Reinforcement learning

CE-477: Machine Learning - CS-828: Theory of Machine Learning
Sharif University of Technology
Fall 2024

Fatemeh Seyyedsalehi

References of the lecture are mentioned in the last slide

Reinforcement Learning (RL)

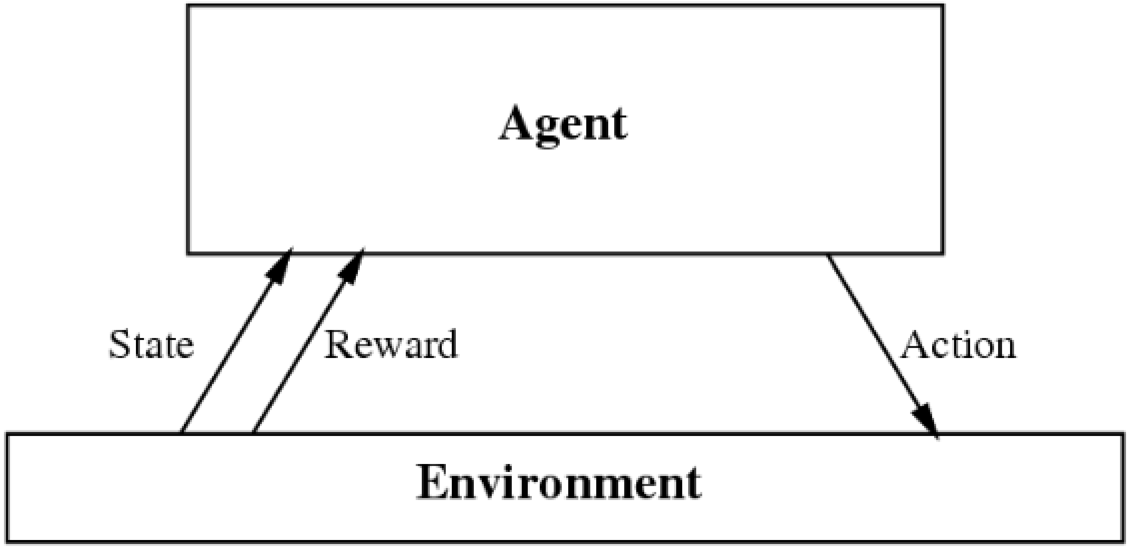
- Learning as a result of interaction with an environment
  - to improve the agent's ability to behave optimally in the future to achieve the goal.
- The first idea when we think about the nature of learning
- Examples:
  - Baby movements
  - Learning to drive car
    - Environment's response affects our subsequent actions
    - We find out the effects of our actions later

Paradigms of learning

- Supervised learning
  - Training data: features and labels for N samples $\{(\boldsymbol{x}^{(n)}, y^{(n)})\}_{n=1}^{N}$
- Unsupervised learning
  - Training data: only features for N samples $\{\boldsymbol{x}^{(n)}\}_{n=1}^{N}$
- Reinforcement learning
  - Training data: a sequence of (s, a, r)
    - (state, some action, reward)
  - Agent acts on its environment, it receives some evaluation of its action via reinforcement signal
    - it is not told of which action is the correct one to achieve its goal

Reinforcement Learning (RL)

- S: Set of states
- A: Set of actions



- Goal: Learning an optimal policy (mapping from states to actions) in order to maximize its long-term reward
  - The agent's objective is to maximize amount of reward it receives over time.

Main characteristics and applications of RL

- Main characteristics of RL
  - Agent must learn from interactions with environment
    - Agent must be able to learn from its own experience
    - Not entirely supervised, but interactive
      - by trial-and-error
    - Opportunity for active exploration
      - Needs trade-off between exploration and exploitation
      - Exploration: you have to try unknown actions to get information
      - Exploitation: eventually, you have to use what you know
- Goal: map states to actions, so as to maximize reward over time (optimal policy)

Main elements of RL

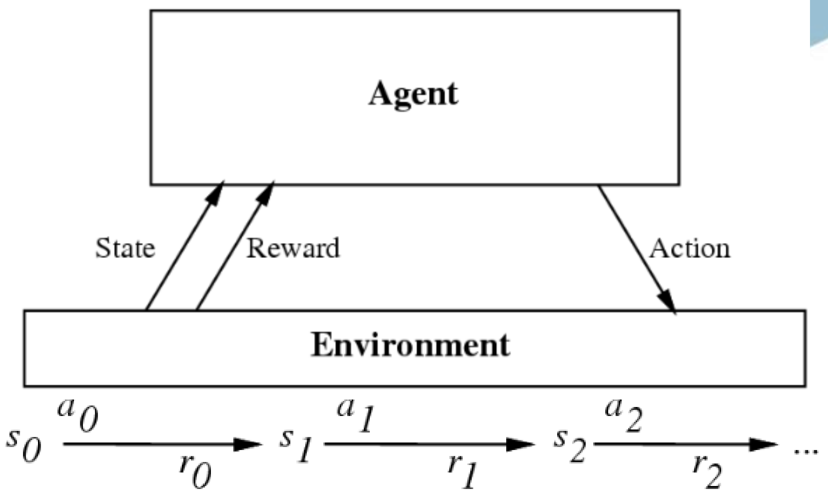
- A reward function
  - It maps each state (or, state-action pair) to a real number, called reward.

- Policy: A map from state space to action space.
  - May be stochastic.

- A value function
  - Value of a state (or state-action) is the total expected reward, starting from that state (or state-action).

RL problem: deterministic environment

- Deterministic
  - Transition and reward functions



- At time t:
  - Agent observes state $s_t \in S$
  - Then chooses action $a_t \in A$
  - Then receives reward r_t, and state changes to s_{t+1}

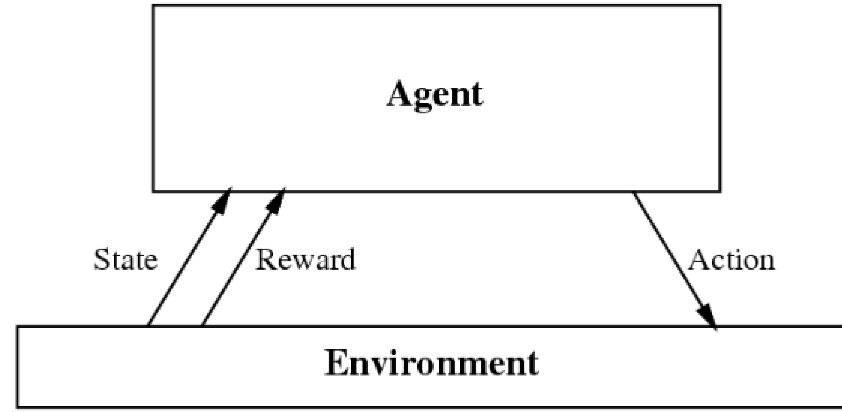
- Learn to choose action a_t in state s_t that maximizes the discounted return:

$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k}, \qquad 0 < \gamma \leq 1$$

Upon visiting the sequence of states $s_t, s_{t+1}, \ldots$ with actions $a_t, a_{t+1}, \ldots$ it shows the total payoff

RL problem: stochastic environment

- Stochastic environment
  - Stochastic transition and/or reward



- Learn to choose a policy that maximizes the expected discounted **return**:

$$E[R_t] = E[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots]$$

starting from state s_t

$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$$

RL: Autonomous Agent

- Execute actions in environment, observe results, and learn
  - Learn (perhaps stochastic) policy that maximizes $E[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \,|s_t = s, \pi]$ for every state $s \in S$

- Function to be learned is the policy $\pi: S \times A \rightarrow [0,1]$ (when the policy is deterministic $\pi: S \rightarrow A$)
  - Training examples in supervised learning: $\langle s, a \rangle$ pairs
  - RL training data shows the amount of reward for a pair $\langle s, a \rangle$.
    - training data are of the form $\langle \langle s, a \rangle, r \rangle$

Markov Decision Process (RL Setting)

- Markov assumption:

$$P(s_{t+1}, r_t | s_t, a_t, r_{t-1}, s_{t-1}, a_{t-1}, , r_{t-2}, \dots) = P(s_{t+1}, r_t | s_t, a_t)$$

- Markov property: Transition probabilities depend on state only, not on the path to the state.

