CS 4341

Introduction to Artificial Intelligence

C-Term 2018

Reinforcement Learning

Adapted from slides by Peter Bodik at UC Berkeley

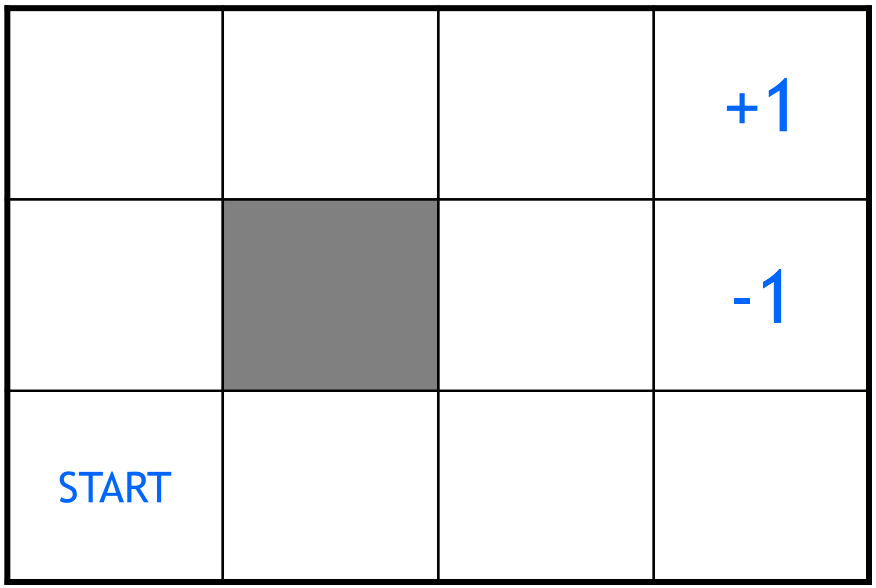
Ahmedul Kabir

WPI

Overview

- Examples
- Defining an RL problem
  - Markov Decision Processes
- Solving an RL problem
  - Dynamic Programming
  - Monte Carlo methods
  - Temporal-Difference learning

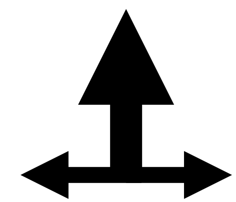
Robot in a room



actions: UP, DOWN, LEFT, RIGHT

UP

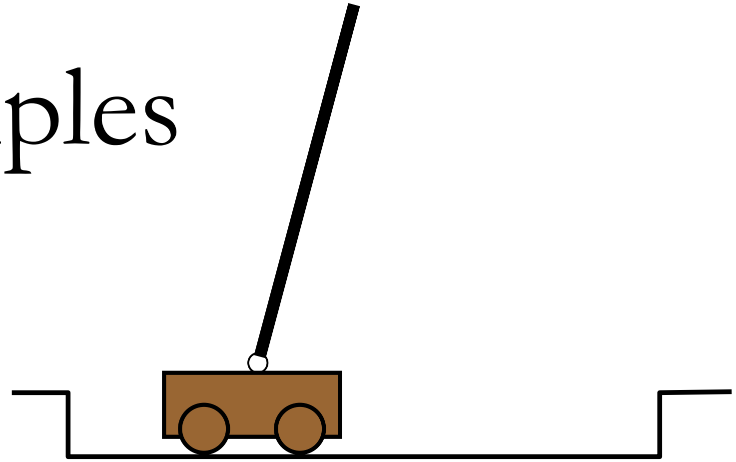
80% move UP
10% move LEFT
10% move RIGHT



- reward +1 at [4,3], -1 at [4,2]
- reward -0.04 for each step

- what's the strategy to achieve max reward?
- what if the actions were deterministic?

