Reinforcement Learning

Dr. Demetrios Glinos
University of Central Florida

CAP4630 – Artificial Intelligence

Topics

- Motivation
- Reinforcement Learning Context
- Direct Utility Estimation
- Model-Based Learning
- Q-Learning
- Exploration
- Generalization

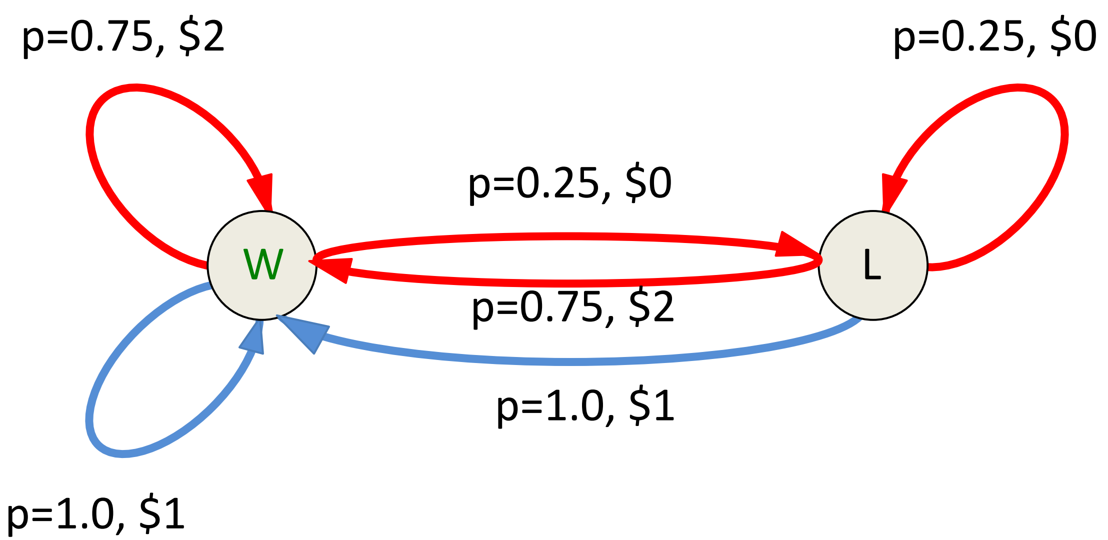
Consider Two One-Armed Bandits

Left: Pays $1 every time

Right: Pays either $2 or $0

- Game Assumptions:
  - Costs nothing to play either machine
  - Game ends after 100 turns
  - Goal: to maximize expected return

Game as an MDP

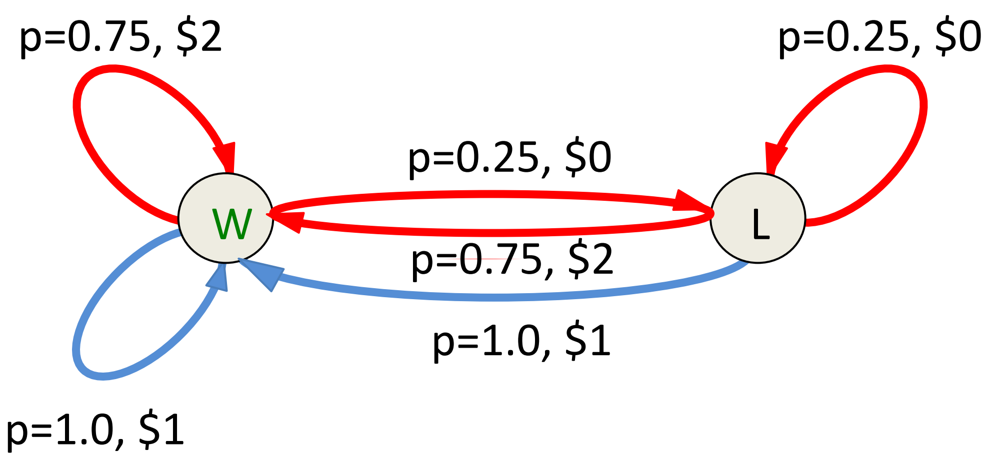


Actions: Play left, Play right
States: Won, Lost
Start state: either

Solving the MDP

- Can be solved offline
  - as a simulation, just like planning
  - need to know details of MDP
  - you don’t actually play the game

Policy	Expected payoff after 100 turns
Play left	100
Play right	150



In-Game Experience

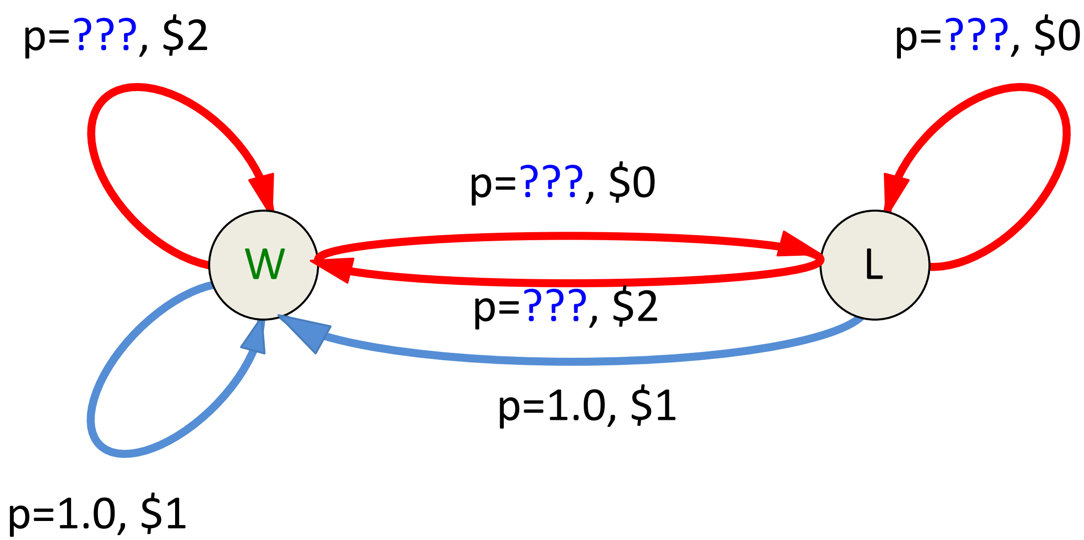
Left: Pays $1 every time

Right: Pays either $2 or $0

- Assume solved MDP per previous slide
- Based on solution, choose policy: Always play right
- Suppose we get these results: $0 $2 $0 $2 $0 $0 $0 $0 $2 $0

What we now know

- Win chance for “Play right” is different from what we expected
- But we don’t know what it is



- We cannot solve this offline
- But sooner or later we will stop always playing right

Huh? What Happened?

