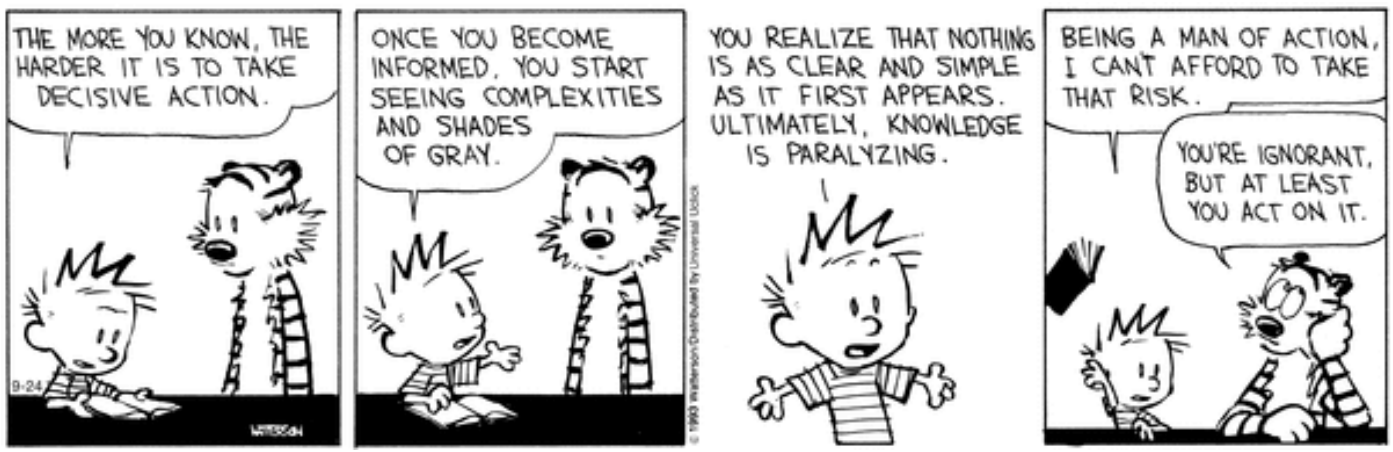
Week 22: Markov Decision Processes

Martha Lewis
(based on slides from Raul Santos Rodriguez)

Have a look at ...

... Russell and Norvig (Ch. 17.1)

... Gym: `https://gym.openai.com/`



Outline

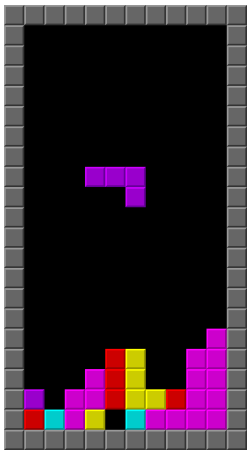
This week we discuss complex decision making. The objective is to present the foundations of Markov Decision Processes and reinforcement learning.

- Sequential decision problems
- Rewards, Utility and Policies
- Value iteration
- Policy iteration
- Reinforcement learning

Motivation

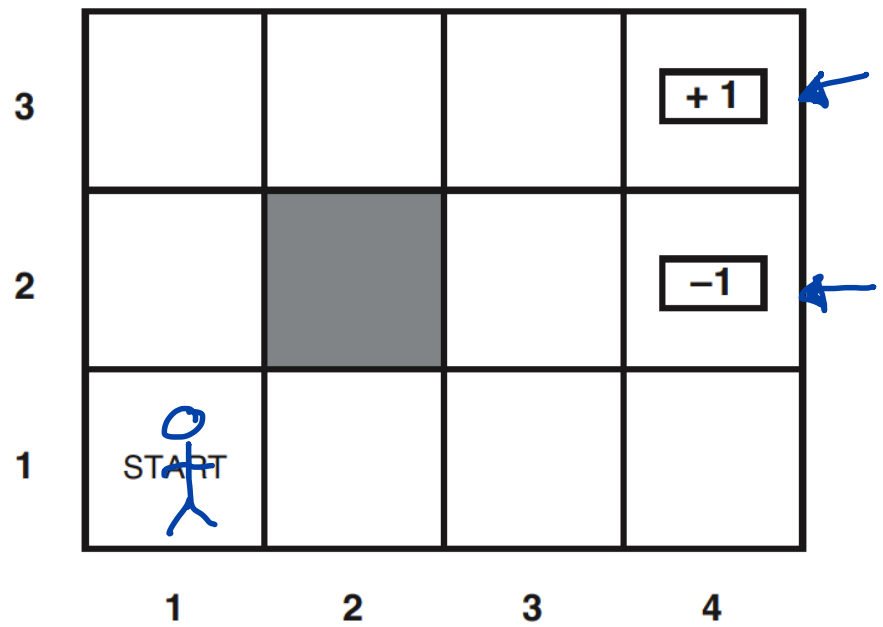
Many important problems are MDPs ...

- Cleaning robot
- Autonomous aircraft navigation
- Games
- Network switching and routing
- Travel route planning
- Models of animals



Motivation

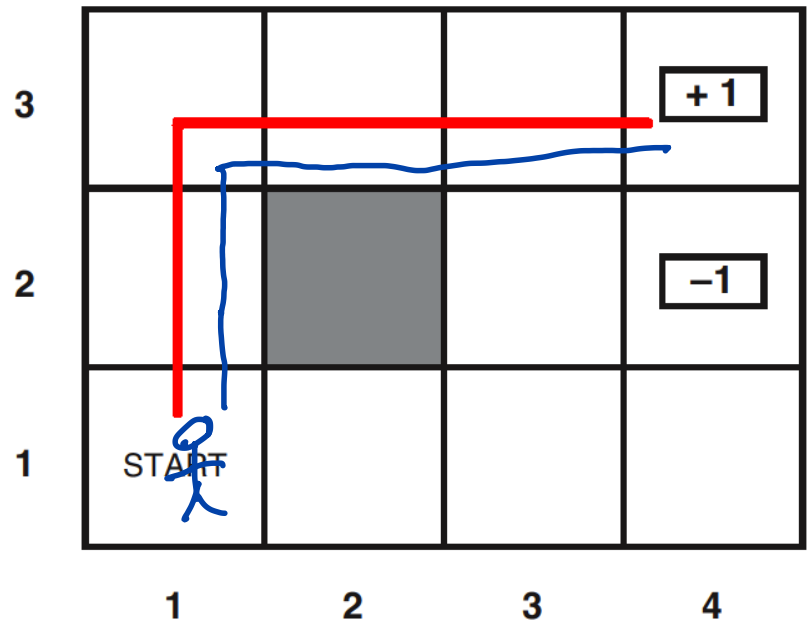
Example: Deterministic Grid World



Action(s)

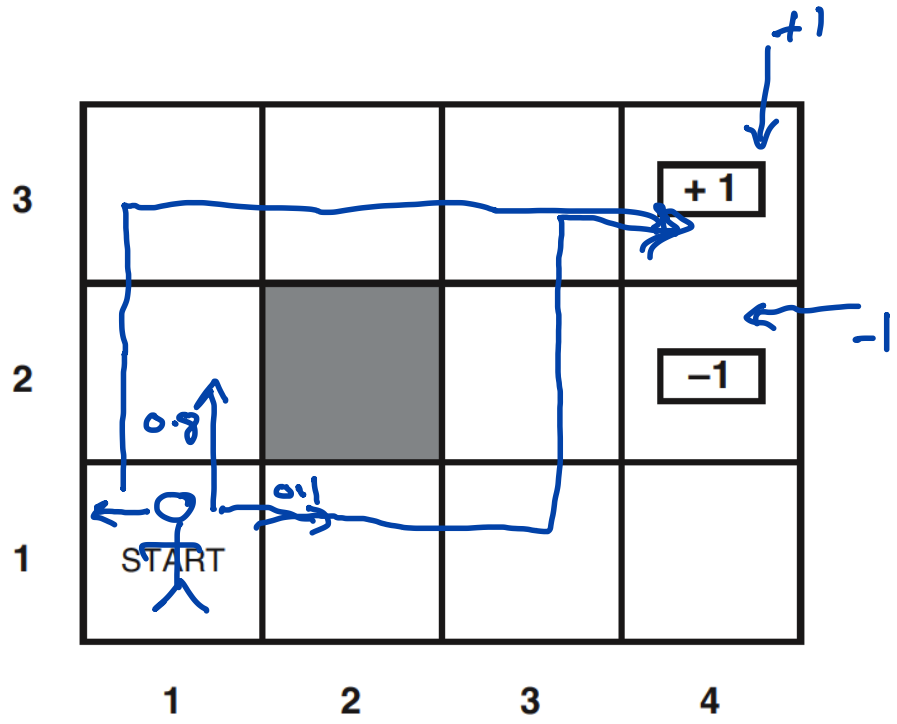
U, D, L, R

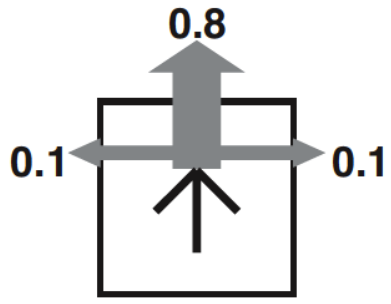
Example: Deterministic Grid World



U, U, R, R, R

Example





Up

$0.8^5 = 0.32768$

Definitions

[Transition model] describes the outcome of each *action* in each *state*.

