CS 221: Artificial Intelligence

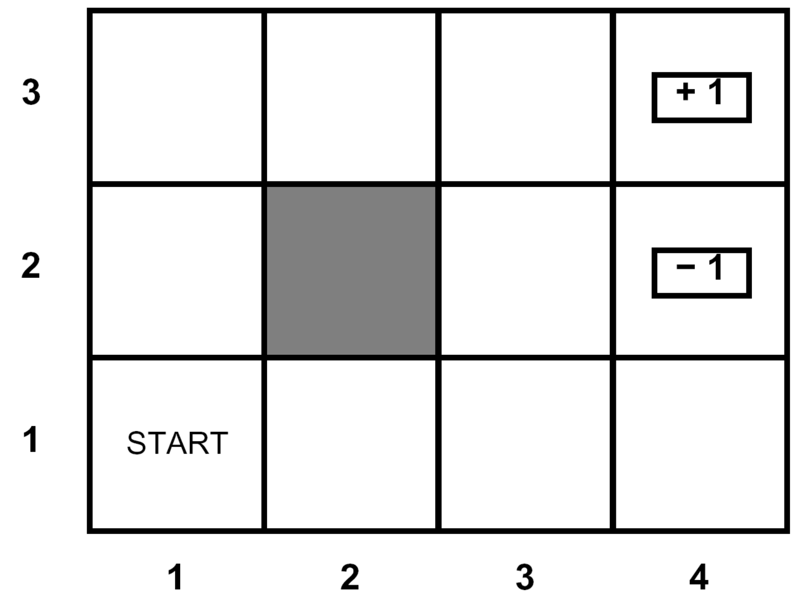
Reinforcement Learning

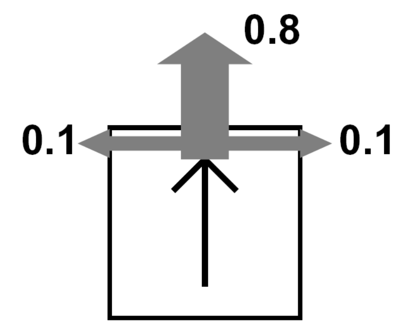
Peter Norvig and Sebastian Thrun

Slide credit: Dan Klein, Stuart Russell, Andrew Moore

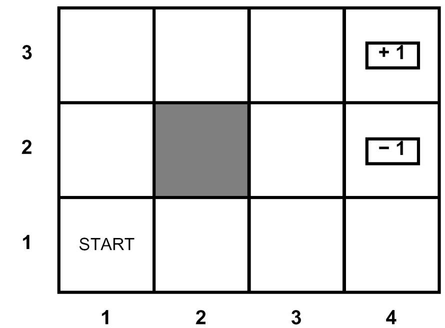
Review: Markov Decision Processes

- An MDP is defined by:
  - set of states $s \in S$
  - set of actions $a \in A$
  - transition function $P(s' \mid s, a)$ aka $T(s,a,s')$
  - reward function $R(s, a, s')$ or $R(s, a)$ or $R(s')$
- Problem:
  - Find policy $\pi(s)$ to maximize $\sum \gamma^t R(s, \pi(s), s')$





Value Iteration



- Idea:
  - Start with $U_0(s) = 0$
  - Given $U_i(s)$, calculate:

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

  - Repeat until convergence

- Theorem: will converge to unique optimal utilities

$$U^{\pi}(s) = R(s) + \gamma \sum_{s'} P(s' \mid s, \pi(s)) U^{\pi}(s') \, .$$

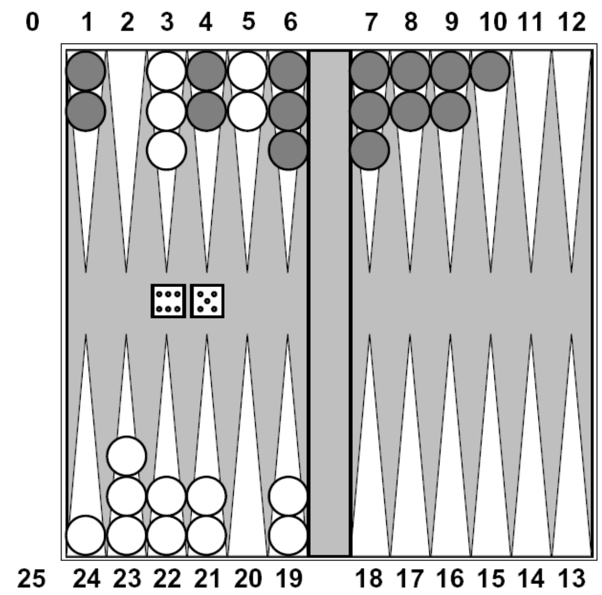
- But: Requires model (R and P)

Reinforcement Learning

- What if the state transition function P and the reward function R are *unknown*?

Example: Backgammon

- Reward only for win / loss in terminal states, zero otherwise
- TD-Gammon (Tesauro, 1992) learns a function approximation to U(s) using a neural network
- Combined with depth 3 search, one of the top 3 human players in the world – good at positional judgment, but mistakes in end game beyond depth 3



Example: Animal Learning

- RL studied experimentally for more than 60 years in psychology
  - Rewards: food, pain, hunger, drugs, etc.
  - Mechanisms and sophistication debated

- Example: foraging
  - Bees learn near-optimal foraging plan in field of artificial flowers with controlled nectar supplies
  - Bees have a direct neural connection from nectar intake measurement to motor planning area

What to Learn

- If you don't know P, R, what you learn depends on what you want to do
  - **Utility-based agent:** learn U(s)
  - **Q-learning agent:** learn utility Q(s, a)
  - **Reflex agent:** learn policy $\pi(s)$
- And on how patient/reckless you are
  - **Passive:** use a fixed policy π
  - **Active:** modify policy as you go