Other examples

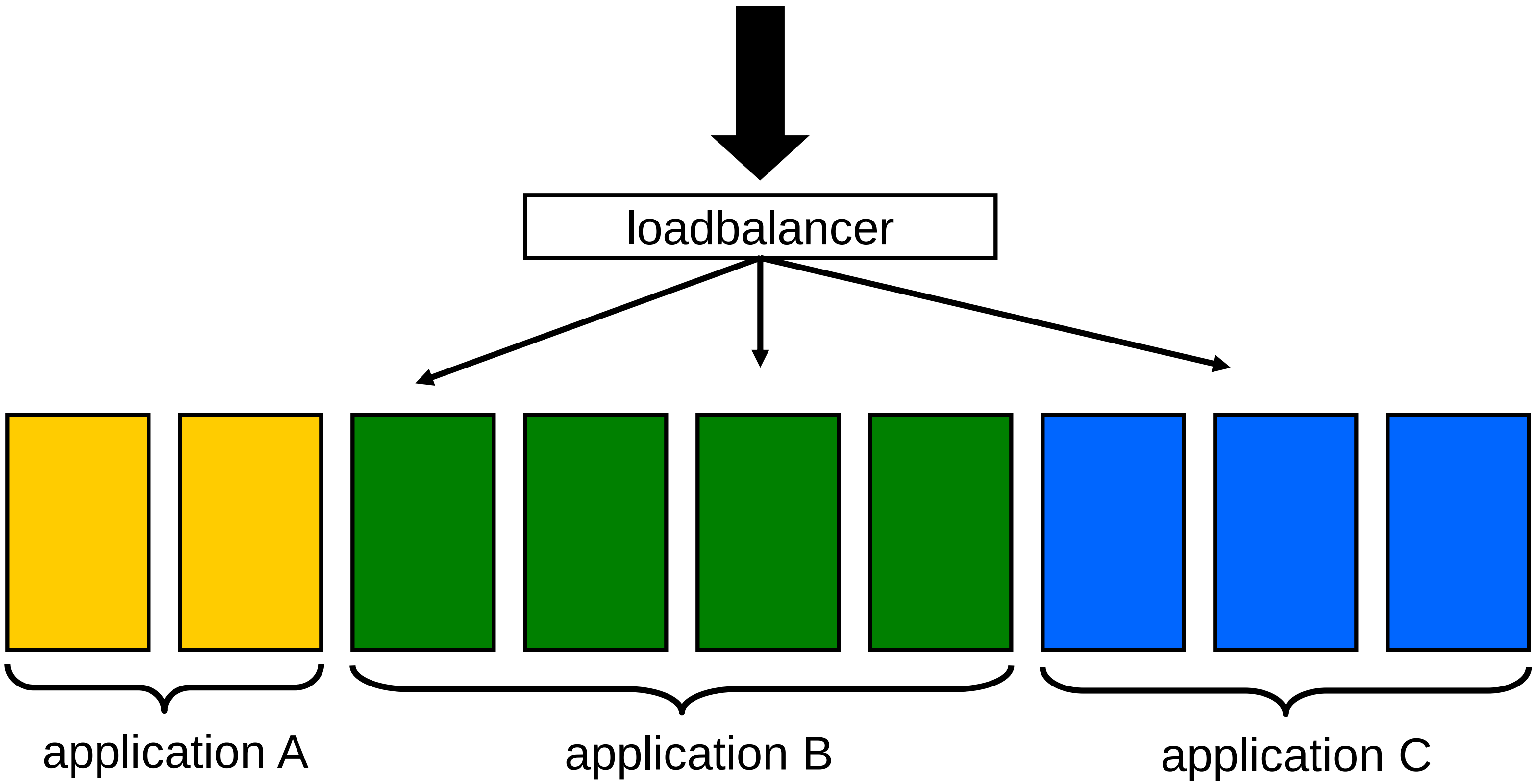


- pole-balancing
- TD-Gammon [Gerry Tesauro]
- helicopter [Andrew Ng]

- no teacher who would say "good" or "bad"
  - is reward "10" good or bad?
  - rewards could be delayed

- explore the environment and learn from experience
  - not just blind search, try to be smart about it

