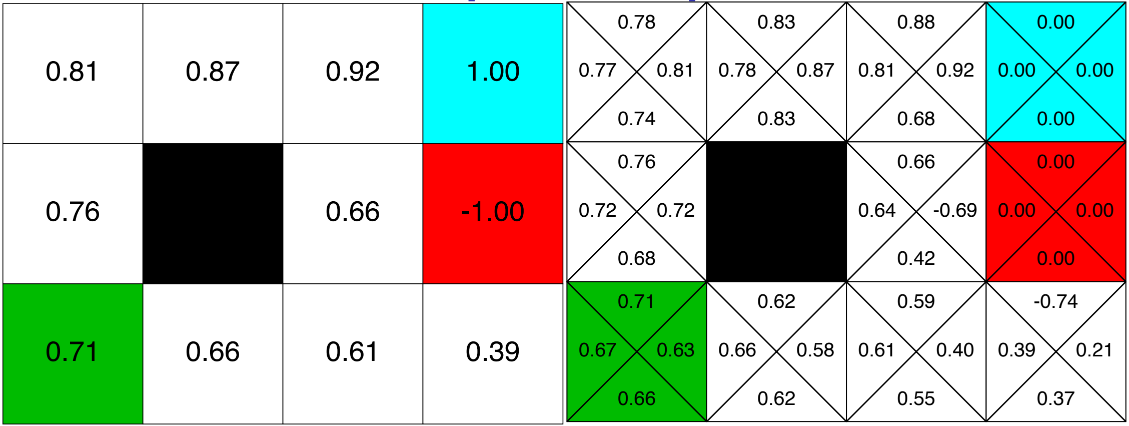
Maze: v^* vs. q^*, deterministic robot, $\mathcal{A} = \{\leftarrow, \uparrow, \downarrow, \rightarrow\}$





$$
\begin{aligned}
q^*(s,a) &= \sum_{s'} p(s'|a,s)\left[r(s,a,s') + \gamma\, v^*(s')\right] \\
v^*(s) &= \max_a q^*(s,a)
\end{aligned}
$$

Maze: v^* vs. q^*, $\gamma = 1$, $T = [0.8, 0.1, 0.1, 0]$



0.81	0.87	0.92	1.00
0.76	■	0.66	-1.00
0.71	0.66	0.61	0.39

0.78 0.77 0.81 0.74	0.83 0.78 0.87 0.83	0.88 0.81 0.92 0.68	0.00 0.00 0.00 0.00
0.76 0.72 0.72 0.68	■	0.66 0.64 -0.69 0.42	0.00 0.00 0.00 0.00
0.71 0.67 0.63 0.66	0.62 0.66 0.58 0.62	0.59 0.61 0.40 0.55	-0.74 0.39 0.21 0.37

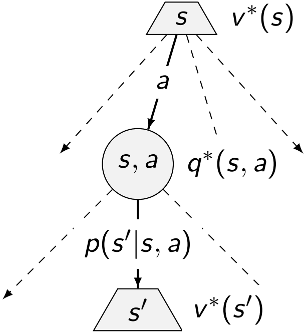
$$q^*(s,a) = \sum_{s'} p(s'|a,s)\left[r(s,a,s') + \gamma\, v^*(s')\right]$$

$$v^*(s) = \max_a q^*(s,a)$$

Value iteration

- Bellman equations **characterize** the optimal values

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



- Value iteration **computes** them:

$$V_{k+1}(s) \leftarrow \max_{a \in A(s)} \sum_{s'} p(s'|s,a)\left[r(s,a,s') + \gamma V_k(s')\right]$$

Value iteration is a fixed point solution method.

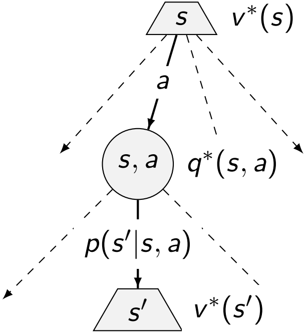
Value iteration

- Bellman equations **characterize** the optimal values

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

- Value iteration **computes** them:

$$V_{k+1}(s) \leftarrow \max_{a \in A(s)} \sum_{s'} p(s'|s,a)\left[r(s,a,s') + \gamma V_k(s')\right]$$



Value iteration is a fixed point solution method.