- We learned from experience!
  - this was *reinforcement learning*
  - *there is an MDP, but we don't know enough to solve it*
  - *so, we must use experience to infer the details*

Right: Pays either $2 or $0

- We touched on these aspects of reinforcement learning

• Exploration	need to try things out to find out what happens
• Exploitation	using what you know so far
• Regret	making mistakes
• Sampling	how much is enough
• Difficulty	learning can be much harder than offline analysis

Topics

- Motivation
- Reinforcement Learning Context
- Direct Utility Estimation
- Model-Based Learning
- Q-Learning
- Exploration
- Generalization

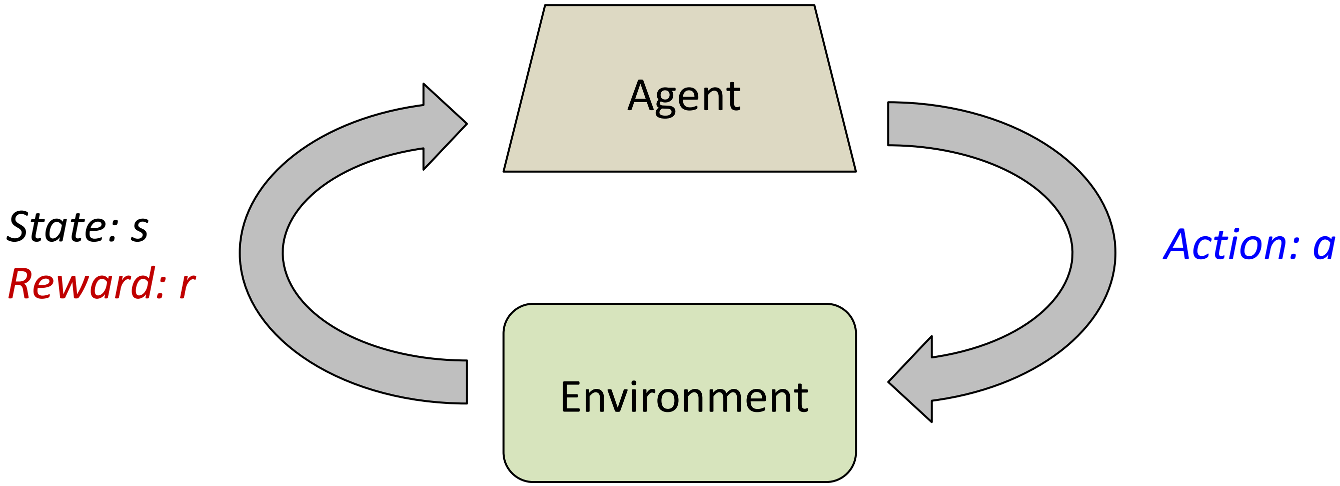
The RL Problem

- Imagine playing a new game whose rules you do not know
  - After a large number of turns, opponent declares that you have lost
  - Making sense of this is reinforcement learning

- RL can sometimes be
  - the only way (supervised learning infeasible)
  - the best way

- Examples:
  - TD-Gammon
  - Program to fly a drone

RL Paradigm



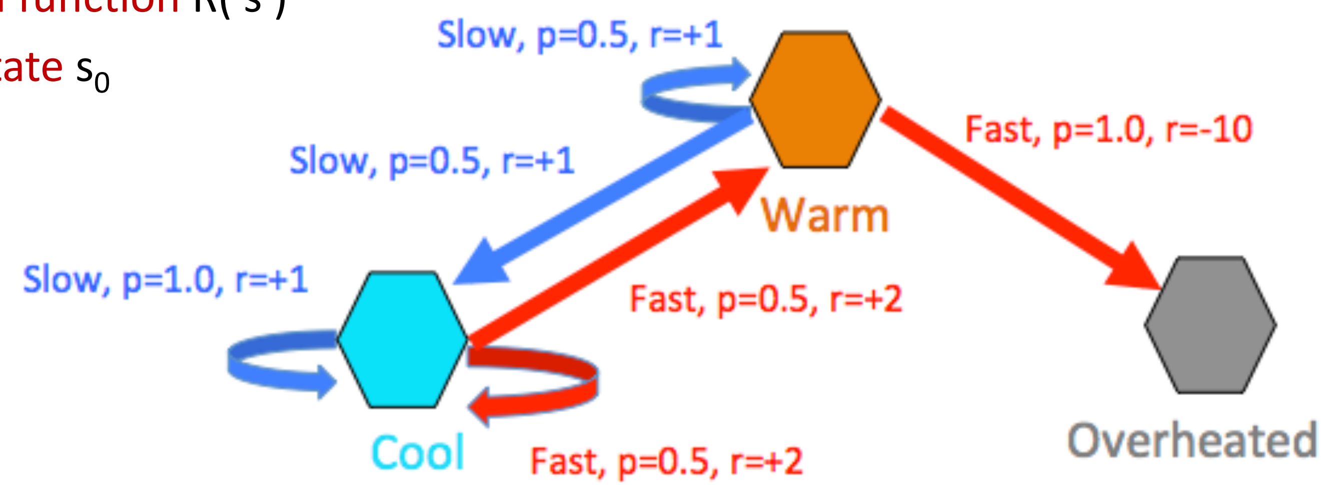
Basic idea:

- Feedback is in the form of rewards
- Unknown reward function determines the utilities
- Agent must learn how to act to maximize expected rewards
- All learning based on observed samples of outcomes

