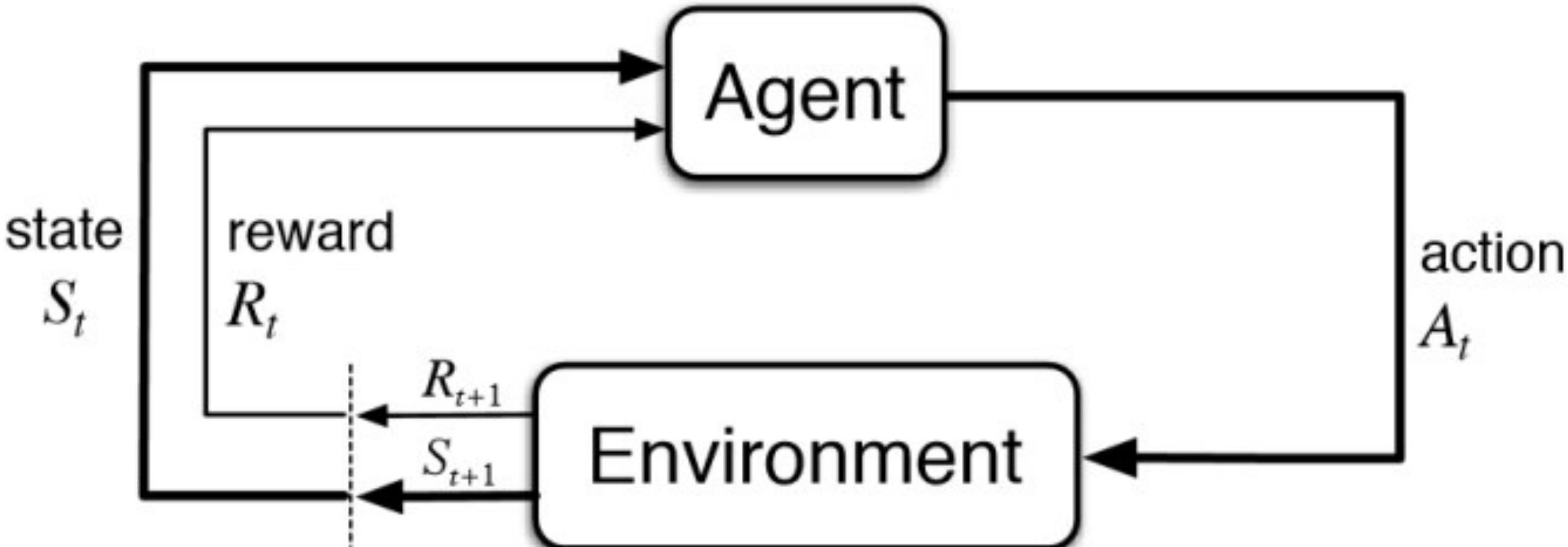
Lecture 19
Reinforcement Learning

Lei Li, **Yu-Xiang Wang**

An RL agent learns interactively through the feedbacks of an environment.



- Learning how the world works (dynamics) and how to maximize the long-term reward (control) at the same time.

Reinforcement learning problem setup

- State, Action, Reward and Observation

$$S_t \in \mathcal{S} \quad A_t \in \mathcal{A} \quad R_t \in \mathbb{R} \quad O_t \in \mathcal{O}$$

- Policy: $\pi : \mathcal{S} \rightarrow \mathcal{A}$
  - When the state is observable:
  - Or when the state is not observable

$$\pi_t : (\mathcal{O} \times \mathcal{A} \times \mathbb{R})^{t-1} \rightarrow \mathcal{A}$$

- Learn the best policy that maximizes the expected reward
  - Finite horizon (episodic) RL: $\pi^* = \arg\max_{\pi \in \Pi} \mathbb{E}[\sum_{t=1}^{T} R_t]$
  - Infinite horizon RL:

$$\pi^* = \arg\max_{\pi \in \Pi} \mathbb{E}[\sum_{t=1}^{\infty} \gamma^{t-1} R_t]$$

T: horizon

γ: discount factor

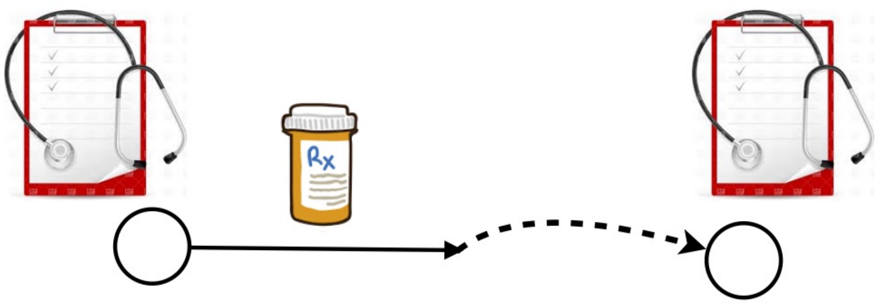
RL for robot control

- States: The physical world, e.g., location/speed/acceleration and so on.
- Observations: camera images, joint angles
- Actions: joint torques
- Rewards: stay balanced, navigate to target locations, serve and protect humans, etc.

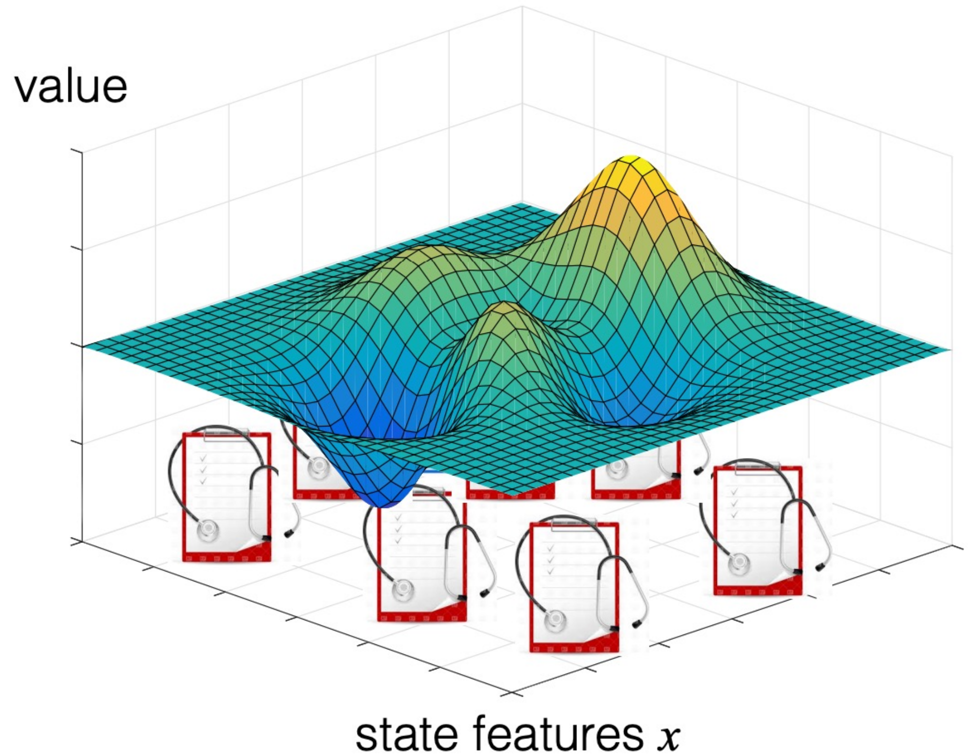
RL for Inventory Management

- State: Inventory level, customer demand, competitor’s inventory
- Observations: current inventory levels and sales history
- Actions: amount of each item to purchase
- Rewards: profit

RL for Adaptive medical treatment



- State: diagnosis
- Action: treatment
- Reward: progress in recovery



(example / illustration due to Nan Jiang)

Example: Supervised learning vs RL in movie recommendation

- Bob is described by a feature vector
  - s =[Previous movies watched / Rating / Written reviews]

- Supervised learning predicts how likely Bob will click on “aliens vs predators”

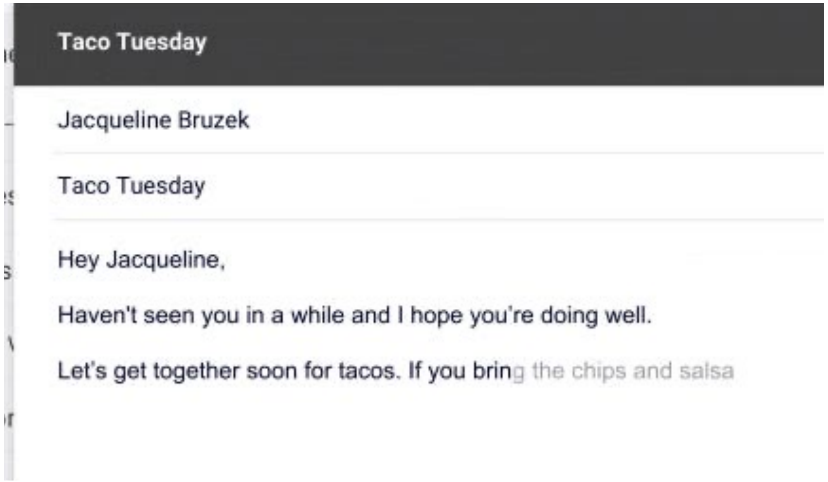
- Reinforcement learning aims at controlling Bob
  - So in the future, Bob will develop a taste for “aliens vs predators” (e.g., from having watched “aliens” and “predators” both).

A broader view: Let’s consider a few other machine learning tasks

- Hospitals need to decide **who to test** based on symptoms and other patient attributes

- Train a classifier on historic records to predict the test outcome.
- The accuracy is high on a holdout set!

- Large tech wants to improve user experience on their popular email service



- Train a **large language model** with user data to **complete sentences**
- It seems to work great!

What could go wrong?

Every machine learning problem is secretly a control (or RL) problem

- If I test patients using the new rule, the distribution of patients receiving the test will be different!
- Should I still trust my classifier?

- If I deploy the new "Guess what you will write" prompt, what users will enter may change!
- Is the model fulfilling its own prophecy?

The ultimate goal is NOT prediction, but to:
minimize disease transmission / maximize user experience!