$P(s'|s,a)$ $\rightarrow$ probability of reaching state s' if action a is done in state s.

[Markov assumption] The probability of reaching s' from s depends only on s and not on the history of earlier states.

Transition model can be represented as a Dynamic Bayesian Network.

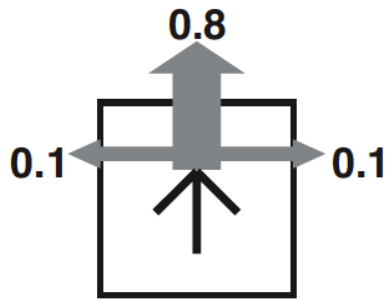
[Rewards] In each state s, we receive a reward $R(s)$ (positive or negative but bounded).

[Utility or Value function] For now, the utility U_h (or V_h) is the sum of the rewards received. The utility function will depend on a sequence of states rather than on a single state.

Example: Stochastic Grid World

3	-0.04	-0.04	-0.04	+1
2	-0.04		-0.04	−1
1	-0.04	-0.04	-0.04	-0.04
	1	2	3	4

$-0.04 \times 10 + 1 = 0.6$



MDP formulation

MDP requires a structure to keep track of the decision sequences:

MDP

- s: state
- s_{start}: starting state
- Actions(s): possible actions
- $P(s'|s,a)$ (or $T(s,a,s')$): probability of s' if take action a in state s
- Reward(s), R(s), r(s): reward for the state s
- Goal(s): whether at the end of the process
- $U_h([s_1, s_2, \ldots])$: utility or value of a sequence of states

Policies

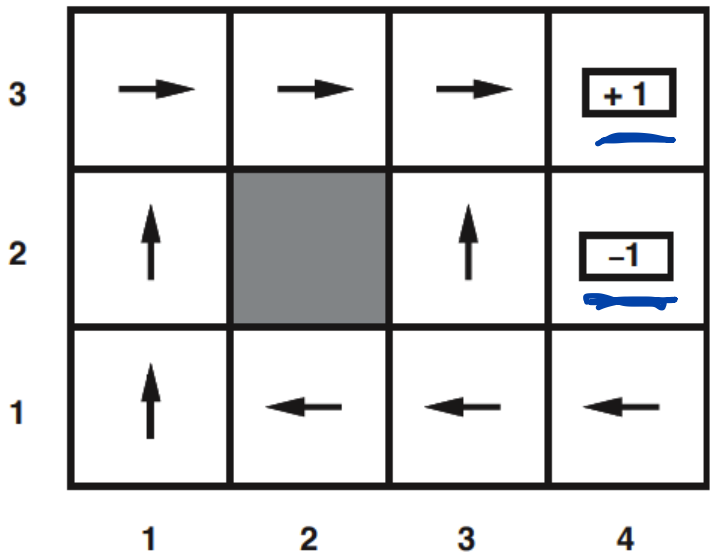
A solution should describe what the robot does in every state: this is called a policy, π.

- $\pi(s)$ for an individual state describes which action should be taken in s.

Each time a given policy is executed starting from the initial state, the stochastic nature of the environment may lead to a different environment history.

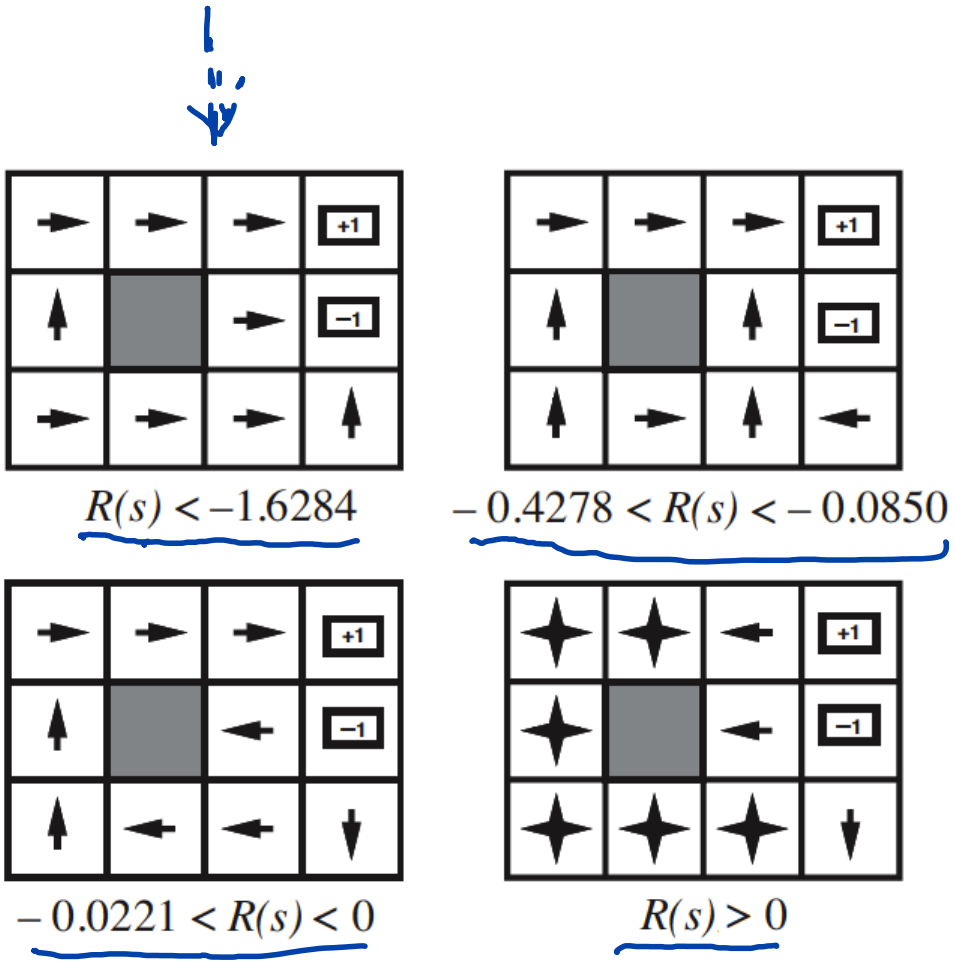
Optimal policy is one that yields the highest *expected utility*, denoted by π^*

Policies: Example



(a)

$R(s) = -0.04$



(b)

Utilities over time

Finite horizon or infinite horizon?

Finite horizon

There is a fixed time N after which nothing matters:

- $\forall k \quad U_h([s_0, s_1, ..., s_{N+k}]) = U_h([s_0, s_1, \ldots, s_N])$
- Leads to non-stationary optimal policies (N matters)

Infinite horizon

Stationary optimal policies (time at state doesn't matter):

- Does **not** mean that all state sequences are infinite; it just means that there is no fixed deadline.
- If two state sequences $[s_0, s_1, s_2, \ldots]$ and $[s'_0, s'_1, s'_2, \ldots]$ begin with the same state (i.e., $s_0 = s'_0$), then the two sequences should be preference-ordered the same way as the sequences $[s_1, s_2, \ldots]$ and $[s'_1, s'_2, \ldots]$.

Additive rewards

Additive rewards

The utility of a state sequence is

$$U_h([s_0, s_1, s_2, \ldots]) = R(s_0) + R(s_1) + R(s_2) + \ldots .$$

Additive rewards

A lecturer gets paid, say, 20K per year.
How much, in total, will the lecturer earn in his/her life?

$$20 + 20 + 20 + 20 + 20 + \ldots = \infty$$

What's wrong with this argument?

Discounted rewards

A reward (payment) in the future is not worth quite as much as a reward now.

- Because of chance of obliteration
- Because of inflation

Discounted rewards

The utility of a state sequence is

$$U_h([s_0, s_1, s_2, \ldots]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \ldots,$$

where the discount factor γ is a number between 0 and 1.

Discount factor makes more distant future rewards less significant!

Infinite horizon rewards

Choosing infinite horizon rewards creates a problem: some sequences will be infinite with infinite reward, **how do we compare them?**

[Solution 1] With discounted rewards, the utility of an infinite sequence is finite. In fact, if $\gamma < 1$ and rewards are bounded by $\pm R_{max}$, we have

$$U_h([s_0, s_1, s_2, \ldots]) = \sum_{t=0}^{\infty} \gamma^t R(s_t) \leq \sum_{t=0}^{\infty} \gamma^t R_{max} = \frac{R_{max}}{(1-\gamma)}$$

geometric sequence

[Solution 2] Under proper policies, i.e. if we will eventually visit terminal state, additive rewards are finite.

[Solution 3] Compare average reward per time step.

Problem

For each round $r = 1, 2, \ldots$

- You choose stay or quit.
- If quit, you get £10 and we end the game.
- If stay, you get £4 and then I roll a 6-sided dice.
  - If the dice results in 1 or 2, we end the game.
  - Otherwise, continue to the next round.

Question

What is the expected utility if we follow the policy stay? and if we follow the policy quit?

