Machine Learning (CS 181): 20. Markov Decision Processes

David C. Parkes and Sasha Rush

Spring 2017

Contents

Overview

Supervised learning

$$D = \{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_n, \mathbf{y}_n)\}$$

Neural networks, Naive Bayes, SVMs, random forests, linear regression, ...

Unsupervised learning

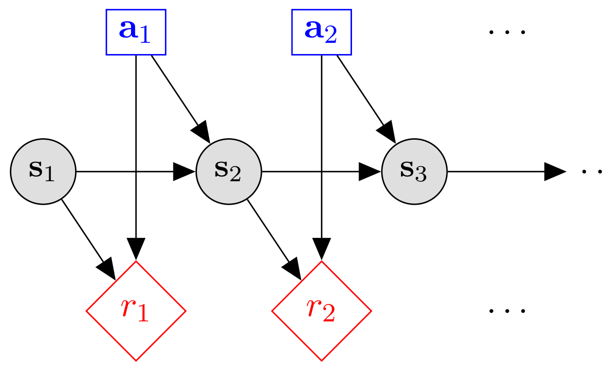
$$D = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$$

K-means, HAC, Bayesian Networks, topic models, Gaussian mixture models, HMMs...

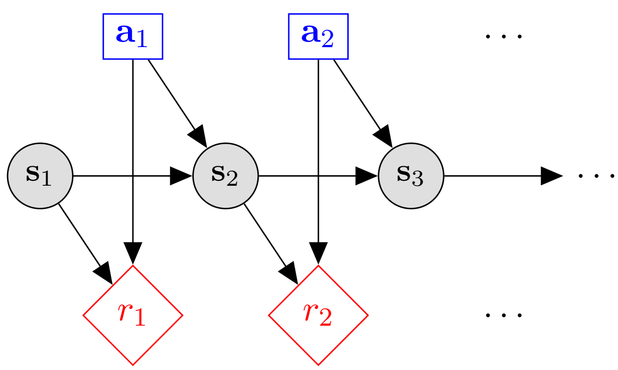
Learning to act:

embodied agents

$$D = (s_1, a_1, r_1, s_2, a_2, r_2, \ldots)$$



Markov Decision Process

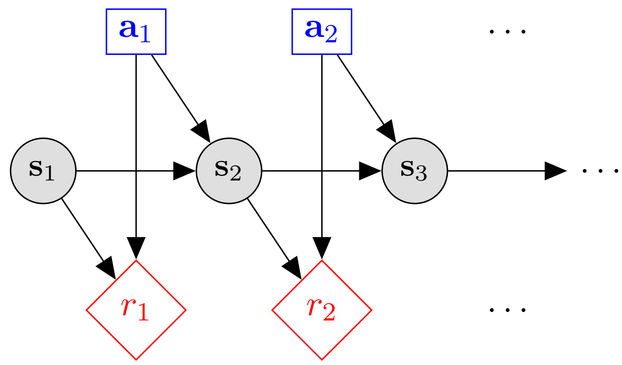


An MDP is specified by (S, A, r, p):

- $S = \{1, \ldots, |S|\}$ states
- $A = \{1, \ldots, |A|\}$ actions
- reward function $r(s, a) \in \mathbb{R}$, for all states s, all actions a
- transition model $p(s' \mid s, a)$, for all states s, actions a, next states s'

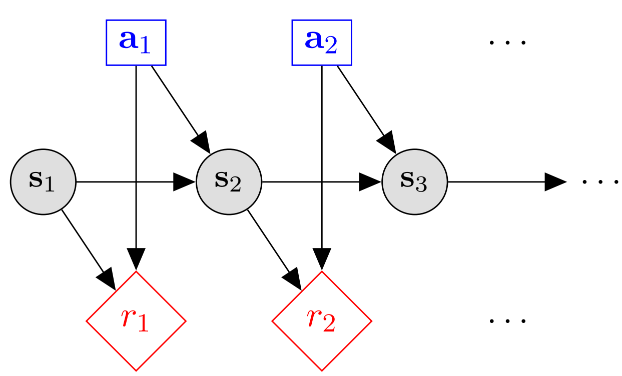
A policy π is a mapping from states to actions. Want to find 'rewarding' policies..

Application 1: Robots



- States: physical location, objects in environment
- Actions: move, pick-up, drop, ...
- Reward: +1 if pick up dirty clothes, -1 if break dish, ...
- Transition model: describe actuators and uncertain environment

Application 2: Game of Go



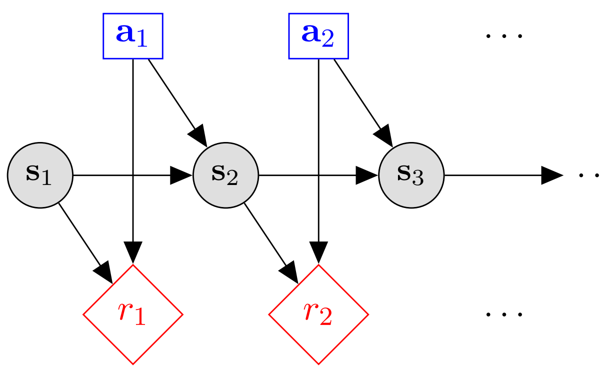
- States: board position
- Actions: move a piece
- Reward: +1 if win the game, 0 if draw, -1 if lose the game
- Transition model: rules of game, response of other player

AlphaGo vs. Lee Sedol

- AlphaGo (DeepMind) defeated Lee Sedol, 4-1 in March 2016, the top Go player in the world
- AlphaGo combines Monte-Carlo tree search with deep neural nets (trained by supervised learning), with reinforcement learning.
- Learns both a 'policy network' (which action to play in which state) and a 'value network' (estimate of value of an action under self-play).