Reinforcement learning is very challenging

- The agent needs to:
  - Learn the state-transitions ----- How the world works
  - Learning the costs / rewards ----- Cost of actions
  - Learning how to search ----- Come up with a good strategy

- All at the same time

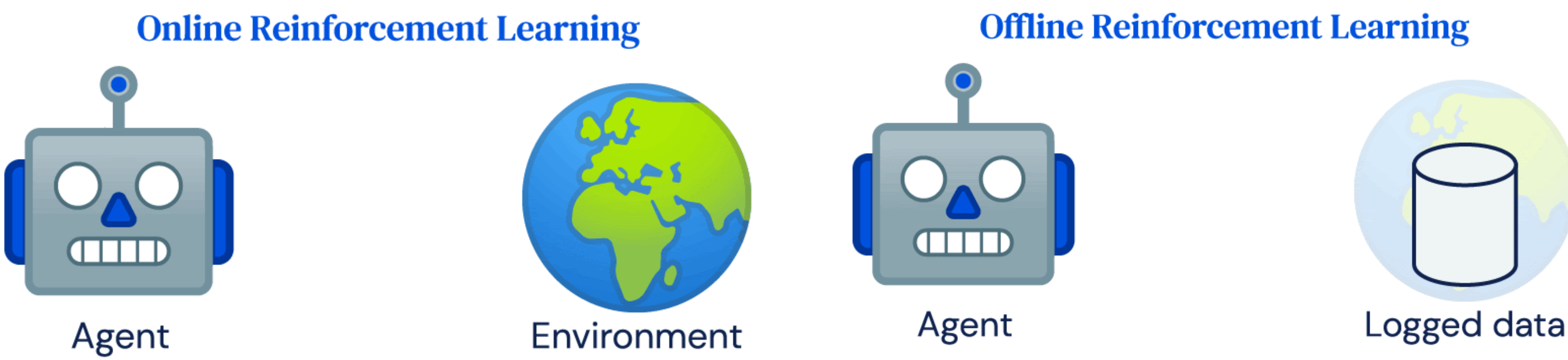
Let us tackle different aspects of the RL problem one at a time

- **Markov Decision Processes: (this lecture)**
  - Dynamics are given no need to learn. planning only.

- RL algorithms (this lecture and the next)
  - Model-based RL vs Model-free RL
  - Temporal difference learning
  - Function approximation

- Exploration (final lecture if time permits)
  - Bandits: Explore-Exploit in simple settings
  - RL: Explore-Exploit in Learning MDPs

Online RL vs Offline RL



Exploration is often **expensive**, **unsafe**, **unethical** or **illegal** in practice, e.g., in self-driving cars, or in medical applications.

Can we learn a policy from already **logged interaction data**?

***Offline RL won't be covered, but it's an important problem**

Let's start by formulating Markov Decision processes (MDP).

- Infinite horizon / discounted setting

$$\mathcal{M}(\mathcal{S}, \mathcal{A}, P, r, \gamma, \mu)$$

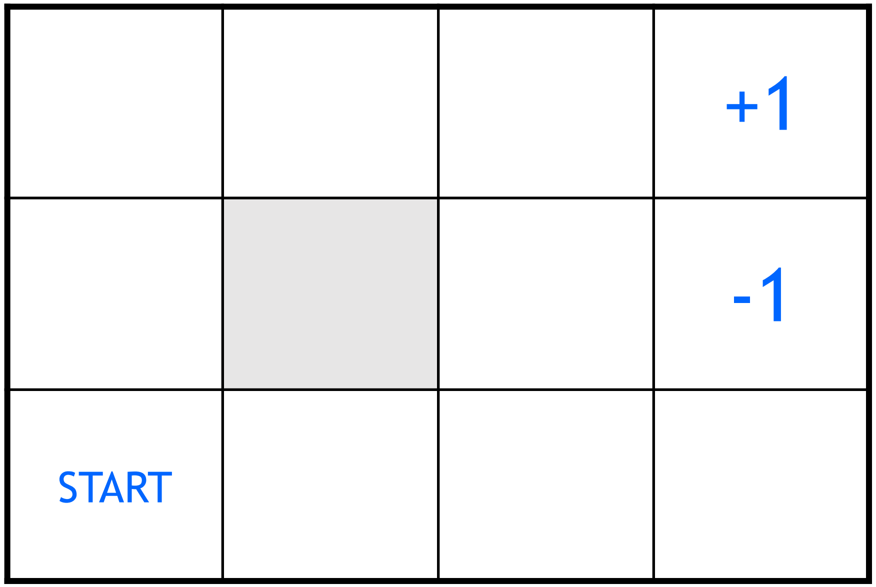
Transition kernel: $P: \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ i.e. $P(s'|s,a)$

(Expected) reward function: $r: \mathcal{S} \times \mathcal{A} \to \mathbb{R} / [0, R_{max}]$ $\quad \mathbb{E}[R_t \mid S_t = s, A_t = a] =: r(s,a)$

Initial state distribution $\mu \in \Delta(\mathcal{S})$

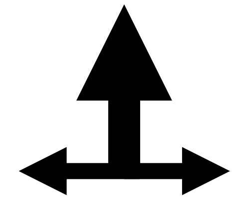
Discounting factor: γ

Example: Frozen lake.



actions: UP, DOWN, LEFT, RIGHT

UP e.g.,



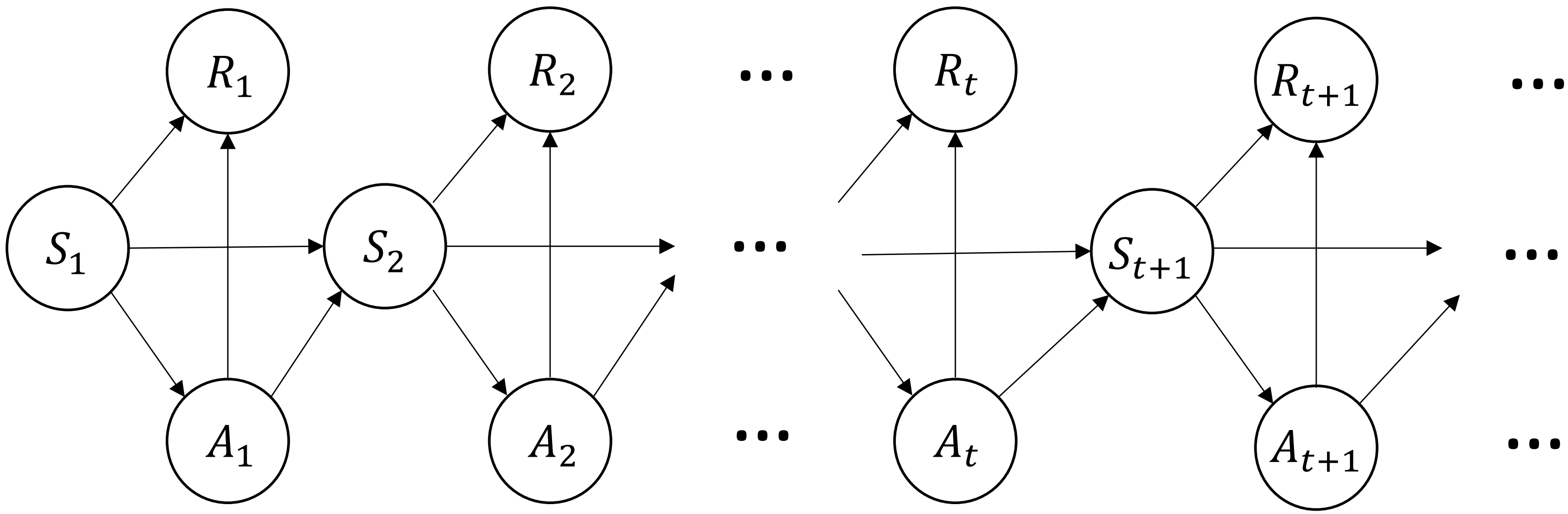
State-transitions with action **UP**:

80% move up
10% move left
10% move right

*If you bump into a wall, you stay where you are.

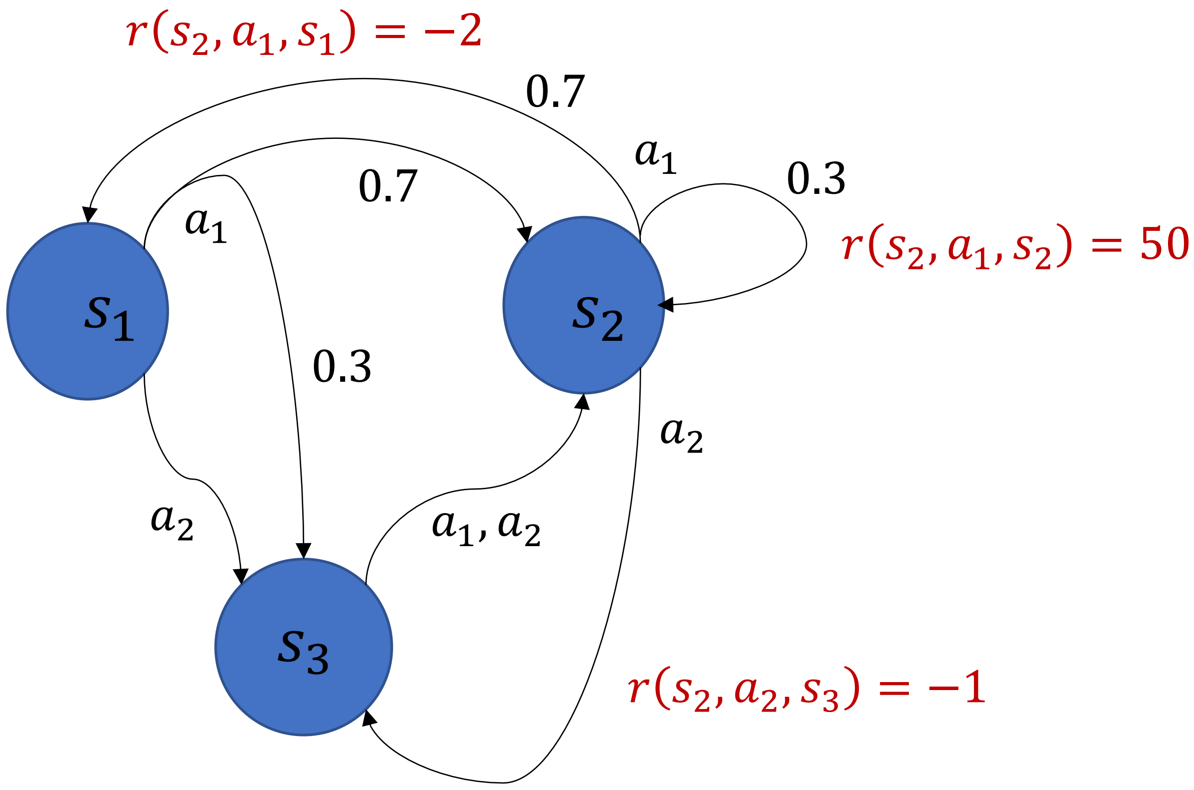
- reward +1 at [4,3], -1 at [4,2]
- reward -0.04 for each step
- Finite horizon or infinite horizon?
- What is a good policy?

Parameters of an MDP are factorizations of the joint distribution



- Initial state distribution
- Transition dynamics
- Reward distribution

State-space diagram representation of an MDP: An example with 3 states and 2 actions.



* The reward can be associated with only the state s' you transition into.
* Or the state that you transition from s and the action a you take.
* Or all three at the same time.