Passive Temporal-Difference

function PASSIVE-TD-AGENT(*percept*) **returns** an action
  inputs: *percept*, a percept indicating the current state s' and reward signal r'
  persistent: π, a fixed policy
    U, a table of utilities, initially empty
    N_s, a table of frequencies for states, initially zero
    s, a, r, the previous state, action, and reward, initially null

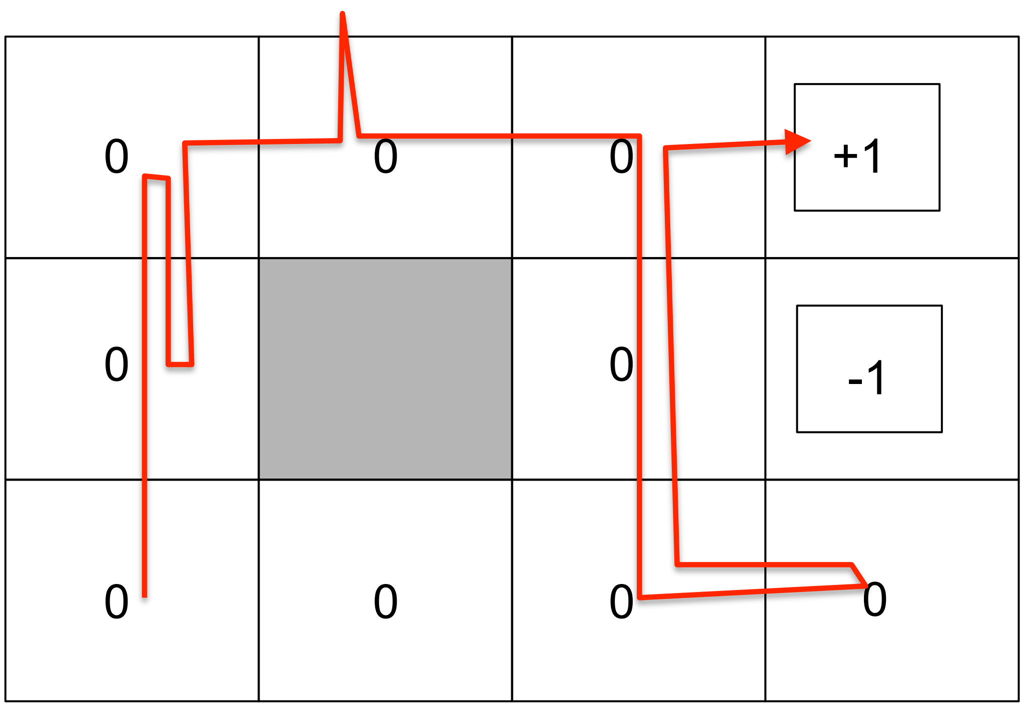
  if s' is new **then** $U[s'] \leftarrow r'$
  if s is not null **then**
    increment $N_s[s]$
    $U[s] \leftarrow U[s] + \alpha(N_s[s])(r + \gamma\, U[s'] - U[s])$
  if s'.TERMINAL? **then** $s, a, r \leftarrow$ null **else** $s, a, r \leftarrow s', \pi[s'], r'$
  return a