Resource allocation in datacenters

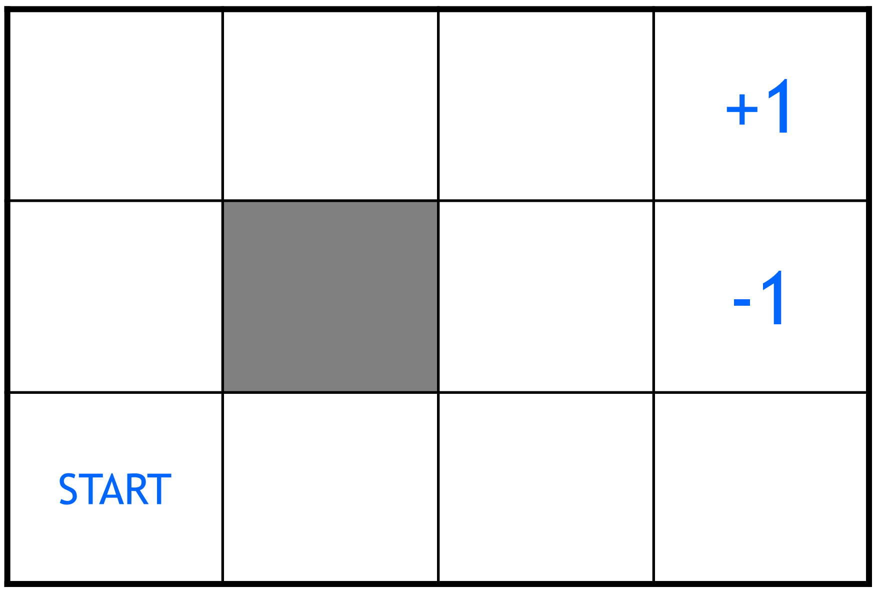


- A Hybrid Reinforcement Learning Approach to Autonomic Resource Allocation
  - Tesauro, Jong, Das, Bennani (IBM)
  - ICAC 2006

Outline

- Examples
- Defining an RL problem
  - Markov Decision Processes
- Solving an RL problem
  - Dynamic Programming
  - Monte Carlo methods
  - Temporal-Difference learning

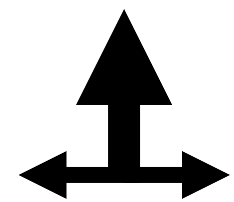
Robot in a room



actions: UP, DOWN, LEFT, RIGHT

UP

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

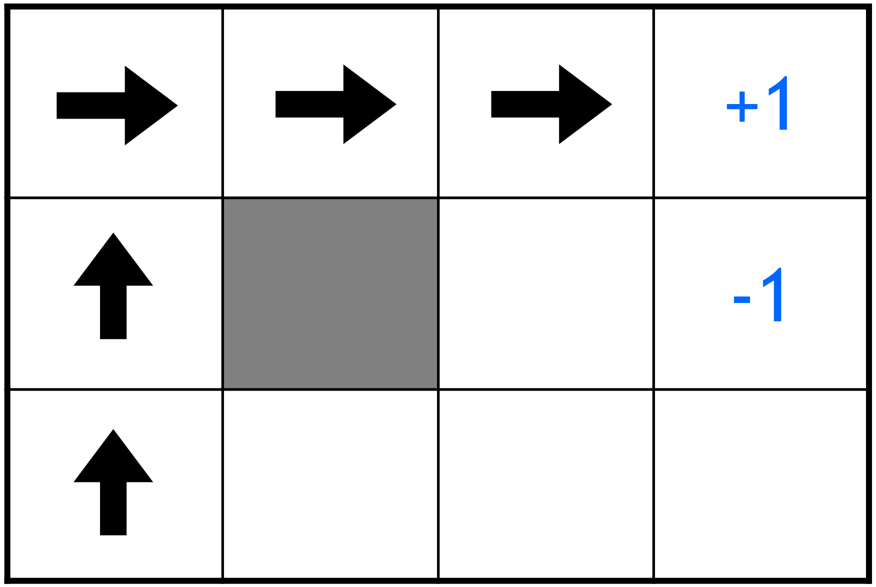


reward +1 at [4,3], -1 at [4,2]
reward -0.04 for each step

- states
- actions
- rewards

- what is the solution?