Summary

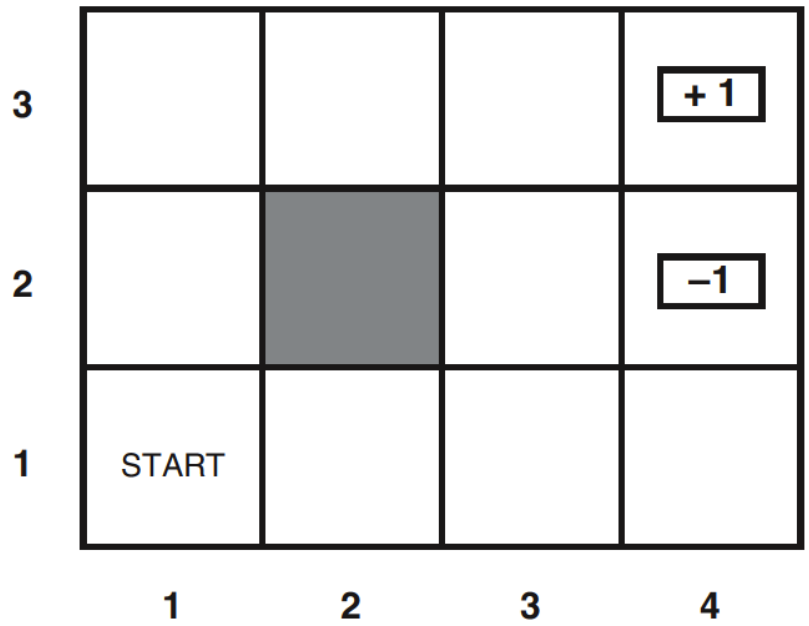
- Markov decision process allows us to model complex decision making
- The probability of moving to state s' depends only on current state s
- Each state has an associated reward
- We can calculate the utility of a sequence of states

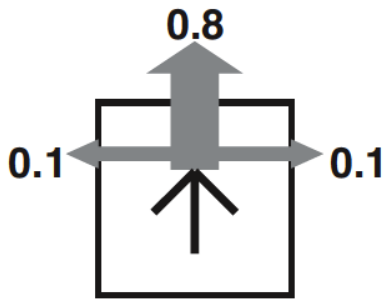
Outline

This video continues with the discussion on complex decision making. The objective is to present two alternatives to deal with Markov Decision Processes:

- Value iteration
- Policy iteration

Example





MDP formulation

MDP requires a structure to keep track of the decision sequences:

MDP

- s: state
- Actions(s): possible actions
- $P(s'|s,a)$ (or $T(s,a,s')$): probability of s' if take action a in state s
- Reward(s): reward for the state s
- Goal(s): whether at the end of the process
- $U_h([s_1, s_2, \ldots])$: utility or value of a sequence of states
- $\pi(s)$: policy describing which action should be taken in s

Optimal policies

Expected utility/value

The expected utility obtained by executing π starting in s is given by

$$U^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^{t} R(s_t)\right]$$

Optimal policies

Out of all the policies we could choose to execute starting in s, one will have higher expected utilities than all the others. This is an optimal policy when s is the starting state,

$$\pi_s^* = \arg\max_{\pi} U^{\pi}(s)$$

Optimal policies

Discounted utilities + infinite horizons $\rightarrow$ optimal policy is independent of the starting state

Note

If policy π_a^* is optimal starting in state a and policy π_b^* is optimal starting in b, then, when they reach a third state c, there's no good reason for them to disagree with each other, or with π_c^*, about what to do next. So we will write π^* for an optimal policy.

True utility of a state is $U^{\pi^*}(s)$ or just $U(s)$: *expected* sum of *discounted rewards* if the robot executes an *optimal policy*.

Value iteration

Idea: Calculate the utility of each state and then select optimal action based on these utilities

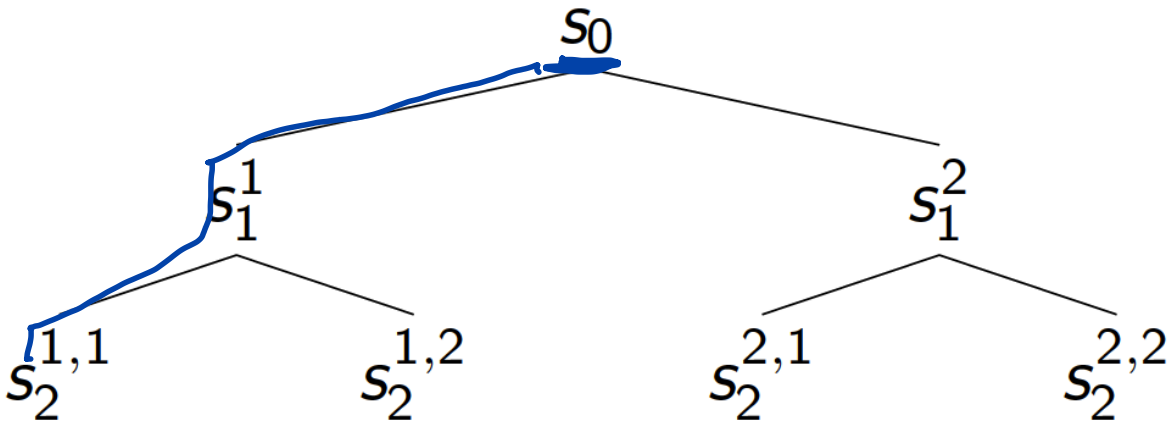
Criterion for optimal policy

Assuming discounted rewards,

$$\pi^* = \arg\max_{\pi} E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t)|\pi\right]$$

Value iteration: criterion

Each policy π yields a tree, with root node s_0, where the children of node s are the possible successor states given the action $\pi(s)$.



$P(s'|a, s)$ gives the probability of traversing an edge from s to s'.

The **expectation** involves two steps:

1. For each path in the tree, getting the product of the (joint) probability of the path in this tree with its discounted reward, and then
2. Summing over all the products from (1)

Value iteration: utilities of states

Some remarks

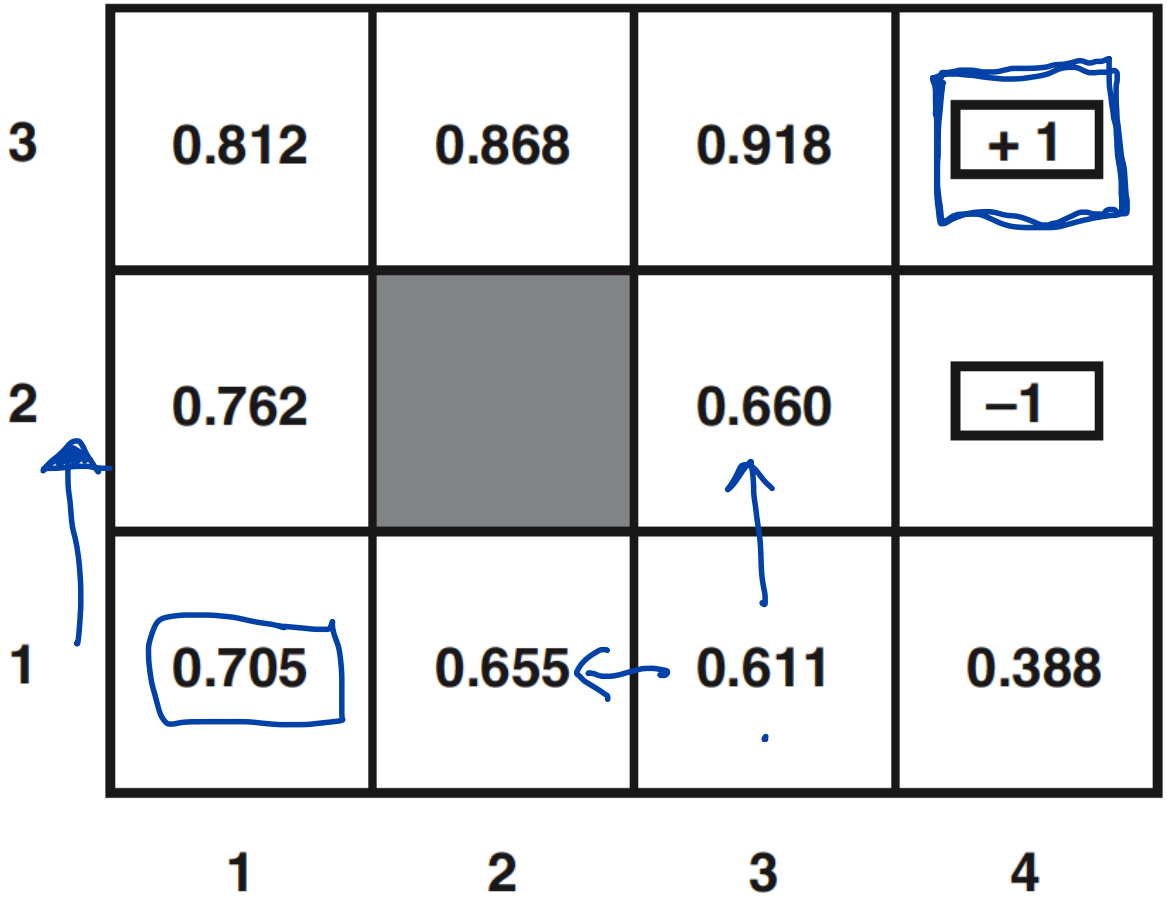
- $R(s)$ is reward for being in s *now*: is the short term reward for being in s
- $U(s)$ is utility of the states that might follow s: $U(s)$ captures long term advantages from being in s
- $U(s)$ reflects what you can do from s;
- $R(s)$ does not.

States that follow depend on π. Utility of s given π is,

$$U^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t) | \pi, s_0 = s\right]$$

Value iteration: example

$\gamma = 1$ and $R(s) = -0.04$ for nonterminal states.