Example

0	0	0	+1
0		0	-1
0	0	0	0

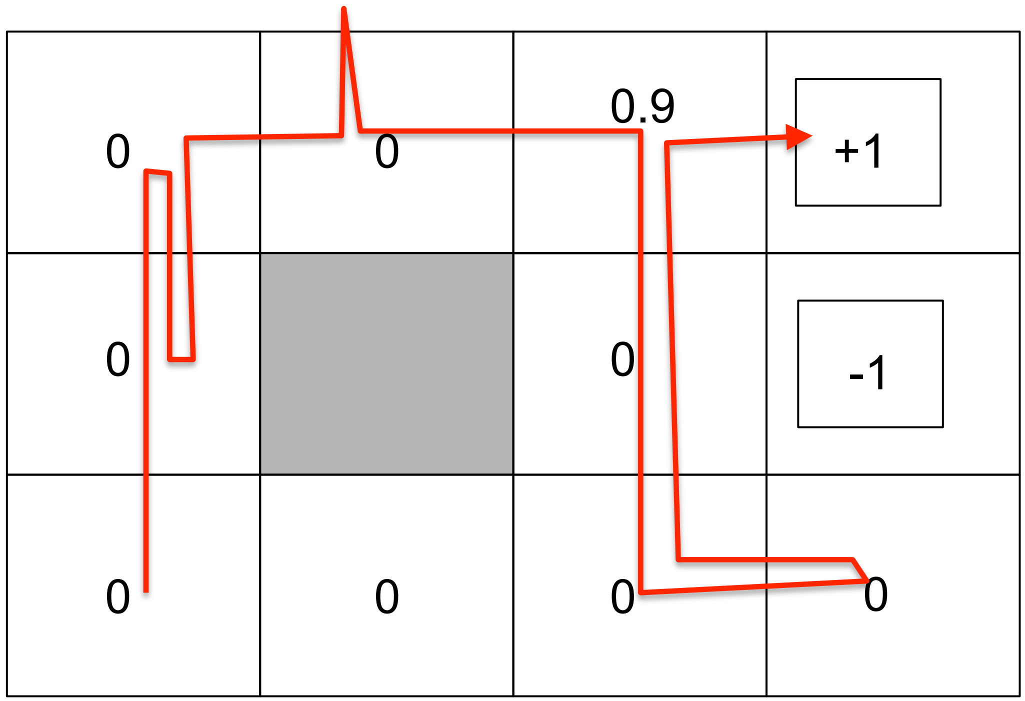
Example

$$U^\pi(s) \leftarrow U^\pi(s) + \alpha(R(s) + \gamma U^\pi(s') - U^\pi(s))$$



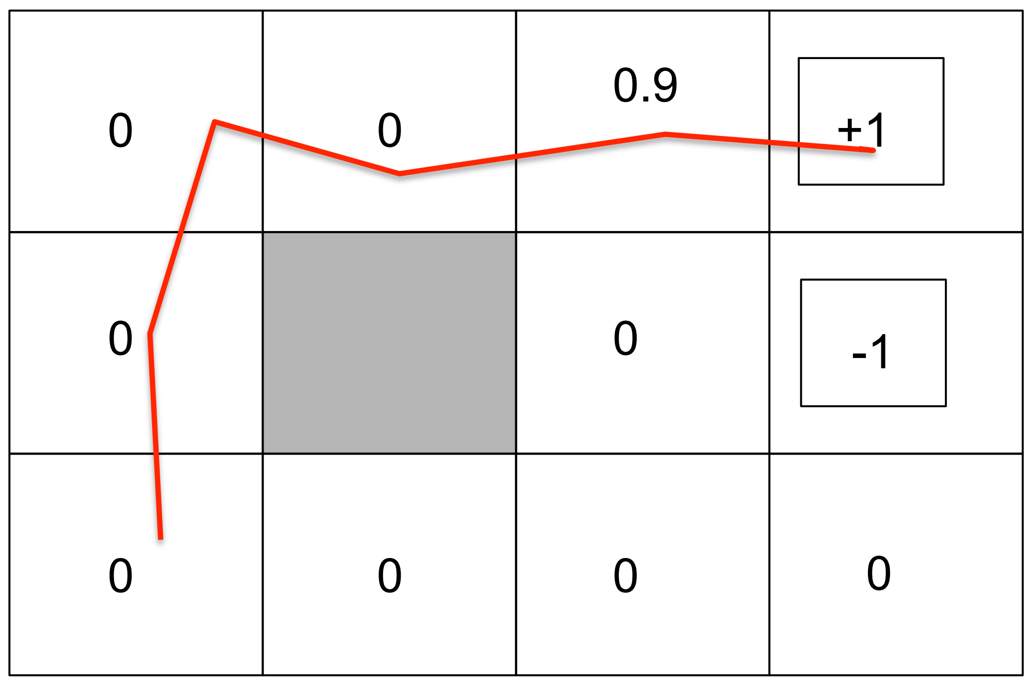
Example

$$U^\pi(s) \leftarrow U^\pi(s) + \alpha(R(s) + \gamma U^\pi(s') - U^\pi(s))$$



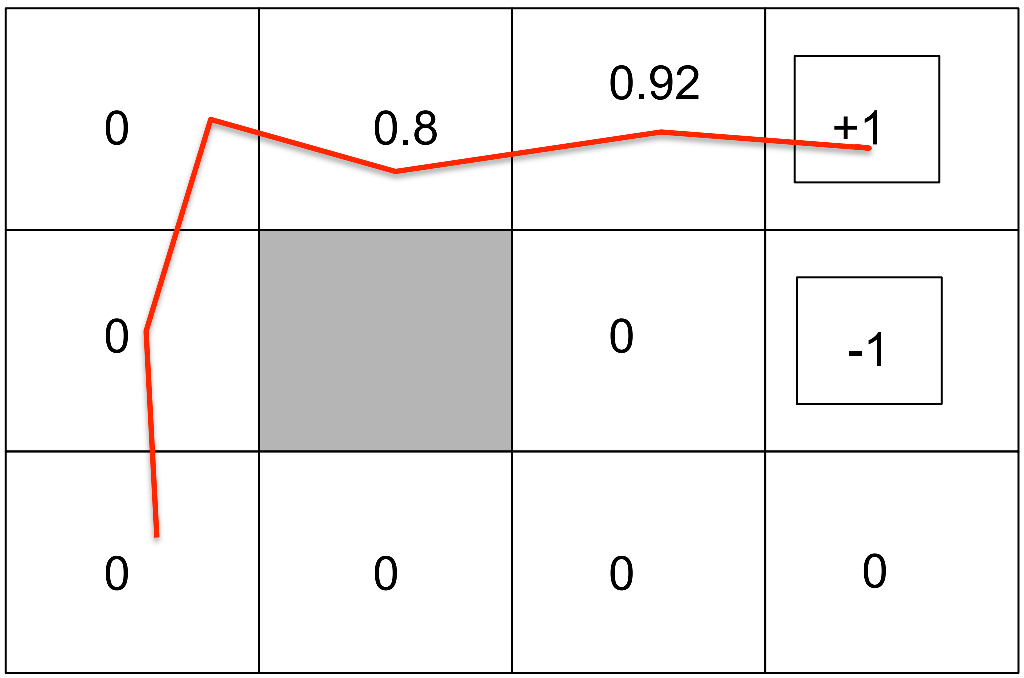
Example

$$U^\pi(s) \leftarrow U^\pi(s) + \alpha(R(s) + \gamma U^\pi(s') - U^\pi(s))$$



Example

$$U^\pi(s) \leftarrow U^\pi(s) + \alpha(R(s) + \gamma U^\pi(s') - U^\pi(s))$$



Example

$$U^\pi(s) \leftarrow U^\pi(s) + \alpha(R(s) + \gamma U^\pi(s') - U^\pi(s))$$

.20	.28	.36	+1