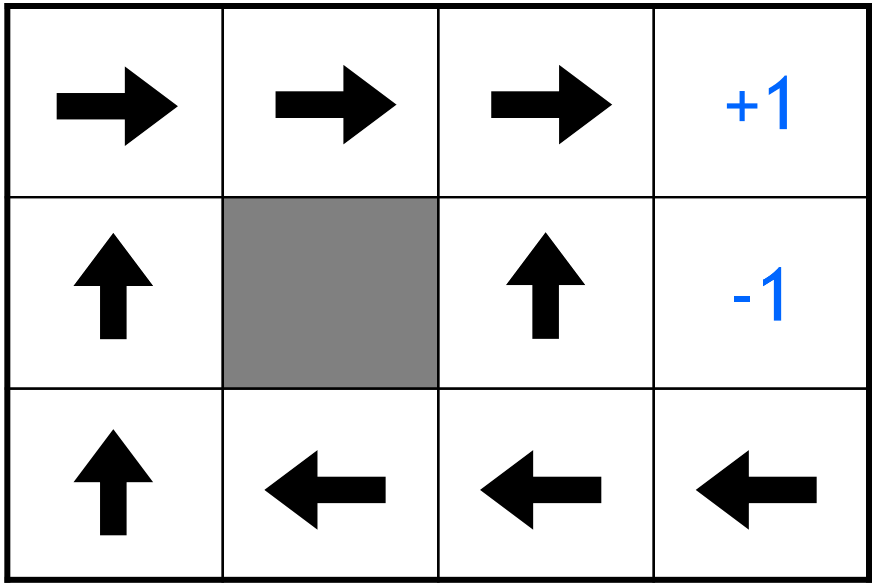
Is this a solution?



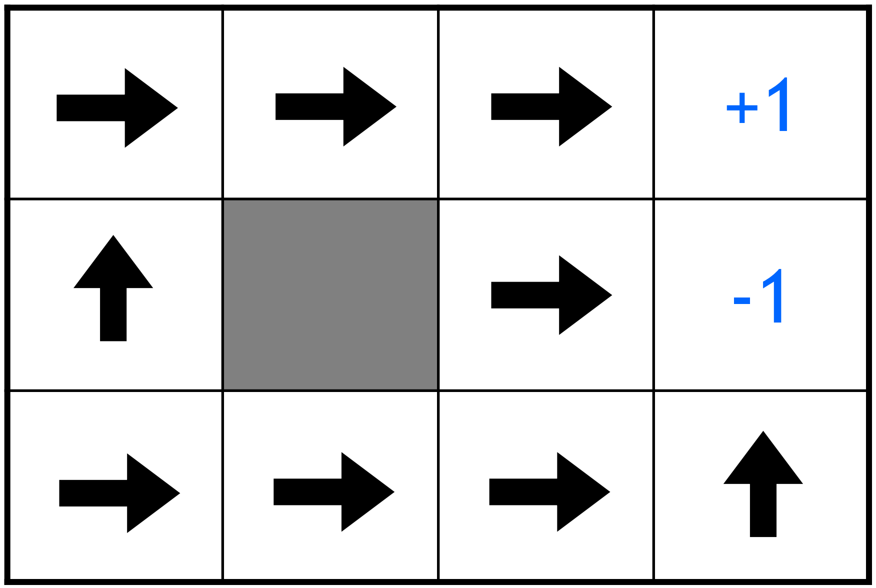
- only if actions deterministic
  - not in this case (actions are stochastic)