Reward function and Value functions

- Immediate reward function r(s,a)
  - expected **immediate** reward

    $$r(s,a) = \mathbb{E}[R_1 | S_1 = s, A_1 = a]$$

    $$r^\pi(s) = \mathbb{E}_{a \sim \pi(a|s)}[R_1 | S_1 = s]$$

- state value function: $V^\pi(s)$
  - expected **long-term** return when starting in *s* and following π

    $$V^\pi(s) = \mathbb{E}_\pi[R_1 + \gamma R_2 + ... + \gamma^{t-1} R_t + ... | S_1 = s]$$

- state-action value function: $Q^\pi(s,a)$
  - expected **long-term** return when starting in *s*, performing *a*, and following π

$$Q^\pi(s,a) = \mathbb{E}_\pi[R_1 + \gamma R_2 + ... + \gamma^{t-1} R_t + ... | S_1 = s, A_1 = a]$$

Optimal value function and the MDP planning problem

$$V^\star(s) := \sup_{\pi \in \Pi} V^\pi(s)$$
$$Q^\star(s,a) := \sup_{\pi \in \Pi} Q^\pi(s,a).$$

Goal of MDP planning:

$$\text{Find } \pi^* \text{ such that } V^\pi(s) = V^*(s) \quad \forall s$$

Approximate solution:

$$\pi \text{ is } \epsilon\text{-optimal if } V^\pi \geq V^*(s) - \epsilon \mathbf{1}$$

General policy, Stationary policy, Deterministic policy

- General policy could depend on the entire history

$$\pi : (\mathcal{S} \times \mathcal{A} \times \mathbb{R})^* \times \mathcal{S} \to \Delta(\mathcal{A})$$

- Stationary policy

$$\pi : \mathcal{S} \to \Delta(\mathcal{A})$$

- Stationary, Deterministic policy

$$\pi : \mathcal{S} \to \mathcal{A}$$

Two surprising facts about MDPs

1. It suffices to consider stationary / deterministic policies.

2. There exists a stationary / deterministic policy that is optimal simultaneously for all initial state distributions.

Bellman equations – the fundamental equations of MDP and RL

- An alternative, recursive and more useful way of defining the V-function and Q function

$$V^\pi(s) = \sum_a \pi(a|s) \sum_{s'} P(s'|s,a)[r(s,a,s') + \gamma V^\pi(s')] = \sum_a \pi(a|s) Q^\pi(s,a)$$

- Exercise:
  - Prove Bellman equation from the definition.
  - Write down the Bellman equation using Q function alone.

$$Q^\pi(s,a) = ?$$

Bellman optimality equations characterizes the optimal policy

$$V^*(s) = \max_a \sum_{s'} P(s'|s,a)[r(s,a,s') + \gamma V^*(s')]$$

- system of n non-linear equations
- solve for V*(s)
- easy to extract the optimal policy

- having Q*(s,a) makes it even simpler

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

Bellman equations in matrix forms

- Lemma (Bellman consistency): For stationary policies, we have

$$V^\pi(s) = Q^\pi(s, \pi(s)).$$
$$Q^\pi(s,a) = r(s,a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)} \left[V^\pi(s') \right].$$

- In matrix forms:

$$V^\pi = r^\pi + \gamma P^\pi V^\pi$$
$$Q^\pi = r + \gamma P V^\pi$$
$$Q^\pi = r + \gamma P^\pi Q^\pi .$$

Value iterations for MDP planning

- Recall: Bellman optimality equations

$$V^*(s) = \max_a \sum_{s'} P(s'|s,a)[r(s,a,s') + \gamma V^*(s')]$$

$$Q(s,a) = r(s,a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)} \left[\max_{a' \in \mathcal{A}} Q(s',a') \right].$$

$$\mathcal{T}Q = r + PV_Q$$ where $$V_Q(s) := \max_{a \in \mathcal{A}} Q(s,a).$$

Theorem: Q = Q* if and only if Q satisfies the Bellman optimality equations.

Value iterations for MDP planning

- The value iteration algorithm iteratively applies the Bellman operator until it converges.
  1. Initialize Q_0 arbitrarily
  2. for i in 1,2,3,…, k, update $Q_i = \mathcal{T} Q_{i-1}$
  3. Return Q_k
- **What is the right question to ask here?**

Convergence of value iteration for solving MDPs

- Lemma 1. The Bellman operator is a γ-contraction.

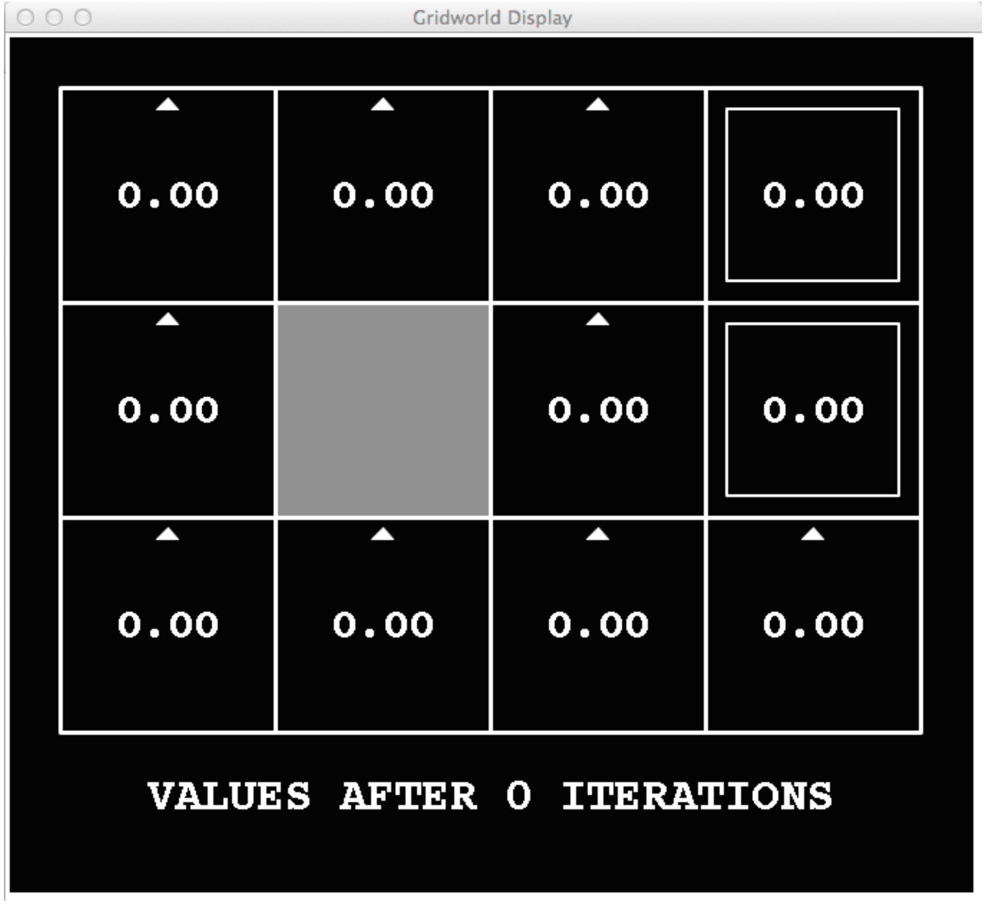
$$\textit{For any two vectors } Q, Q' \in \mathbb{R}^{|\mathcal{S}||\mathcal{A}|},$$

$$\|\mathcal{T}Q - \mathcal{T}Q'\|_\infty \leq \gamma \|Q - Q'\|_\infty$$

  - Prove this in the optional HW4.

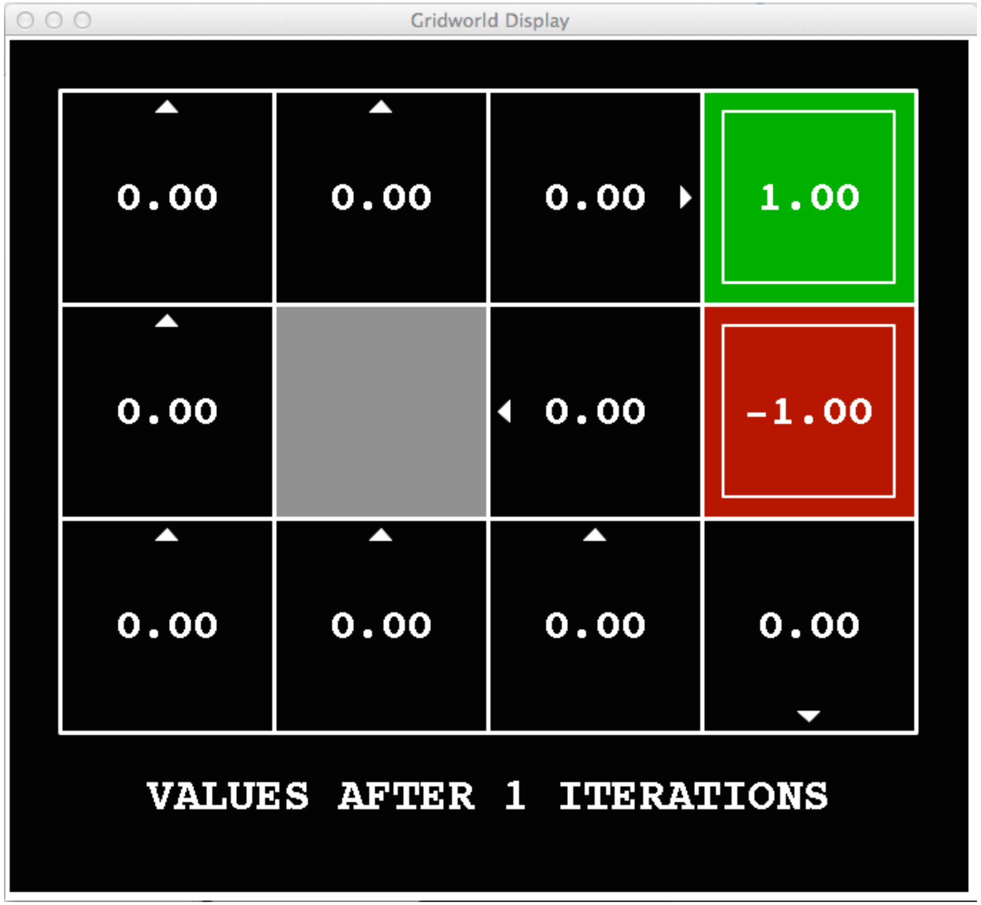
- Fast convergence of value iterations to Q*:

k=0



Noise = 0.2
Discount = 0.9
Living reward = 0

k=1



Noise = 0.2
Discount = 0.9
Living reward = 0

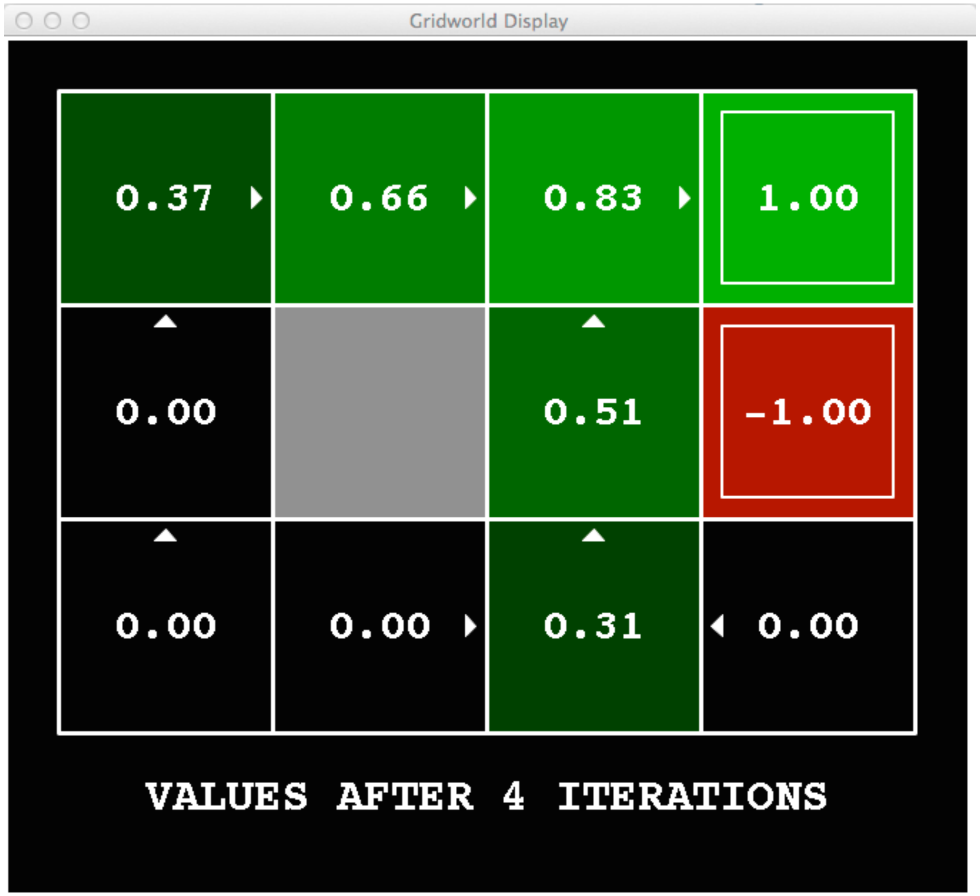
k=2

Noise = 0.2
Discount = 0.9
Living reward = 0

k=3

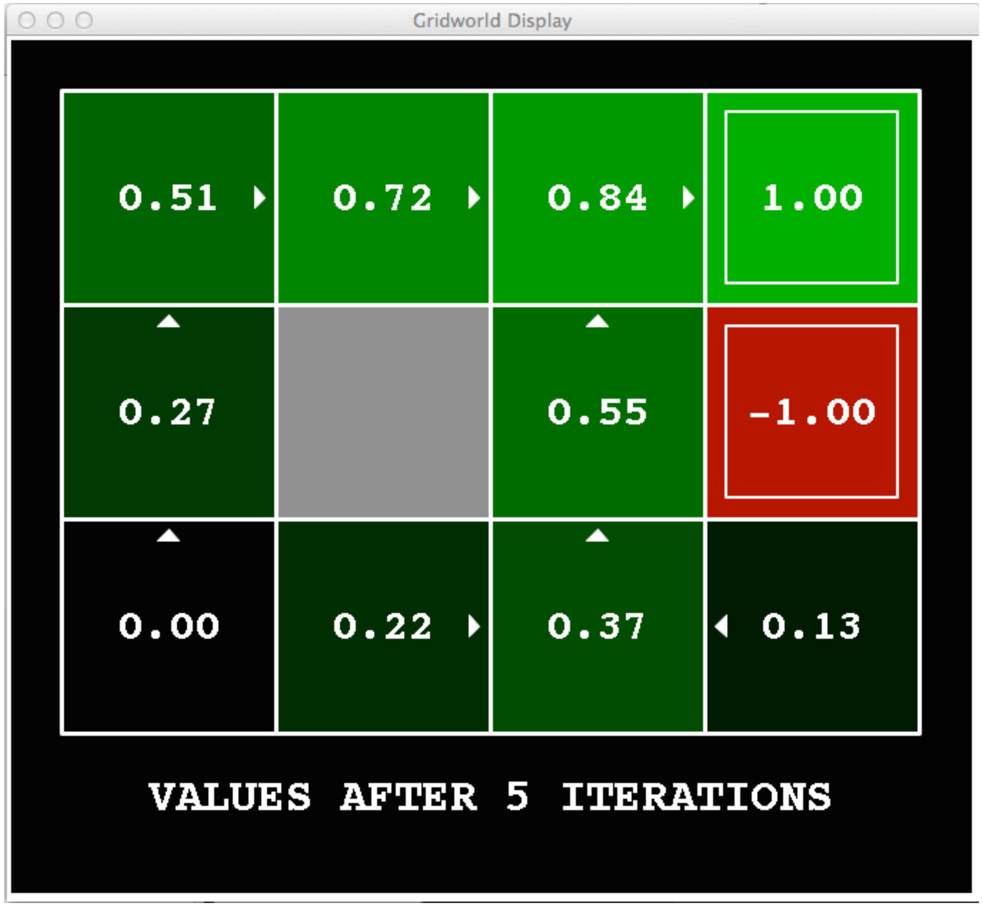
Noise = 0.2
Discount = 0.9
Living reward = 0

k=4



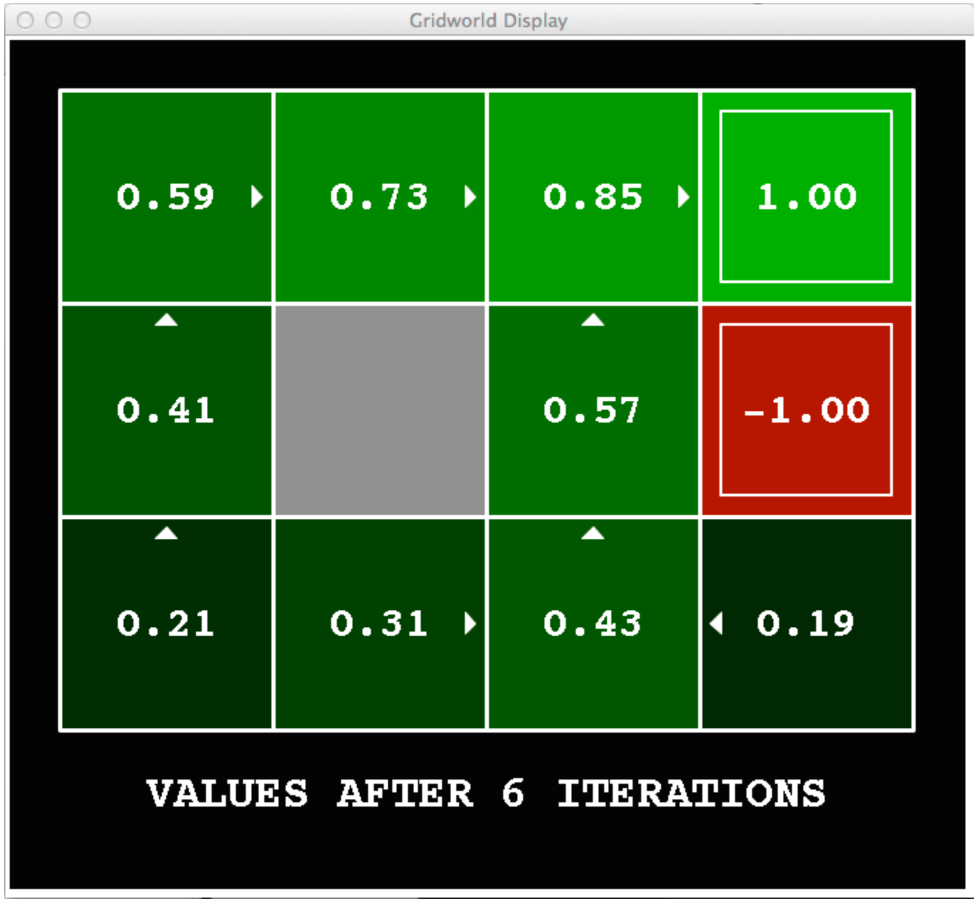
Noise = 0.2
Discount = 0.9
Living reward = 0

k=5



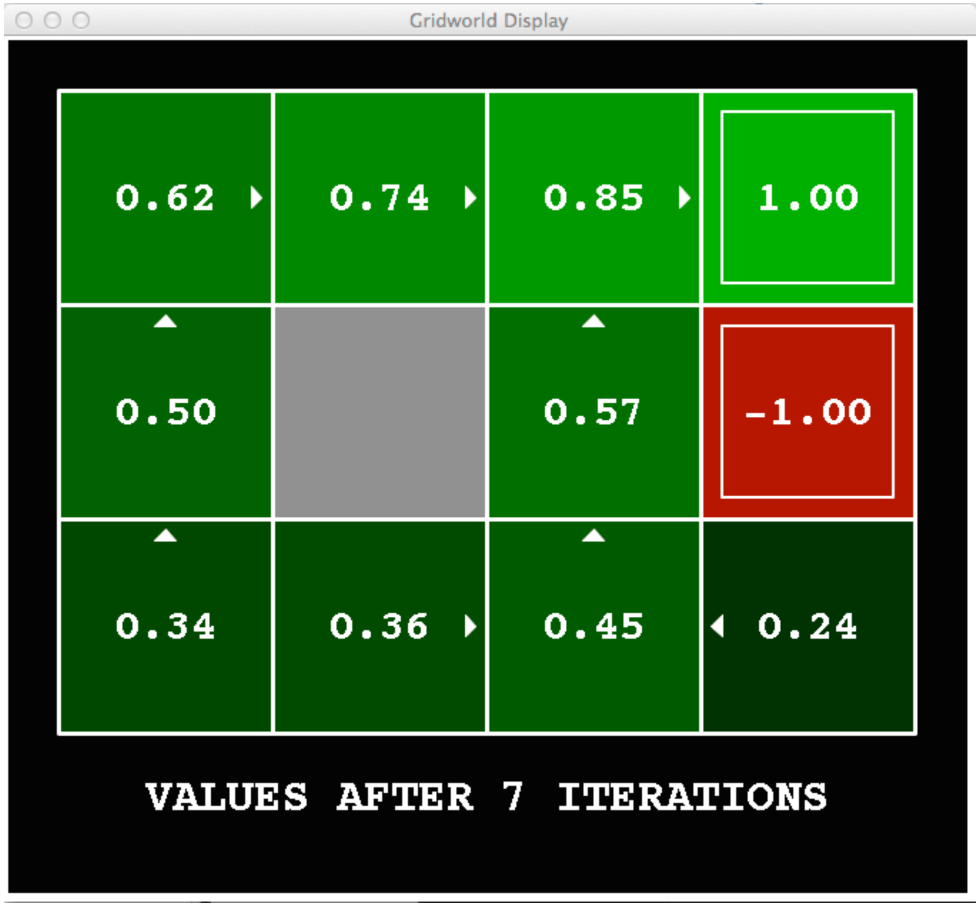
Noise = 0.2
Discount = 0.9
Living reward = 0