'Mastering the game of Go with deep neural networks and tree search', Silver et al., Nature **529**:484–582 (2016)

Application 3: Customer Service Agent



- States: summary of conversation so far
- Actions: words to utter
- Reward: +1 if solve caller's problem, -1 if need to go to human, -10 if caller hangs up angry
- Transition model: effect of words on state, next words or action from caller.

Working with MDPs

An MDP is a general probabilistic framework, and can be utilized in many different scenarios.

- Planning:
  - Full access to the MDP, compute an optimal policy.
  - "How do I act in a known world?"
- Policy Evaluation:
  - Full access to the MDP, compute the 'value' of a fixed policy.
  - "How will this plan perform under uncertainty?"
- Reinforcement Learning (next lecture):
  - Limited access to the MDP.
  - "Can I learn to act in an uncertain world?"

Different Objective Criteria

- Sequence of $s_1, a_1, r_1, s_2, a_2, r_2, \ldots$; discrete time t
- Finite horizon, $T \geq 1$ steps

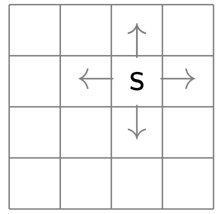
$$\text{utility} = \sum_{t=1}^{T} r(s_t, a_t)$$

- Infinite horizon, discount factor $\gamma \in (0, 1]$

$$\text{utility} = r(s_1, a_1) + \gamma \cdot r(s_2, a_2) + \gamma^2 \cdot r(s_3, a_3) + \ldots$$

(Long-run average, $\lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{\infty} r(s_t, a_t)$ is another objective criterion.)

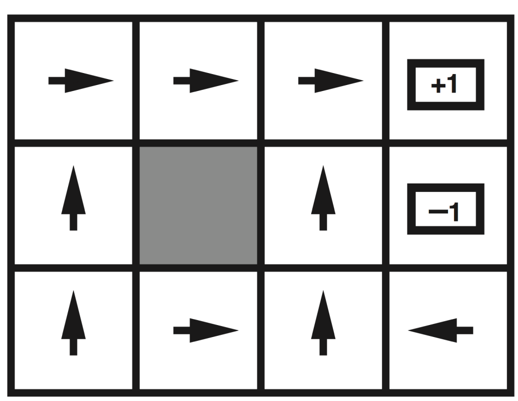
Running Illustration: MDP on Gridworld



S	Location of the grid (x_1, x_2)
A	Local movements $\leftarrow, \rightarrow, \uparrow, \downarrow$
$r : S \times A \mapsto \mathbb{R}$	Reward function, e.g. make it to goal
$p(s' \mid s, a)$	Transition model, e.g deterministic or slippages

Example Gridworld (perfect actuator)

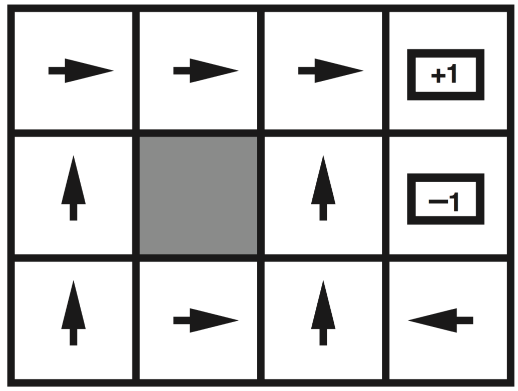
Optimal policy:



- $r(s, a) = -0.04$ for all states, actions except $(4, 2), (4, 3)$
- Bounce off obstacles
- Stop when get to $(4, 2), (4, 3)$ ('episodic')
- Perfect actuator

Gridworld Example (perfect actuator)

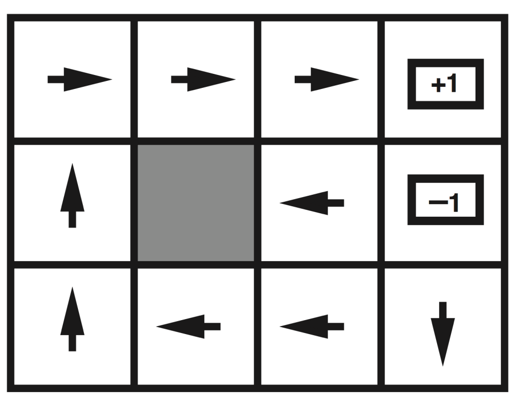
Optimal policy:



- $r(s,a) = -0.04$ for all states, actions except $(4,2),(4,3)$
- Bounce off obstacles
- Stop when get to $(4,2),(4,3)$ ('episodic')
- ~~Perfect actuator~~ imperfect actuator (prob. 0.1 in direction 90° left, prob. 0.1 in direction 90° right)?