- solution/policy
  - mapping from each state to an action

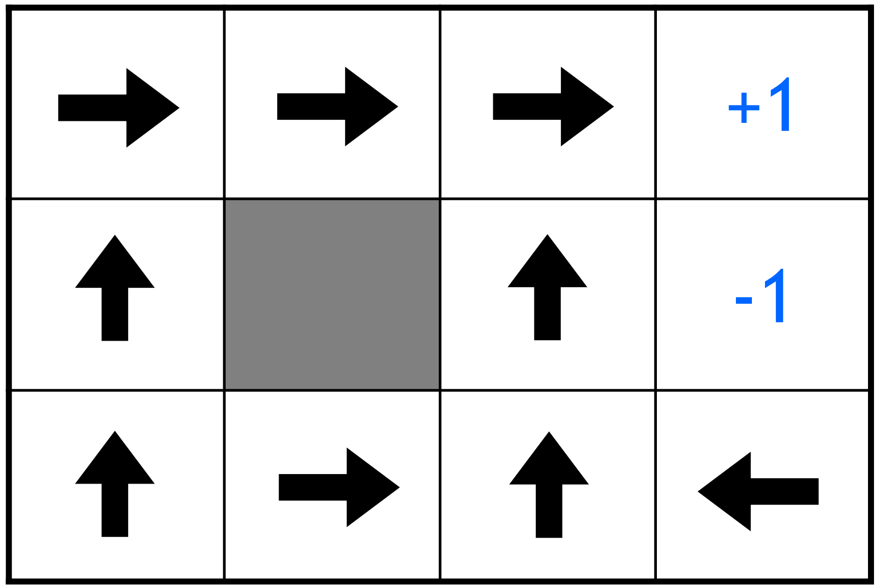
Optimal policy



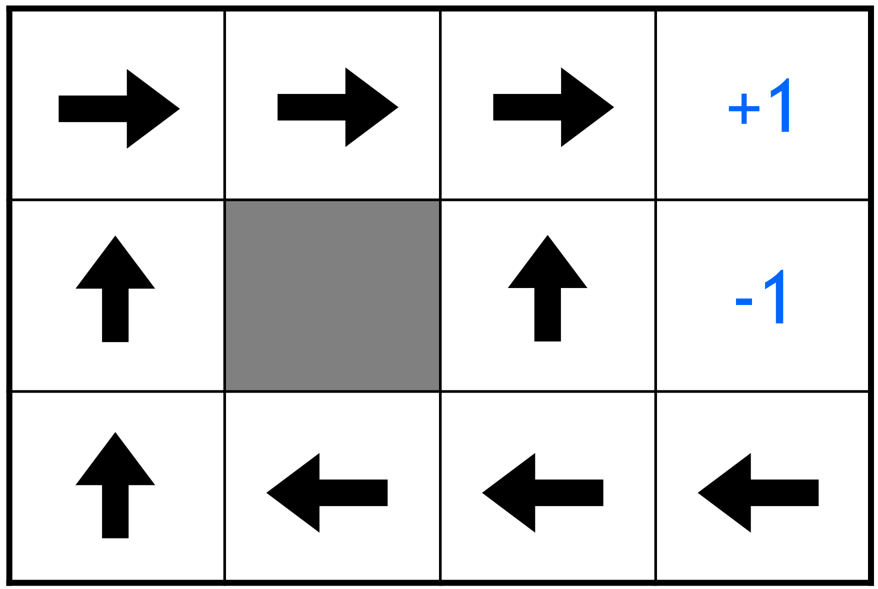
Reward for each step: -2



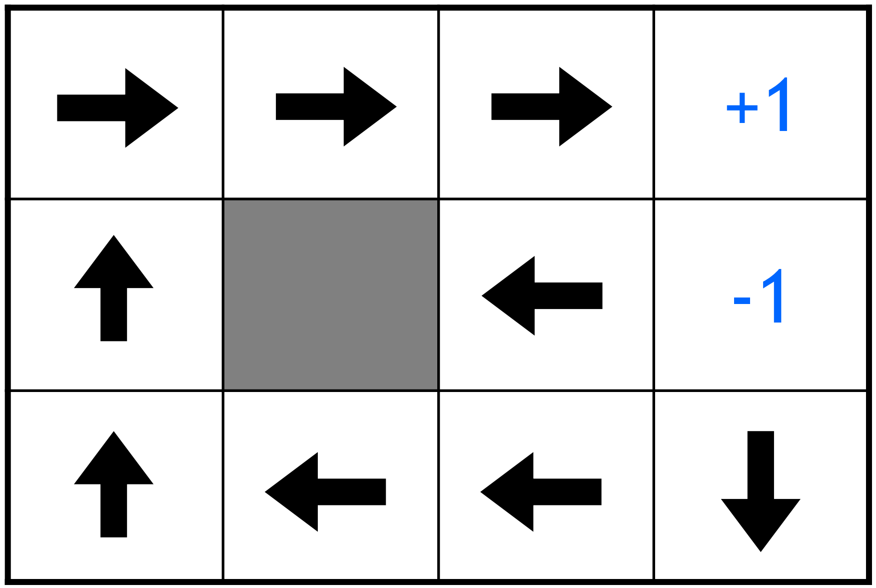
Reward for each step: -0.1



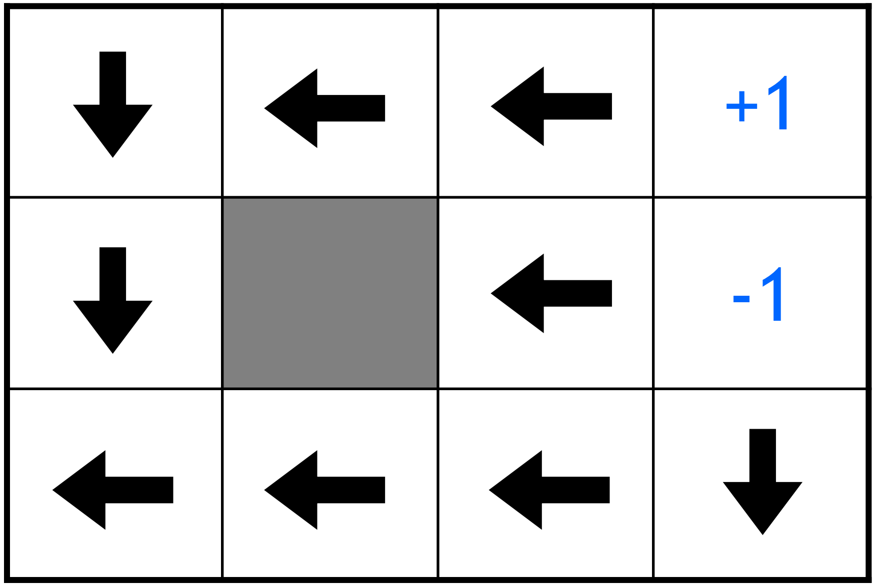
Reward for each step: -0.04



Reward for each step: -0.01

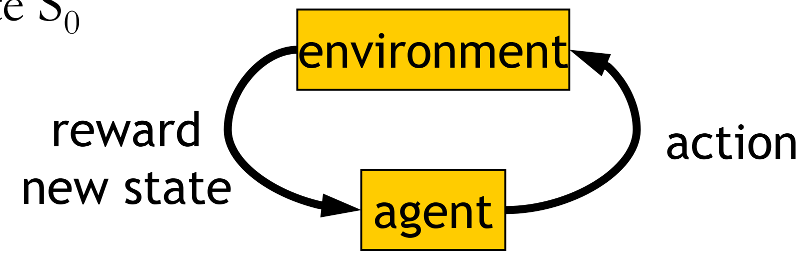


Reward for each step: +0.01



Markov Decision Process (MDP)

- set of states S, set of actions A, initial state S_0
- transition model P(s,a,s')
  - P([1,1], up, [1,2]) = 0.8
- reward function r(s)
  - r([4,3]) = +1
- goal: maximize cumulative reward in the long run



- policy: mapping from S to A
  - $\pi(s)$ or $\pi(s,a)$ (deterministic vs. stochastic)

- reinforcement learning
  - transitions and rewards usually not available
  - how to change the policy based on experience
  - how to explore the environment

Computing return from rewards

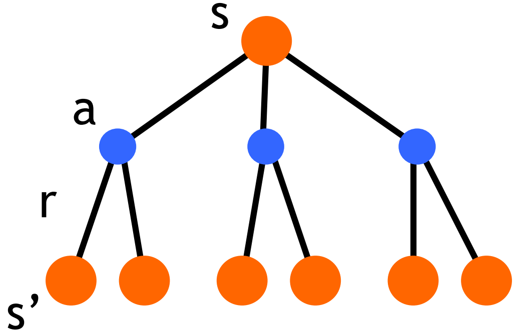
- episodic (vs. continuing) tasks
  - "game over" after N steps
  - optimal policy depends on N; harder to analyze

- additive rewards
  - $V(s_0, s_1, \ldots) = r(s_0) + r(s_1) + r(s_2) + \ldots$
  - infinite value for continuing tasks

- discounted rewards
  - $V(s_0, s_1, \ldots) = r(s_0) + \gamma^* r(s_1) + \gamma^{2*} r(s_2) + \ldots$
  - value bounded if rewards bounded

Value functions

- state utility function: $U^{\pi}(s)$
  - expected return when starting in *s* and following π
- state-action value function: $Q^{\pi}(s,a)$
  - expected return when starting in *s*, performing *a*, and following π
- useful for finding the optimal policy
  - can estimate from experience
  - pick the best action using $Q^{\pi}(s,a)$



Outline

- examples
- defining an RL problem
  - Markov Decision Processes
- solving an RL problem
  - Dynamic Programming
  - Monte Carlo methods
  - Temporal-Difference learning

Different approaches to RL

- Passive vs Active Learning
- Model-based vs Model-free learning
- Exploration vs Exploitation

Passive vs Active RL

- Passive learning
  - The agent simply watches the world going by and tries to learn the utilities of being in various states

- Active learning
  - The agent not simply watches, but also acts

Model-free vs Model-based RL

- Model based approach:
  - learn the model, and use it to derive the optimal policy
  - e.g Adaptive dynamic programming (ADP)
- Model free approach:
  - derive the optimal policy without learning the model.
  - e.g Temporal difference approach