- Goal: for every possible state $s \in S$ learn a policy π for choosing actions that maximizes (infinite-horizon sum of discounted reward):

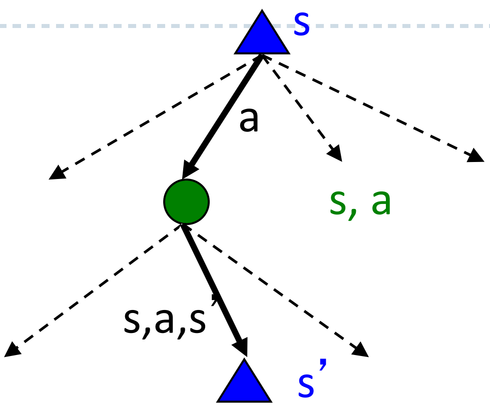
$$E[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots | s_t = s, \pi], \qquad 0 < \gamma \leq 1$$

Markov Decision Process

- MDPs provide a way to think about how we can act optimally under uncertainty

- Components:
  - states
  - actions
  - state transition probabilities
  - reward function
  - discount factor

MDP: Definition

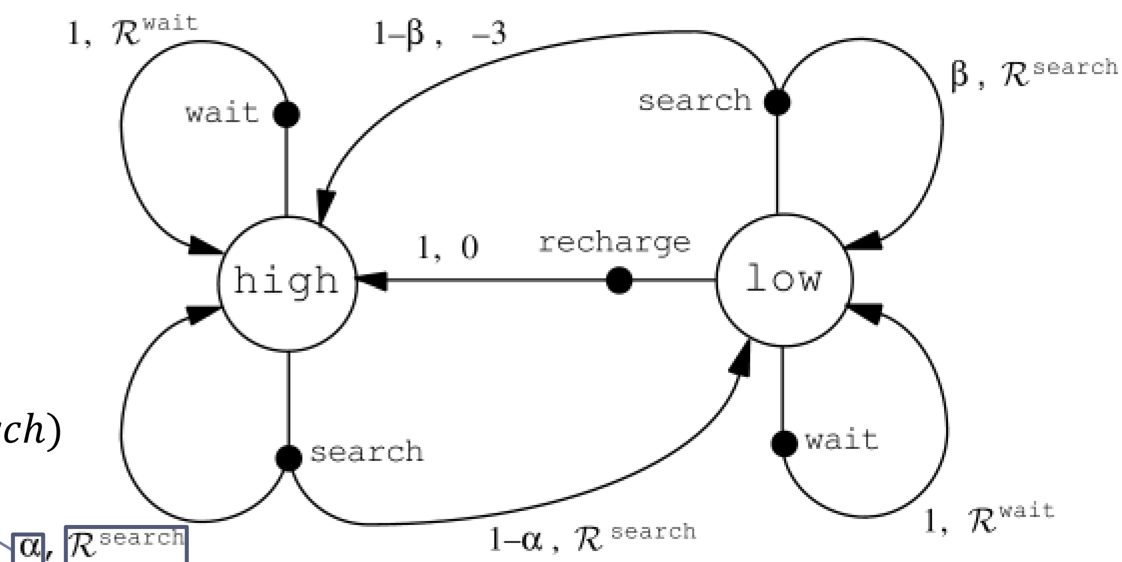


- A Markov decision process is composed:
  - $\mathcal{S}$: a finite set of states
  - $\mathcal{A}$: a finite set of actions
  - Transition probabilities
    - $\mathcal{P}_{ss'}^{a} = P(s_{t+1} = s' | s_t = s, a_t = a)$ as the probability that action a in state s at time t will lead to state s' at time $t+1$
  - Immediate rewards:
    - $\mathcal{R}_{ss'}^{a} = E\{r_t | s_t = s, a_t = a, s_{t+1} = s'\}$ as the immediate reward received after transition to state s' from state s with action a
  - $\gamma \in [0,1]$: discount factor
    - represents the difference in importance between future rewards and present rewards.

MDP: Recycling Robot example

- $S = \{high, low\}$
- $A = \{search, wait, recharge\}$
  - $\mathcal{A}(high) = \{search, wait\}$ → Available actions in the 'high' state
  - $\mathcal{A}(low) = \{search, wait, recharge\}$
- $\mathcal{R}_{search} > \mathcal{R}_{wait}$

$P(s_{t+1} = high | s_t = high, a_t = search)$



reward

State-value function for policy π

- Given a policy π, define value function

$$V^{\pi}(s) = E\{\sum_{k=0}^{\infty} \gamma^k r_{t+k} \,|s_t = s, \pi\}$$

- $V^{\pi}(s)$: How good for the agent to be in the state s when its policy is π
  - It is simply the expected sum of discounted rewards upon starting in state s and taking actions according to π

Recursive definition for $V^{\pi}(S)$

$$V^{\pi}(s) = E\{\sum_{k=0}^{\infty} \gamma^{k} r_{t+k} \,|s_t = s, \pi\}$$

$$= E\left\{r_t + \gamma \sum_{k=1}^{\infty} \gamma^{k-1} r_{t+k} \,|s_t = s, \pi\right\}$$

$$= E\left\{r_t + \gamma \sum_{k=0}^{\infty} \gamma^{k} r_{t+k+1} \,|s_t = s, \pi\right\}$$

$$= \sum_{s'} \mathcal{P}_{ss'}^{\pi(s)} \left(\mathcal{R}_{ss'}^{\pi(s)} + \gamma \underbrace{E\{\sum_{k=0}^{\infty} \gamma^{k} r_{t+k+1} \,|s_{t+1} = s', \pi\}}_{V^{\pi}(s')}\right)$$

$$\mathcal{P}_{ss'}^{a} = P(s_{t+1} = s' | s_t = s, a_t = a)$$

$$\mathcal{R}_{ss'}^{a} = E\{r_t | s_t = s, a_t = a, s_{t+1} = s'\}$$

Bellman Equations

$$V^{\pi}(s) = \sum_{s'} \mathcal{P}_{ss'}^{\pi(s)} \left(\mathcal{R}_{ss'}^{\pi(s)} + \gamma V^{\pi}(s')\right)$$

Base equation for dynamic programming approaches

State-action value function for policy π

$$Q^\pi(s,a) = E\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k} \,\middle|\, s_t = s, a_t = a, \pi\right\}$$

$$= \sum_{s'} \mathcal{P}_{ss'}^{a} \Big(\mathcal{R}_{ss'}^{a} + \gamma \underbrace{E\{\textstyle\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,|s_{t+1} = s', \pi\}}_{V^\pi(s')}\Big)$$

$$Q^\pi(s,a) = \sum_{s'} \mathcal{P}_{ss'}^{a} \left(\mathcal{R}_{ss'}^{a} + \gamma V^\pi(s')\right)$$

Optimal policy

- Policy π is better than (or equal to) π' (i.e. $\pi \geq \pi'$) iff

$$V^{\pi}(s) \geq V^{\pi'}(s), \qquad \forall s \in S$$

- Optimal policy π^* is better than (or equal to) all other policies ($\forall \pi, \pi^* \geq \pi$)

- **Optimal policy π^*:**

$$\pi^*(s) = \underset{\pi}{\operatorname{argmax}}\, V^{\pi}(s), \qquad \forall s \in S$$

MDP: Optimal policy state-value and action-value functions

- Optimal policies share the same optimal state-value function ($V^{\pi^*}(s)$ will be abbreviated as $V^*(s)$):

$$V^*(s) = \max_{\pi} V^{\pi}(s), \qquad \forall s \in S$$

- And the same optimal action-value function:

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a), \qquad \forall s \in S, a \in \mathcal{A}(s)$$

- For any MDP, a deterministic optimal policy exists!

Bellman optimality equation

- For optimal policy, the Bellman equation can be written as follows,
- Remember these equations until the end of the lecture !!

$$V^*(s) = \max_{a \in \mathcal{A}(s)} \sum_{s'} \mathcal{P}^a_{ss'} \left(\mathcal{R}^a_{ss'} + \gamma V^*(s') \right)$$

$$Q^*(s,a) = \sum_{s'} \mathcal{P}^a_{ss'} \left(\mathcal{R}^a_{ss'} + \gamma \max_{a'} Q^*(s', a') \right)$$

$$V^*(s) = \max_{a \in \mathcal{A}(s)} Q^*(s,a)$$

$$Q^*(s,a) = \sum_{s'} \mathcal{P}^a_{ss'} \left(\mathcal{R}^a_{ss'} + \gamma V^*(s') \right)$$

Optimal policy

- If we have $V^*(s)$ and $P(s_{t+1}|s_t, a_t)$ we can compute $\pi^*(s)$

$$\pi^*(s) = \underset{a}{\operatorname{argmax}} \left\{ \sum_{s'} P(s'|s,a) \left(R(s,a,s') + \gamma V^*(s') \right) \right\}$$

- It can also be computed as:

$$\pi^*(s) = \underset{a \in \mathcal{A}(s)}{\operatorname{argmax}} Q^*(s,a)$$

  - It is not dependent on the initial state.

Known Environments

- We first assume that the MDP is known although in RL problems, we do not know it
  - $\mathcal{P}_{ss'}^{a}$ and $\mathcal{R}_{ss'}^{a}$ is known

- Dynamic programming methods are used for solving the problems when MDP is known
  - Value iteration and Policy iteration are two more classic approaches to this problem.