Example (imperfect actuator)

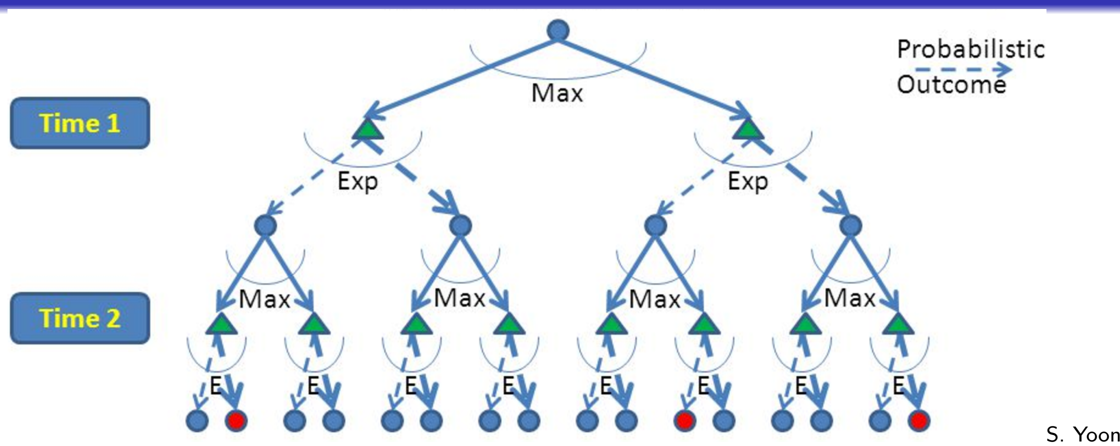
In this case, optimal policy becomes:



- $r(s,a) = -0.04$ for all states, actions except $(4,2),(4,3)$
- Bounce off obstacles
- Stop when get to $(4,2),(4,3)$ ('episodic')
- ~~Perfect actuator~~ imperfect actuator (prob. 0.1 in direction 90° left, prob. 0.1 in direction 90° right)?

Contents

Warm-up: Expectimax



- Build out a look-ahead tree to the decision horizon; $\max$ over actions, exp over next states.
- Solve from the leaves, backing-up the expectimax values.
- Problem: computation is exponential in horizon.
- May expand the same subtree multiple times. (e.g., s_1, a_1 and s_1, a_2 may lead to same state.)

Finite-Horizon Planning: Value iteration

A dynamic programming approach. Let $V^*_{(t)}(s)$ denote the total value from state s under optimal policy with t-steps-to-go, $\pi^*_{(t)}(s)$ the optimal action with t-periods-to-go.

Base case (for all states s):

$$V^*_{(1)}(s) = \max_a \; r(s,a).$$

Inductive case (for all states s, time-to-go $t = 2, \ldots, T$):

$$V^*_{(t)}(s) = \max_{a \in A} \left[r(s,a) + \sum_{s' \in S} p(s' \mid a, s) V^*_{(t-1)}(s') \right]$$

Work back from last period to present. Can read-off the optimal policy.

Let $L = \max\#$ states reachable from any state under any action.

Computational complexity is $O(|A| \cdot |S| \cdot L \cdot T)$.

Example: Value iteration

$$V^*_{(t)}(s) = \max_{a \in A}(r(s,a) + \sum_{s' \in S} p(s' \mid a, s) V^*_{(t-1)}(s'))$$

Simple 5-state, 2-action gridworld. Stop when get to states 1 or 5.

$r(s,a)$	10	-1	-1	-1	5	optimal policy?
$V^*_{(1)}(s)$	10	-1	-1	-1	5	
$V^*_{(2)}(s)$	10	9	-2	4	5	e.g., $9 = \max(-1+10, -1-1)$, $-2 = \max(-1-1, -1-1)$
$V^*_{(3)}(s)$	10	9	8	4	5	e.g., $8 = \max(-1+9, -1+4)$
$V^*_{(4)}(s)$	10	9	8	7	5	e.g., $7 = \max(-1+8, -1+5)$

Contents

MDP Value function

Consider an infinite time horizon, and a stationary and deterministic policy $\pi(s) \in A$.

This is without loss of generality (for discounted objective criterion).

Definition (MDP value function)

The MDP value function of a policy π from state s is

$$V^{\pi}(s) = \mathbf{E}\left[\sum_{t=1}^{\infty} \gamma^{t-1} r(s_t, \pi(s_t))\right]$$

where $s_1 \triangleq s$, and $s_{t+1} \sim p(s' \mid s_t, \pi(s_t))$.

Policy Evaluation

We can expand this MDP value function as:

$$V^{\pi}(s) = \underbrace{r(s, \pi(s))}_{\text{reward now}} + \underbrace{\gamma \sum_{s' \in S} p(s' \mid s, \pi(s)) V^{\pi}(s)}_{\text{expected, discounted future reward}} \tag{1}$$

Definition (Policy evaluation)

For a given policy π, infinite time horizon, and discounting, evaluate the MDP value function.

We can solve system of linear equations (1) in time $O(|S|^3)$ via Gaussian elimination.

Bellman equations

The planning problem for an MDP is:

$$\pi^* \in \arg\max_{\pi} V^{\pi}(s).$$

(exists a solution that is optimal for every state s).