Value iteration: Bellman equations

Optimal policy

Given $U(s)$, we can easily determine the optimal policy

$$\pi^*(s) = \arg\max_{a \in A(s)} \sum_{s'} P(s'|a,s)U(s') \tag{1}$$

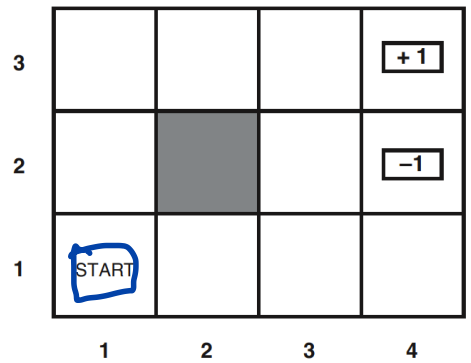
There is a direct relationship between the utility of a state and the utility of its neighbours.

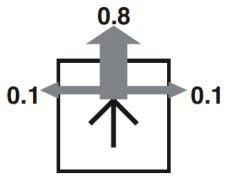
Bellman equations (1957)

Utility of a state is the immediate reward plus expected utility of subsequent states if we choose the optimal action,

$$U(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|a,s)U(s')$$

Value iteration: example





Bellman equation for the state (1,1) is

$$U(1,1) = -0.04 + \gamma \max\{0.8U(1,2) + 0.1U(2,1) + 0.1U(1,1), (Up)$$
$$0.9U(1,1) + 0.1U(1,2), (Left)$$
$$0.9U(1,1) + 0.1U(2,1), (Down)$$
$$0.8U(2,1) + 0.1U(1,2) + 0.1U(1,1), (Right)\}$$

***Up* is the best action**

Value iteration: algorithm

For n states we have n Bellman equations with n unknowns (utilities of states)

Value iteration is an *iterative* approach to solving the n equations.

Intuition

We start with arbitrary values and update them as follows

$$U_{i+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|a, s) U_i(s')$$

The algorithm converges to right and unique solution.

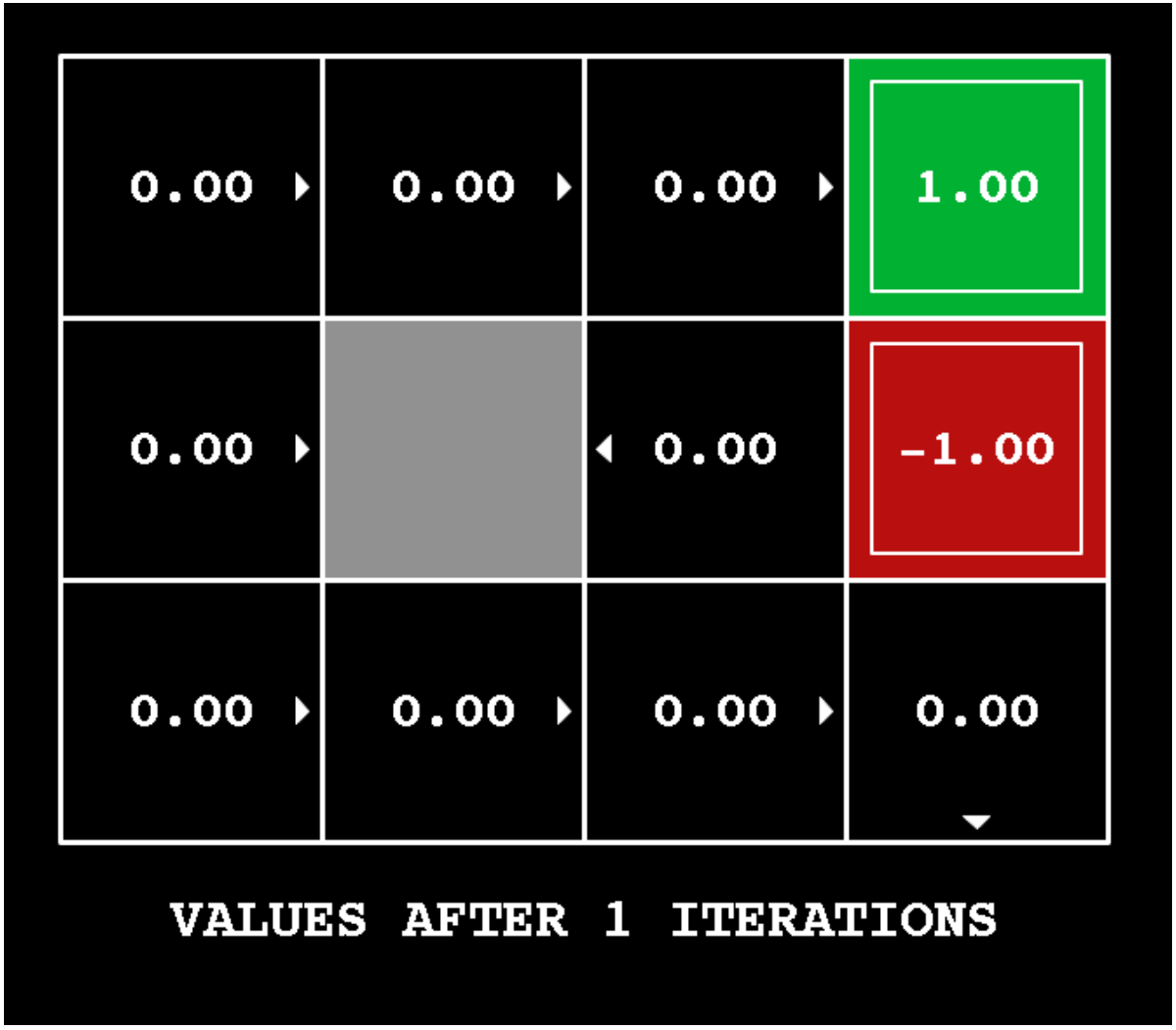
Value iteration: algorithm

function VALUE-ITERATION(mdp, ϵ) **returns** a utility function
  inputs: mdp, an MDP with states S, actions $A(s)$, transition model $P(s' \mid s, a)$,
    rewards $R(s)$, discount γ
    ϵ, the maximum error allowed in the utility of any state
  local variables: U, U', vectors of utilities for states in S, initially zero
    δ, the maximum change in the utility of any state in an iteration

  repeat
    $U \leftarrow U'$; $\delta \leftarrow 0$
    for each state s **in** S **do**
      $U'[s] \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\, U[s']$
      if $|U'[s] - U[s]| > \delta$ **then** $\delta \leftarrow |U'[s] - U[s]|$
  until $\delta < \epsilon(1-\gamma)/\gamma$
  return U