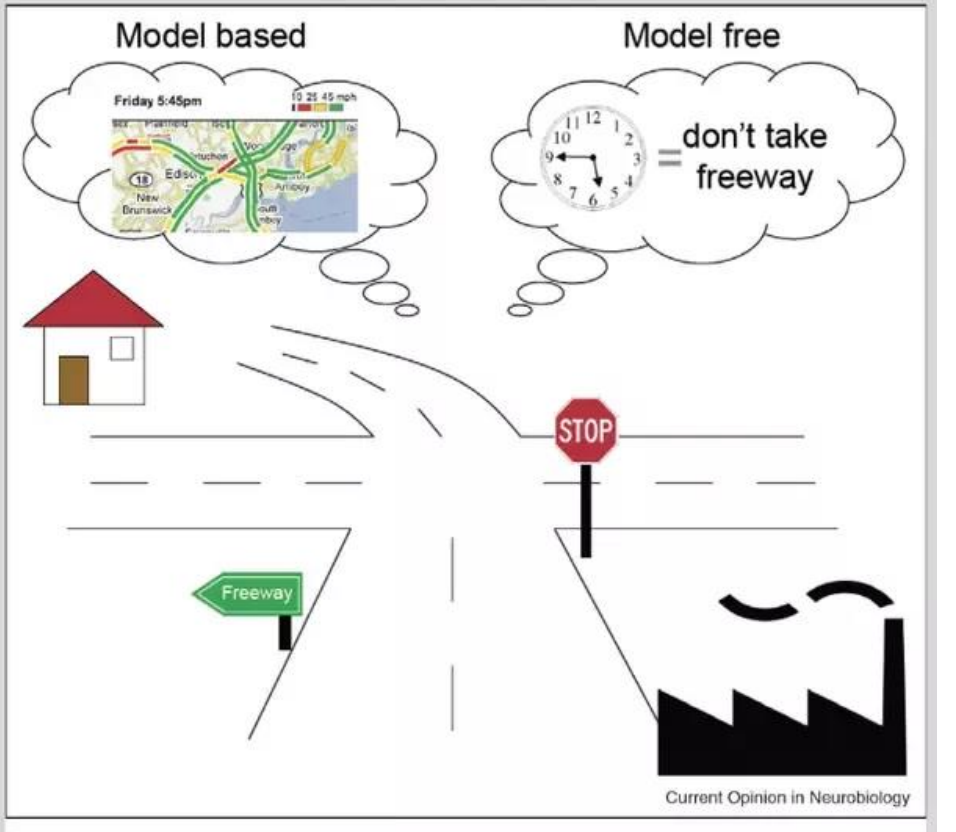


Figure 1: Two ways to choose which route to take when traveling home from work on friday evening.

Exploration vs Exploitation

- deterministic/greedy policy won't explore all actions
  - don't know anything about the environment at the beginning
  - need to try all actions to find the optimal one

- maintain exploration
  - use *soft* policies instead: $\pi(s,a)>0$ (for all s,a)

- ε-greedy policy
  - with probability 1-ε perform the optimal/greedy action
  - with probability ε perform a random action

  - will keep exploring the environment
  - slowly move it towards greedy policy: ε -> 0

Dynamic programming

- main idea
  - use value functions to structure the search for good policies
  - need a perfect model of the environment

- two main components
  - policy evaluation: compute U^π from π
  - policy improvement: improve π based on U^π

  - start with an arbitrary policy
  - repeat evaluation/improvement until convergence

Monte Carlo methods

- don't need full knowledge of environment
  - just experience, or
  - simulated experience

- but similar to DP
  - policy evaluation, policy improvement

- averaging sample returns
  - defined only for episodic tasks

Temporal Difference Learning

- combines ideas from MC and DP
  - like MC: learn directly from experience (don't need a model)
  - like DP: learn from values of successors
  - works for continuous tasks, usually faster than MC

- Examples
  - SARSA, Q-Learning

Summary

- Reinforcement learning
  - use when need to make decisions in uncertain environment
- solution methods
  - dynamic programming
    - need complete model
  - Monte Carlo
  - time-difference learning (Sarsa, Q-learning)
- most work
  - algorithms simple
  - need to design features, state representation, rewards