Value Iteration

- Bellman equations characterize the optimal values:

$$V^*(s) = \max_a \sum_{s'} P(s'|s,a)\left(R(s,a,s') + \gamma V^*(s')\right)$$

- Value iteration computes them:

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} P(s'|s,a)\left(R(s,a,s') + \gamma V_k(s')\right)$$

Value Iteration algorithm

Consider only MDPs with finite state and action spaces:

1) Initialize all $V(s)$ to zero
2) Repeat until convergence

   for $s \in S$

$$V^{new}(s) \leftarrow \max_a \sum_{s'} P(s'|s,a)\left(R(s,a,s') + \gamma V(s')\right)$$

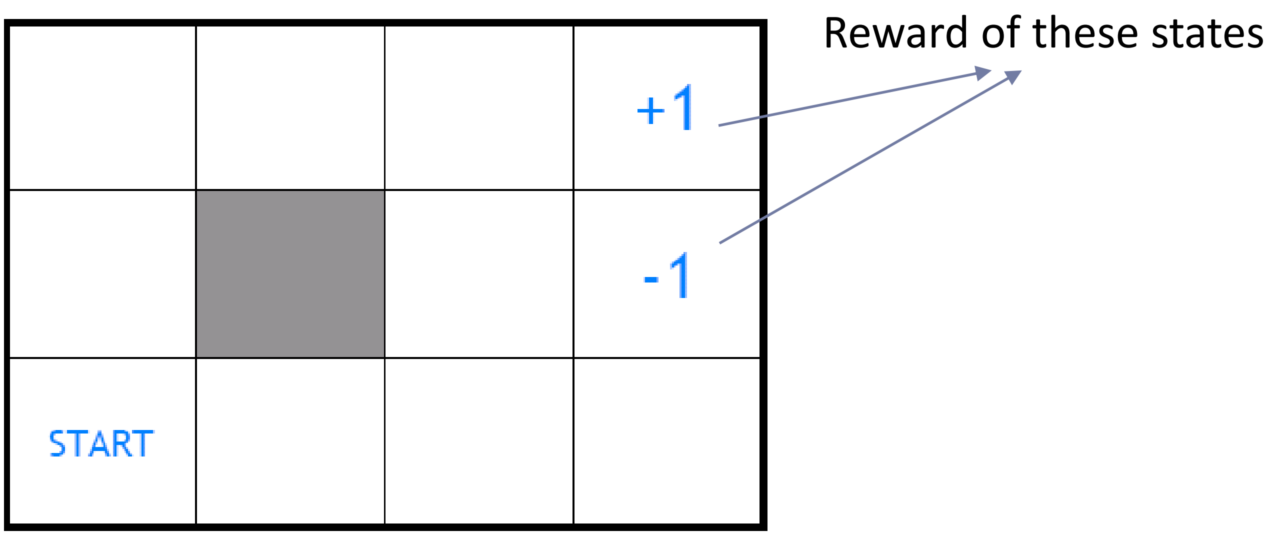
   $V = V^{new}$

3) for $s \in S$

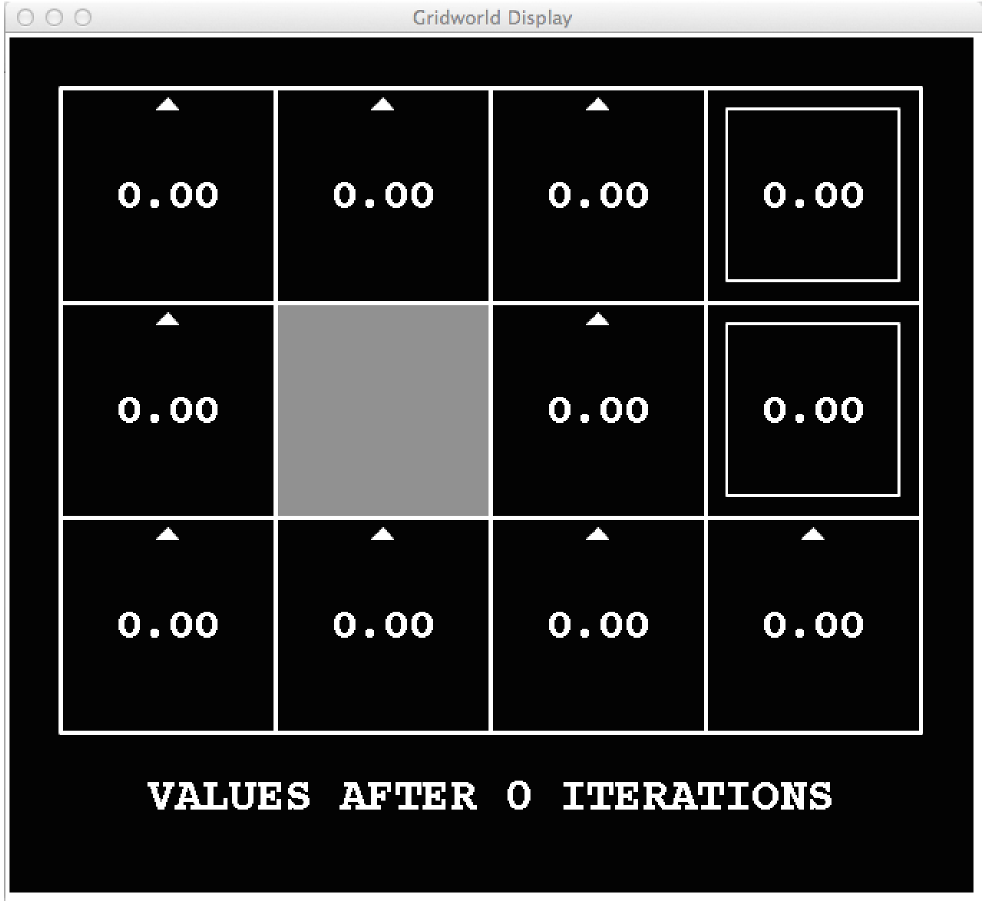
$$\pi(s) \leftarrow \operatorname*{argmax}_a \textstyle\sum_{s'} P(s'|s,a)\left(R(s,a,s') + \gamma V(s')\right)$$

$V(s)$ converges to $V^*(s)$

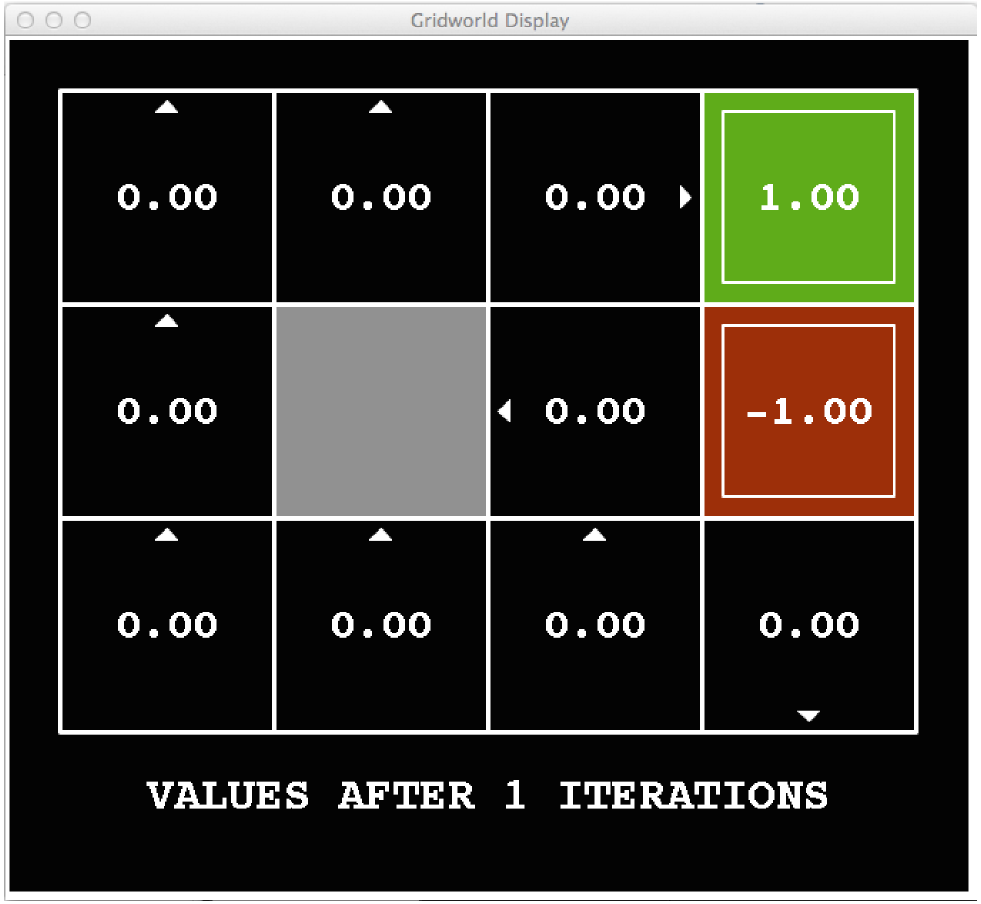
k=0



k=0


Gridworld Display
0.00
0.00
0.00
0.00
0.00
0.00
0.00
0.00
0.00
0.00
0.00
VALUES AFTER 0 ITERATIONS