Definition (Bellman equations)

For an optimal policy π^*, we have

$$V^*(s) = \max_{a \in A} \left[r(s, a) + \gamma \sum_{s' \in S} p(s' \mid s, a) V^*(s') \right], \quad \forall s \tag{2}$$

This system of (non-linear) equations capture the principle of optimality. The value of an optimal policy = value of doing the right thing now, considering the value that comes from optimal 'continuation.'

Value iteration

The Bellman equations suggest the following approach to planning:

- Initialize: $V(s) = 0$, for all states s
- Update step ('Bellman operator'):

$$V'(s) \leftarrow \max_{a \in A} \left[r(s,a) + \gamma \sum_{s' \in S} p(s' \,|\, s,a) V(s') \right], \quad \forall s$$

  update value function using one-step look-ahead

- $V \leftarrow V'$

Continue until converge, find the fixpoint. Can then read-off the optimal policy via (2).

Computation $O(|S| \cdot |A| \cdot L)$ per iteration, where $L = \max\#$ states reachable from any state under any action.

Convergence of Value Iteration

- Contraction property for update $x' \leftarrow f(x)$:

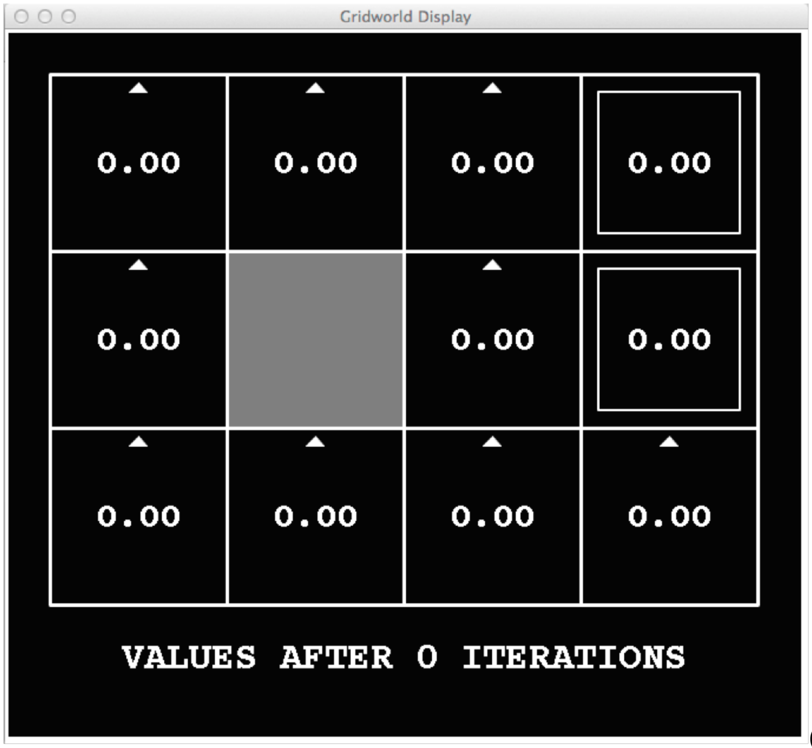
$$||f(x) - f(y)|| < ||x - y||, \qquad \text{for all } x \neq y$$

  - e.g., $x' \leftarrow f(x) = x/2$, fixpoint $x^* = f(x^*) \Leftrightarrow x^* = 0$
  - contraction: $(2, 8), (1, 4), (1/2, 2), \ldots$

- By contraction property:
  - f has a unique fixpoint, else $||f(x^*) - f(y^*)|| = ||x^* - y^*||$ (violation of contraction)
  - update converges to the fixpoint, consider $x \neq x^*$, $||f(x) - x^*|| = ||f(x) - f(x^*)|| < ||x - x^*||$.

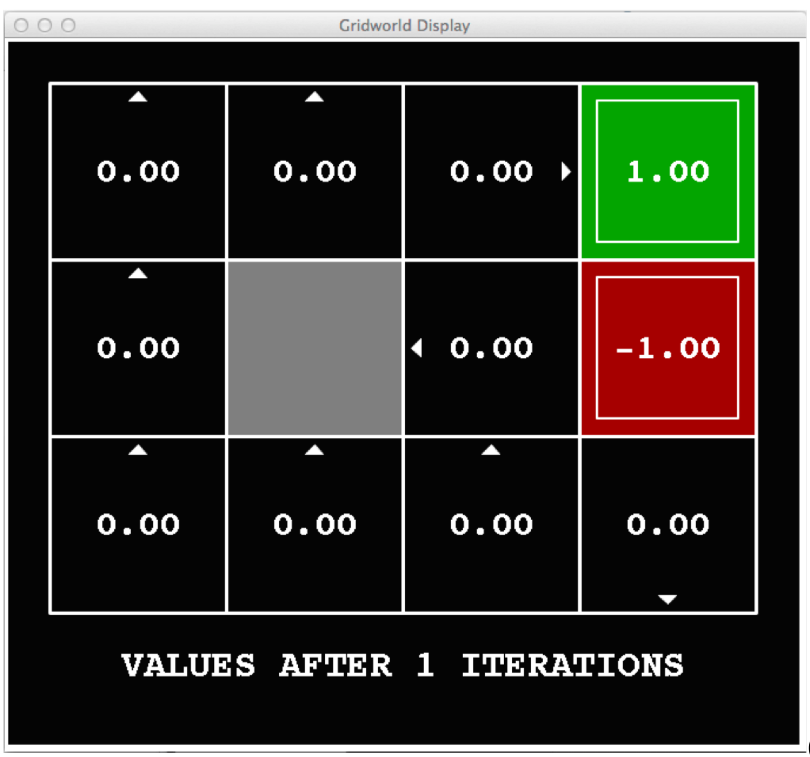
- The Bellman operator is a contraction when discount factor $\gamma < 1$, and where $||\mathbf{V}|| = \max_s |V(s)|$ ('max-norm')

