Sequential decisions under uncertainty
Policy iteration

Tomáš Svoboda & Matej Hoffmann

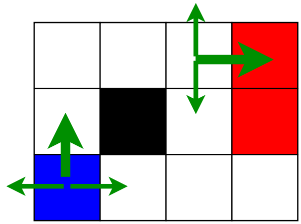
Vision for Robots and Autonomous Systems, Center for Machine Perception
Department of Cybernetics
Faculty of Electrical Engineering, Czech Technical University in Prague

March 31, 2021

Notes

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



Notes

This is a recap slide, we already know this from the last lecture.

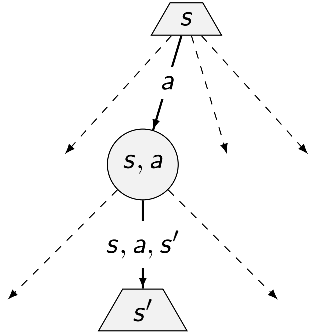
MDPs recap

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 $\pi(s) : \mathcal{S} \to \mathcal{A}$
- 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), $q(s,a)$



Notes

Q-values – like values but given that I have commited to do action a from state s.

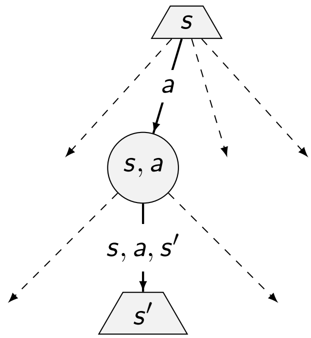
MDPs recap

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), $q(s,a)$



Notes

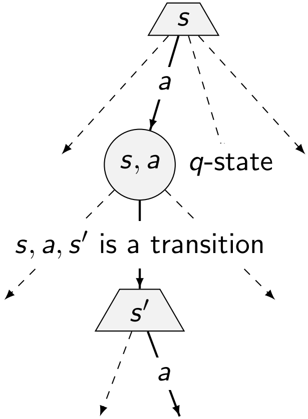
Q-values – like values but given that I have commited to do action a from state s.

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.

Notes

Remember: Discounted return G_t
Returns are successive steps related to each other

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

$G_t \doteq \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$ including the possibility that $T = \infty$ or $\gamma = 1$, but not both.

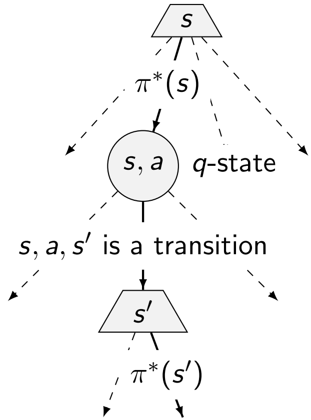
Optimal quantities

- 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.



Notes

Remember: Discounted return G_t
Returns are successive steps related to each other

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

$G_t \doteq \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$ including the possibility that $T = \infty$ or $\gamma = 1$, but not both.

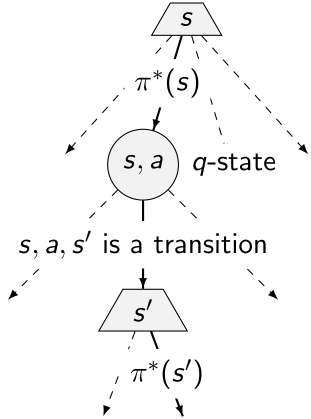
Optimal quantities

- 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.



Notes

Remember: Discounted return G_t
Returns are successive steps related to each other

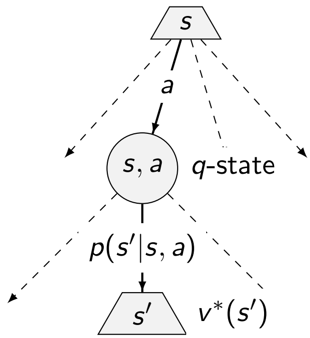
$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \\ &= R_{t+1} + \gamma(R_{t+2} + \gamma^1 R_{t+3} + \gamma^2 R_{t+4} + \cdots) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned}$$

$G_t \doteq \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$ including the possibility that $T = \infty$ or $\gamma = 1$, but not both.