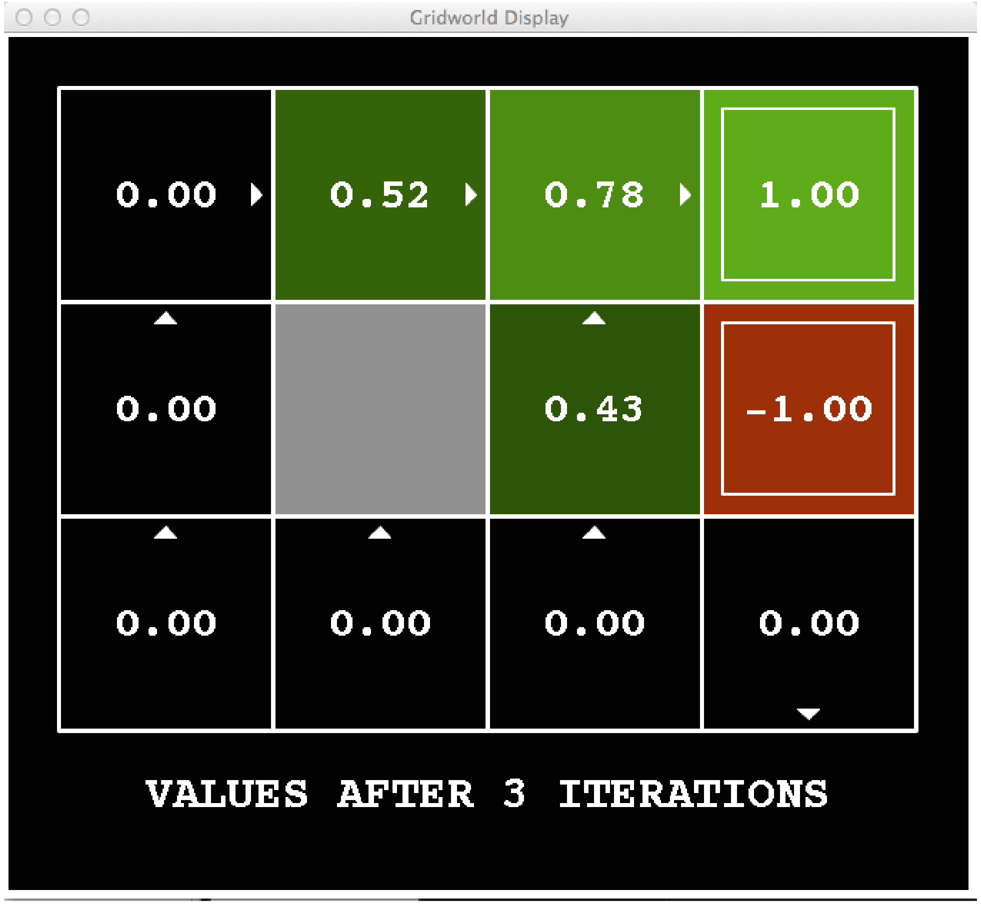
k=1


Gridworld Display
0.00
0.00
0.00
1.00
0.00
0.00
-1.00
0.00
0.00
0.00
0.00
VALUES AFTER 1 ITERATIONS

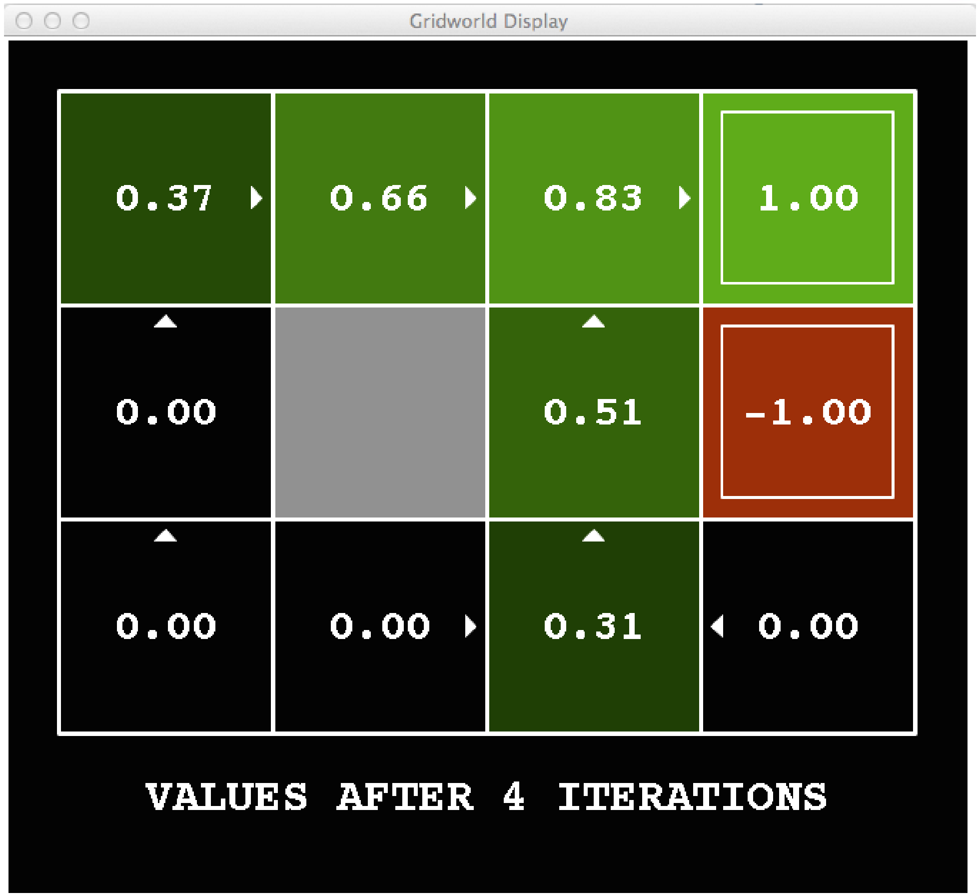
k=2


Gridworld Display
0.00
0.00
0.72
1.00
0.00
0.00
-1.00
0.00
0.00
0.00
0.00
VALUES AFTER 2 ITERATIONS

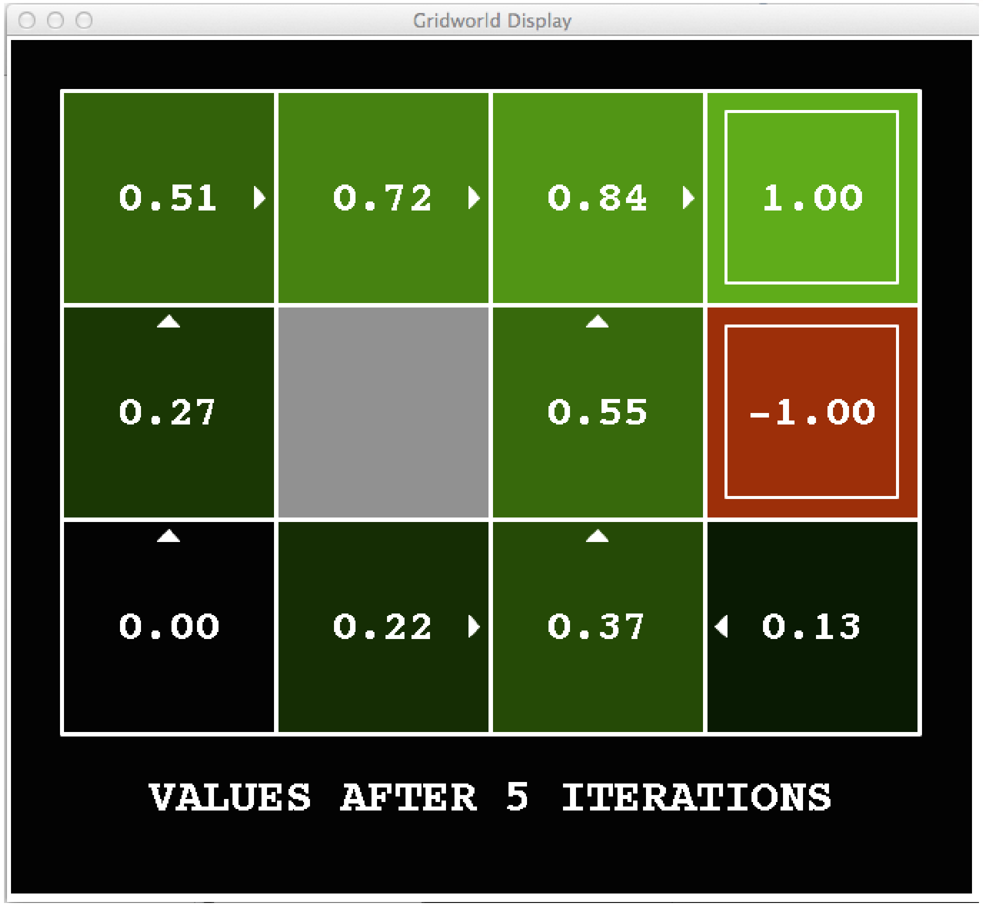
k=3


Gridworld Display
0.00
0.52
0.78
1.00
0.00
0.43
-1.00
0.00
0.00
0.00
0.00
VALUES AFTER 3 ITERATIONS

k=4


Gridworld Display
0.37
0.66
0.83
1.00
0.00
0.51
-1.00
0.00
0.00
0.31
0.00
VALUES AFTER 4 ITERATIONS