k=6



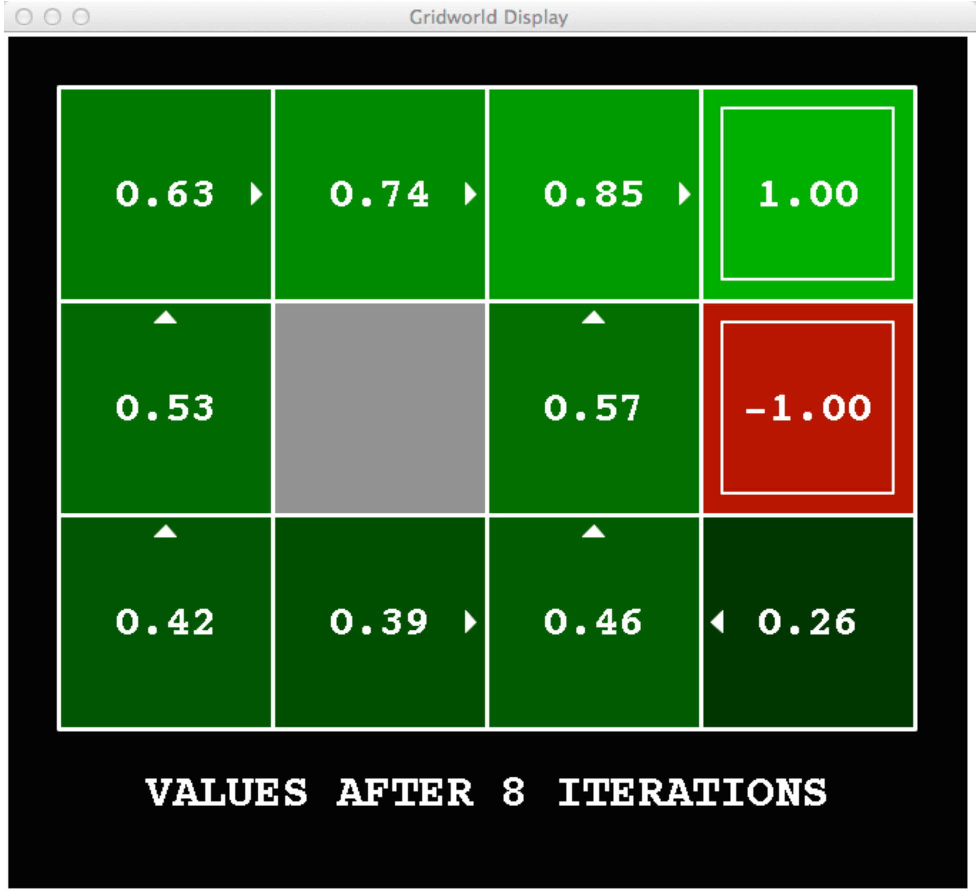
Noise = 0.2
Discount = 0.9
Living reward = 0

k=7



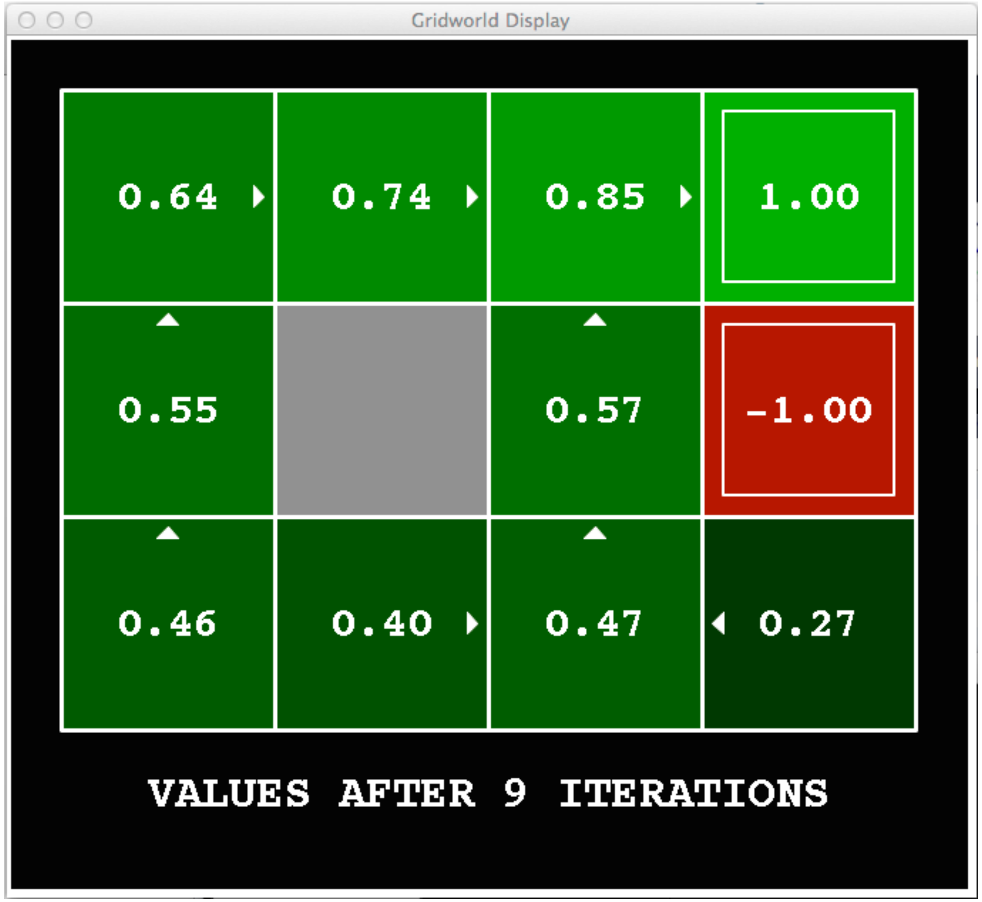
Noise = 0.2
Discount = 0.9
Living reward = 0

k=8



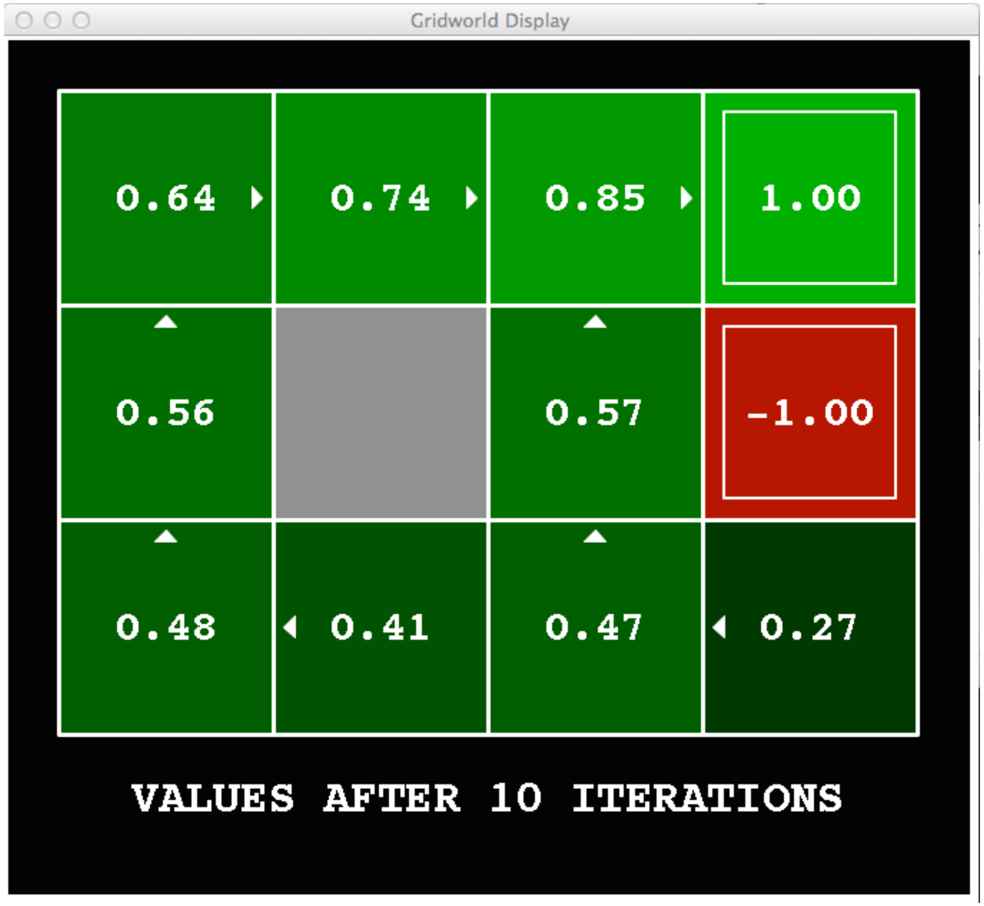
Noise = 0.2
Discount = 0.9
Living reward = 0

k=9



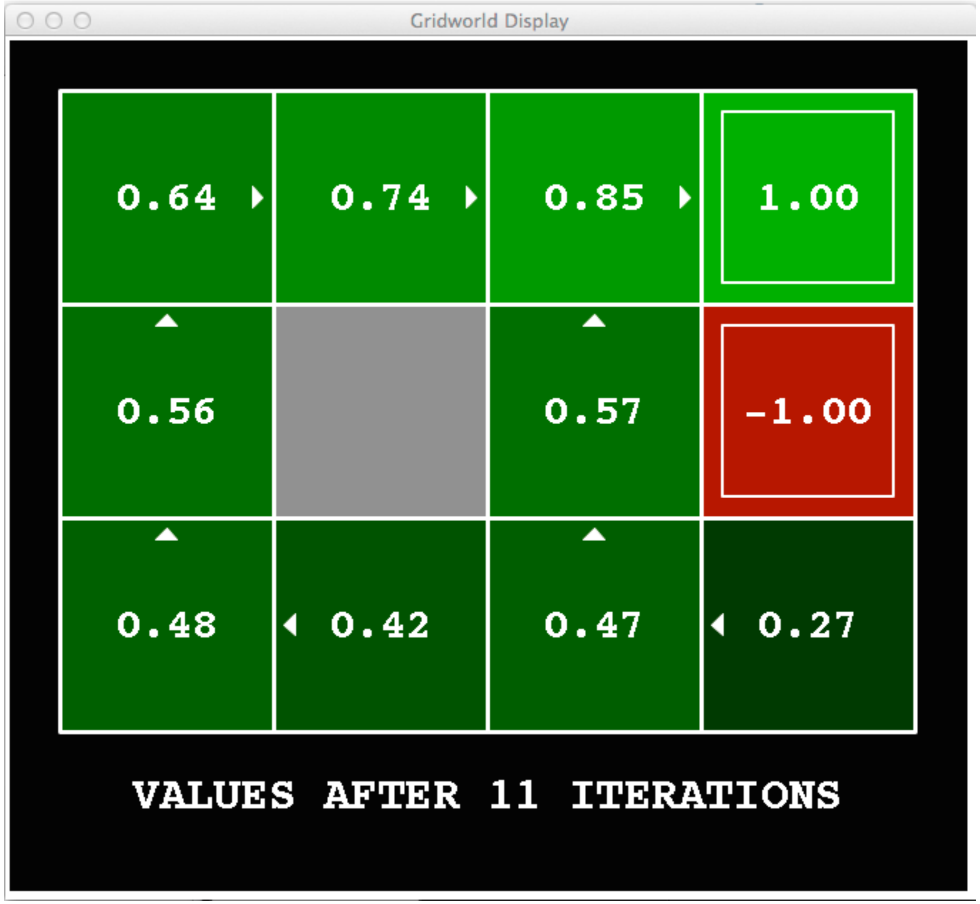
Noise = 0.2
Discount = 0.9
Living reward = 0