.17		-.12	-1
.12	-0.01	-.16	-.2

Sample results

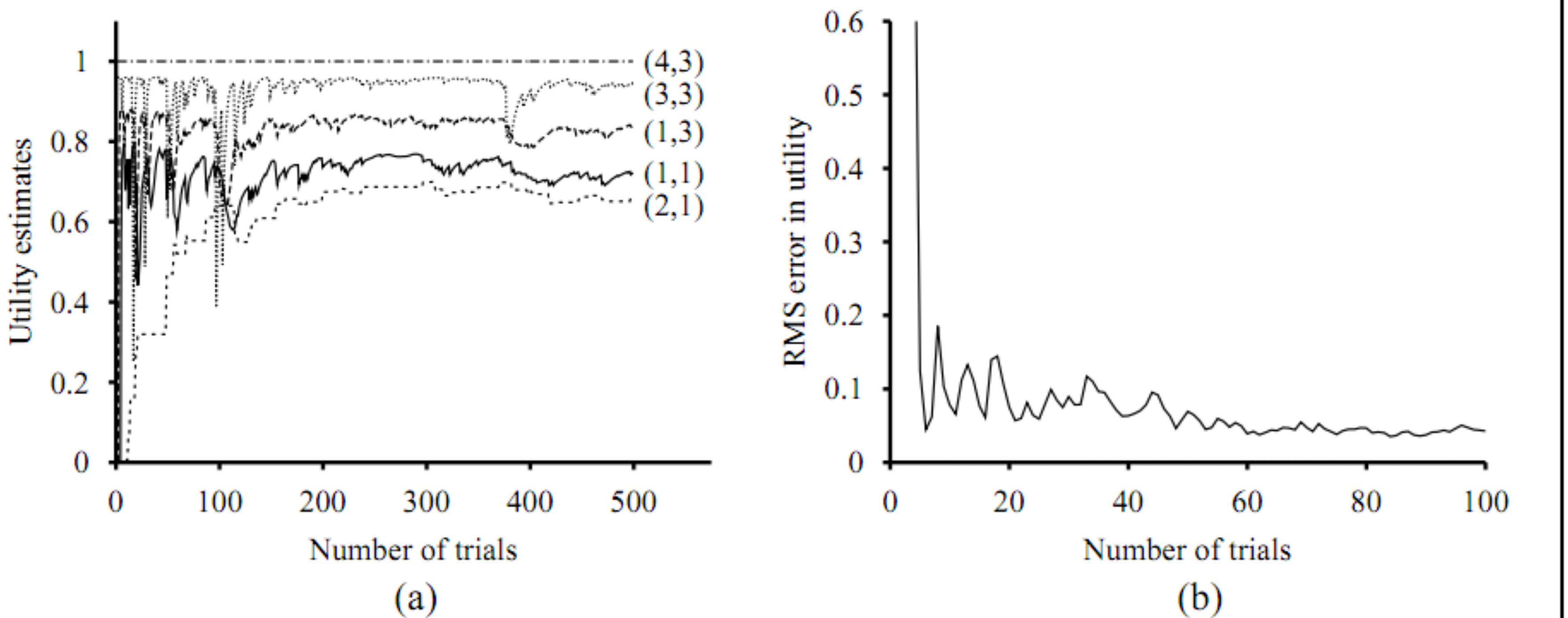


Figure 21.5 The TD learning curves for the 4×3 world. (a) The utility estimates for a selected subset of states, as a function of the number of trials. (b) The root-mean-square error in the estimate for $U(1,1)$, averaged over 20 runs of 500 trials each. Only the first 100 trials are shown to enable comparison with Figure 21.3.

Problem

- The problem
  - Algorithm computes the value of one specific policy
  - May never try the best moves
  - May never visit some states
  - May waste time computing unneeded utilities