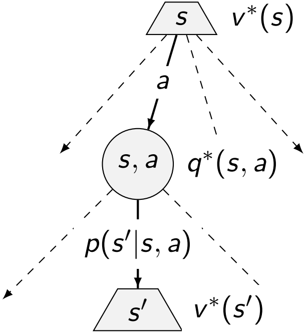
Value iteration

- Bellman equations **characterize** the optimal values

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

- Value iteration **computes** them:

$$V_{k+1}(s) \leftarrow \max_{a \in A(s)} \sum_{s'} p(s'|s,a) \left[r(s,a,s') + \gamma V_k(s') \right]$$

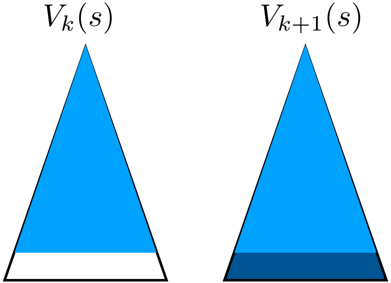


Value iteration is a fixed point solution method.

Convergence

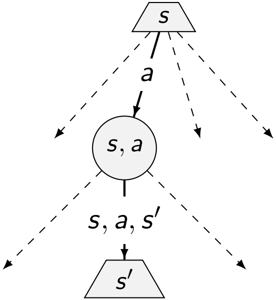
$$V_{k+1}(s) \leftarrow \max_{a \in A(s)} \sum_{s'} p(s'|s,a)\left[r(s,a,s') + \gamma V_k(s')\right]$$

- Thinking about special cases: deterministic world, $\gamma = 0$, $\gamma = 1$.
- For all s, $V_k(s)$ and $V_{k+1}(s)$ can be seen as expectimax search trees of depth k and $k+1$

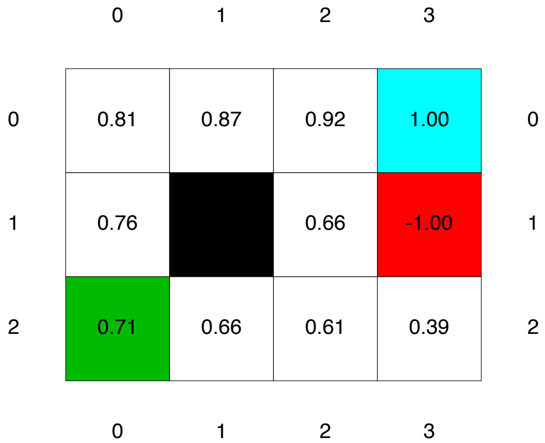


From Values to Policy

Policy extraction - computing actions from Values



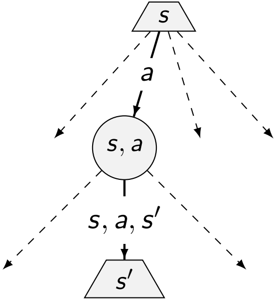
- Assume we have $v^*(s)$
- What is the optimal action?
- We need a one-step expectimax:



	0	1	2	3	
0	0.81	0.87	0.92	1.00	0
1	0.76		0.66	-1.00	1
2	0.71	0.66	0.61	0.39	2
	0	1	2	3	

$$\pi^*(s) = \arg\max_{a\in\mathcal{A}(s)} \sum_{s'} p(s' \mid s, a)\left[r(s, a, s') + \gamma v^*(s')\right]$$

Policy extraction - computing actions from Values



	0	1	2	3	
0	0.81	0.87	0.92	1.00	0
1	0.76	■	0.66	-1.00	1
2	0.71	0.66	0.61	0.39	2
	0	1	2	3	

- Assume we have $v^*(s)$
- What is the optimal action?
- We need a one-step expectimax:

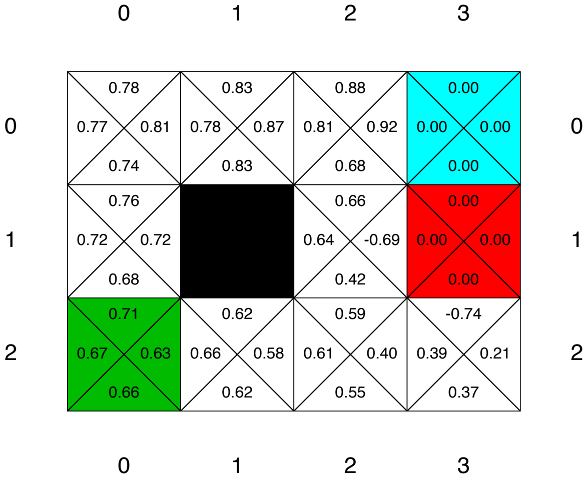
$$\pi^*(s) = \underset{a \in \mathcal{A}(s)}{\arg\max} \sum_{s'} p(s' \mid s, a) \left[r(s, a, s') + \gamma v^*(s') \right]$$

Policy extraction - computing actions from q-Values

- Assume we have $q^*(s, a)$
- What is the optimal action?
- Just take the (arg) max:

$$\pi^*(s) = \arg\max_{a \in \mathcal{A}(s)} q^*(s, a)$$

Actions are easier to extract from q-values.

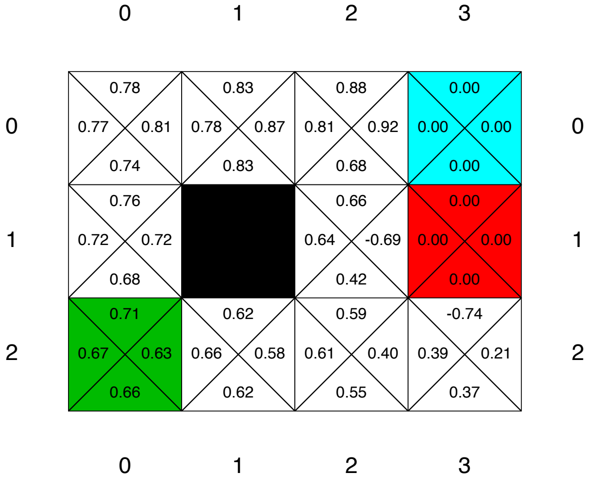


Policy extraction - computing actions from q-Values

- Assume we have $q^*(s, a)$
- What is the optimal action?
- Just take the (arg) max:

$$\pi^*(s) = \arg\max_{a \in \mathcal{A}(s)} q^*(s, a)$$

Actions are easier to extract from q-values.



What is wrong with the Value iteration?

$$V_{k+1}(s) \leftarrow \max_{a\in\mathcal{A}(s)} \sum_{s'} p(s' \mid s, a) \left[r(s, a, s') + \gamma V_k(s') \right]$$

- What is complexity of one iteration - over all S states?
- When does the iteration stop?
- When the does the policy converge?
- Can we compute the policy directly?

What is wrong with the Value iteration?

$$V_{k+1}(s) \leftarrow \max_{a \in \mathcal{A}(s)} \sum_{s'} p(s' \mid s, a) \left[r(s, a, s') + \gamma V_k(s') \right]$$

- What is complexity of one iteration - over all S states?
- When does the iteration stop?
- When the does the policy converge?
- Can we compute the policy directly?

What is wrong with the Value iteration?

$$V_{k+1}(s) \leftarrow \max_{a \in \mathcal{A}(s)} \sum_{s'} p(s' \mid s, a) \left[r(s, a, s') + \gamma V_k(s') \right]$$

- What is complexity of one iteration - over all S states?
- When does the iteration stop?
- When the does the policy converge?
- Can we compute the policy directly?