Example: Value iteration in GridWorld



(D. Klein and P. Abbeel)

Example: Value iteration in GridWorld

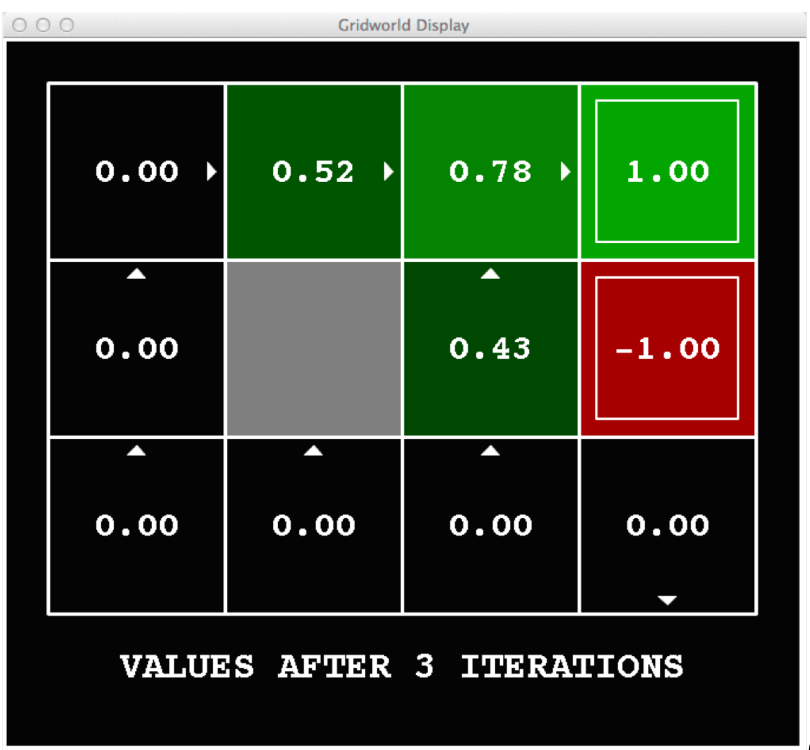


(D. Klein and P. Abbeel)

Example: Value iteration in GridWorld

(D. Klein and P. Abbeel)

Example: Value iteration in GridWorld



(D. Klein and P. Abbeel)

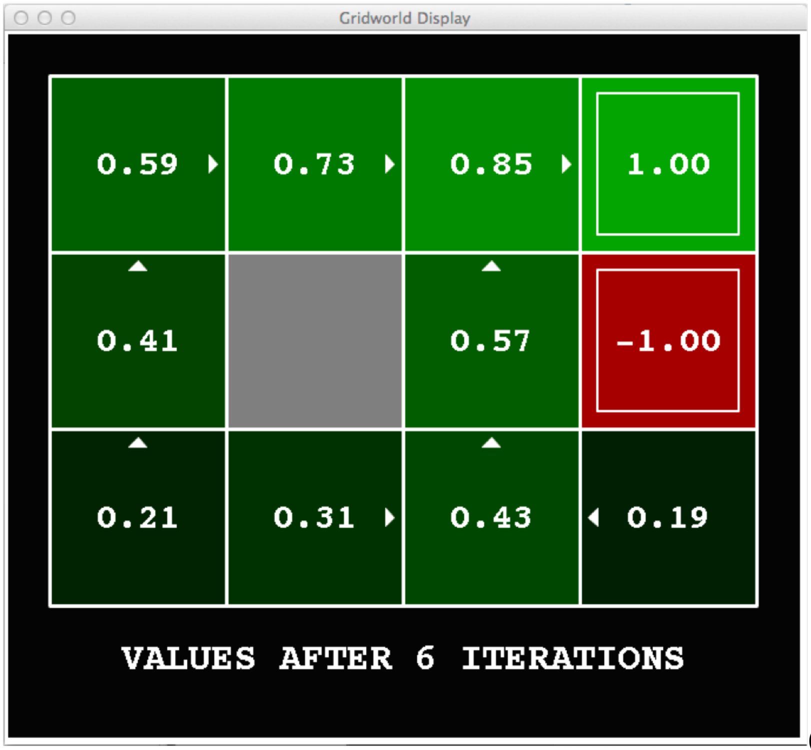
Example: Value iteration in GridWorld

(D. Klein and P. Abbeel)