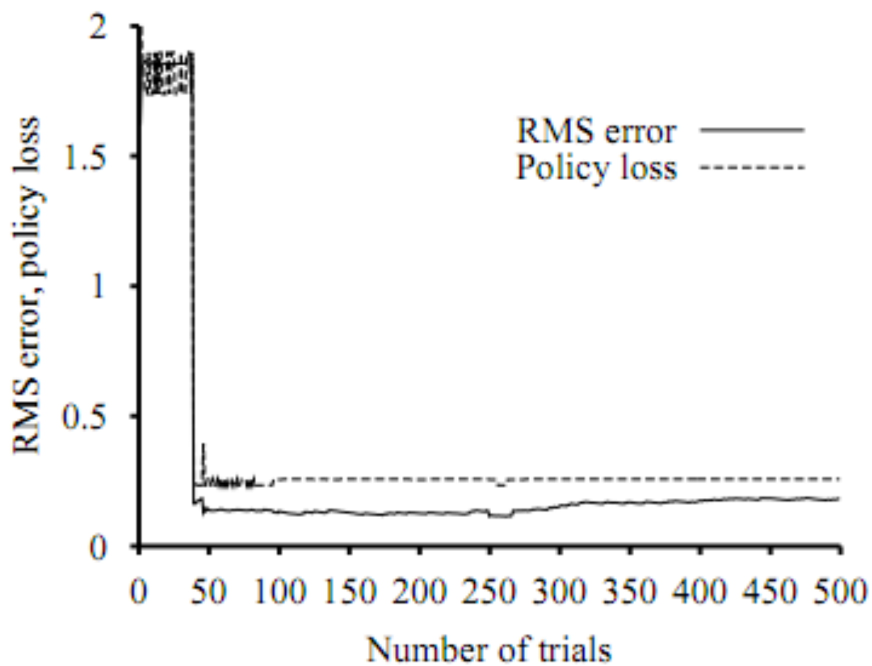
Quiz

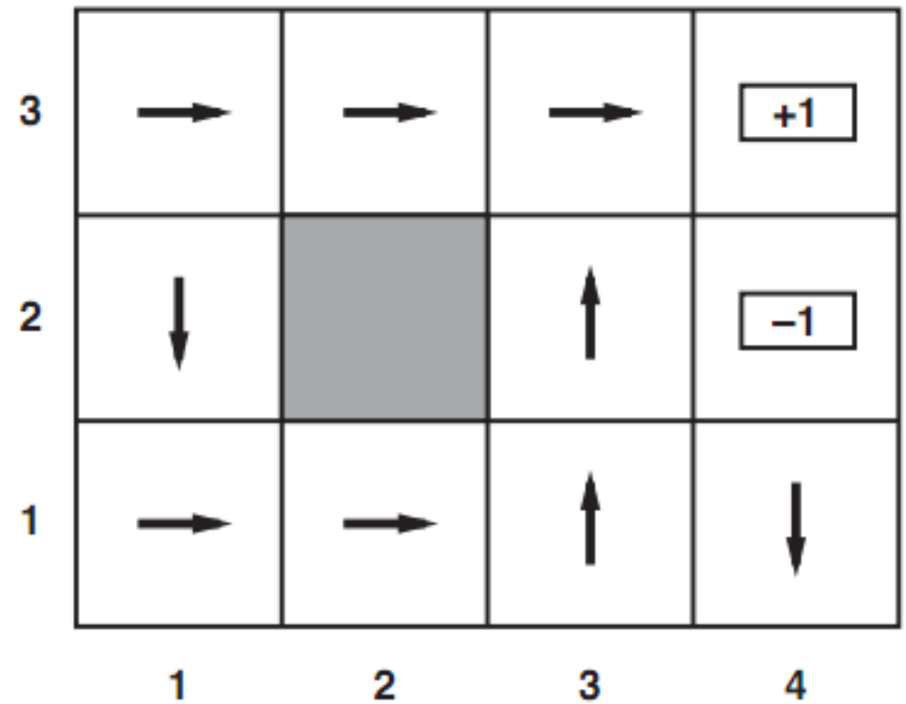
- Is there a fix?
- Can we explore different policies (active reinforcement learning), yet still learn the optimal policy?

Greedy TD-Agent

- Learn U with TD-Learning
- After each change to U, solve MDP to compute new optimal policy $\pi(s)$
- Continue learning with new policy

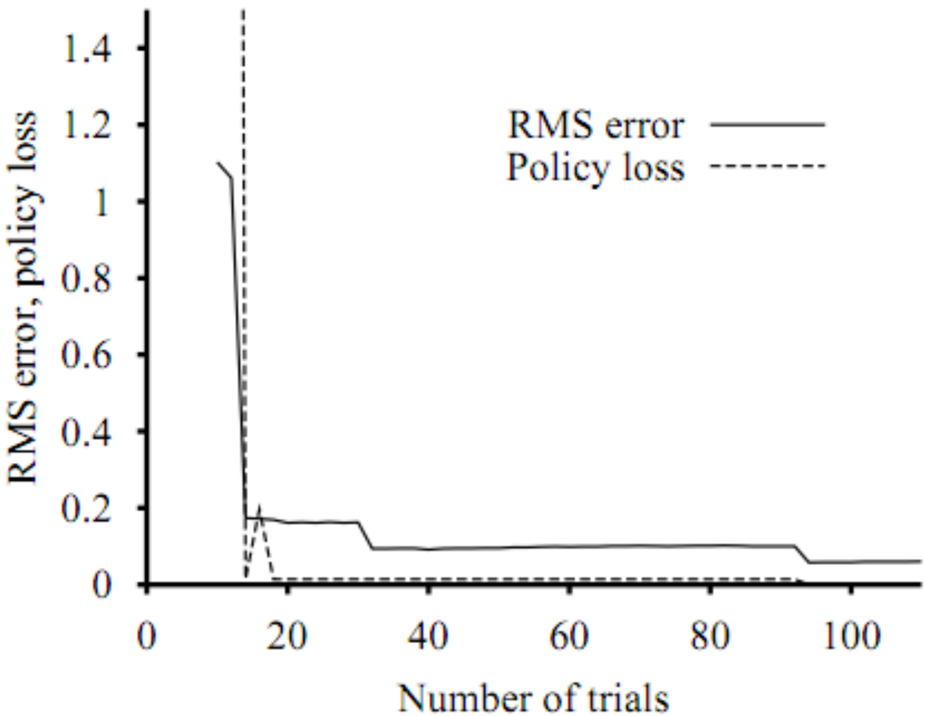
Greedy Agent Results

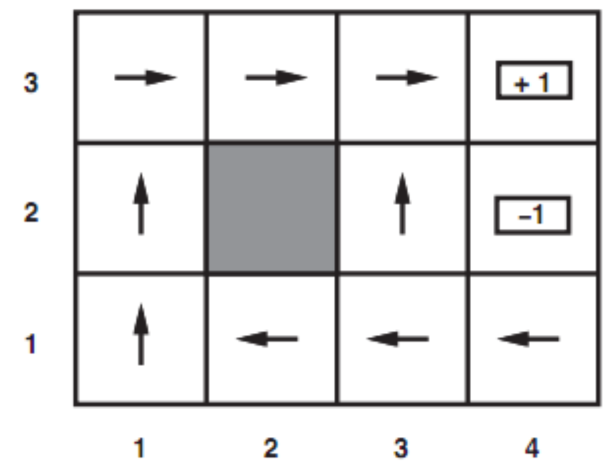




- Wrong policy
  - Too much exploitation, not enough exploration
  - Fix: higher U(s) for unexplored states