What is wrong with the Value iteration?

$$V_{k+1}(s) \leftarrow \max_{a \in \mathcal{A}(s)} \sum_{s'} p(s' \mid s, a) \left[r(s, a, s') + \gamma V_k(s') \right]$$

- What is complexity of one iteration - over all S states?
- When does the iteration stop?
- When the does the policy converge?
- Can we compute the policy directly?

What is wrong with the Value iteration?

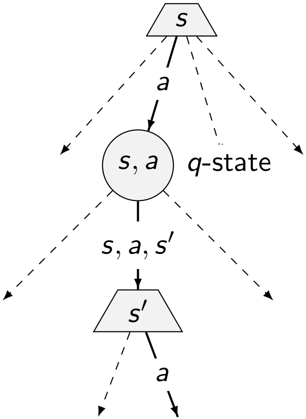
$$V_{k+1}(s) \leftarrow \max_{a \in \mathcal{A}(s)} \sum_{s'} p(s' \mid s, a) \left[r(s, a, s') + \gamma V_k(s') \right]$$

- What is complexity of one iteration - over all S states?
- When does the iteration stop?
- When the does the policy converge?
- Can we compute the policy directly?

Policy evaluation

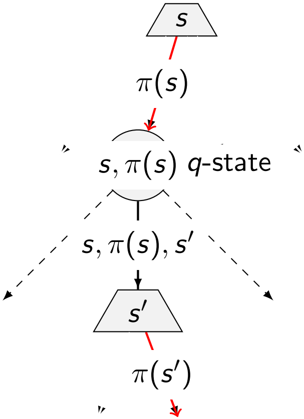
- Assume $\pi(s)$ given.
- How to evaluate (compare)?

Fixed policy, do what π says



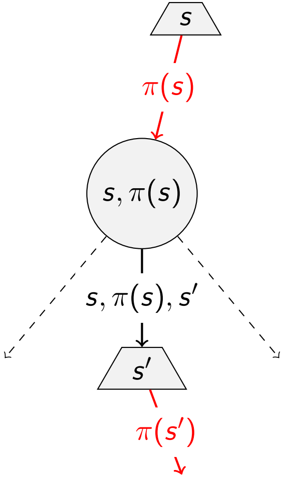
- Expectimax trees “max” over all actions . . .
- Fixed π for each state → no “max” operator!

Fixed policy, do what π says



- Expectimax trees "max" over all actions ...
- Fixed π for each state → no "max" operator!

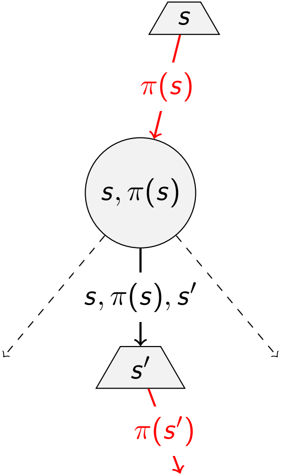
State values under a fixed policy



- Expectimax trees "max" over all actions . . .
- Fixed π for each state → no "max" operator!

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

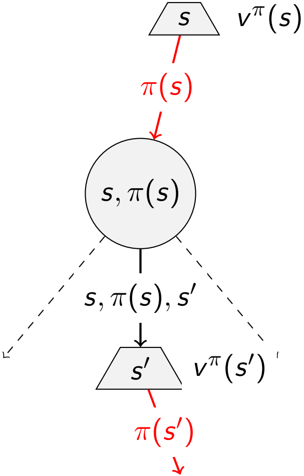
State values under a fixed policy



- Expectimax trees “max” over all actions . . .
- Fixed π for each state → no “max” operator!