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:

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

Notes

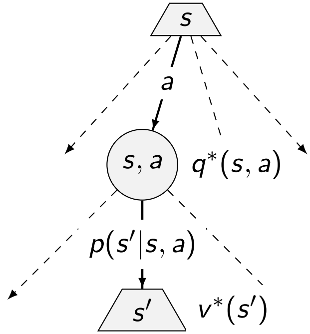
v^* and q^*

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:

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



Notes

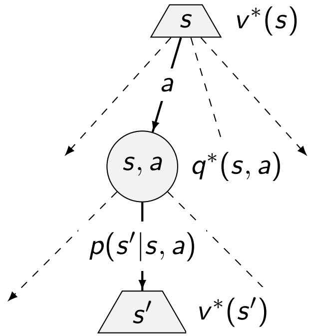
v^* and q^*

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:

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



Notes

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

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

Notes

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

Notes

	0	1	2	3	4	
0	-10.00	-1.00	-1.00	9.00	10.00	0
	0	1	2	3	4	

Live: Calculate q-values (it's easy, robot is deterministic) and then choose max for every state.

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

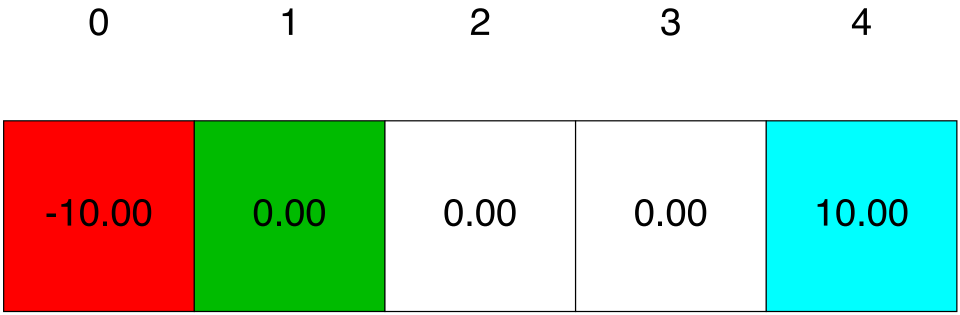
C: $V_2^* = [-2, 8, 9]^\top$

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

Notes

	0	1	2	3	4	
0	-10.00	-2.00	8.00	9.00	10.00	0
	0	1	2	3	4	

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



Pick the option which is *wrong*:

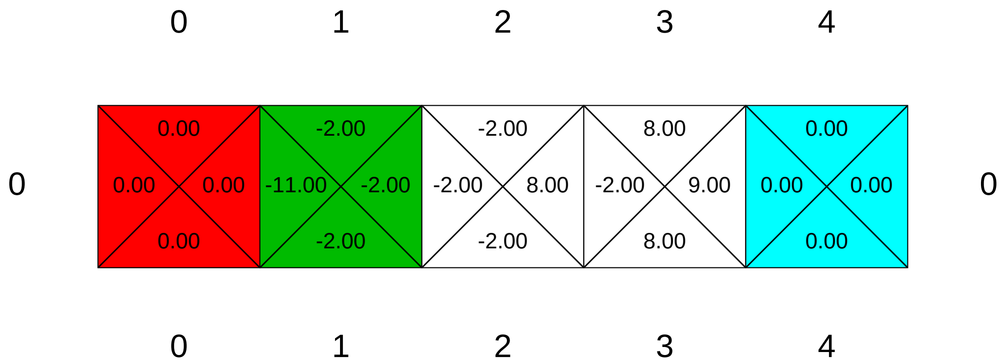
A: $q^*(1, \uparrow) = -2$

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

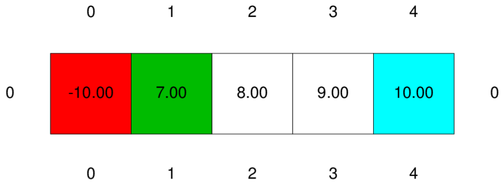
C: $q^*(2, \rightarrow) = -2$

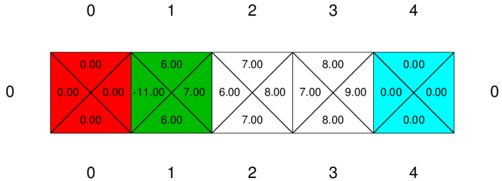
D: $q^*(3, \leftarrow) = -2$

Notes



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





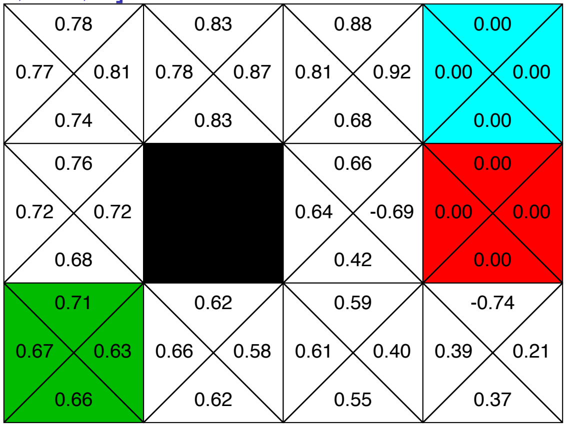
$$
\begin{aligned}
q^*(s,a) &= \sum_{s'} p(s'|a,s)\left[r(s,a,s') + \gamma\, v^*(s')\right] \\
v^*(s) &= \max_a q^*(s,a)
\end{aligned}
$$

Notes

Deterministic robot/agent with four possible actions.

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

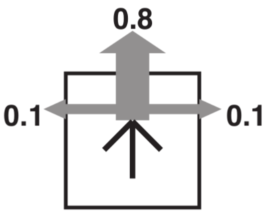


$$q^*(s,a) = \sum_{s'} p(s'|a,s)\left[r(s,a,s') + \gamma\, v^*(s')\right]$$

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

Notes

This is the $R = -0.04$ for nonterminal states maze ([1] Fig. 17.3).



, $\gamma = 1$

Note that the Value of a state takes into account a number of things:

- the policy – which action will chosen in s
- the fact that the goal may be far away and
  - there will be a number of living penalties incured before reaching it
  - the final reward may be discounted (not the case here)
- the transition probabilities

Q-values - useful for choosing the best action – getting the policy.

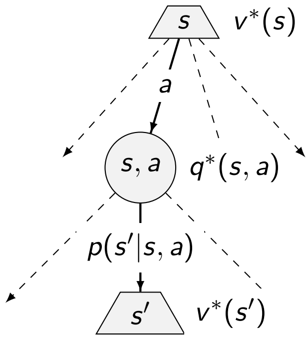
Value iration

- 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.

Notes

Bellman equations:

1. Take correct first action (1 ply of Expectimax)
2. Keep being optimal (recursion $v^*(s')$)

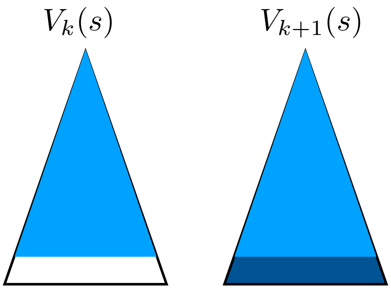
Recall that we may simplify equations by marginalizing rewards if all $r(s,a,s')$ are the same.

$$r(s) = \sum_{s'} p(s'|a,s) r(s,a,s')$$

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 $k+1$



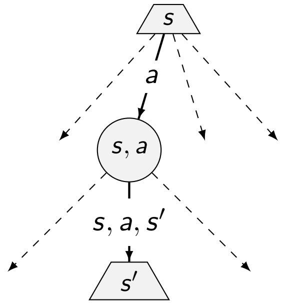
Notes

- Bottom (last) layer, zeros for $V_k(s)$, true rewards for $V_{k+1}(s)$
- Last layer $\langle R_{min}, R_{max} \rangle$
- But the last layer is γ^k discounted ...
- hence, V_k and V_{k+1} are no more than $\gamma^k \max|R|$ apart.
- The k increases, the values converge.

From Values to Policy

Notes

Policy extraction - computing actions from Values



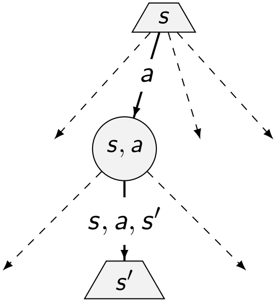
- Assume we have $v^*(s)$
- What is the optimal action?
- We need a one-step expectimax:

	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	

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

Notes

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	

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

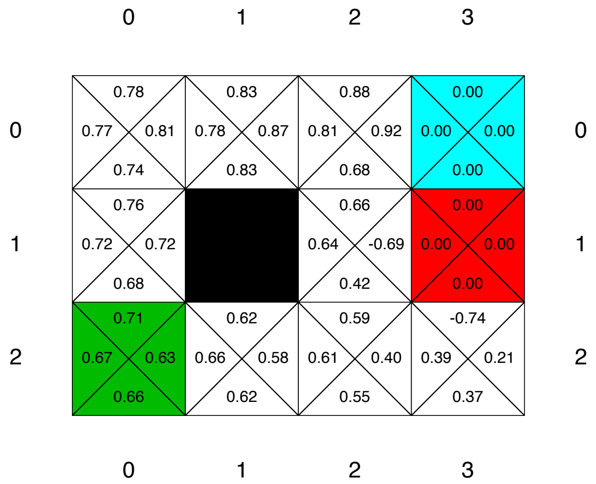
Notes

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.