Assumption: There is an underlying MDP

- The universe is not random

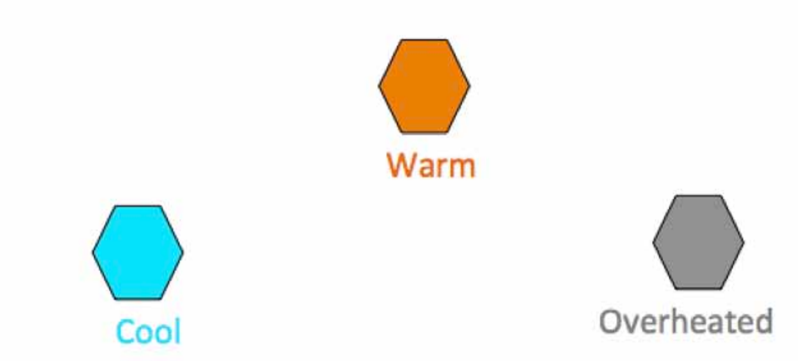
- What is going on is still an MDP
  - A set of states $s \in S$
  - A set of actions $a \in A$
  - A transition model P(s' | s, a)
  - A reward function R(s)
  - A start state s_0



What We Know About the MDP

- We don't know everything

- What we know of the MDP
  - A set of states $s \in S$
  - A set of actions $a \in A$
  - A start state s_0



- We don't know
  - The transition model P(s' | s, a)
  - The reward function R(s)

- We still want to find an optimal policy **π(s)**
- But we can't figure it out by analysis (offline)
- We must take action and see what happens (online)

How we Can Use Experience

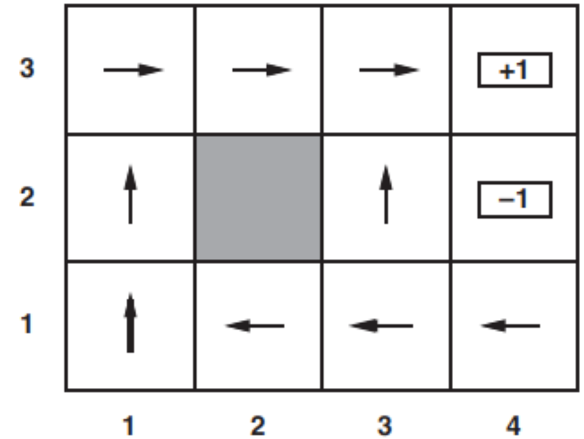
- Direct Utility Estimation
  - A simplified RL task
  - Follow a *given* policy and experience rewards to determine state values
- Model-Based Learning
  - Follow an initial policy
  - Take actions and develop an approximate model
  - Use Value Iteration to evaluate the model and determine state values
- Q-Learning
  - This is active RL
  - Take actions, experience rewards, and develop q-values
  - Must explore, as well as exploit

Topics

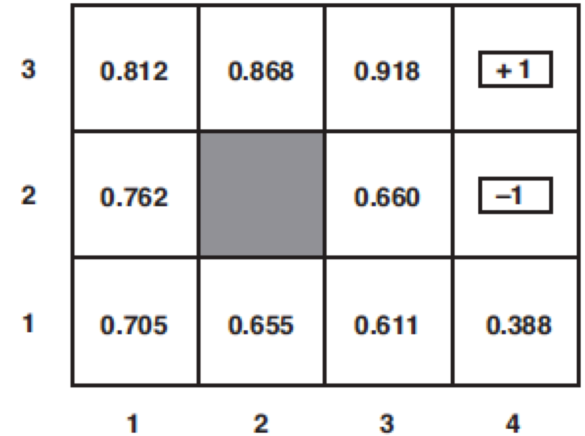
- Motivation
- Reinforcement Learning Context
- Direct Utility Estimation
- Model-Based Learning
- Q-Learning
- Exploration
- Generalization

Direct Utility Estimation

- **Direct Utility Estimation** aka **Direct Evaluation**
- Goal: Compute values for each state, for a *given* policy π
- Simple procedure:
  - Act strictly according to π
  - Record sum of discounted rewards for each state visited (all the way to end)
  - Average the samples collected
- Use the "reward-to-go" definition of the utility of a state:
  - i.e., the expected total reward from that state onward



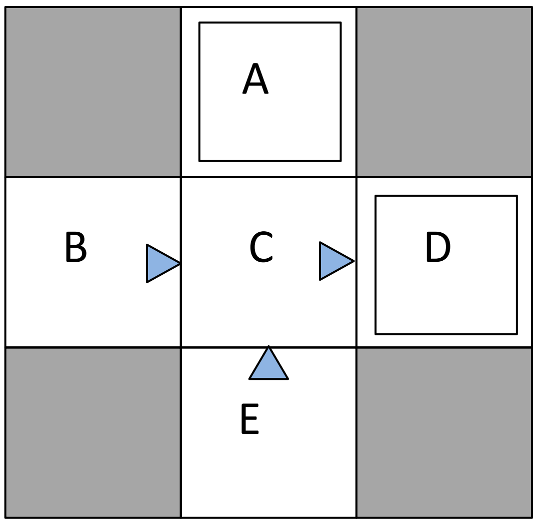
source: Fig 17.2(a)



source: Fig 17.3

Example: Direct Utility Estimation

Input Policy π



assume γ = 1

Q: How do we get V(C)= +4?

Trial episodes

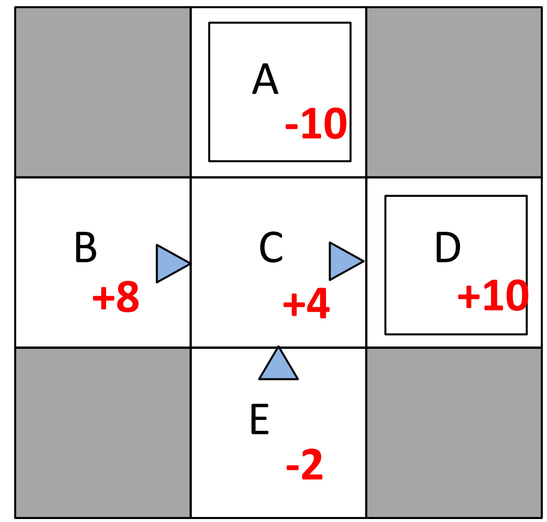
B, right, C, -1
C, right, D, -1
D, exit, _, +10