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

State values under a fixed policy

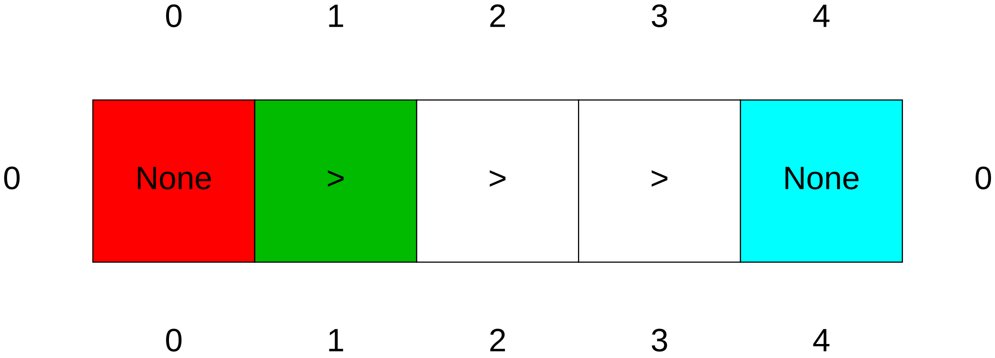


- Expectimax trees “max” over all actions . . .
- Fixed π for each state → no “max” operator!

$$v^\pi(s) = \sum_{s'} p(s' \mid s, \pi(s)) \big[r(s, \pi(s), s') + \gamma v^\pi(s') \big]$$

How to compute $v^\pi(s)$?

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



	0	1	2	3	4	
0	None	>	>	>	None	0
	0	1	2	3	4	

Policy iteration

- Start with a random policy.
- Step 1: Evaluate it.
- Step 2: Improve it.
- Repeat steps until policy converges.

Policy iteration

- Policy π evaluation. Solve equations or iterate until convergence.

$$V_{k+1}^{\pi_i}(s) \leftarrow \sum_{s'} p(s' \mid s, \pi(s)) \left[r(s, \pi(s), s') + \gamma V_k^{\pi_i}(s') \right]$$

- Policy improvement. Look-ahead and keep optimality. Policy extraction from fixed values.

$$\pi_{i+1}(s) = \arg\max_{a \in \mathcal{A}(s)} \sum_{s'} p(s' \mid s, a) \left[r(s, a, s') + \gamma V_k^{\pi_i}(s') \right]$$

Policy iteration algorithm

```
function POLICY-ITERATION(env) returns: policy $\pi$
    input: env - MDP problem
    $\pi(s) \leftarrow$ random $a \in A(s)$ in all states
    $V(s) \leftarrow 0$ in all states
    repeat                                ▷ iterate values until no change in policy
        $V \leftarrow$ POLICY-EVALUATION($\pi$, $V$, env)
        unchanged $\leftarrow$ True
        for each state $s$ in $S$ do
            if $\max_{a \in A(s)} \sum_{s'} P(s'|a,s)V(s') > \sum_{s'} P(s'|s,\pi(s))V(s')$ then
                $\pi(s) \leftarrow \arg\max_{a \in A(s)} \sum_{s'} P(s'|a,s)V(s')$
                unchanged $\leftarrow$ False
            end if
        end for
    until unchanged
end function
```

Policy iteration algorithm

```
function POLICY-ITERATION(env) returns: policy $\pi$
    input: env - MDP problem
    $\pi(s) \leftarrow$ random $a \in A(s)$ in all states
    $V(s) \leftarrow 0$ in all states
    repeat                                                  ▷ iterate values until no change in policy
        $V \leftarrow$ POLICY-EVALUATION($\pi$, $V$, env)
        unchanged $\leftarrow$ True
        for each state $s$ in $S$ do
            if $\max_{a \in A(s)} \sum_{s'} P(s'|a,s)V(s') > \sum_{s'} P(s'|s,\pi(s))V(s')$ then
                $\pi(s) \leftarrow \arg\max_{a \in A(s)} \sum_{s'} P(s'|a,s)V(s')$
                unchanged $\leftarrow$ False
            end if
        end for
    until unchanged
end function
```