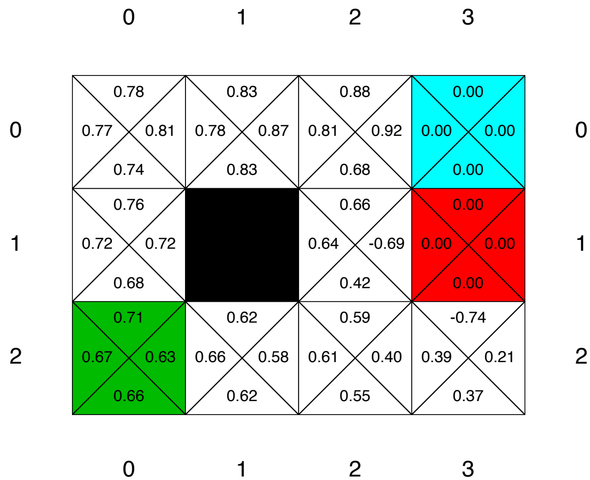
Notes

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.



Notes

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?

Notes

- Complexity: $O(AS^2)$ per iteration For every state (LHS), there can be up to $\sharp S$ also on RHS – if every other state was reachable from the current state. In addition, all actions from every state need to be considered.
- Iteration stops when max difference between V_k and V_{k+1} becomes small enough.
- However, (change in) policy depends on change in $\max(\mathcal{A})$.
- $\max(\mathcal{A})$ **does not** change often.
- Policy often converges long before the values.

Notes for teacher: Run "AIMA Fig. 17.2 / 17.3 demo" with $R = -0.04$
mdp_agents.py, value iteration with $eps = 0.03, discount = 0.999999$

- verbosity=SHOW.UTILS
- verbosity=SHOW.QVALS - max does not change often...

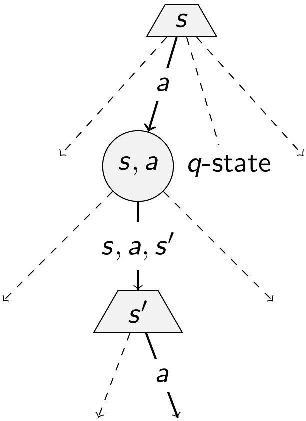
Policy evaluation

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

Notes

Remember last week's quiz?