Exploratory Agent





- Converges to correct policy

Alternative: Q-Learning

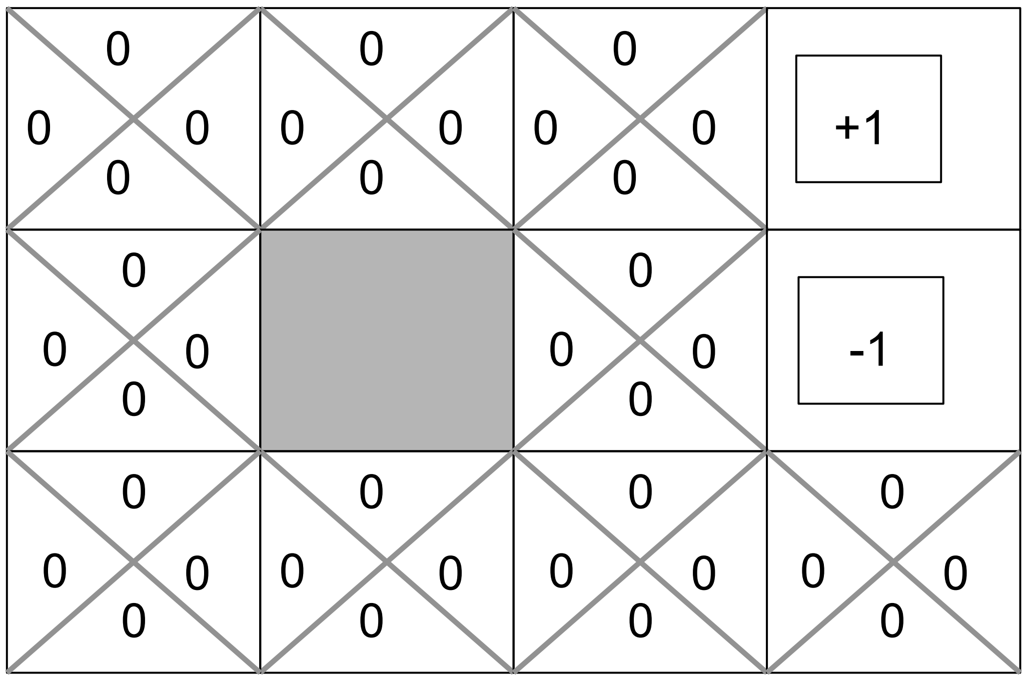
- Instead of learning U(s), learn **Q(a,s)**: the utility of action a in state s
- With utilities, need a model, P, to act:

$$\pi^*(s) = \operatorname*{argmax}_{a \in A(s)} \sum_{s'} P(s' \mid s, a) U(s')$$

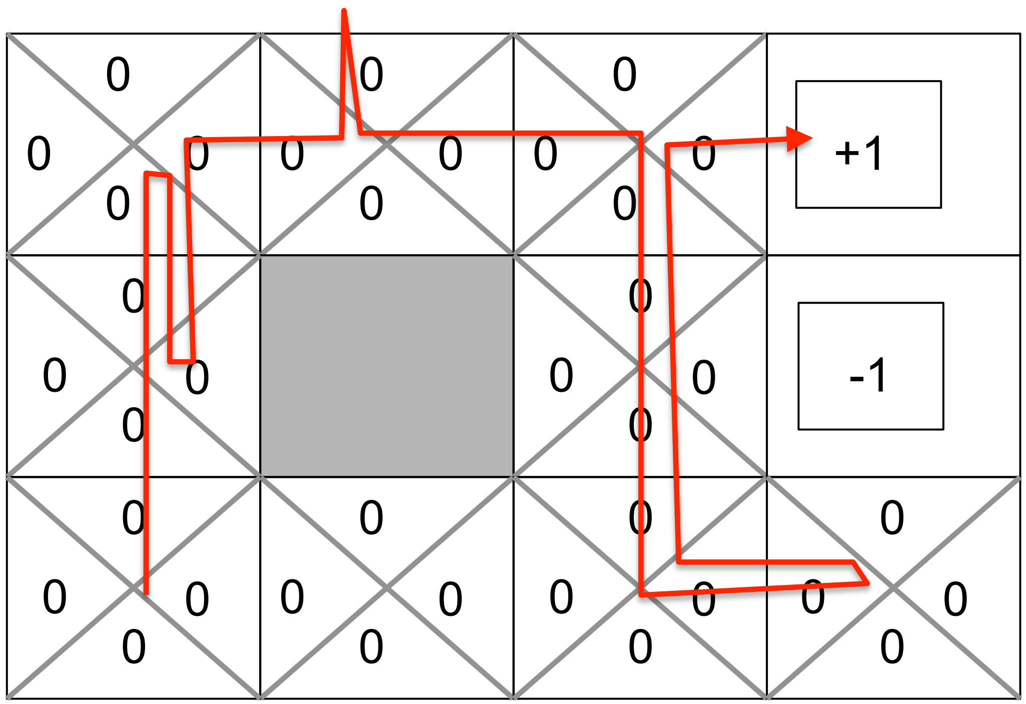
- With Q-values, no model needed
  - Compute Q-values with TD/exploration algorithm