Example: Value iteration in GridWorld

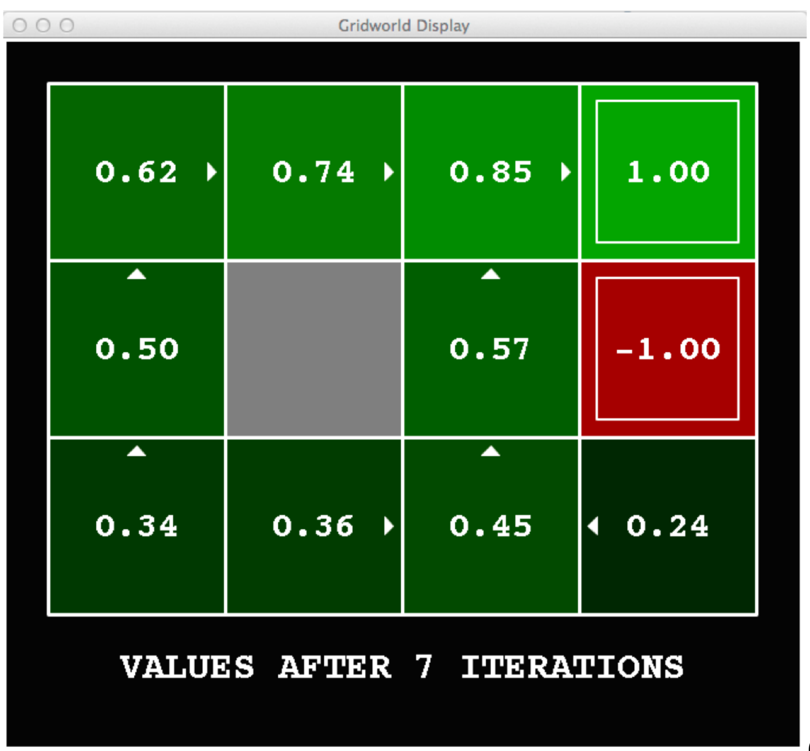
(D. Klein and P. Abbeel)

Example: Value iteration in GridWorld



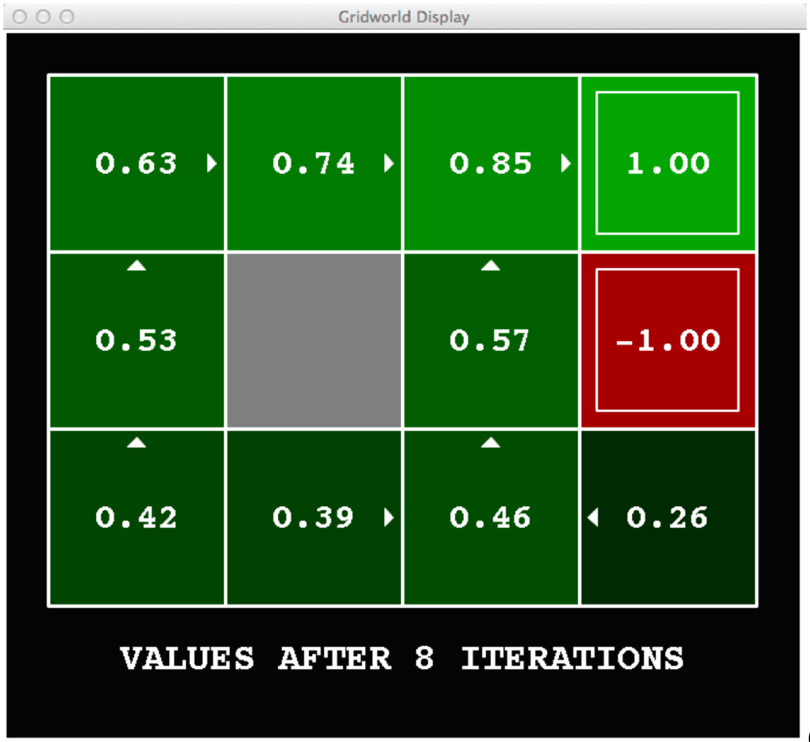
(D. Klein and P. Abbeel)

Example: Value iteration in GridWorld



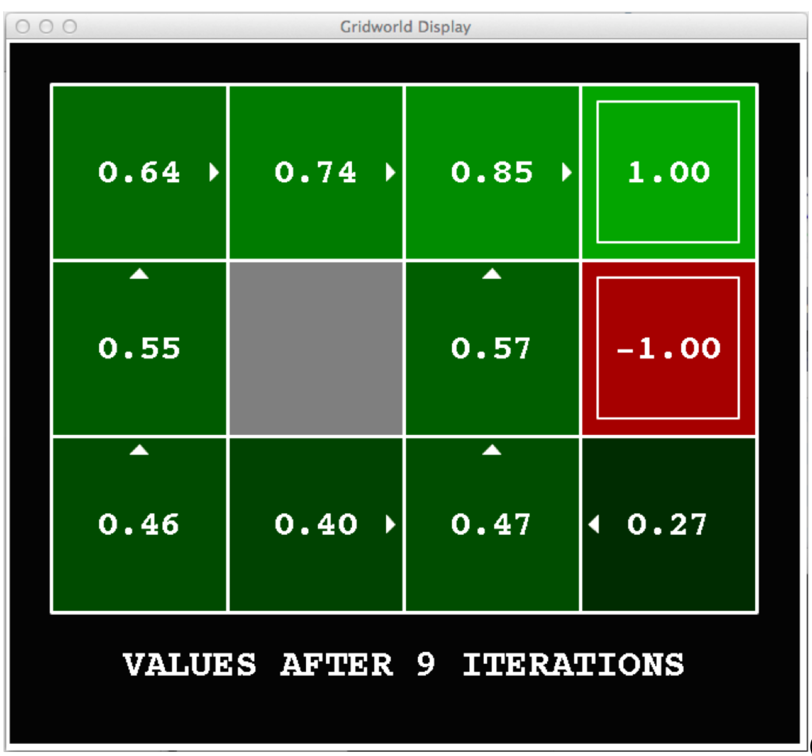
(D. Klein and P. Abbeel)