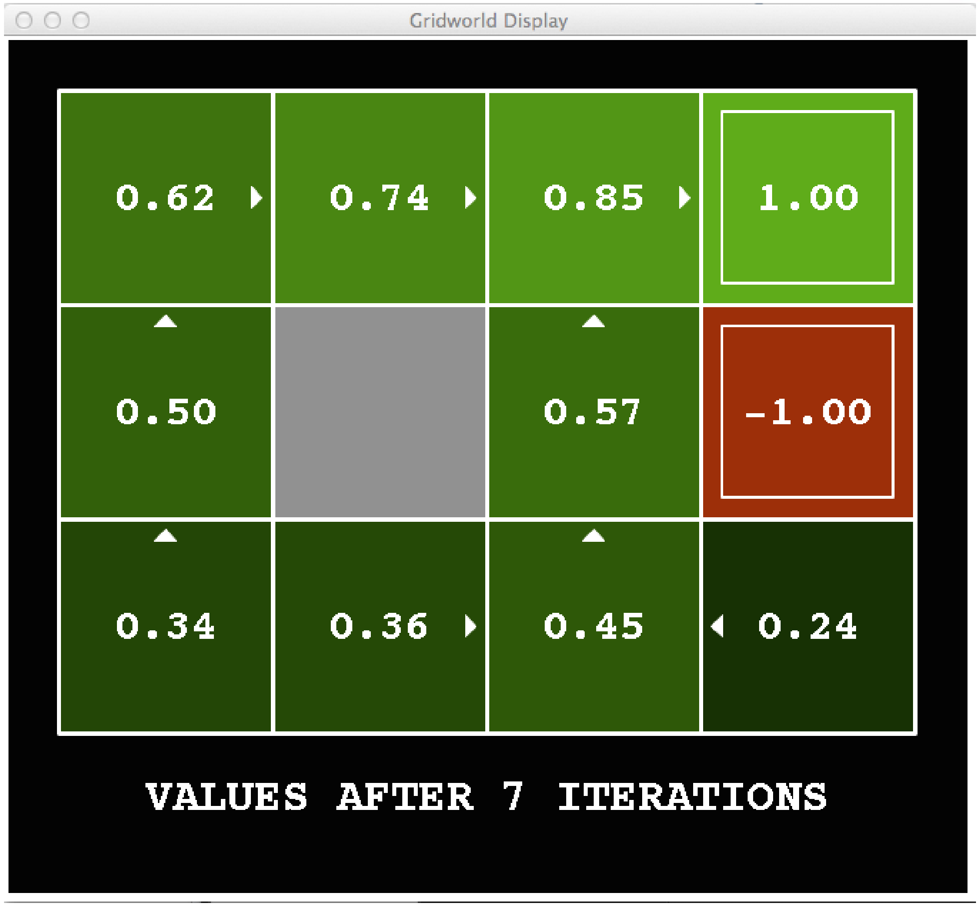
k=5


Gridworld Display
0.51
0.72
0.84
1.00
0.27
0.55
-1.00
0.00
0.22
0.37
0.13
VALUES AFTER 5 ITERATIONS

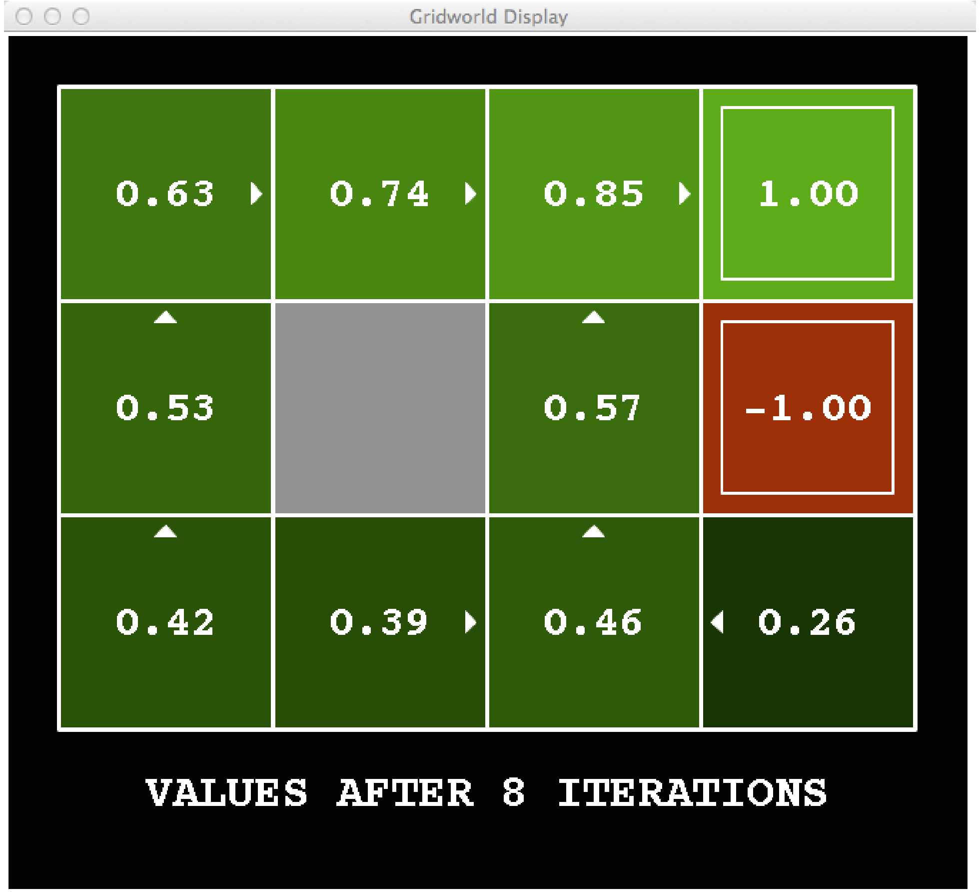
k=6


Gridworld Display
0.59
0.73
0.85
1.00
0.41
0.57
-1.00
0.21
0.31
0.43
0.19
VALUES AFTER 6 ITERATIONS

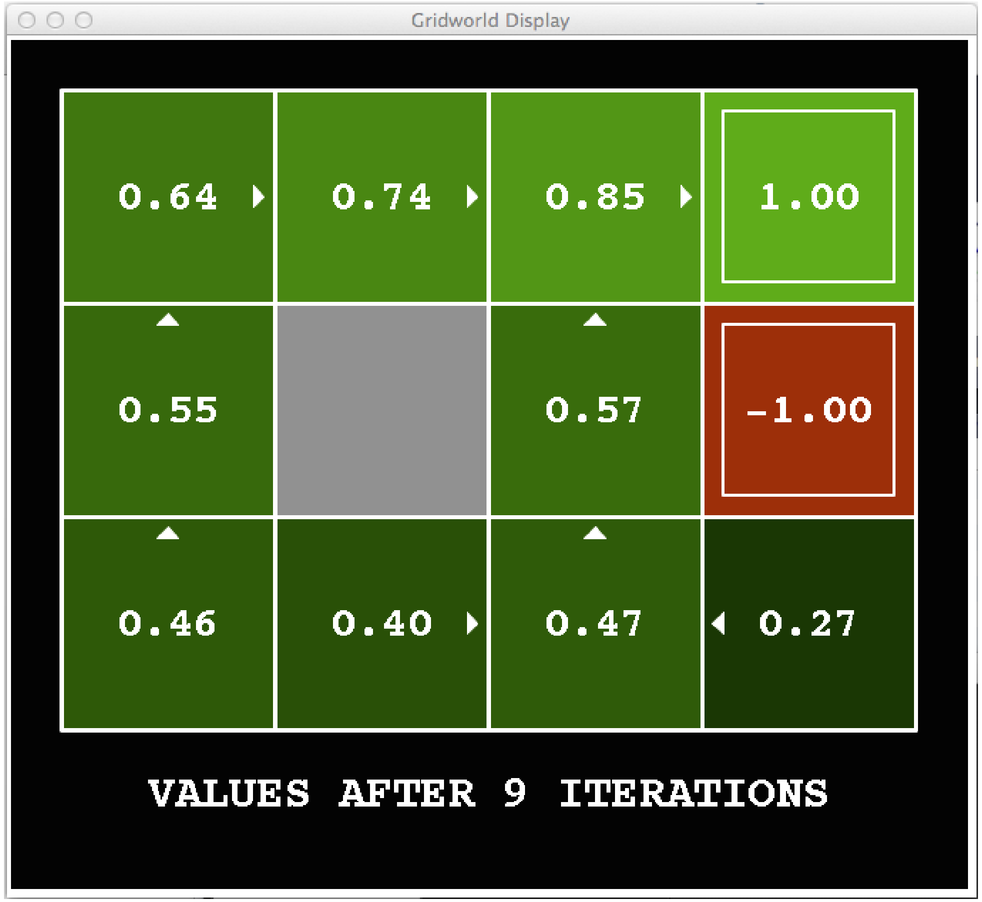
k=7


Gridworld Display
0.62
0.74
0.85
1.00
0.50
0.57
-1.00
0.34
0.36
0.45
0.24
VALUES AFTER 7 ITERATIONS