k=10



Noise = 0.2
Discount = 0.9
Living reward = 0

k=11

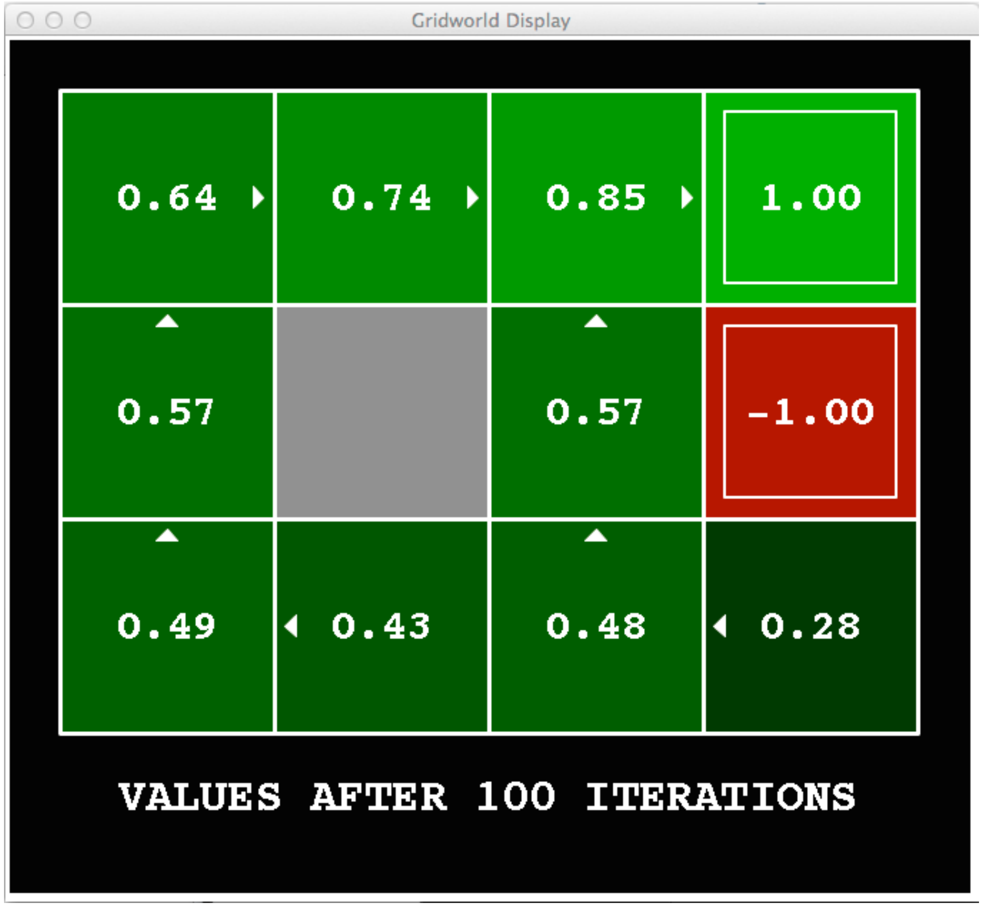


Noise = 0.2
Discount = 0.9
Living reward = 0

k=12

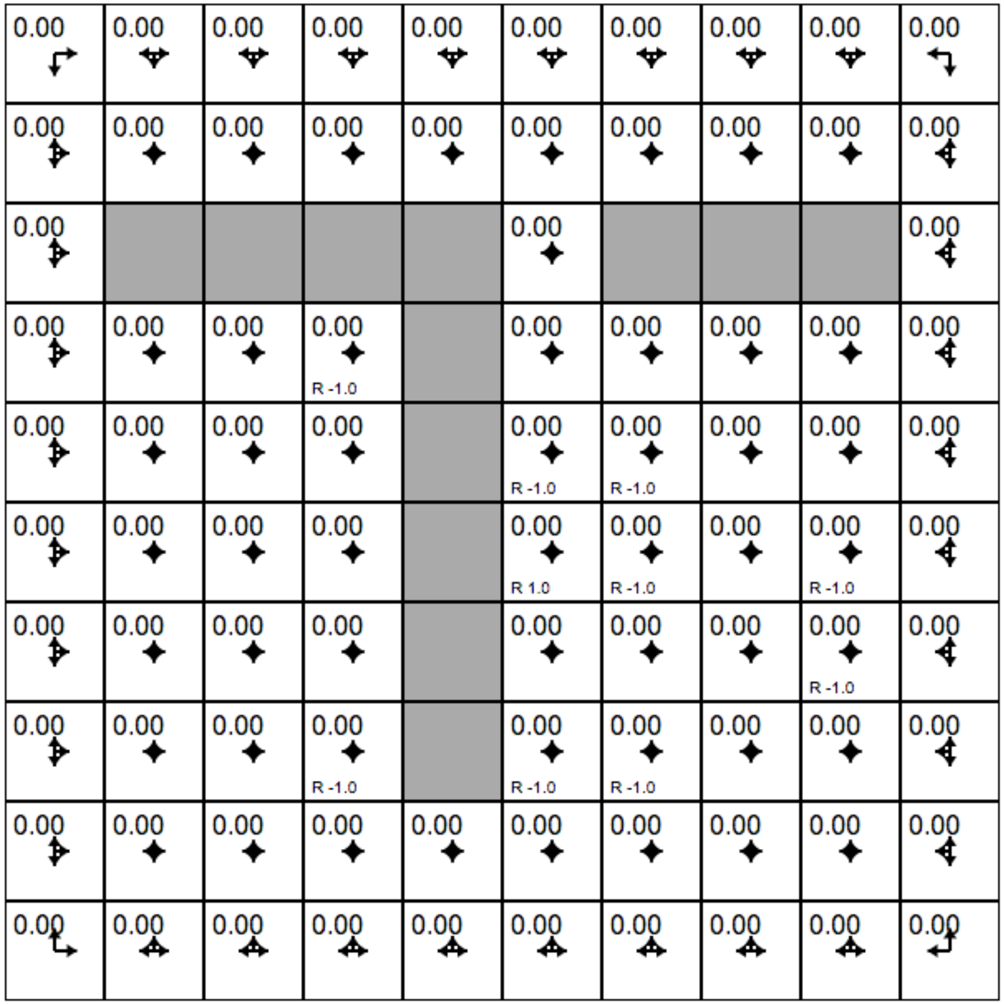
Noise = 0.2
Discount = 0.9
Living reward = 0

k=100



Noise = 0.2
Discount = 0.9
Living reward = 0

Demo: grid worlds



https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld_dp.html

Checkpoint

- What is RL? What are its motivating applications?
- A model of RL --- Markov Decision Processes
  - Value functions: Q functions and V functions
  - Bellman equations
- MDP planning / inference problem
  - Value iterations

Remainder of this lecture

- RL algorithms
  - Model-based RL vs Model-free RL
  - Monte Carlo
  - Temporal Difference Learning
  - Linear function approximation

Recap: Policy Iterations and Value Iterations

- What are these algorithms for?
  - Algorithms of computing the V* and Q* functions from MDP parameters
- Policy Iterations

$$\pi_0 \rightarrow^E V^{\pi_0} \rightarrow^I \pi_1 \rightarrow^E V^{\pi_1} \rightarrow^I \dots \rightarrow^I \pi^* \rightarrow^E V^*$$

- Value iterations

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} P(s'|s,a)[r(s,a,s') + \gamma V_k(s')]$$

- How do we make sense of them?
  - Recursively applying the Bellman equations until convergence.

*These methods are called "Dynamic Programming" approaches in Chap 4 of Sutton and Barto.

They are no longer valid in RL

- Policy Evaluation

$$V^{\pi}_{k+1}(s) \leftarrow \sum_a \pi(a|s) \sum_{s'} \cancel{P(s'|s,a)}[\cancel{r(s,a,s')} + \gamma V^{\pi}_k(s')]$$

- Policy improvement

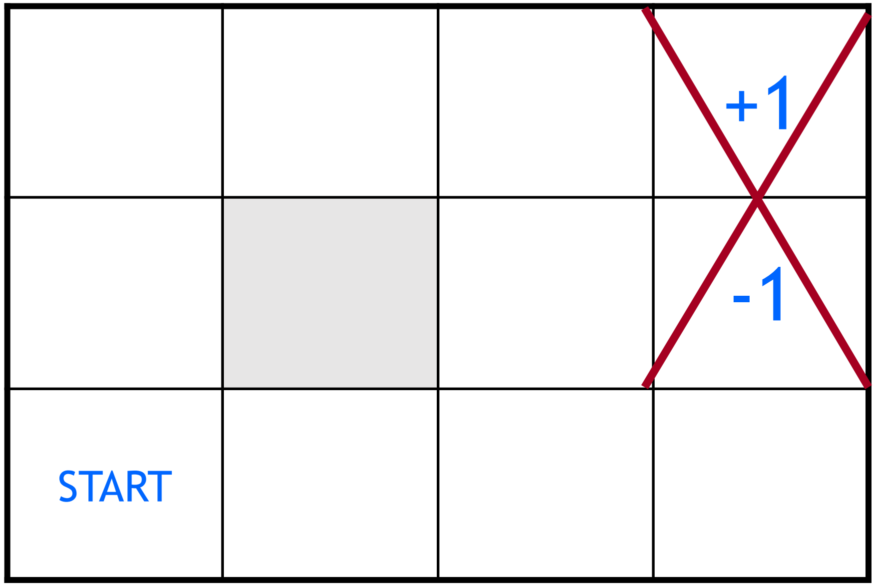
$$\pi'(s) = \arg\max_a Q^{\pi}(s,a)$$
$$= \arg\max_a \sum_{s'} \cancel{P(s'|s,a)}[\cancel{r(s,a,s')} + \gamma V^{\pi}_k(s')]$$