B, right, C, -1
C, right, D, -1
D, exit, _, +10

E, up, C, -1
C, right, D, -1
D, exit, _, +10

E, up, C, -1
C, right, A, -1
A, exit, _, -10

Output values



→ This approach misses learning opportunities

- utilities are not independent
- convergence often slow

Topics

- Motivation
- Reinforcement Learning Context
- Direct Utility Estimation
- Model-Based Learning
- Q-Learning
- Exploration
- Generalization

Model-Based Learning

- Basic idea:
  - Learn an ***approximate model*** from observations
    - follow some initial policy
  - Once learned, assume the model is correct
  - Use the model offline to solve for values
  - Revised values give us a better policy

- Learning the approximate model
  - count outcomes s' for each (s,a)
  - normalize to estimate $\hat{P}(s' \mid s, a)$
  - Discover $\hat{R}(s')$ when s' occurs

- Solving the learned MDP
  - Use value iteration (or policy iteration)

Example:

Suppose in state s and choose action *a* 10 times, and 6 of those times end up in state s', then

$$\hat{P}(s' \mid s, a) = 6/10 = 0.60$$

Example: Model-Based Learning

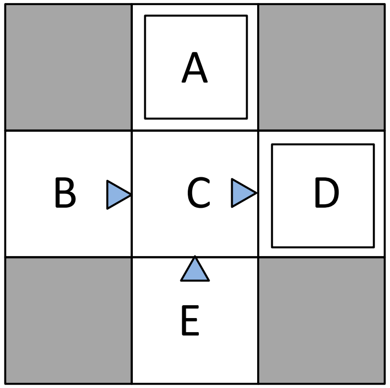
Same game as before:

What we know:

- board configuration
- A and D are terminal states
- Actions: up, down, left, right
- start in State B

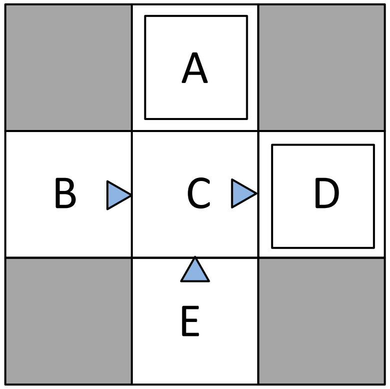
We are given:

- the input policy π(s) (shown by triangles)
- we also assume γ = 1



Example: Model-Based Learning (cont'd)

Input Policy π



Trial observations

B, right, C, -1
C, right, D, -1
D, exit, _, +10

B, right, C, -1
C, right, D, -1
D, exit, _, +10

E, up, C, -1
C, right, D, -1
D, exit, _, +10

E, up, C, -1
C, right, A, -1
A, exit, _, -10

Learned model

P(C|B, right) = 1.00
P(D|C, right) = 0.75
P(A|C, right) = 0.25
P(_|D, exit) = 1.0
P(_|A, exit) = 1.0

R(C) = -1
R(D) = -1
R(A) = -1
R(exit from D) = +10
R(exit from A) = -10

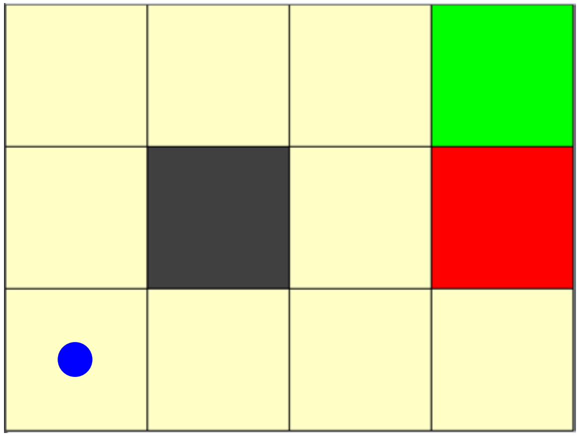
Given the approximate learned model, we can use value iteration to compute the values, from which we can obtain a better policy

Topics

- Motivation
- Reinforcement Learning Context
- Direct Utility Estimation
- Model-Based Learning
- Q-Learning
- Exploration
- Generalization

Active Reinforcement Learning

- This is full reinforcement learning
  - still don't know transition function
  - still don't know reward function
  - this time, policy π(s) is NOT fixed
  - agent actively chooses actions



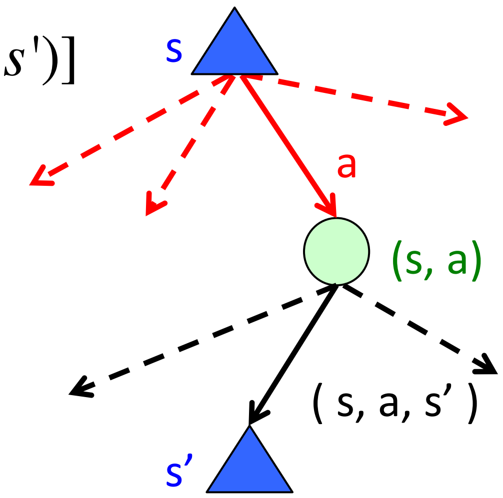
- Goal: learn an optimal policy / values

- How this works
  - Agent makes choices and gains experience
  - Must try different things (*"Nothing ventured, nothing gained"*)
  - Balance exploration v. exploitation
  - This is NOT offline planning

Q-Value Iteration

- Recall: Value Iteration
  - Start with zero vector: $V_0(s) = 0$, for all s
  - Iterate:

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



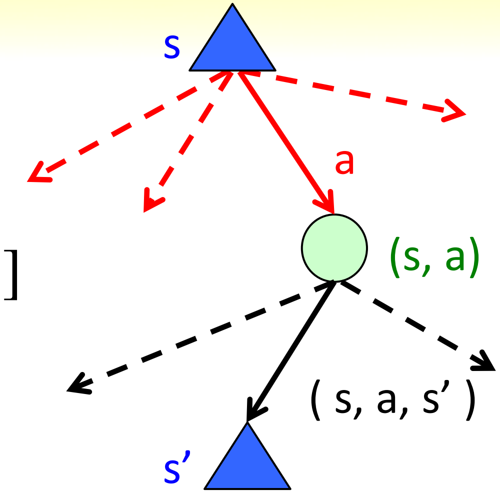
- Big idea: drop down a half level and do expectimax from the chance node instead

- This allows Q-value iteration, which is more useful
  - Start with zero vector: $Q_0(s,a) = 0$, for all (s,a)
  - Iterate:

$$\boxed{Q_{k+1}(s,a) \leftarrow \sum_{s'} P(s' \mid s,a)[R(s') + \gamma \max_{a'} Q_k(s',a')]}$$

- **Q:** Why would this be more useful?