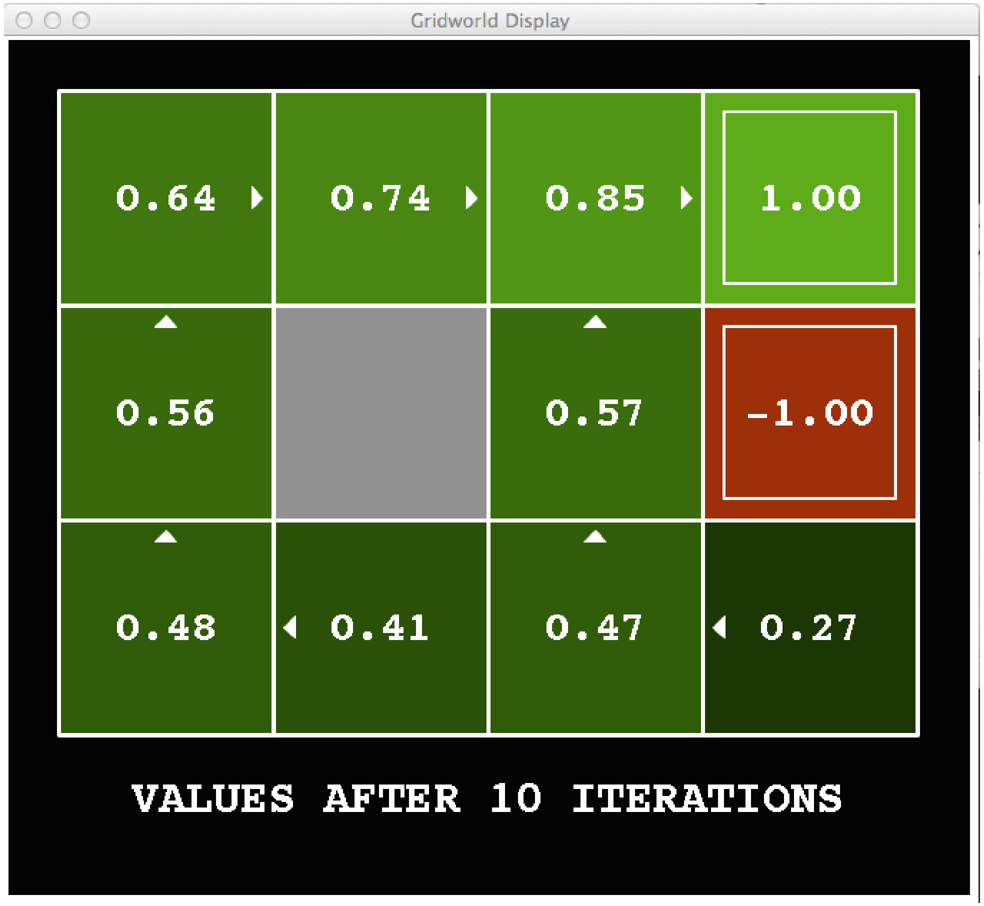
k=8


Gridworld Display
0.63
0.74
0.85
1.00
0.53
0.57
-1.00
0.42
0.39
0.46
0.26
VALUES AFTER 8 ITERATIONS

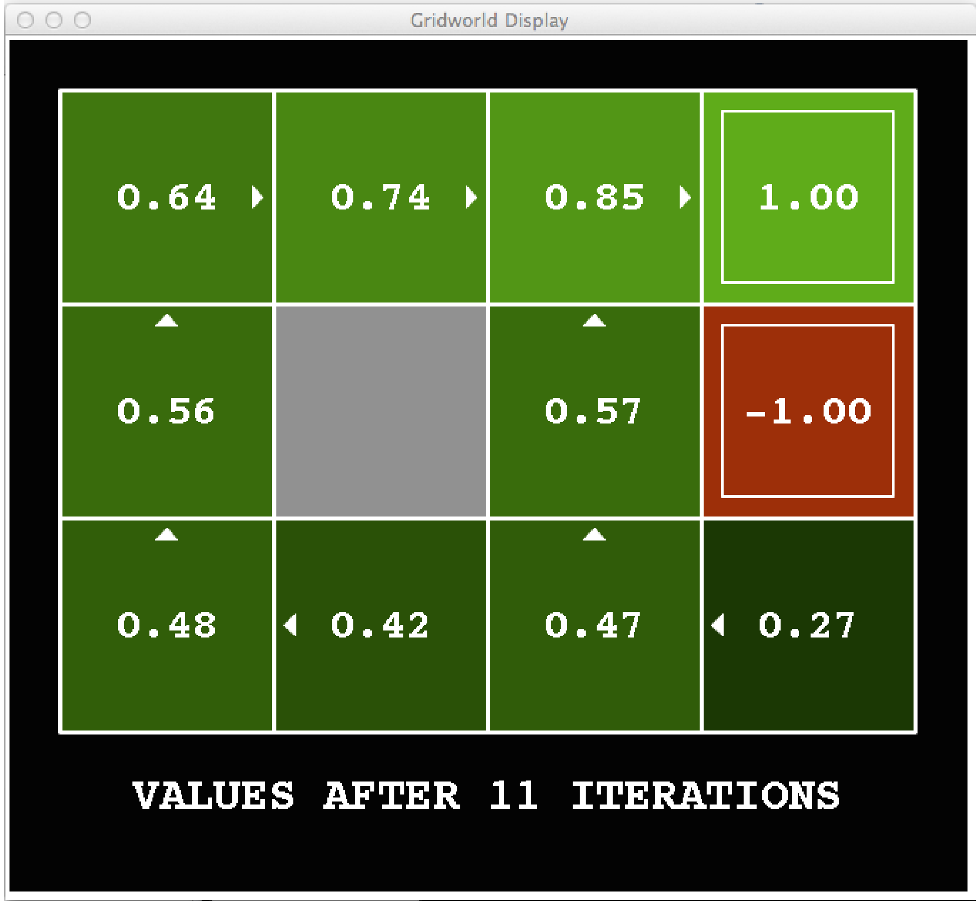
k=9


Gridworld Display
0.64
0.74
0.85
1.00
0.55
0.57
-1.00
0.46
0.40
0.47
0.27
VALUES AFTER 9 ITERATIONS

k=10


Gridworld Display
0.64
0.74
0.85
1.00
0.56
0.57
-1.00
0.48
0.41
0.47
0.27
VALUES AFTER 10 ITERATIONS

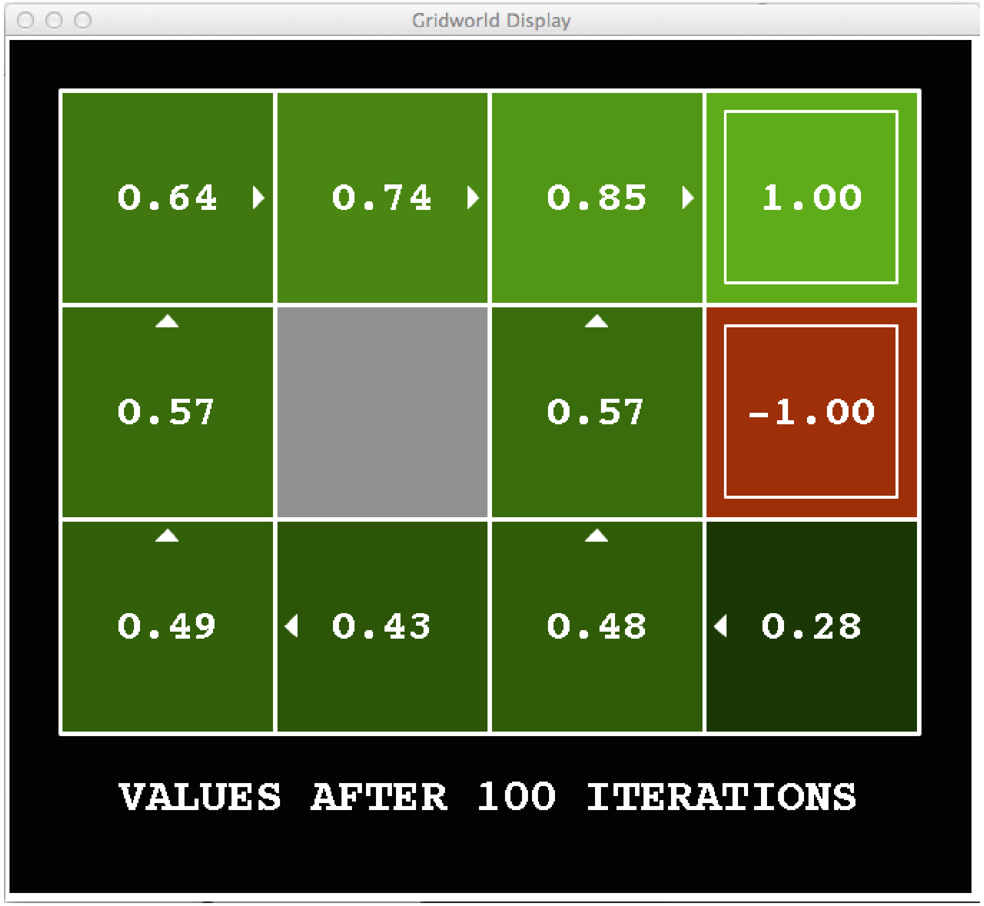
k=11


Gridworld Display
0.64
0.74
0.85
1.00
0.56
0.57
-1.00
0.48
0.42
0.47
0.27
VALUES AFTER 11 ITERATIONS

k=12

Gridworld Display
0.64
0.74
0.85
1.00
0.57
0.57
-1.00
0.49
0.42
0.47
0.28
VALUES AFTER 12 ITERATIONS

k=100


Gridworld Display
0.64
0.74
0.85
1.00
0.57
0.57
-1.00
0.49
0.43
0.48
0.28
VALUES AFTER 100 ITERATIONS