Example: Value iteration in GridWorld



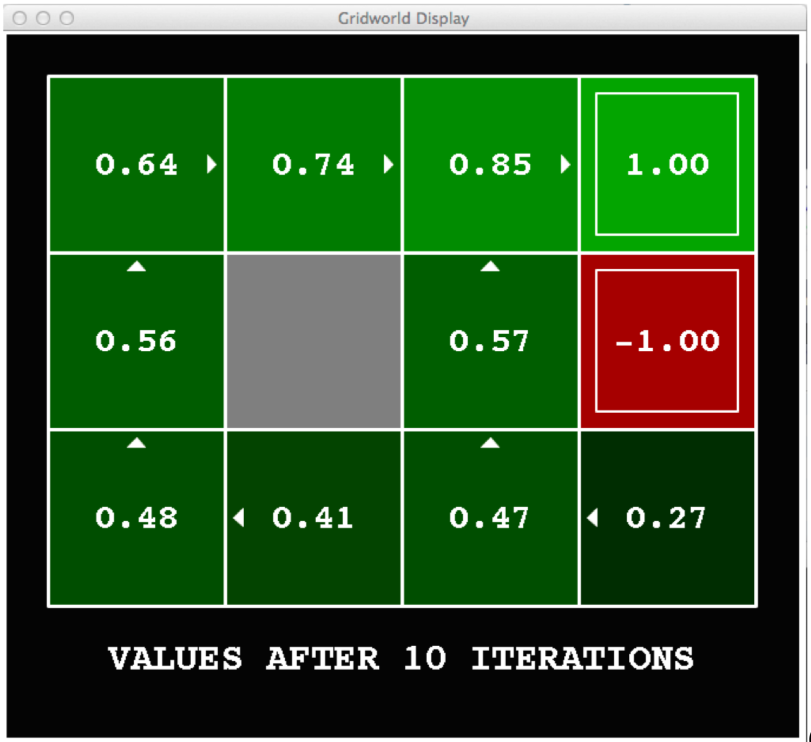
(D. Klein and P. Abbeel)

Example: Value iteration in GridWorld



(D. Klein and P. Abbeel)

Example: Value iteration in GridWorld

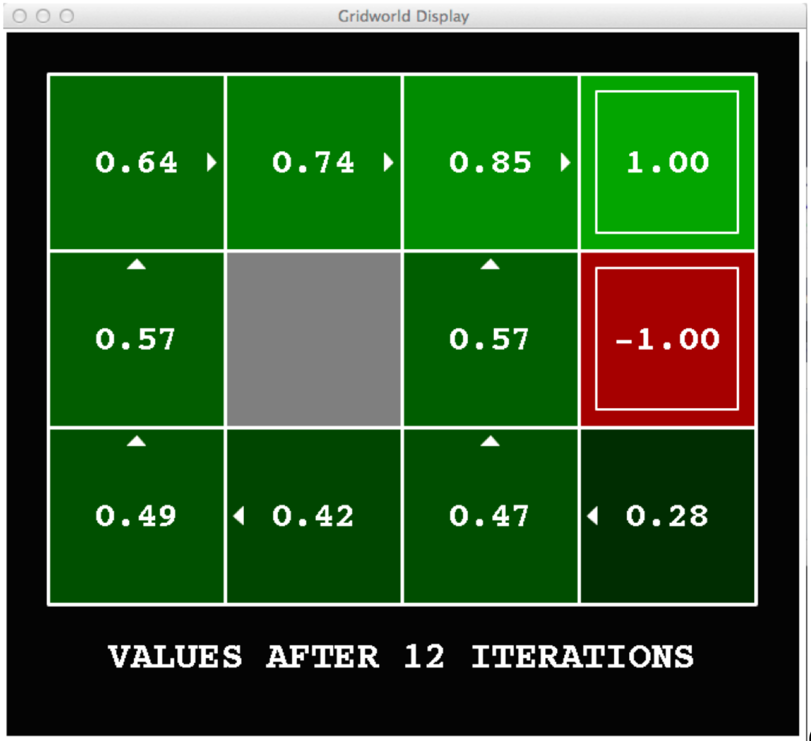


(D. Klein and P. Abbeel)

Example: Value iteration in GridWorld

(D. Klein and P. Abbeel)

Example: Value iteration in GridWorld



(D. Klein and P. Abbeel)

Example: Value iteration in GridWorld

(D. Klein and P. Abbeel)

Problems with Value Iteration

- The 'max' value at each state rarely changes
- The policy often converges long before the values converge

Policy iteration is an alternative approach, which is still optimal and can converge much more quickly.