Q-Learning



- Q-Learning is **sample-based Q-value iteration**

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

- Learn Q(s,a) values as you go
  - Start with Q(s,a)=0 for all q-states
  - Get trial observations: (s,a,s',r)
  - Start with old estimate: Q(s,a)
  - Interpret sample:

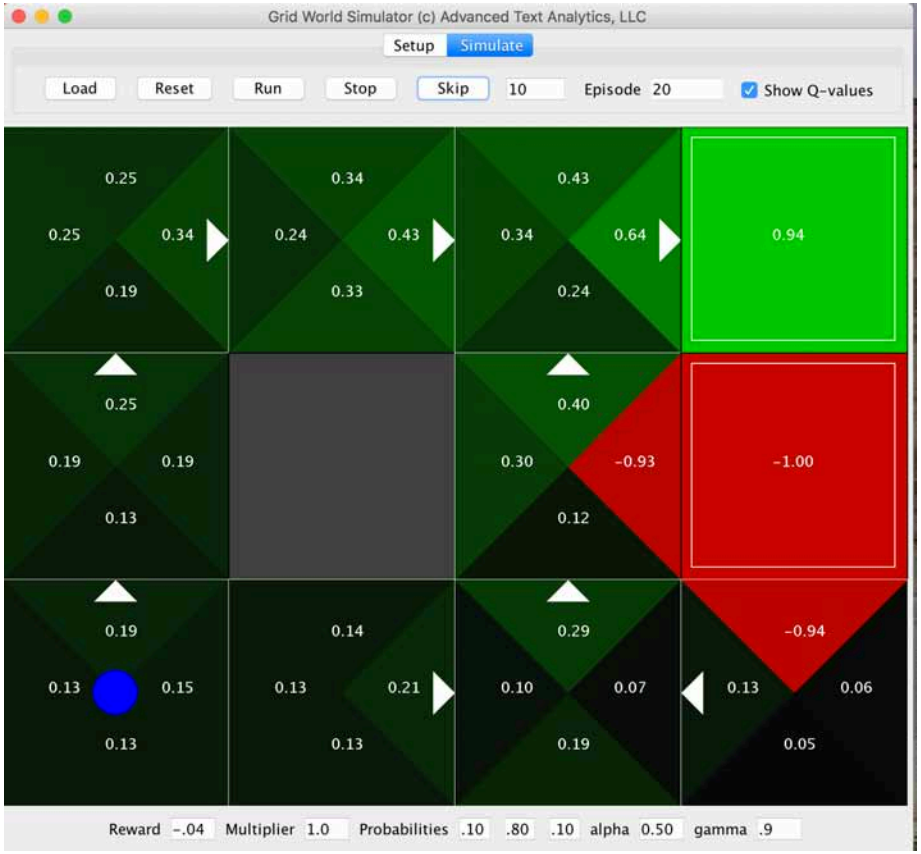
Note: $max_{a'}Q$ (s',a') is just the approximate V(s')*

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

  - Fold new sample into running average Q-value using learning rate α:

$$\boxed{Q_{new}(s,a) \leftarrow (1 - \alpha)\, Q_{old}(s,a) + (\alpha)\,(\text{sample})}$$

Q-Learning Demo



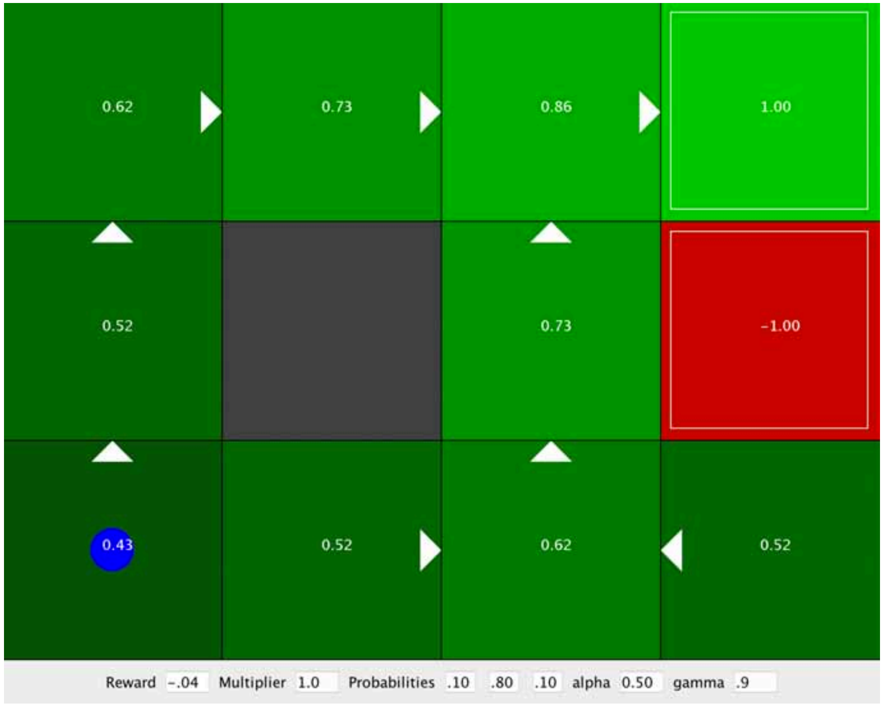

```
From [ 0, 2 ]  try NORTH do NORTH end [ 0, 1 ] reward -0.04
From [ 0, 1 ]  try EAST do EAST end [ 0, 1 ] reward -0.04
From [ 0, 1 ]  try NORTH do WEST end [ 0, 1 ] reward -0.04
From [ 0, 1 ]  try NORTH do NORTH end [ 0, 0 ] reward -0.04
From [ 0, 0 ]  try EAST do NORTH end [ 0, 0 ] reward -0.04
From [ 0, 0 ]  try EAST do EAST end [ 1, 0 ] reward -0.04
From [ 1, 0 ]  try EAST do EAST end [ 2, 0 ] reward -0.04
From [ 2, 0 ]  try EAST do SOUTH end [ 2, 1 ] reward -0.04
From [ 2, 1 ]  try NORTH do WEST end [ 2, 1 ] reward -0.04
From [ 2, 1 ]  try NORTH do NORTH end [ 2, 0 ] reward -0.04
From [ 2, 0 ]  try EAST do EAST end [ 3, 0 ] reward -0.04
EXIT from [ 0, 2 ] reward 1.00
```