Problems with Value Iteration

- Value iteration repeats the Bellman updates:

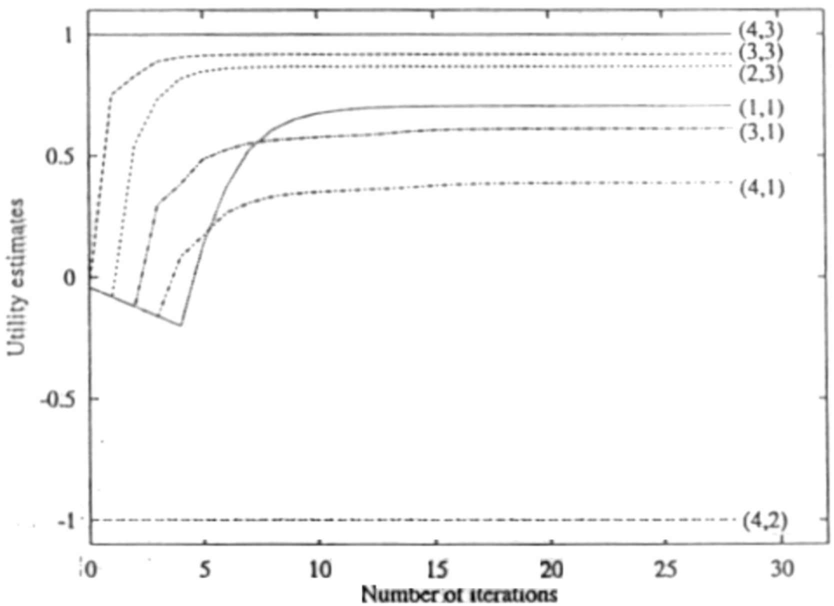
$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} P(s'|s,a)\left(R(s,a,s') + \gamma V_k(s')\right)$$

- Problem 1: It's slow – $O(S^2A)$ per iteration

- Problem 2: The "max" at each state rarely changes

- Problem 3: The policy often converges long before the values

Convergence: Example



3	0.812	0.868	0.912	+1
2	0.762		0.660	-1
1	0.705	0.655	0.611	0.388
	1	2	3	4

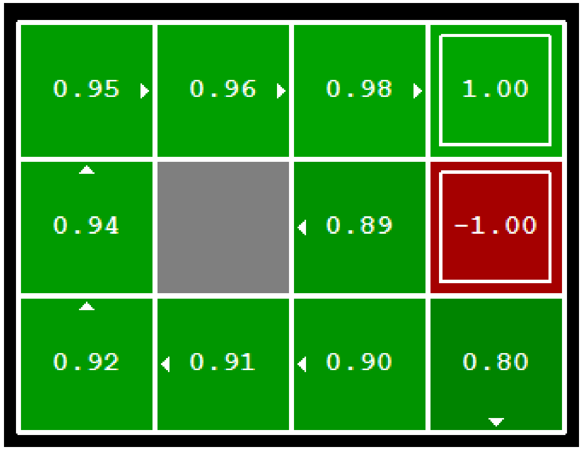
[Russel, AIMA, 2010]

Computing Actions from Values

- Let's imagine we have the optimal values V*(s)

- How should we act?
  - It's not obvious!

- We need to do (one step)

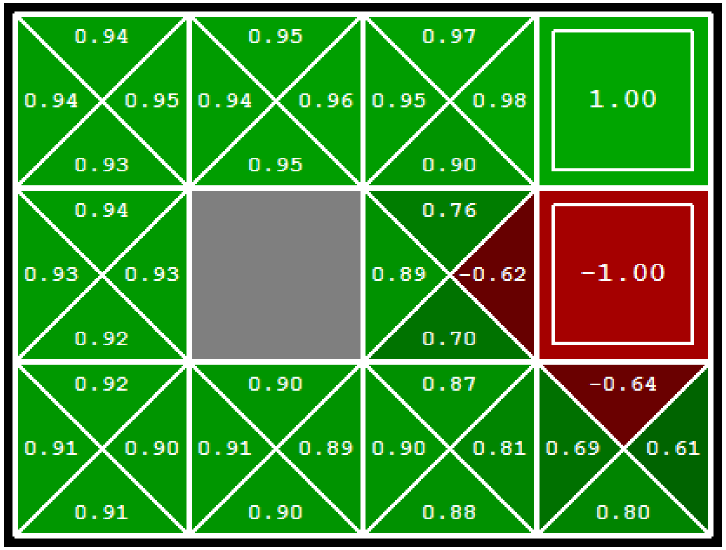


$$\pi^*(s) = \underset{a}{\operatorname{argmax}} \sum_{s'} P(s'|s,a)\left(R(s,a,s') + \gamma V^*(s')\right)$$

- This is called policy extraction, since it gets the policy implied by the values

Computing Actions from Q-Values

- Let's imagine we have the optimal q-values:



- How should we act?
  - Completely trivial to decide!

$$\pi^*(s) = \arg\max_a Q^*(s,a)$$

- Important lesson: actions are easier to select from q-values than values!

Policy Iteration Algorithm

1) Initialize $\pi(s)$ arbitrarily
2) Repeat until convergence

   - (policy evaluation step) Compute the value function for the current policy π (i.e. V^{π})

   $V \leftarrow V^{\pi}$

   - For $s \in S$

   $$\pi(s) \leftarrow \underset{a}{\operatorname{argmax}} \sum_{s'} P(s'|s,a)\left(R(s,a,s') + \gamma V(s')\right)$$

updates the policy (greedily) using the current value function.

-$\pi(s)$ converges to $\pi^*(s)$

-Policy evaluation step = using a linear equation we can compute the value function in each iteration

When to stop iterations:

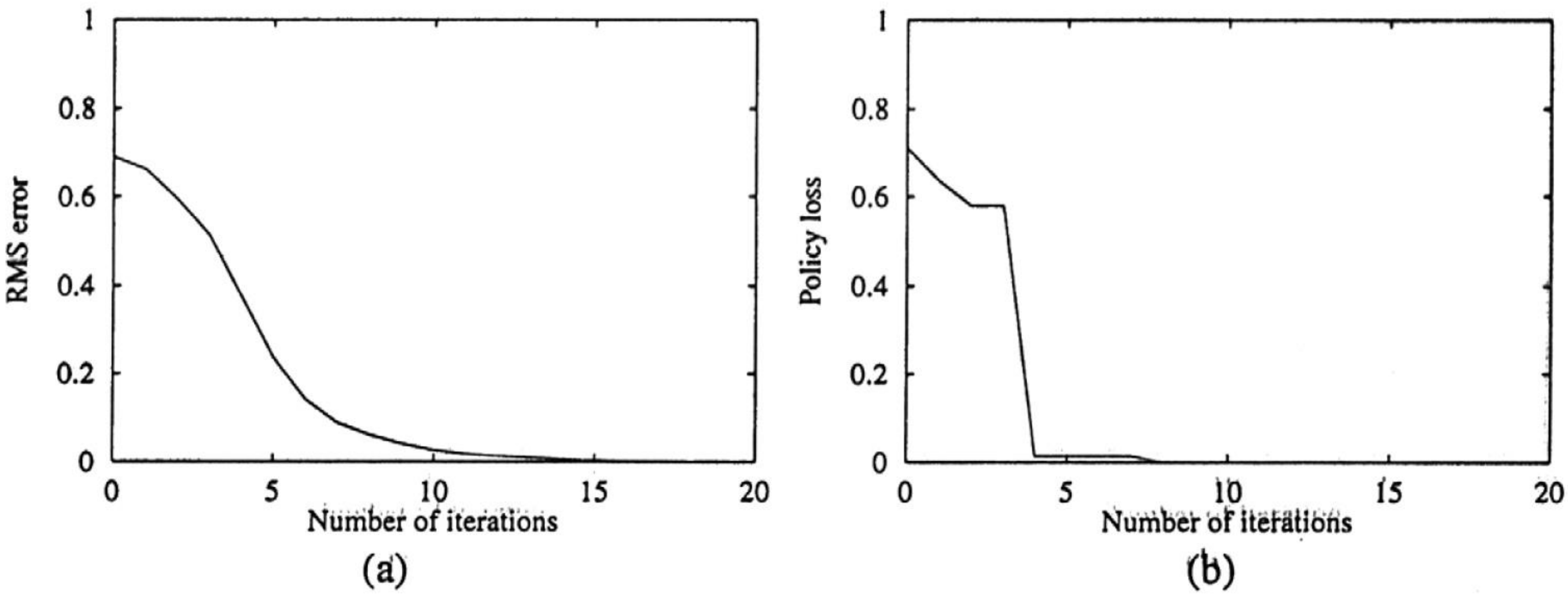


Figure 17.6 (a) The RMS (root mean square) error of the utility estimates compared to the correct values, as a function of iteration number during value iteration. (b) The expected policy loss compared to the optimal policy.