Policy iteration

$$\pi^{(0)} \xrightarrow{E} V^{\pi^{(0)}} \xrightarrow{I} \pi^{(1)} \xrightarrow{E} V^{\pi^{(1)}} \xrightarrow{I} \pi^{(2)} \xrightarrow{E} \ldots$$

Repeat (until policy converges):

- Evaluate (E) V^π (where π is current policy)
- Policy improvement (I):

$$\pi'(s) \leftarrow \arg\max_{a \in A} \left[r(s,a) + \gamma \sum_{s' \in S} p(s' \mid s, a) V^\pi(s') \right], \quad \forall s$$

  update policy using one-step look-ahead with V^π as future values

- $\pi \leftarrow \pi'$

Proof of convergence shows $V^{\pi^{(k+1)}} > V^{\pi^{(k)}}$ (if policy changes).

Example: Policy iteration

Example on a different grid world, initialized with $\pi(s) = \uparrow$ (all states).

0	0	0	1
0		0	-100
0	0	0	0

Z. Kolter

Original reward function

Example: Policy iteration

Example on a different grid world, initialized with $\pi(s) = \uparrow$ (all states).

0.418	0.884	2.331	6.367
0.367		-8.610	-105.7
-0.168	-4.641	-14.27	-85.05

Z. Kolter

V^π at one iteration

Example: Policy iteration

Example on a different grid world, initialized with $\pi(s) = \uparrow$ (all states).

5.414	6.248	7.116	8.634
4.753		2.881	-102.7
2.251	1.977	1.849	-8.701

Z. Kolter

V^π at two iterations

Example: Policy iteration

Example on a different grid world, initialized with $\pi(s) = \uparrow$ (all states).

5.470	6.313	7.190	8.669
4.803		3.347	-96.67
4.161	3.654	3.222	1.526

Z. Kolter

V^π at three iterations (converged!)

Typical Gridworld results

- Approximation of value function
  - Policy iteration: exact value function after three iterations
  - Value iteration: $||\mathbf{V} - \mathbf{V}^*||_2 < 10^{-4}$ after 100 iterations
- Approximation of optimal policy
  - Policy iteration: optimal policy after three iterations
  - Value iteration: optimal policy after 12 iterations

What is the difference?

Value iteration

$$V'(s) \leftarrow \max_{a \in A} \left[r(s,a) + \gamma \sum_{s' \in S} p(s' \mid s,a) V(s') \right], \quad \forall s$$

Policy iteration

$$\pi'(s) \leftarrow \arg\max_{a \in A} \left[r(s,a) + \gamma \sum_{s' \in S} p(s' \mid s,a) V^{\pi}(s') \right], \quad \forall s$$

Policy iteration or Value iteration?

Both converge to the optimal policy in a finite number of steps.

- Value iteration:
  - $O(|S| \cdot |A| \cdot L)$ per iteration
  - less work per iteration (no policy evaluation!)
- Policy iteration:
  - policy changes every iteration
  - $O(|S| \cdot |A| \cdot L + |S|^3)$ computation per iteration
  - tends to require less steps (larger changes each step)

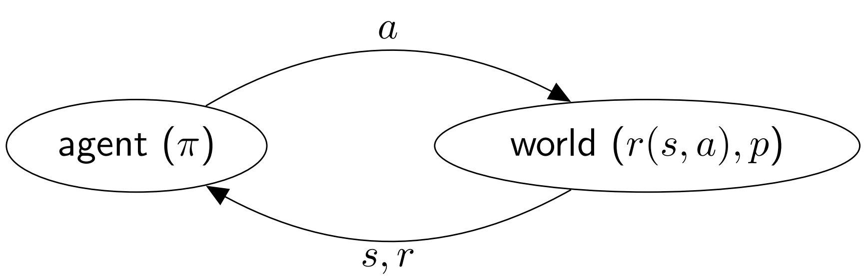
In practice, PI tends to be faster, especially if transition matrix is sparse so that policy evaluation is fast.

Other solution approaches

- Can take derivatives of a policy that is parameterized (good for large/continuous action spaces)
- Tree search: can “roll out,” or simulate policies. Good for large state spaces. (Approximate form of expectimax).
- Linear programming.

Contents

Next Class: Learning a Policy



- Agent knows current state s takes actions a, and gets reward r.
- Only access to reward model $r(s,a)$, transition model $p(s' \mid s,a)$ via feedback
- Very challenging problem to learn π while uncertain about model of the world.

Conclusion

- MDPs are a general, probabilistic model for acting in an uncertain environment
- The main assumptions in the model are:
  - Markovian: $p_t(s_{t+1} \mid s_1, \ldots, s_t, a_1, \ldots, a_t) = p_t(s_{t+1} \mid s_t, a_t)$
  - Stationarity: $p_t(s_{t+1} \mid s_t, a_t) = p(s_{t+1} \mid s_t, a_t)$
- Planning is the problem of deciding how to act, given knowledge of the MDP (S, A, r, p)
- For the infinite time horizon, discounted setting, we can use value iteration and policy iteration.