$$Q(s,a) \leftarrow Q(s,a) + \alpha(R(s) + \gamma\, Q(s',a') - Q(s,a))$$

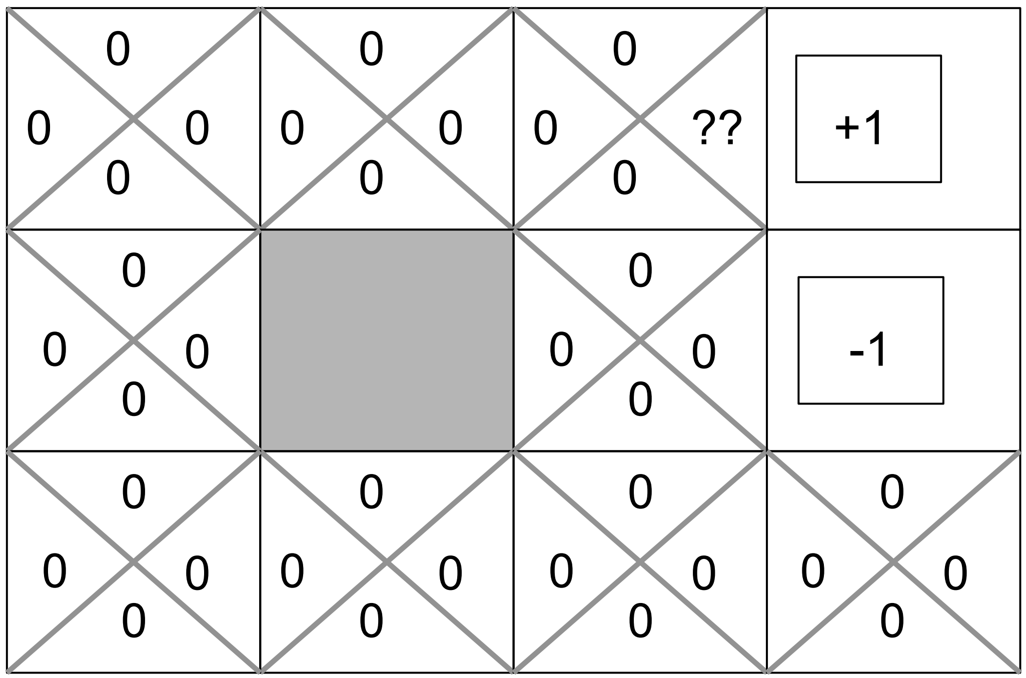
Q-Learning



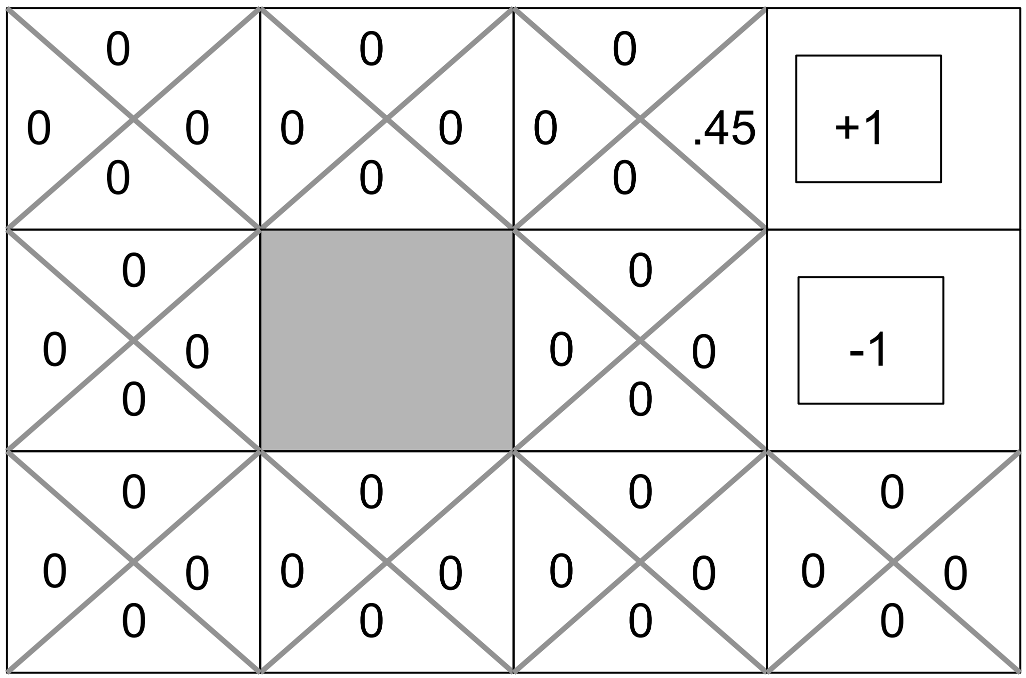
Q-Learning



Q-Learning

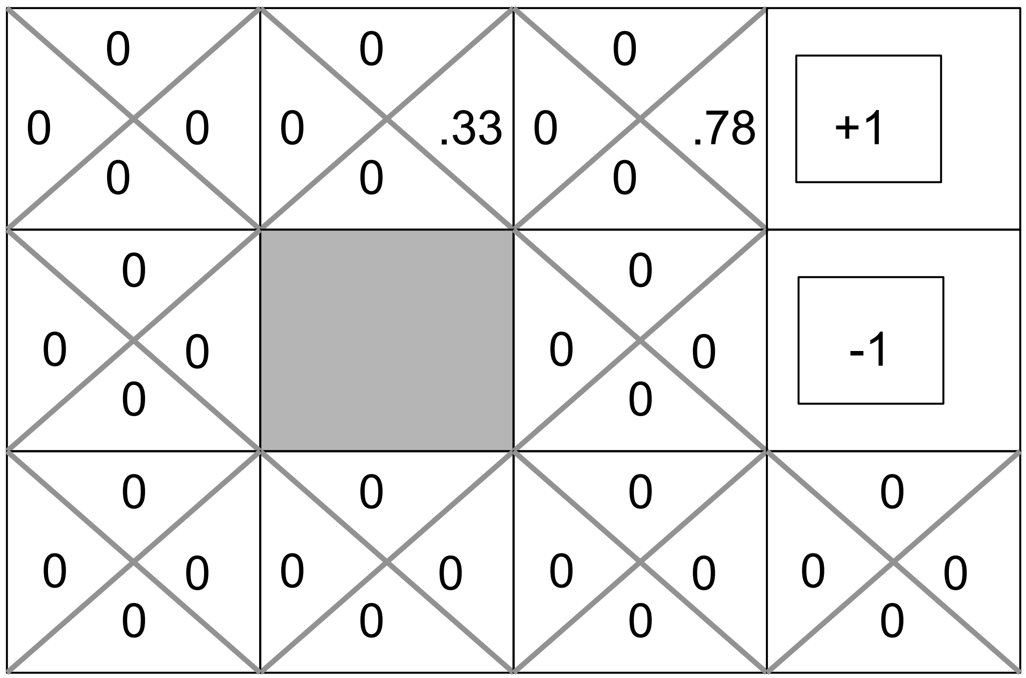


Q-Learning



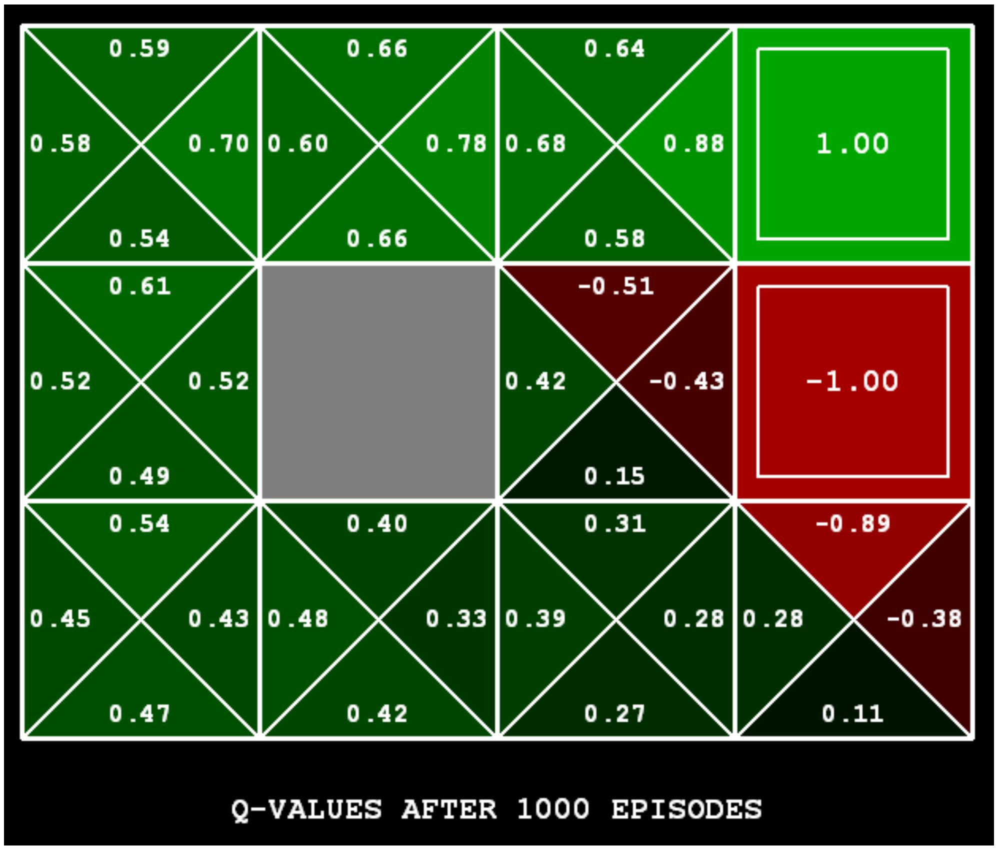
$$Q(s,a) \leftarrow Q(s,a) + \alpha(R(s) + \gamma\, Q(s',a') - Q(s,a))$$

Q-Learning



Q-Learning

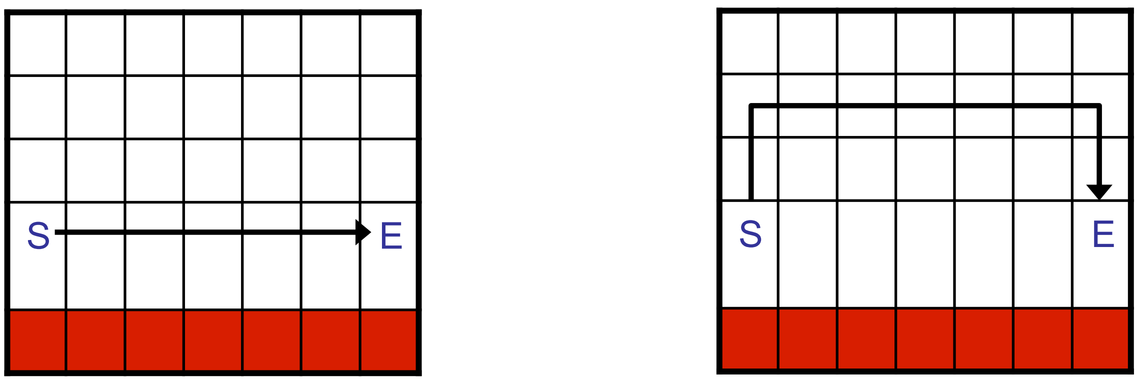
- Q-learning produces tables of q-values:



Q-Learning Properties

- Amazing result: Q-learning converges to optimal policy
  - If you explore enough
  - If you make the learning rate small enough
  - … but not decrease it too quickly!