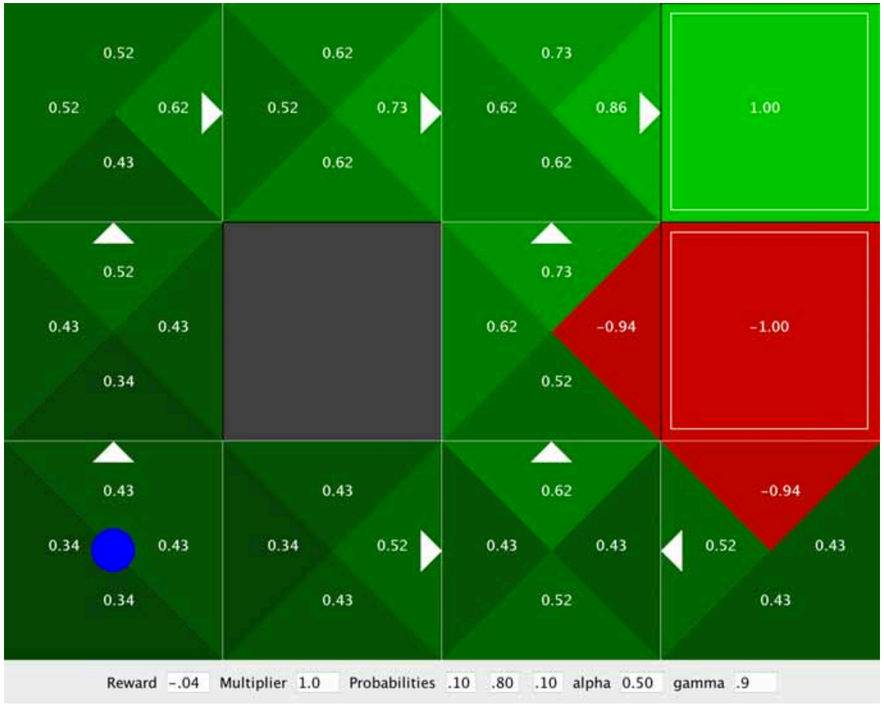
demo: GridSim

Using Q-values

- Once the Q-values are learned, the state values are easily computed
  - just take the max of the Q-values over all actions from the state

- The optimal policy is to choose the action that produces the highest Q-value for the state





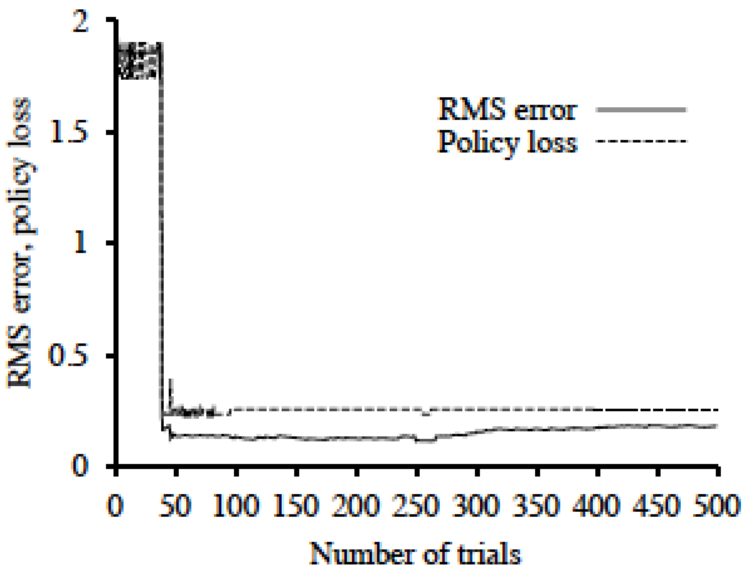
Q-Learning Convergence

- Q-Learning converges to optimal policy
  - even if not acting optimally
  - this is called "off-policy learning"
- Requirements
  - Must explore enough
  - Must eventually lower the learning rate sufficiently to stop learning
- Bottom line:
  - How you select actions while you learn does not really matter *in the limit*
  - OK to start with zero values

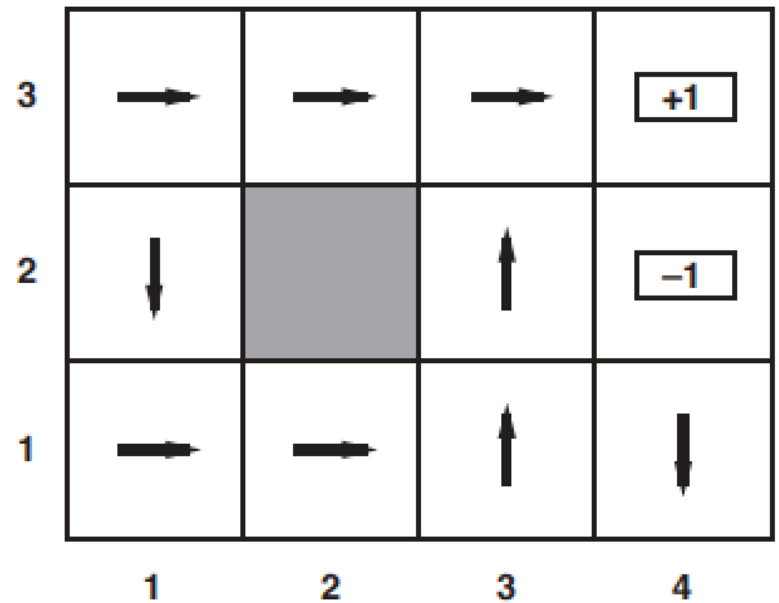
Topics

- Motivation
- Reinforcement Learning Context
- Direct Utility Estimation
- Model-Based Learning
- Q-Learning
- Exploration
- Generalization