- Value iterations

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} \cancel{P(s'|s,a)}[\cancel{r(s,a,s')} + \gamma V_k(s')]$$

***We do not have the MDP parameters in RL!**

Example: Frozen lake

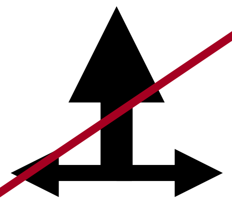


Action 1, Action 2, Action 3, Action 4

actions: ~~UP, DOWN, LEFT, RIGHT~~

UP

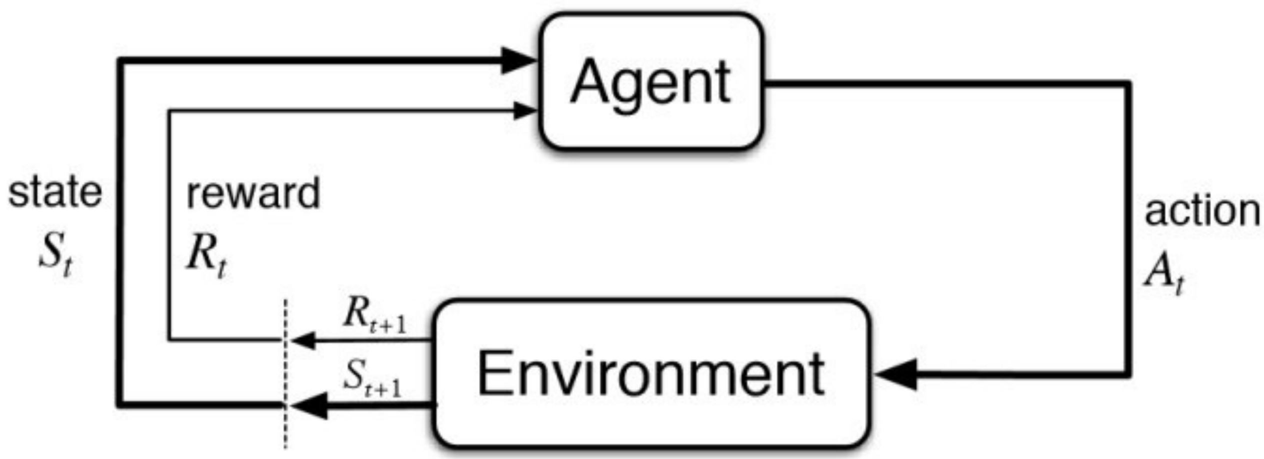
80% move UP
10% move LEFT
10% move RIGHT



- ~~reward +1 at [4,3], -1 at [4,2]~~
- ~~reward -0.04 for each step~~
- what's the strategy to achieve max reward?

Instead, reinforcement learning agents have "online" access to an environment

- State, Action, Reward
- Unknown reward function, unknown state-transitions.
- Agents can "act" and "experiment", rather than only doing offline planning.



Idea 1: **Model-based** Reinforcement Learning

- Model-based idea
  - Let's approximate the model based on experiences
  - Then solve for the values as if the learned model were correct

- Step 1: Get data by running the agent to explore
  - Many data points of the form: $\{(s_1, a_1, s_2, r_1), \dots, (s_N, a_N, s_{N+1}, r_N)\}$

- Step 2: Estimate the model parameters
  - $\hat{P}(s'|s,a)$ --- plug-in / MLE. We need to observe the transition many times for each s, a
  - $\hat{r}(s', a, s)$ --- this is an estimate of the empirical rewards.

Then we can plug in these estimates and then use dynamic programming for policy evaluation / improvements.

$$V_{k+1}^{\pi}(s) \leftarrow \sum_a \pi(a|s) \sum_{s'} \boxed{\hat{P}(s'|s,a)[\hat{r}(s,a,s')} + \gamma V_k^{\pi}(s')]$$

$$\pi' \leftarrow \arg\max_a \sum_{s'} \boxed{\hat{P}(s'|s,a)[\hat{r}(s,a,s')} + \gamma V_k^{\pi}(s')]$$

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} \boxed{\hat{P}(s'|s,a)[\hat{r}(s,a,s')} + \gamma V_k(s')]$$

* As usual, **“hat”** indicates empirical estimates.

* These iterations will produce $\hat{V}^*$ and $\hat{Q}^*$ functions, and then $\hat{\pi}^*$

This is OK if we have a generative model! But there are complications.

- For MDPs
  - Often we need to take a carefully chosen sequence of actions to reach a state
  - The chance of randomly running into a state can be **exponentially small,** if we decide to take random actions.
  - **Question: What is an example of this?**

*Need to somehow update the “exploration policy” on the fly!

More caveats

- The fitted model is just an approximation of the environment.
- How does the error in the fitted MDP translate into the error in the estimated value functions V* and Q*?
- How does the error in the estimated Q* function affect the suboptimality of the policy that maximizes $\hat{Q}^*$?
- Answered by "Simulation Lemma" (Kearns and Singh, 2002)
  - Resurgence of research on this more recently: Yin and W. (2020), Yin, Bai and W. (2020)

Idea 2: **Model-free** Reinforcement Learning

- Do we need the model? Can we learn the Q function directly?
  - **How many free parameters are there to represent the Q-function?**

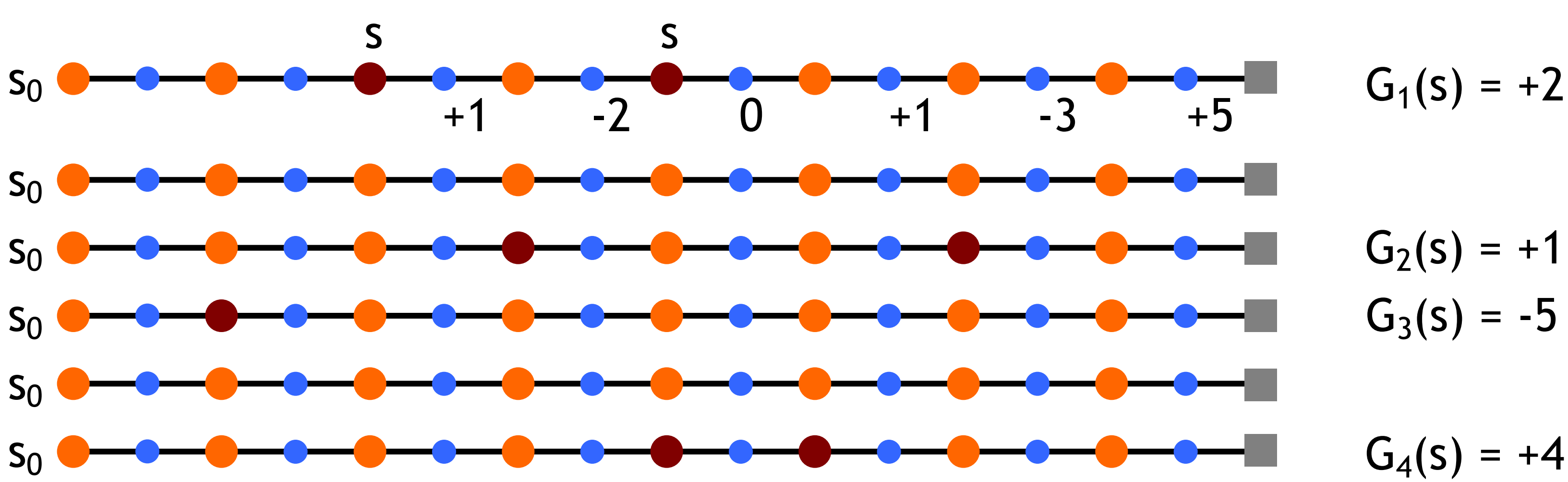
- Recall: Policy iterations

$$\pi_0 \rightarrow^E V^{\pi_0} \rightarrow^I \pi_1 \rightarrow^E V^{\pi_1} \rightarrow^I \ldots \rightarrow^I \pi^* \rightarrow^E V^*$$

  - Maybe we can do policy evaluation / value iterations without estimating the model?

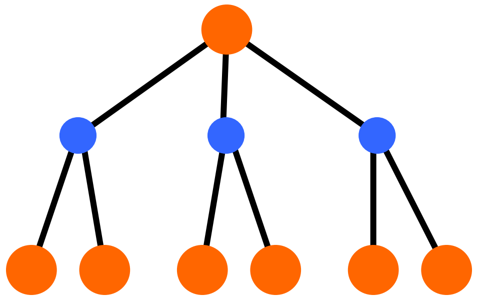
Monte Carlo Policy Evaluation (Prediction)

- want to estimate $V^{\pi}(s)$
  = expected return starting from s and following π
  - estimate as average of observed returns in state s
- We can execute the policy π
- first-visit MC
  - average returns following the first visit to state s



$V^{\pi}(s) \approx (2 + 1 - 5 + 4)/4 = 0.5$

Monte Carlo Policy Optimization (Control)



- V^π not enough for policy improvement
    - need exact model of environment
- estimate $Q^\pi(s,a)$

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

- MC control

$$\pi_0 \longrightarrow^E Q^{\pi_0} \longrightarrow^I \pi_1 \longrightarrow^E Q^{\pi_1} \longrightarrow^I \ldots \longrightarrow^I \pi^* \longrightarrow^E Q^*$$

    - update after each episode
- Two problems
    - greedy policy won't explore all actions   eps-greedy, or bonus design.
    - Requires many independent episodes for the estimated value function to be accurate.