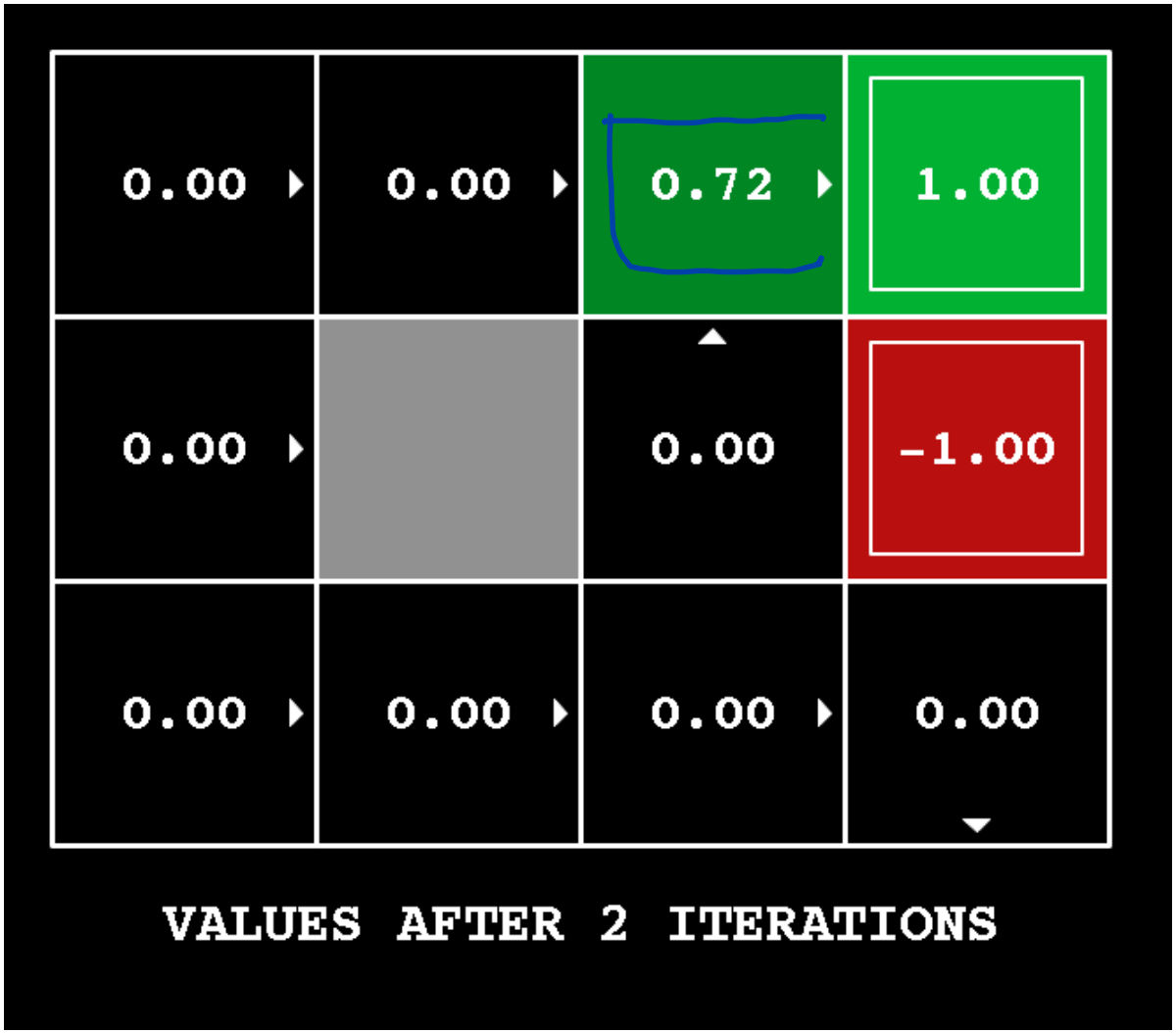
Value iteration: example

4x3 grid world with $\gamma = 0.9$ and $R(s) = 0$ for nonterminal states



Value iteration: example

4x3 grid world with $\gamma = 0.9$ and $R(s) = 0$ for nonterminal states



$R(s) = 0$

$0.8 \times 1 \times 0.9 = 0.72$

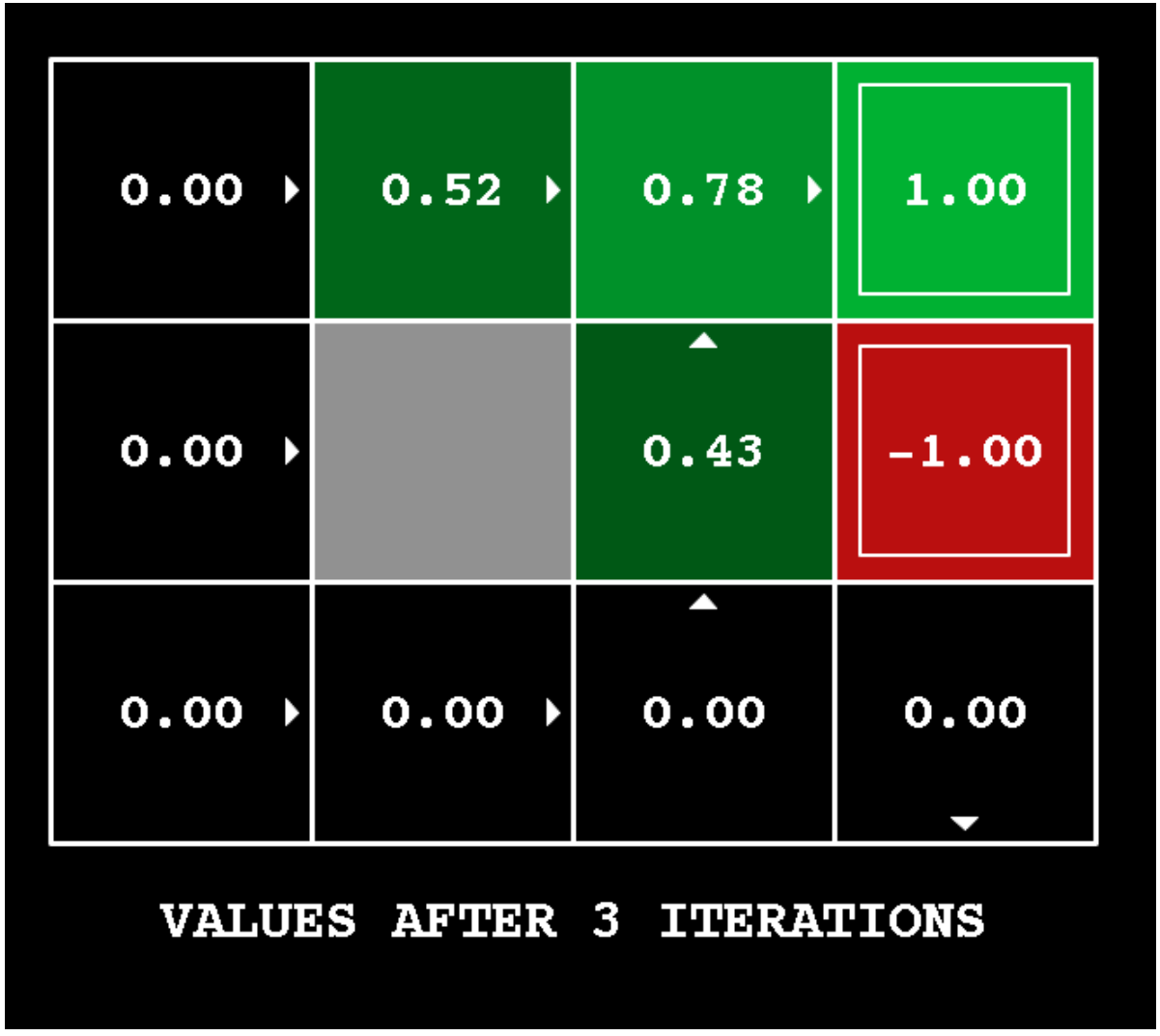
$+ 0.1 \times 0$

$+ 0.1 \times 0$

$0 + 0.72 + 0 + 0$

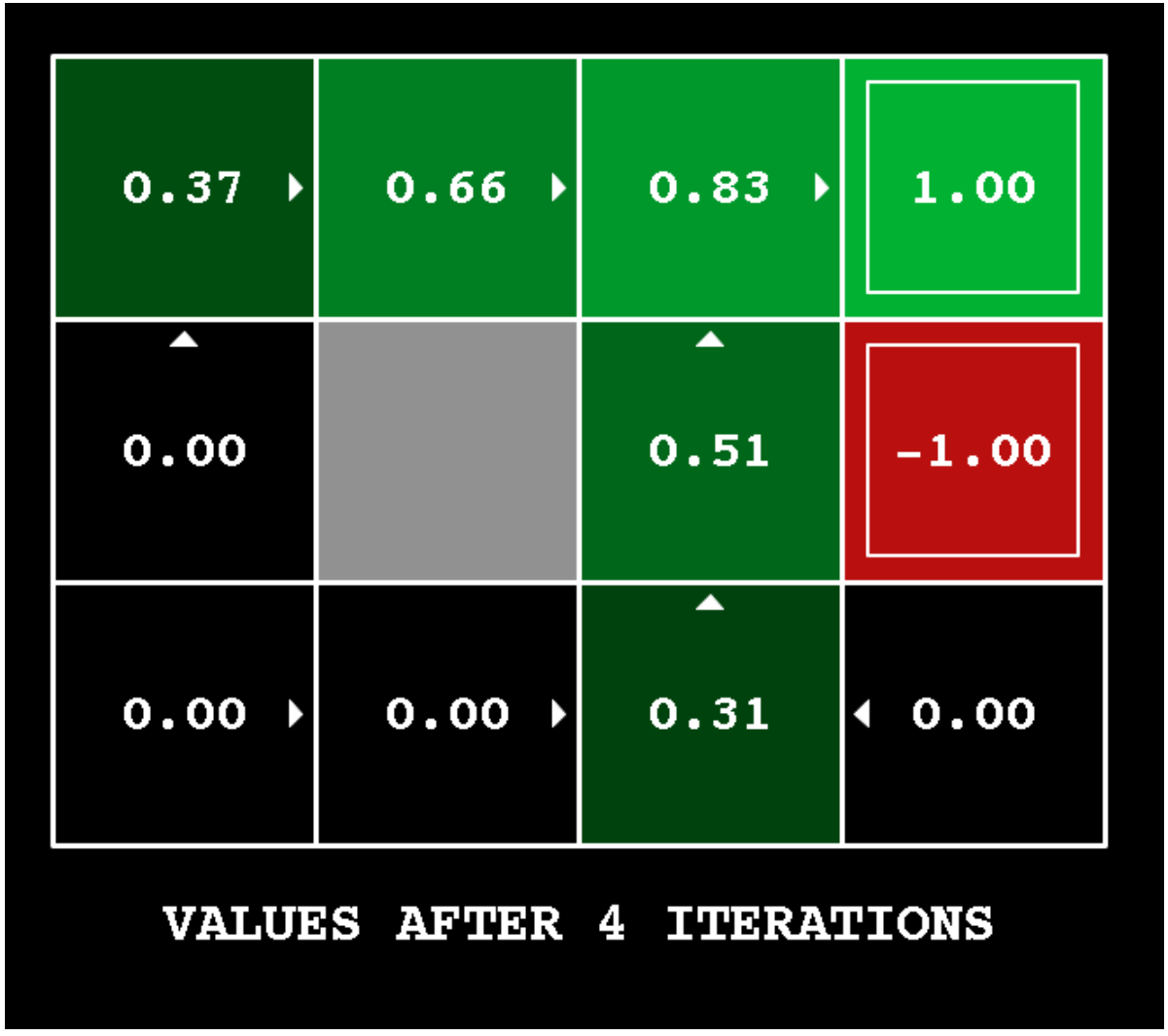
Value iteration: example

4x3 grid world with $\gamma = 0.9$ and $R(s) = 0$ for nonterminal states