Exploration – Exploitation Tradeoff



source: Fig 17.5(a)



source: Fig 17.2(a)

- Greedy agent that follows currently recommended (presumed optimal) action at each step
- Converges to a suboptimal policy

- Exploitation is about maximizing reward
- Exploration is about maximizing long-term well-being

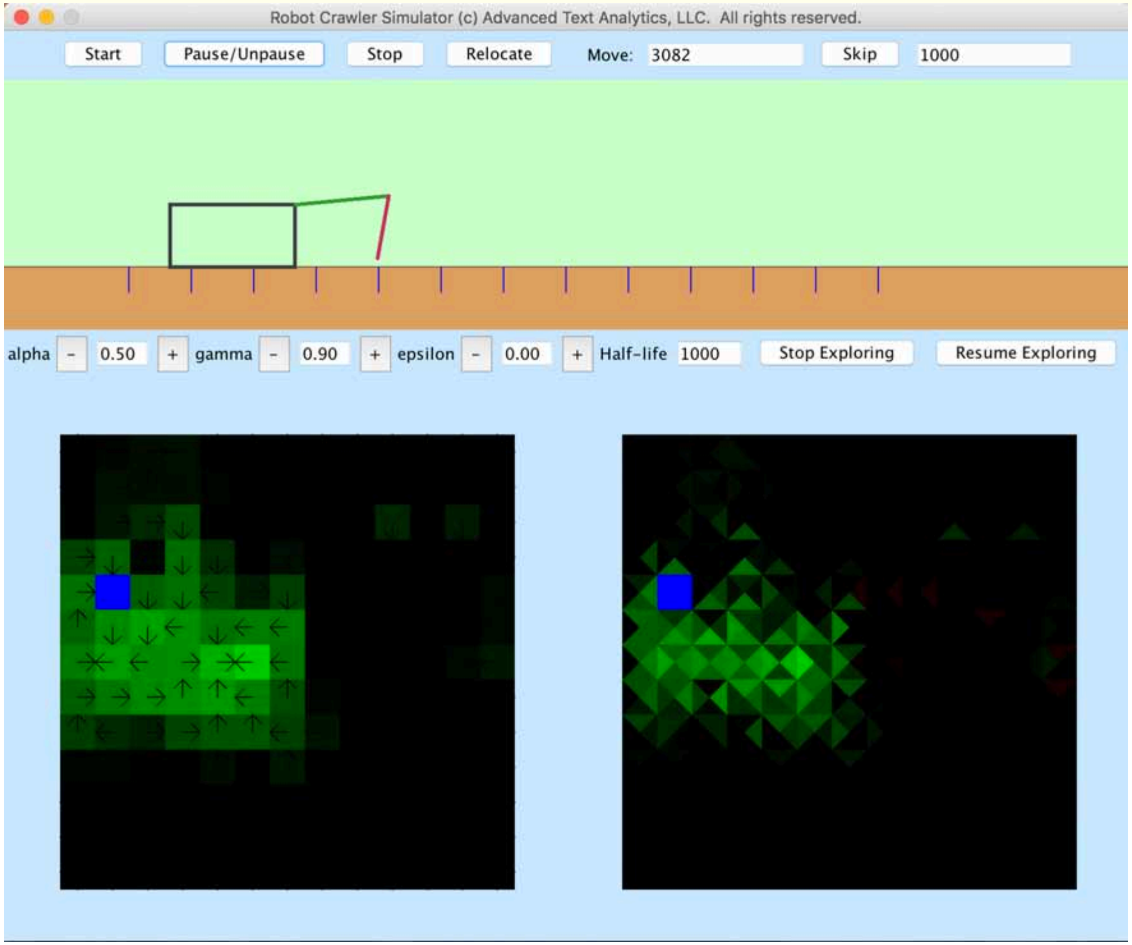
How to Explore

- Must be **GLIE – Greedy in the Limit of Infinite Exploration**
  - must try each action an *infinite* number of times to avoid having a finite probability of missing an optimal action due to a really bad series of outcomes
    - will learn the optimal model
  - must also eventually become greedy
    - must eventually stop exploring and follow the learned (optimal) model to avoid thrashing once done learning

- Simplest GLIE scheme: **"ε-greedy"**
  - choose a random action 1/t of the time and follow current policy otherwise
    - can be extremely slow

- Improving exploration
  - we can lower exploration threshold over time (to avoid thrashing)
  - we can use exploration functions that assign higher utility estimate to relatively unexplored state-action pairs

Q-Learning Example



This model uses an exponential decay exploration function

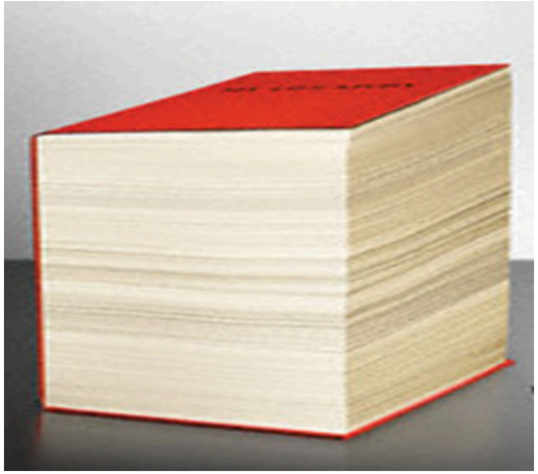
demo: crawl

Topics

- Motivation
- Reinforcement Learning Context
- Direct Utility Estimation
- Model-Based Learning
- Q-Learning
- Exploration
- Generalization

Practical Limits

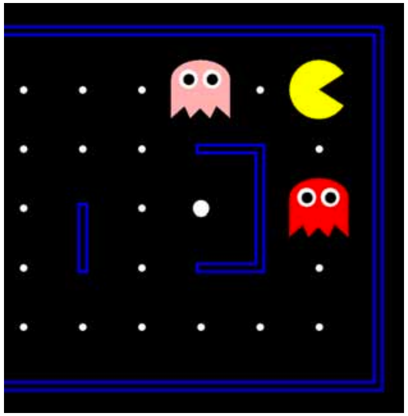
- Basic Q-Learning keeps track of ***all*** Q-values
  - conceptually, as a table
- This is not possible in realistic situations
  - Too many states to store them all in memory
  - Too many states to visit in training
- Instead, we need to generalize
  - Learn what we can from experience
  - Generalize what we learned to new, but similar situations