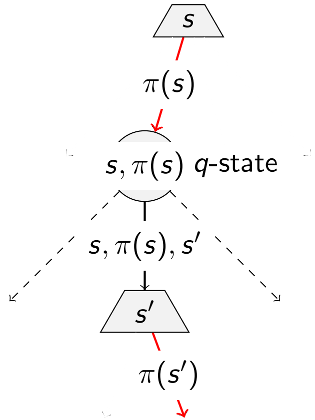
Fixed policy, do what π says



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

Notes

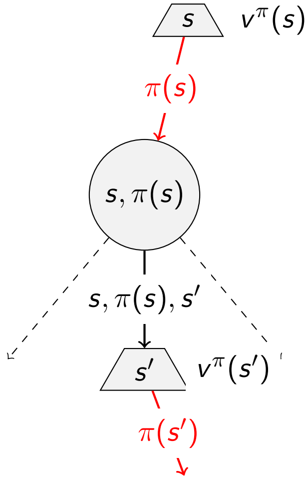
Fixed policy, do what π says



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

Notes

State values under a fixed policy



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

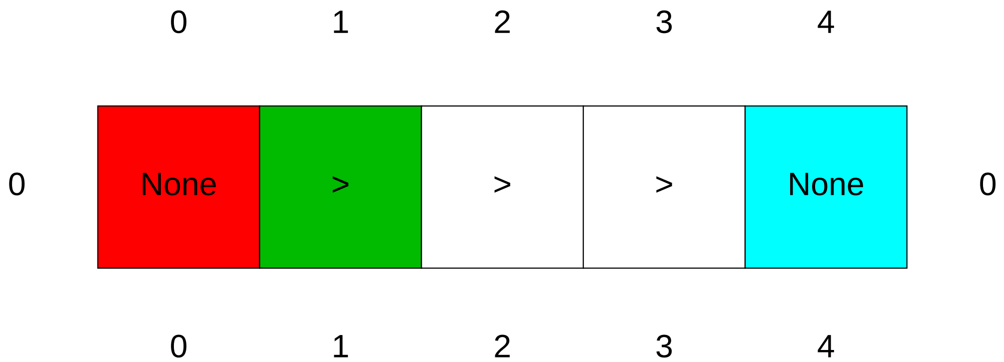
$$v^\pi(s) = \sum_{s'} p(s' \mid s, \pi(s)) \left[r(s, \pi(s), s') + \gamma v^\pi(s') \right]$$

Notes

Recall that $v^\pi(s)$ quantity contains all the future – expected discounted sum of rewards – executing policy from the state s onwards.

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

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



Notes

- by iteration
- solving set of equations

Policy iteration

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

Notes

Policy iteration

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

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

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

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

Notes

A few demo runs of `mdp_agents.py`.
Test of understanding: Policy evaluation: Repeats until convergence. Hmm, just like for Value iteration. So how come we are saving time? Because we do not iterate (and max) over the actions. We can solve directly (system of linear equations), even though for large problems in practice, iterative methods are still used.

Policy improvement: Note that the value is taken from "old policy" on RHS.

Policy iteration algorithm

function POLICY-ITERATION(env) **returns:** policy π
  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(π, 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

Notes

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.

Notes

Complexity (of one iteration step):
Value iteration: $O(S^2 * A)$
For every state (LHS), there can be up to $\sharp S$ also on RHS – if every other state was reachable from the current state.
In addition, all actions from every state need to be considered.
$Max(A)$ **does not** change often.
Policy often converges long before the values.
Policy evaluation: $O(S^3)$ (after AIMA, pg. 657)
The Bellman equations are *linear* because the max operator is gone.
For $\sharp S$ states, we have $\sharp S$ equations, which can be solved exactly in time $O(S^3)$ using standard linear algebra methods.
For small state spaces - ok.
For large state spaces - may be prohibitive $\rightarrow$ *modified policy iteration* with only a certain number of simplified Bellman update.

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.

Notes

Complexity (of one iteration step):
Value iteration: $O(S^2 * A)$
For every state (LHS), there can be up to $\sharp S$ also on RHS – if every other state was reachable from the current state.
In addition, all actions from every state need to be considered.
Max(A) **does not** change often.
Policy often converges long before the values.
Policy evaluation: $O(S^3)$ (after AIMA, pg. 657)
The Bellman equations are *linear* because the max operator is gone.
For $\sharp S$ states, we have $\sharp S$ equations, which can be solved exactly in time $O(S^3)$ using standard linear algebra methods.
For small state spaces - ok.

For large state spaces - may be prohibitive → *modified policy iteration* with only a certain number of simplified Bellman update.

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.

Notes

Complexity (of one iteration step):
Value iteration: $O(S^2 * A)$
For every state (LHS), there can be up to $\sharp S$ also on RHS – if every other state was reachable from the current state.
In addition, all actions from every state need to be considered.
$Max(A)$ **does not** change often.
Policy often converges long before the values.
Policy evaluation: $O(S^3)$ (after AIMA, pg. 657)
The Bellman equations are *linear* because the max operator is gone.
For $\sharp S$ states, we have $\sharp S$ equations, which can be solved exactly in time $O(S^3)$ using standard linear algebra methods.
For small state spaces - ok.
For large state spaces - may be prohibitive $\rightarrow$ *modified policy iteration* with only a certain number of simplified Bellman update.

References

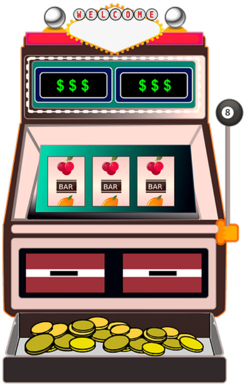
Further reading: Chapter 17 of [1] however, policy iteration is quite compact there. More detailed discussion can be found in chapter Dynamic programming in [2] with slightly different notation, though. This lecture has been also greatly inspired by the 9th lecture of CS 188 at http://ai.berkeley.edu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

Notes

Bandits



Notes