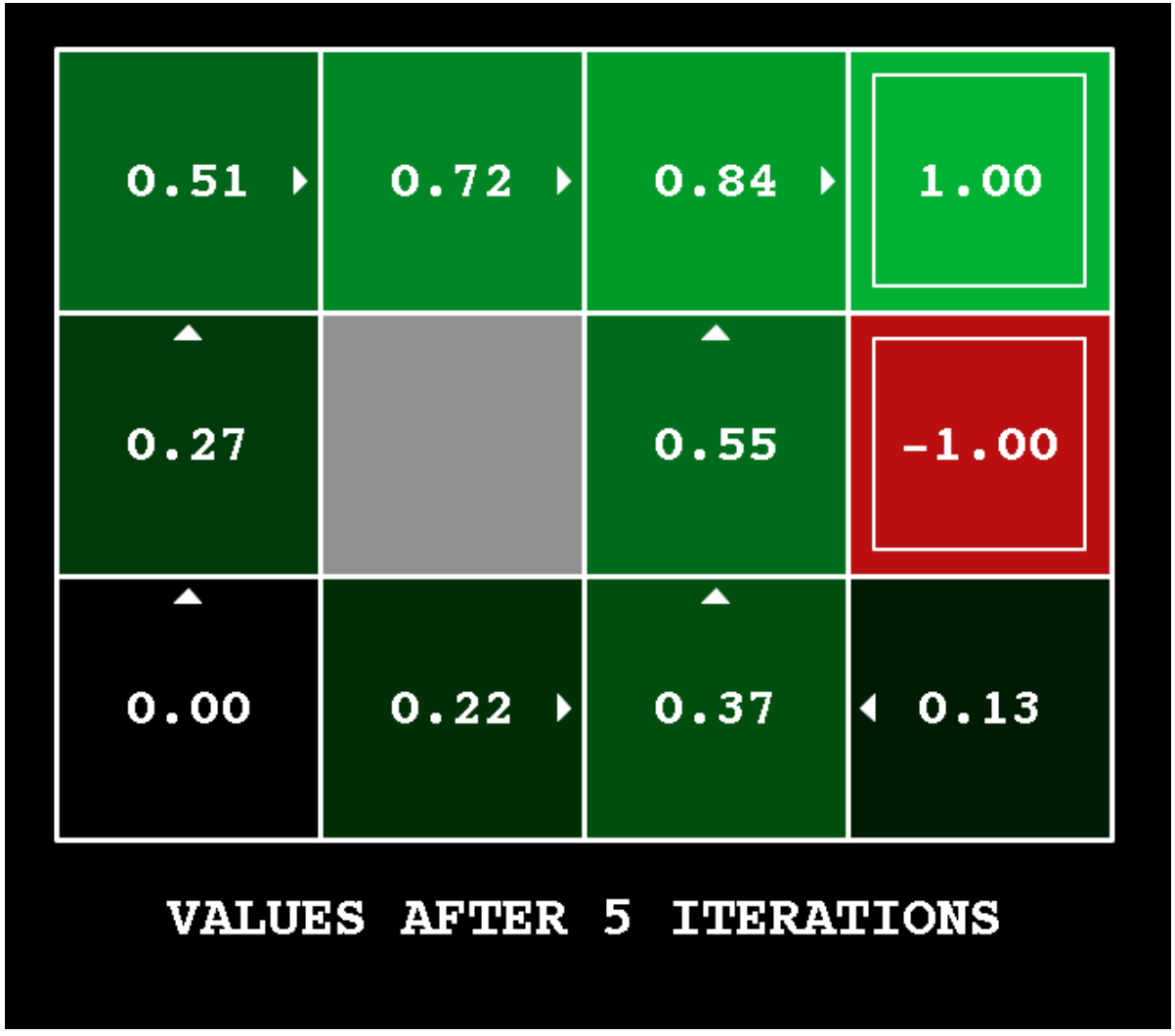
Value iteration: example

4x3 grid world with $\gamma = 0.9$ and $R(s) = 0$ for nonterminal states



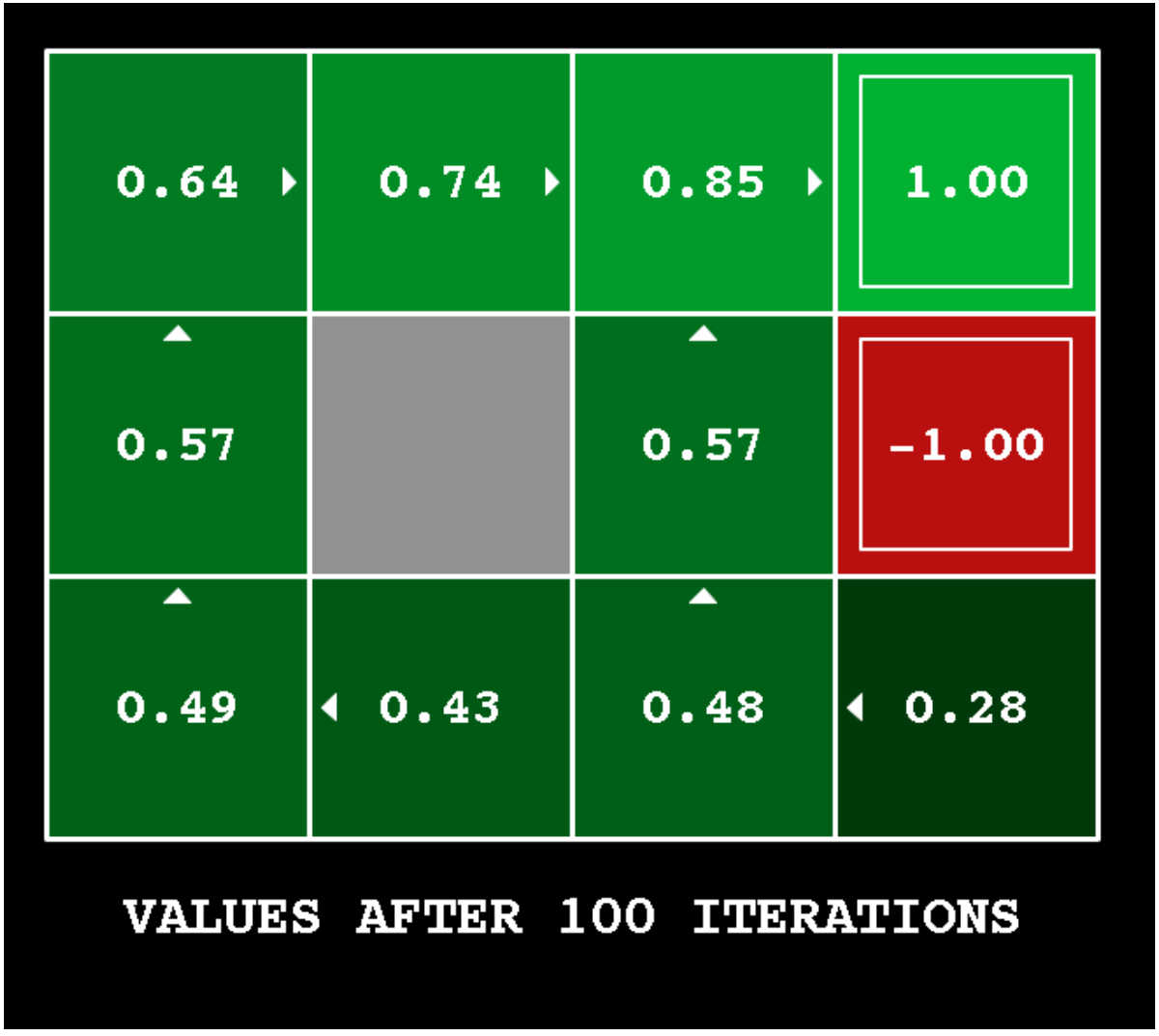
Value iteration: example

4x3 grid world with $\gamma = 0.9$ and $R(s) = 0$ for nonterminal states



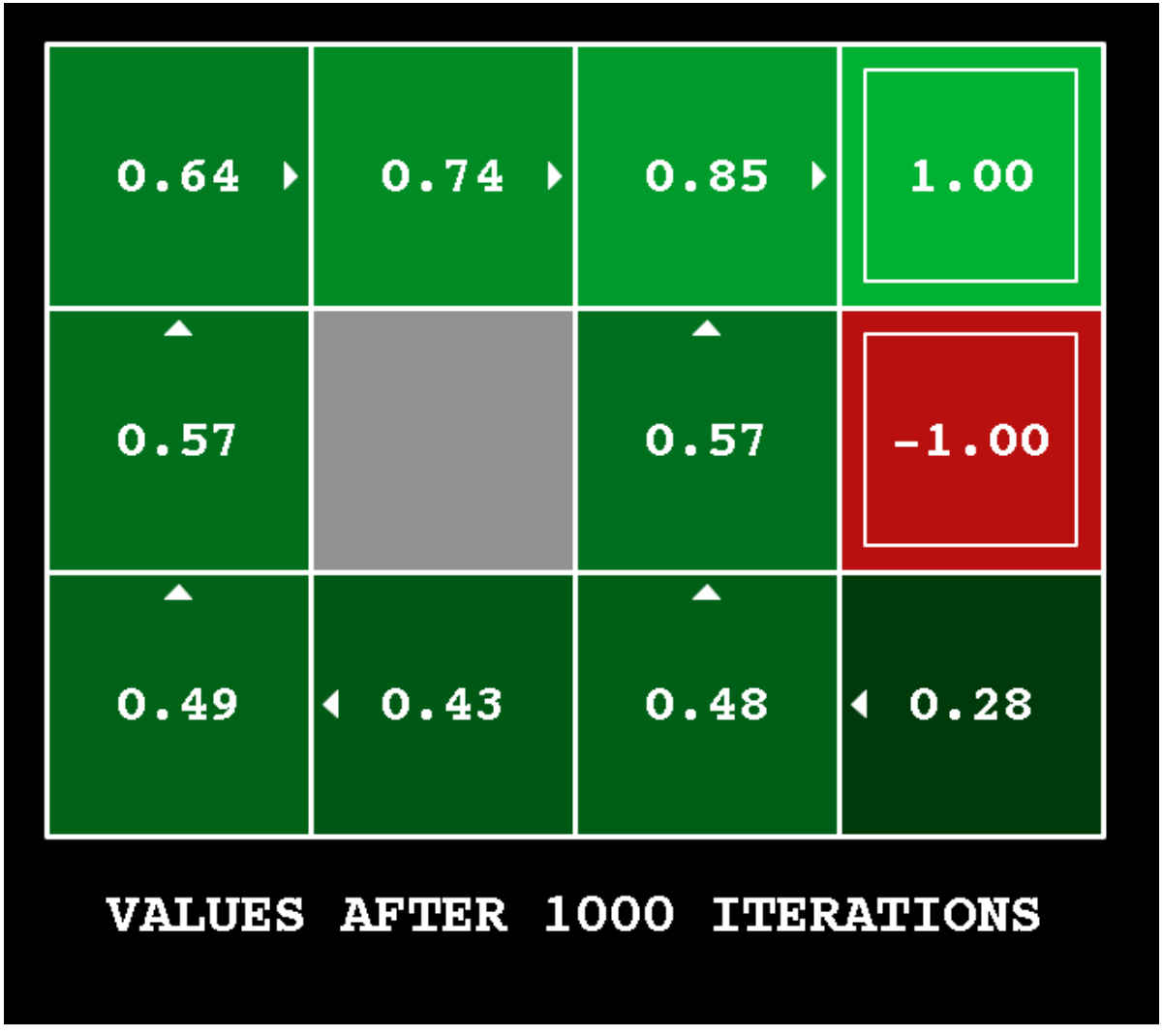
Value iteration: example

4x3 grid world with $\gamma = 0.9$ and $R(s) = 0$ for nonterminal states



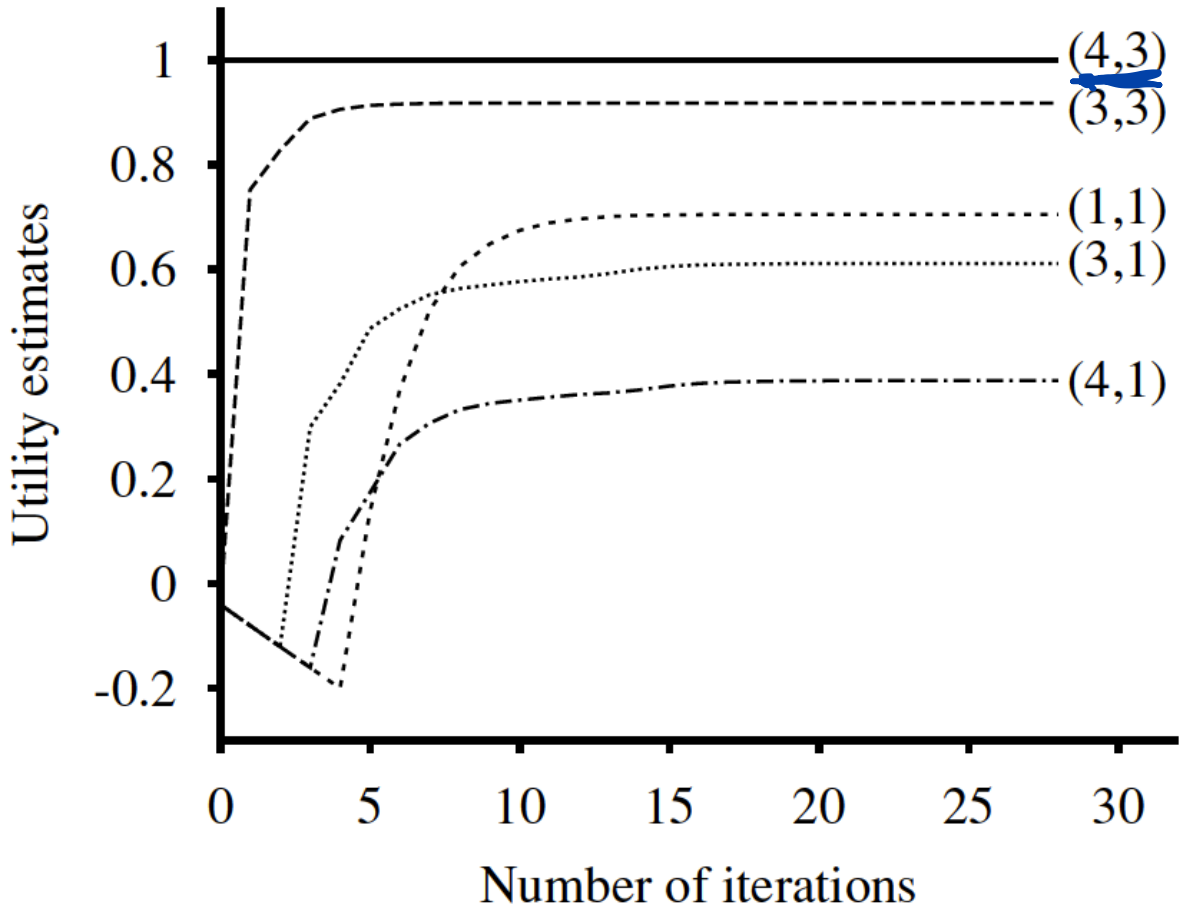