Policy iteration algorithm

```
function POLICY-ITERATION(env) returns: policy $\pi$
    input: env - MDP problem
    $\pi(s) \leftarrow$ random $a \in A(s)$ in all states
    $V(s) \leftarrow 0$ in all states
    repeat                                    ▷ iterate values until no change in policy
        $V \leftarrow$ POLICY-EVALUATION($\pi$, $V$, env)
        unchanged $\leftarrow$ True
        for each state $s$ in $S$ do
            if $\max_{a \in A(s)} \sum_{s'} P(s'|a,s)V(s') > \sum_{s'} P(s'|s,\pi(s))V(s')$ then
                $\pi(s) \leftarrow \arg\max_{a \in A(s)} \sum_{s'} P(s'|a,s)V(s')$
                unchanged $\leftarrow$ False
            end if
        end for
    until unchanged
end function
```

Policy iteration algorithm

```
function POLICY-ITERATION(env) returns: policy $\pi$
    input: env - MDP problem
    $\pi(s) \leftarrow$ random $a \in A(s)$ in all states
    $V(s) \leftarrow 0$ in all states
    repeat                                          ▷ iterate values until no change in policy
        $V \leftarrow$ POLICY-EVALUATION($\pi$, $V$, env)
        unchanged $\leftarrow$ True
        for each state $s$ in $S$ do
            if $\max_{a \in A(s)} \sum_{s'} P(s'|a,s)V(s') > \sum_{s'} P(s'|s,\pi(s))V(s')$ then
                $\pi(s) \leftarrow \arg\max_{a \in A(s)} \sum_{s'} P(s'|a,s)V(s')$
                unchanged $\leftarrow$ False
            end if
        end for
    until unchanged
end function
```

Policy iteration algorithm

```
function POLICY-ITERATION(env) returns: policy $\pi$
    input: env - MDP problem
    $\pi(s) \leftarrow$ random $a \in A(s)$ in all states
    $V(s) \leftarrow 0$ in all states
    repeat                                        ▷ iterate values until no change in policy
        $V \leftarrow$ POLICY-EVALUATION($\pi$, $V$, env)
        unchanged $\leftarrow$ True
        for each state $s$ in $S$ do
            if $\max_{a \in A(s)} \sum_{s'} P(s'|a,s)V(s') > \sum_{s'} P(s'|s,\pi(s))V(s')$ then
                $\pi(s) \leftarrow \arg\max_{a \in A(s)} \sum_{s'} P(s'|a,s)V(s')$
                unchanged $\leftarrow$ False
            end if
        end for
    until unchanged
end function
```

Policy vs. Value iteration

- Value iteration.
  - Iteration updates values and policy. (policy only implicitly – can be extracted from values)
  - No track of policy.
- Policy iteration.
  - Update of values is faster – only one action per state.
  - New policy from values (slower).
  - New policy is better or done.
- Both methods belong to Dynamic programming realm.

Policy vs. Value iteration

- Value iteration.
  - Iteration updates values and policy. (policy only implicitly – can be extracted from values)
  - No track of policy.
- Policy iteration.
  - Update of values is faster – only one action per state.
  - New policy from values (slower).
  - New policy is better or done.
- Both methods belong to Dynamic programming realm.

Policy vs. Value iteration

- Value iteration.
  - Iteration updates values and policy. (policy only implicitly – can be extracted from values)
  - No track of policy.
- Policy iteration.
  - Update of values is faster – only one action per state.
  - New policy from values (slower).
  - New policy is better or done.
- Both methods belong to Dynamic programming realm.

References

Further reading: Chapter 17 of [1] however, policy iteration is quite compact there. More detailed discussion can be found in chapter Dynamic programming in [2] with slightly different notation, though. This lecture has been also greatly inspired by the 9th lecture of CS 188 at http://ai.berkeley.edu as it convincingly motivates policy search and offers an alternative convergence proof of the value iteration method.

[1] Stuart Russell and Peter Norvig.
Artificial Intelligence: A Modern Approach.
Prentice Hall, 3rd edition, 2010.
http://aima.cs.berkeley.edu/.

[2] Richard S. Sutton and Andrew G. Barto.
Reinforcement Learning; an Introduction.
MIT Press, 2nd edition, 2018.
http://www.incompleteideas.net/book/the-book-2nd.html.

Bandits