Improved Monte-Carlo Q-function estimate using Bellman equations

- Recall:

$$Q^\pi(s,a) = \sum_{s'} P(s'|s,a)[r(s,a,s') + \gamma \sum_{a'} \pi(a'|s') Q^\pi(s',a')]$$

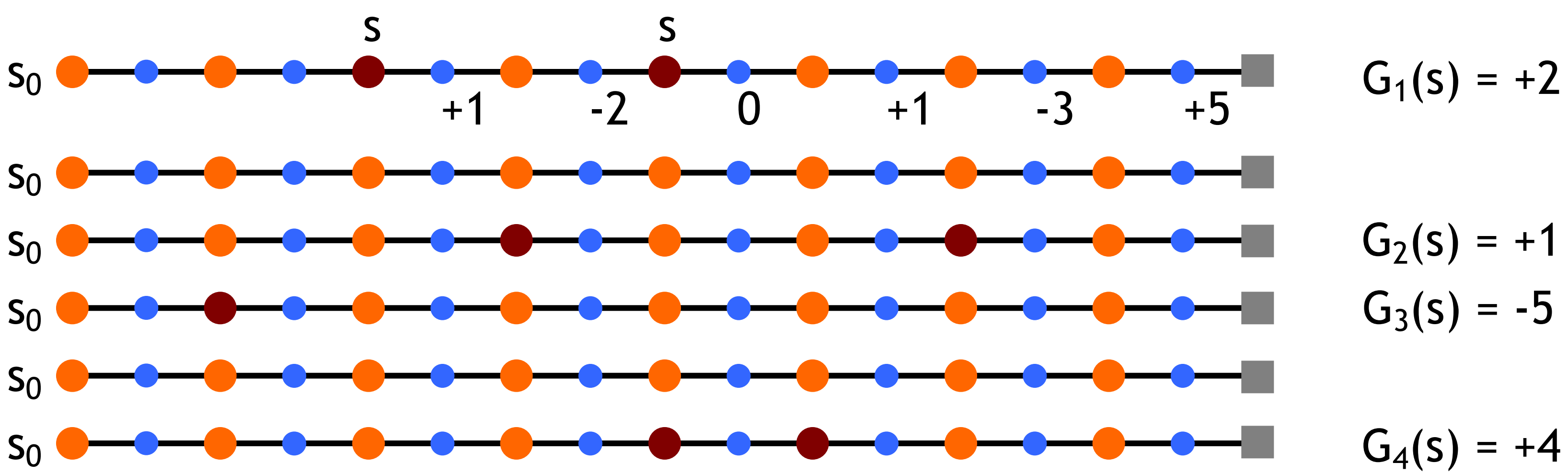
$$Q^\pi(s,a) = r^\pi(s,a) + \gamma \mathbb{E}_{s' \sim P(s'|s,a)}[V^\pi(s')]$$

- We can use the empirical (Monte Carlo) estimate.

$$\widehat{Q^\pi}(s,a) = \widehat{r^\pi}(s,a) + \gamma \widehat{\mathbb{E}}_{s' \sim P(s'|s,a)}[\widehat{V}^\pi(s')]$$

*No need to estimate P(s' | s,a) or r(s,a,s') as intermediate steps.
*Require only O(SA) space, rather than O(S^2A)

Online averaging representation of MC



$V^{\pi}(s) \approx (2 + 1 - 5 + 4)/4 = 0.5$

- Alternative, *online averaging* update

$$V(S_t) \leftarrow V(S_t) + \alpha \Big[G_t - V(S_t) \Big], \quad \text{where } \alpha = 1/N_{S_t}$$

DP + MC = Temporal Difference Learning

- Monte Carlo $V(S_t) \leftarrow V(S_t) + \alpha\Big[G_t - V(S_t)\Big],$

Issue: G_t can only be obtained after the entire episode!

- The idea of TD learning:

$$\mathbb{E}_\pi[G_t] = \mathbb{E}_\pi[R_t|S_t] + \gamma V^\pi(S_{t+1})$$

We only need one step before we can plug-in and estimate the RHS!

- TD-Policy evaluation

Bootstrapping!

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\Big]$$

Bootstrap's origin

- "The Surprising Adventures of Baron Münchausen"
  - Rudolf Erich Raspe, 1785

- In statistics: Brad Efron's resampling methods
- In computing: Booting…
- In RL: It simply means TD learning

TD policy optimization (TD-control)

- SARSA (On-Policy TD-control)
  - Update the Q function by bootstrapping Bellman Equation

$$Q(S, A) \leftarrow Q(S, A) + \alpha \big[R + \gamma Q(S', A') - Q(S, A)\big]$$

  - Choose the next A' using Q, e.g., eps-greedy.

- Q-Learning (Off-policy TD-control)
  - Update the Q function by bootstrapping Bellman Optimality Eq.

$$Q(S, A) \leftarrow Q(S, A) + \alpha \big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$$

  - Choose the next A' using Q, e.g., eps-greedy, or any other policy.

Remarks:

- These are **proven to converge** asymptotically.
- Much more data-efficient in practice, than MC.
- Regret analysis is still active area of research.

Advantage of TD over Monte Carlo

- Given a trajectory, a roll-out, of T steps.
  - MC updates the Q function only once
  - TD updates the Q function (and the policy) T times!

Remark: This is the same kind of improvement from Gradient Descent to Stochastic Gradient Descent (SGD).

Model-free vs Model-based RL algorithms

- Different function approximations

- Different space efficiency

- Which one is more statistically efficient?
  - More or less equivalent in the tabular case.
  - Different challenges in their analysis.

The problem of large state-space is still there

- We need to represent and learn SA parameters in Q-learning and SARSA.

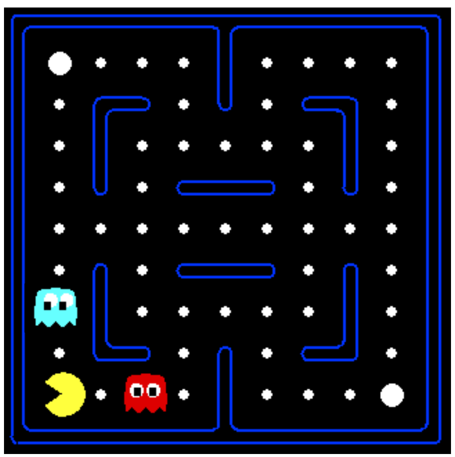
- S is often large
  - 9-puzzle, Tic-Tac-Toe: 9! = 362,800, S^2 = 1.3*10^11
  - PACMAN with 20 by 20 grid. S = O(2^400), S^2 = O(2^800)

- O(S) is not acceptable in some cases.

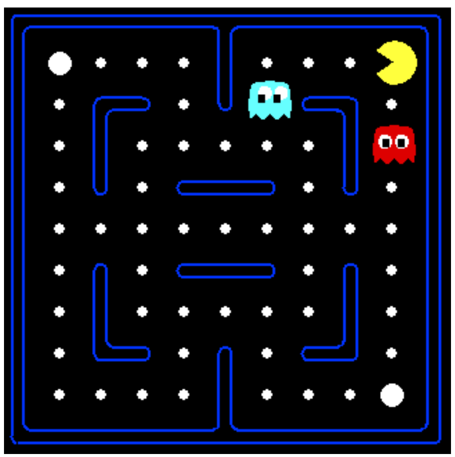
- Need to think of ways to “generalize”/share information across states.

Example: Pacman

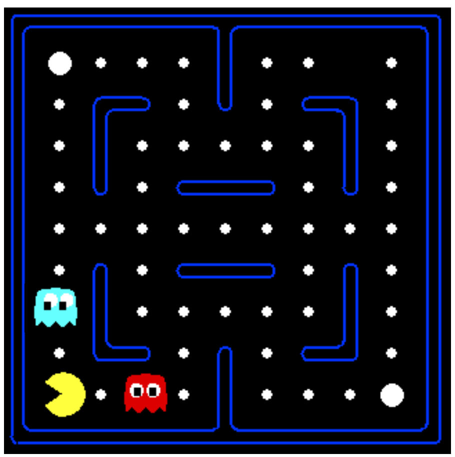
Let's say we discover through experience that this state is bad:



In naïve q-learning, we know nothing about this state:



Or even this one!



(From Dan Klein and Pieter Abbeel)

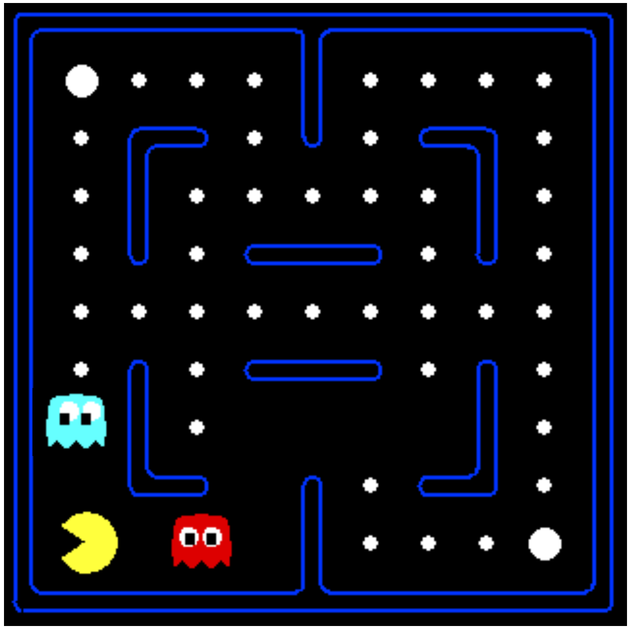
Video of Demo Q-Learning Pacman – Tiny – Watch All

Video of Demo Q-Learning Pacman – Tiny – Silent Train

Video of Demo Q-Learning Pacman – Tricky – Watch All

Why not use an evaluation function? A Feature-Based Representations

- Solution: describe a state using a vector of features (properties)
  - Features are functions from states to real numbers (often 0/1) that capture important properties of the state
  - Example features:
    - Distance to closest ghost
    - Distance to closest dot
    - Number of ghosts
    - 1 / (dist to dot)^2
    - Is Pacman in a tunnel? (0/1)
    - …… etc.
    - Is it the exact state on this slide?
  - Can also describe a q-state (s, a) with features (e.g. action moves closer to food)



Linear Value Functions

- Using a feature representation, we can write a q function (or value function) for any state using a few weights:
  - $V_{\mathbf{w}}(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$
  - $Q_{\mathbf{w}}(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \ldots + w_n f_n(s,a)$
- Advantage: our experience is summed up in a few powerful numbers
- Disadvantage: states may share features but actually be very different in value!

Updating a linear value function

- Original Q learning rule tries to reduce prediction error at s, a:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \cdot [R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a)]$$

- Instead, we update the weights to try to reduce the error at s, a:

$$w_i \longleftarrow w_i + \alpha \cdot [R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a)] \, \partial Q_{\mathbf{w}}(s,a)/\partial w_i$$
$$= w_i + \alpha \cdot [R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a)] \, f_i(s,a)$$

Updating a linear value function

- Original Q learning rule tries to reduce prediction error at s, a:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \cdot [R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a)]$$

- Instead, we update the weights to try to reduce the error at s, a:

$$w_i \longleftarrow w_i + \alpha \cdot [R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a)] \, \partial Q_{\mathbf{w}}(s,a)/\partial w_i$$
$$= w_i + \alpha \cdot [R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a)] \, f_i(s,a)$$

- Qualitative justification:
  - Pleasant surprise: increase weights on positive features, decrease on negative ones
  - Unpleasant surprise: decrease weights on positive features, increase on negative ones

PACMAN Q-Learning (Linear function approx.)

Deriving the TD via incremental optimization that minimizes Bellman errors

- Mean Square Error and Mean Square Bellman error

So far, in RL algorithms

- Model-based approaches
  - Estimate the MDP parameters.
  - Then use policy-iterations, value iterations.
- Monte Carlo methods:
  - estimating the rewards by empirical averages
- Temporal Difference methods:
  - Combine Monte Carlo methods with Dynamic Programming
- Linear function approximation in Q-learning
  - Similar to SGD
  - Learning heuristic function

Final lecture

- Wrap up RL algorithm
- Exploration