Value iteration: example

4x3 grid world with $\gamma = 0.9$ and $R(s) = 0$ for nonterminal states



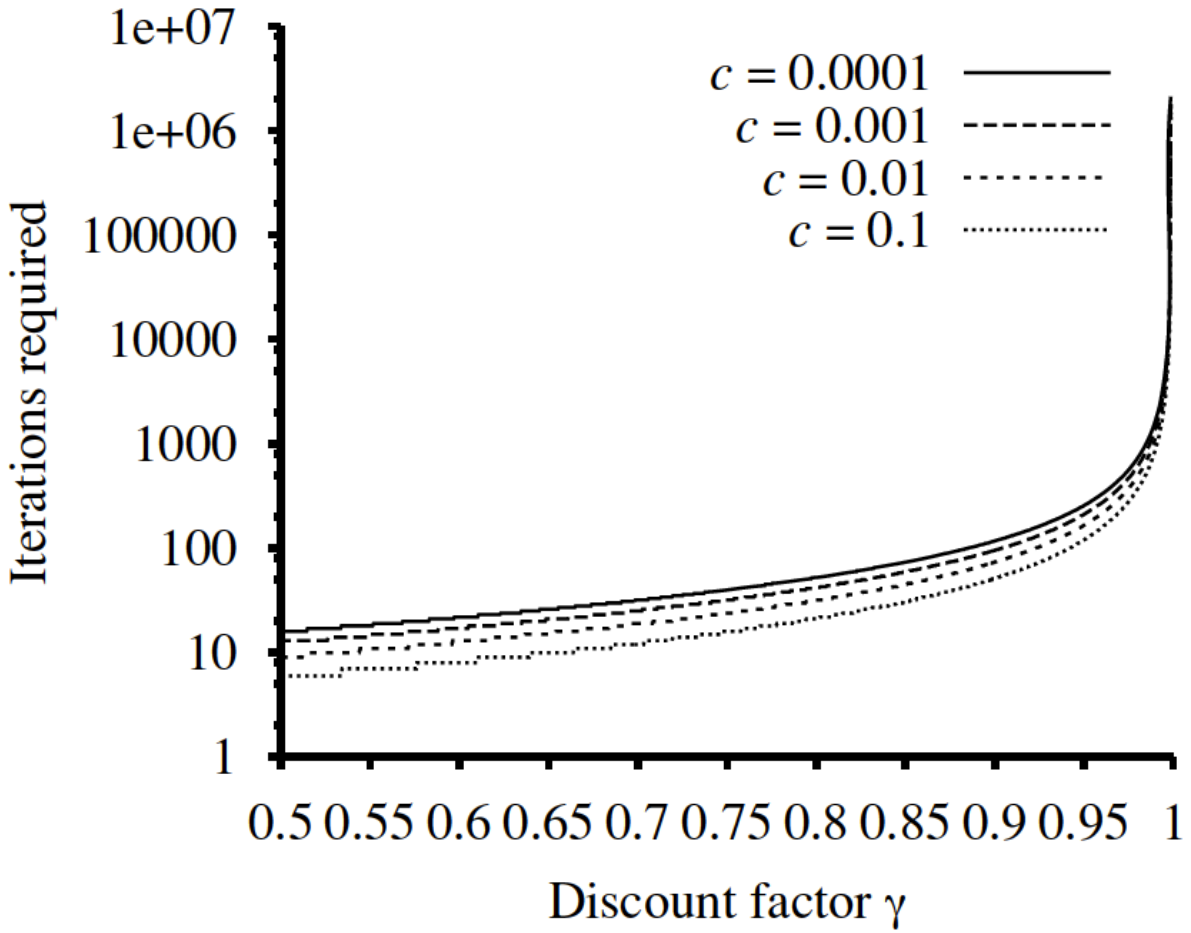
Value iteration: algorithm

4x3 grid world with $R(s) = -0.04$ for nonterminal states



Value iteration: algorithm

4x3 grid world with $R(s) = -0.04$ for nonterminal states



Summary

- Value iteration allows us to find the best policy by iteratively updating the value (utility) at each cell.
- We calculate the value of each cell based on the previous values.