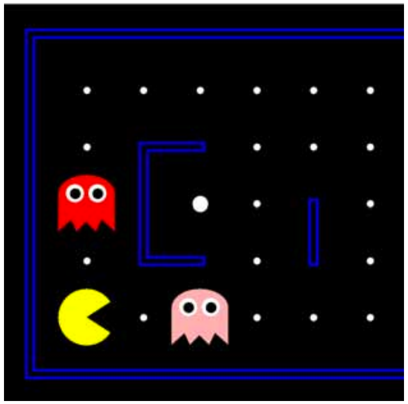
Book of Every Chess Position

Generalization

- If we learn through experience that this state is bad



- Then we know nothing about this one, unless:
  - we learn it separately
  - or, we generalize from the state above



Feature-Based Representation

- **Features** are functions over states to real numbers
  - distance to closest ghost for Pac-Man
  - 1 / (distance to dot)^2
  - is Pac-Man in a tunnel, etc.

Note: the feature functions themselves can encode nonlinear conditions

- We can write state values as *linear combinations* of feature values

$$V(s) = w_1 f_1(s) + w_2 f_2(s) + \dots + w_n f_n(s)$$

- We can also express q-state values as linear combinations of feature values

  e.g., the action moves Pac-Man closer to food, power, or ghost

$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \dots + w_n f_n(s,a)$$

- Advantages: features give us a compact representation of value
- Disadvantages: states with similar features may have very different values

Approximate Q-Learning

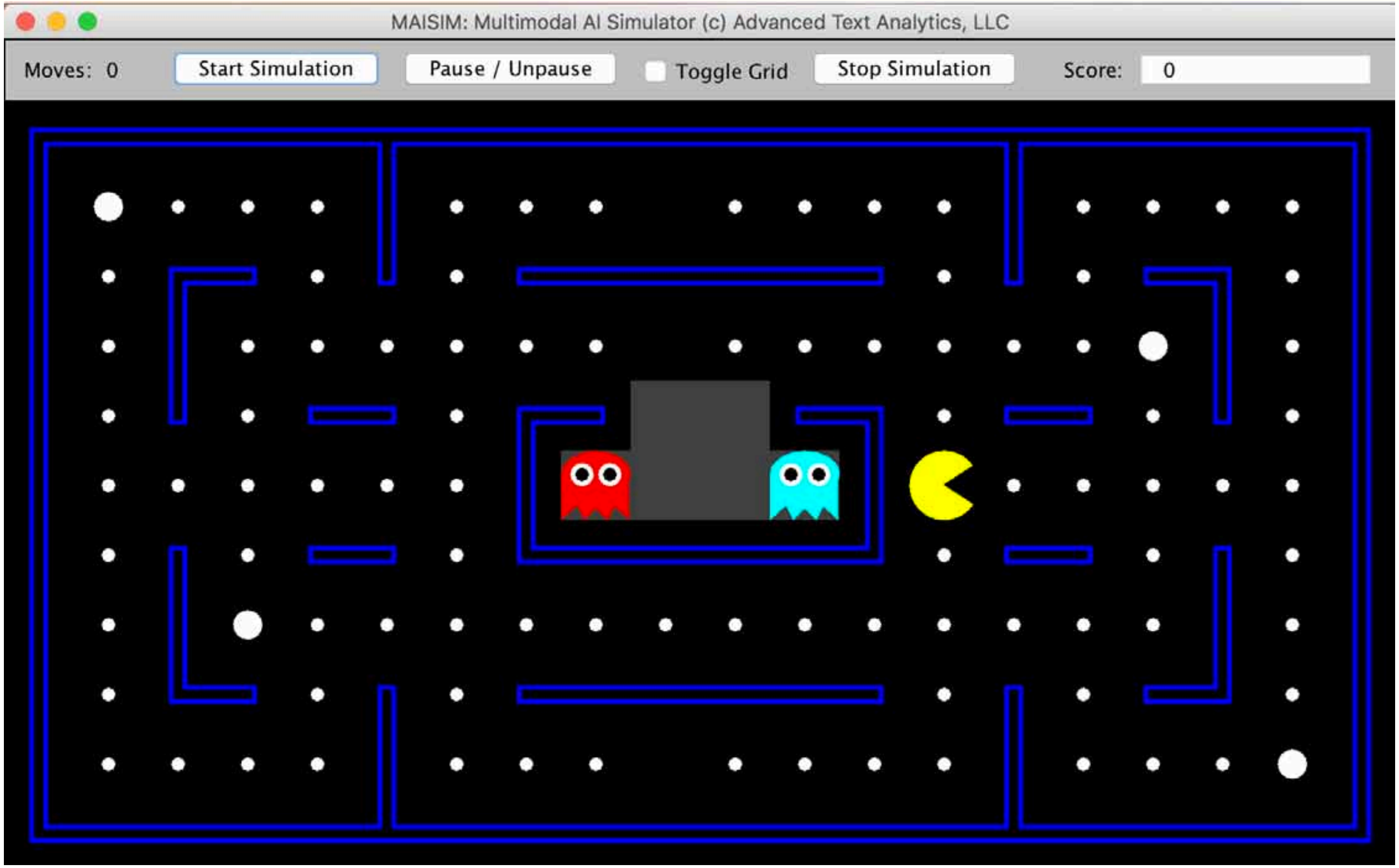
- Compute Q-values using feature function

$$\boxed{Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \ldots + w_n f_n(s,a)}$$

- Q-learning with linear Q-functions:
  - experience a transition: (s,a,r,s')
  - compute difference = [immediate reward + discounted future] – Q(s,a)
  - for exact Q-learner, just update table entry: $Q(s,a) = Q(s,a) + \alpha$ [difference]
  - for approximate Q-learner, update weights: $w_i \leftarrow w_i + \alpha$ [difference] $f_i(s,a)$

→ The learning algorithm finds the values of the weights that maximize our expected utility, so we don't need to hunt around manually to find them

Approximate Q-Learning Demo



This implementation uses only 2 features and wins about 60% of the time

demo: qlearn2