[Russel, AIMA 2010]

Comparison

- Both value iteration and policy iteration compute the same thing (all optimal values)

- In value iteration:
  - Every iteration updates both the values and (implicitly) the policy
  - We don’t track the policy, but taking the max over actions implicitly recomputes it

- In policy iteration:
  - We do several passes that update utilities with fixed policy (each pass is fast because we consider only one action, not all of them)
  - After the policy is evaluated, a new policy is chosen (slow like a value iteration pass)
  - The new policy will be better (or we’re done)

- Both are dynamic programs for solving MDPs

Unknown transition model

- So far: learning optimal policy when we know $\mathcal{P}_{ss'}^{a}$ and $\mathcal{R}_{ss'}^{a}$
  - it requires prior knowledge of the environment's dynamics

- Now, we assume we don't know the environment's dynamics. Two types of algorithms:
  - Model-based (passive)
    - Learn model of environment (transition and reward probabilities)
    - Then, value iteration or policy iteration algorithms
  - Model-free (active)

Model-Based Learning

- Learn an approximate model based on experiences
- Solve for values as if the learned model were correct

- Step 1: Learn empirical MDP model
  - Count outcomes s' for each s, a
  - Normalize to give an estimate of $\hat{P}(s'|s,a)$
  - Discover each $\hat{R}(s,a,s')$ when we experience (s, a, s')

- Step 2: Solve the learned MDP
  - For example, use value iteration, as before

Model free approaches to solve RL problems

- These methods can be categorized into:
  - Monte Carlo methods
  - Temporal-difference learning
    - Q-learning is a more recent approaches to this problem.

Monte Carlo methods

- Require only *experience*
  - Sample sequences of states, actions, and rewards from on-line or simulated interaction with an environment

- Are based on averaging sample returns
  - are defined for episodic tasks

Policy evaluation using samples

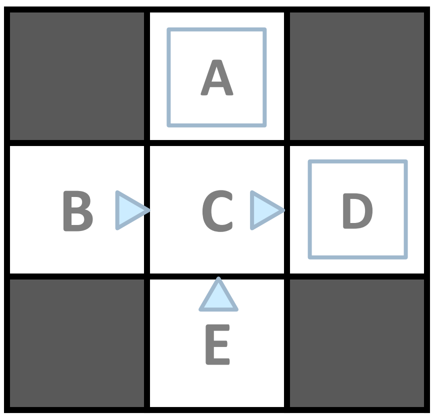
- Before discussing about Monte Carlo approach for finding optimal policy, first we solve a simpler problem “policy evaluation” by Monte Carlo

- Goal: Compute values for each state under π

- Idea: average together observed sample values
  - Act according to π
  - Every time you visit a state, write down what the sum of discounted rewards turned out to be
  - Average those samples

Example: Direct Policy Evaluation

Input Policy π



Assume: $\gamma = 1$

Observed Episodes

Episode 1

B, east, C, -1
C, east, D, -1
D, exit, x, +10

Episode 2

B, east, C, -1
C, east, D, -1
D, exit, x, +10

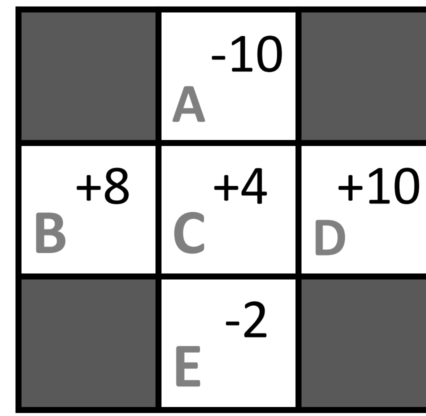
Episode 3

E, north, C, -1
C, east, D, -1
D, exit, x, +10

Episode 4

E, north, C, -1
C, east, A, -1
A, exit, x, -10

Output Values



Monte Carlo RL algorithm

1) Initialize Q and π arbitrarily and $Returns$ to empty lists
2) Repeat

Policy evaluation step

Generate an episode using π and exploring starts

for each pair of s and a appearing in the episode

R ←return following the first occurrence of s, a

Append R to $Returns(s, a)$

$Q(s, a) \leftarrow average(Returns(s, a))$

for each s in the episode

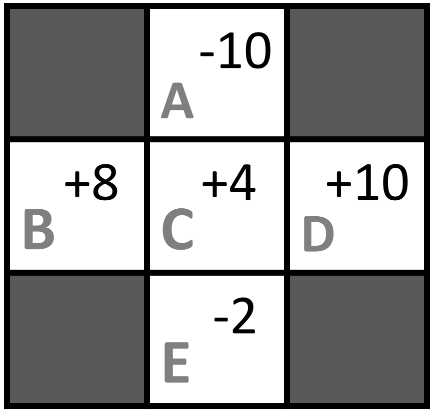
$$\pi(s) \leftarrow \underset{a}{\operatorname{argmax}} Q(s, a)$$

Problems with Direct Evaluation

- What's good about direct evaluation?
  - It's easy to understand
  - It doesn't require any knowledge of P, R
  - It eventually computes the correct average values, using just sample transitions

- What bad about it?
  - It wastes information about state connections
  - Each state must be learned separately
  - So, it takes a long time to learn

Output Values



If B and E both go to C under this policy, how can their values be different?

Temporal Difference Methods

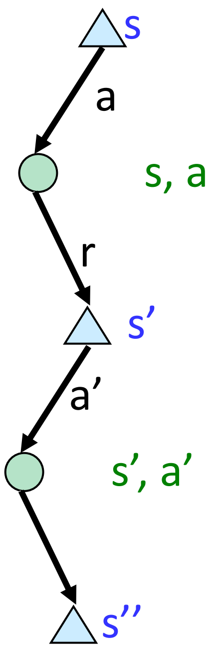
- TD learning is a combination of MC and DP (i.e. Bellman equations) ideas.
  - Like MC methods, can learn directly from raw experience without a model of the environment's dynamics.
  - Like DP, update estimates based in part on other learned estimates, without waiting for a final outcome.

Model-Free Learning

- Model-free (temporal difference) learning
  - Experience world through episodes

$$(s, a, r, s', a', r', s'', a'', r'', s'''' \ldots)$$

  - Update estimates each transition (s, a, r, s')



Q-Learning

- We'd like to do Q-value updates to each Q-state:
  - The Bellman equation for optimal state-action values

$$Q_{k+1}(s,a) \leftarrow \sum_{s'} P(s'|s,a)\left(R(s,a,s') + \gamma \max_{a'} Q_k(s',a)\right)$$

  - But can't compute this update without knowing P, R

- Instead, compute average as we go
  - Receive a sample transition (s,a,r,s')
  - This sample suggests

$$Q(s,a) \approx r + \gamma \max_{a'} Q(s',a')$$

  - But we want to average over results from (s,a) (Why?)
  - So keep a running average

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha)\left[r + \gamma \max_{a'} Q(s',a')\right]$$

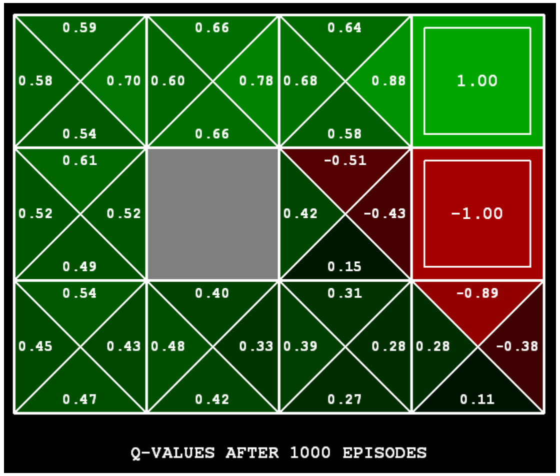
Q-Learning

- Learn Q(s,a) values as you go
  - Receive a sample (s,a,s',r)
  - Consider your old estimate: $Q(s,a)$
  - Consider your new sample estimate:

$$sample = R(s,a,s') + \gamma \max_{a'} Q(s',a')$$

  - Incorporate the new estimate into a running average:

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha)\,[sample]$$



Q-learning Algorithm

Initialize $\hat{Q}(s, a)$ arbitrarily

Repeat (for each episode):

Initialize s

Repeat (for each step of episode):

Choose a from s using a policy derived from $\hat{Q}$

Take action a, receive reward r, observe new state s'

$$\hat{Q}(s, a) \leftarrow \hat{Q}(s, a) + \alpha \left[r + \gamma \max_{a'} \hat{Q}(s', a') - \hat{Q}(s, a) \right]$$

$s \leftarrow s'$

until s is terminal