Policy iteration

Idea: if one action is clearly better than all others, then the exact magnitude of the utilities on the states involved need not be precise.

Algorithm

Policy iteration alternates two steps, beginning from some initial policy π_0:

1. **Policy evaluation**: given a policy π_i, calculate $U_i = U^{\pi_i}$, the utility of each state if π_i were to be executed.
2. **Policy improvement**: Calculate a new MEU policy π_{i+1}, using one-step look-ahead based on U_i (using Eq. 1).

The algorithm terminates when the policy improvement step yields no change in the utilities.

Policy iteration: policy evaluation

Policy improvement is easy, but policy evaluation?

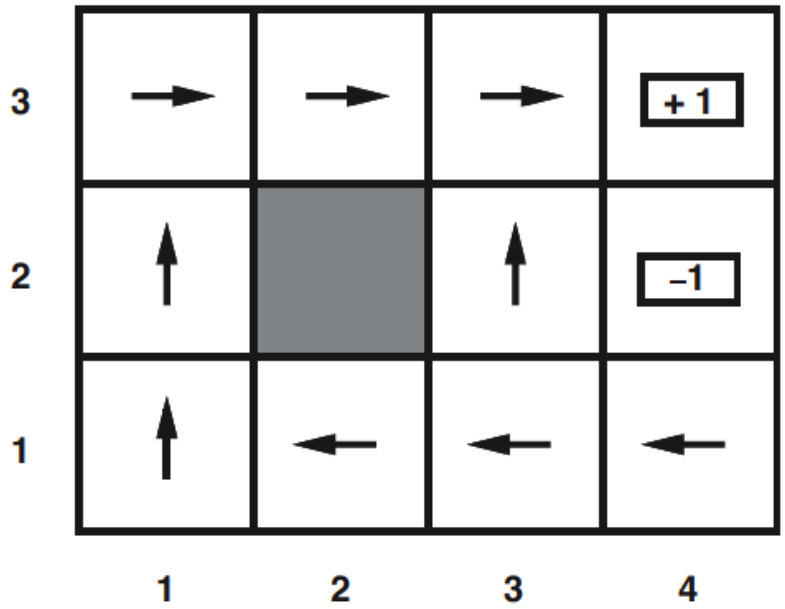
Policy evaluation

- Simpler than solving the standard Bellman equations: the action in each state is fixed by the policy.
- At the ith iteration, the policy π_i specifies the action $\pi_i(s)$ in state s: simplified version of the Bellman equation relating the utility of s (under π_i) to the utilities of its neighbours:

$$U_i(s) = R(s) + \gamma \sum_{s'} P(s'|s, \pi_i(s)) U_i(s')$$

don't have to maximise

Policy iteration: example



E.g. $\pi_i(1,1) = Up$, $\pi_i(1,2) = Up$, ...

The simplified Bellman equations are

$$U_i(1,1) = -0.04 + 0.8U_i(1,2) + 0.1U_i(1,1) + 0.1U_i(2,1),$$

$$U_i(1,2) = -0.04 + 0.8U_i(1,3) + 0.2U_i(1,2),$$

...

Policy iteration: algorithm

function POLICY-ITERATION(mdp) **returns** a policy
  inputs: mdp, an MDP with states S, actions $A(s)$, transition model $P(s' \mid s, a)$
  local variables: U, a vector of utilities for states in S, initially zero
    π, a policy vector indexed by state, initially random

  repeat
    $U \leftarrow$ POLICY-EVALUATION(π, U, mdp)
    $unchanged? \leftarrow$ true
    for each state s **in** S **do**
      if $\max_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\ U[s'] > \sum_{s'} P(s' \mid s, \pi[s])\ U[s']$ **then do**
        $\pi[s] \leftarrow \operatorname{argmax}_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\ U[s']$
        $unchanged? \leftarrow$ false
  until $unchanged?$
  return π

Policy iteration vs Value iteration

The equations are now **linear**: the max operator has been removed.

For n states, we have n linear equations with n unknowns, which can be solved exactly in time $O(n^3)$ by standard linear algebra methods.

When to use Policy iteration?

- For small state spaces: **policy evaluation** using exact solution methods is often the most efficient approach, typically very fast and converges quickly.
- For large state spaces, $O(n^3)$ time might be prohibitive. **Value iteration** is preferred.
- For very large state spaces: use an **approximation** but optimality guarantee is lost.

Summary

The road so far

- We now have (tediously) gathered all the ingredients to build MDPs.

- Transition models are described by a DBN

- Decisions will be made by projecting forward possible action sequences and choosing the best one.

MDPs are great, if

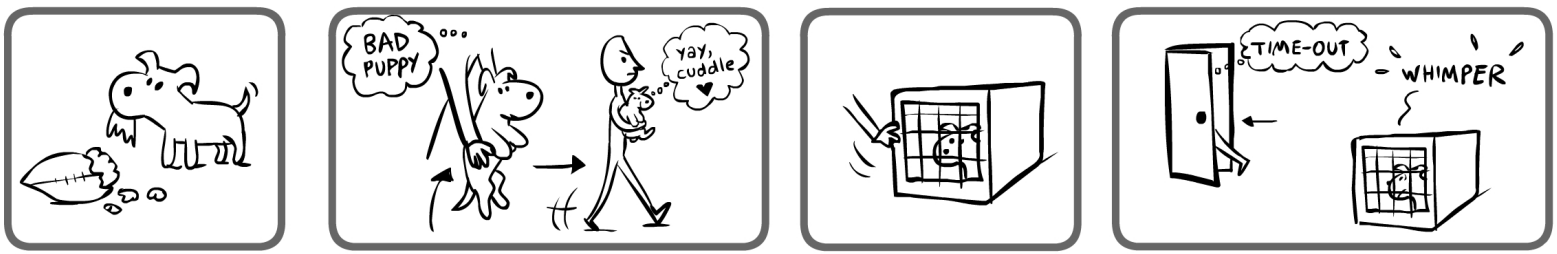
... we know the state transition function $P(s'|a, s)$

... we know the reward function $R(s)$

But what if we don't?

Like when we were babies

Reinforcement learning



What we think the puppy is learning: **CHEWING UP STUFF IS BAD.**
What the puppy is really learning: **CHEWING UP STUFF is AWESOME. CRATE is BAD.**

[https://www.youtube.com/user/stanfordhelicopter]

Reinforcement learning

- Actions have non-deterministic effects: initially unknown
- Rewards / punishments are infrequent and often at the end of long sequences of actions
- Learner must decide what actions to take
- World is large and complex

Reinforcement learning

Reinforcement learning is **on-line** methodology that we use when the **model of world is unknown** and/or **rewards are delayed**.

Reinforcement learning: Naive approach

Naive Approach

1. Act randomly for a while (or systematically explore all possible actions)
2. Learn Transition model and Reward function
3. Use value iteration, policy iteration, ...

Problems?

Reinforcement learning: basic approaches

Exploration vs Exploitation

Exploit: use your learning result to maximise expected utility now, according to your learned model

Explore: choose an action that will help you improve your model

- How to explore
  - choose a random action
  - choose an action you haven't chosen yet
  - choose an action that will take you to unexplored states
- How to exploit: follow policy
- When to explore

Model-based reinforcement learning

- Learn MDP
- Solve the MDP to determine optimal policy
- Treat the difference between expected / actual reward as an error signal

Model-free reinforcement learning

Learn the (utility) function directly: Q-learning, TD learning

Q-learning

An action-value function $Q(s, a)$ says how good it is to be in a state, take an action, and thereafter follow a policy.

Algorithm

1. Initialise $Q(s, a)$ arbitrarily
2. Choose actions in any way such that all actions are taken in all states (infinitely often in the limit)
3. On each time step, change one element of $Q(s, a)$

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(R + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t))$$

4. If desired, reduce the step-size parameter α over time

Theorem

Q-learning control converges to the optimal action-value function.

Deep Q-learning

Approximate the action-value function with a neural network with weights w

$$Q(s, a, w)$$

Define an objective function by mean-squared error in Q-values

$$L(w) = E[(R + \gamma \max_{a'} Q(s', a', w) - Q(s, a, w))]$$

With the following gradient

$$\frac{\partial L(w)}{\partial w} = E\left[(R + \gamma \max_{a'} Q(s', a', w) - Q(s, a, w)) \frac{\partial Q(s, a, w)}{\partial w}\right]$$

Optimise using Stochastic Gradient Descent!

---

https://arxiv.org/pdf/1312.5602.pdf

http://projects.rajivshah.com/rldemo/

Summary

- Markov decision processes allow us to model complex decision making
- If we know the rewards at each state, we can determine the optimal policy using the Bellman equations
- These can be solved via value iteration or policy iteration
- When we do not know the value of each state this can be learned using reinforcement learning