  - Basically doesn't matter how you select actions (!)

- Neat property: off-policy learning
  - learn optimal policy without following it
    (some caveats, see SARSA)



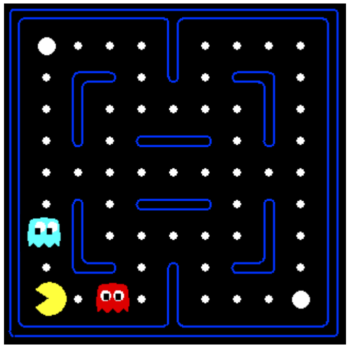
Practical Q-Learning

- In realistic situations, we cannot possibly learn about every single state!
  - Too many states to visit them all in training
  - Too many states to hold the q-tables in memory

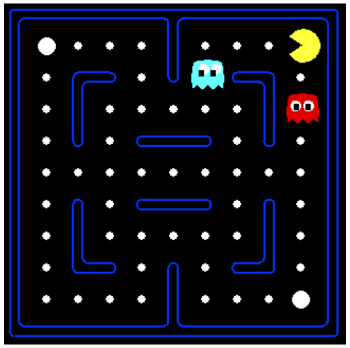
- Instead, we want to generalize:
  - Learn about some small number of training states from experience
  - Generalize that experience to new, similar states
  - This is a fundamental idea in machine learning, and we'll see it over and over again

Generalization Example

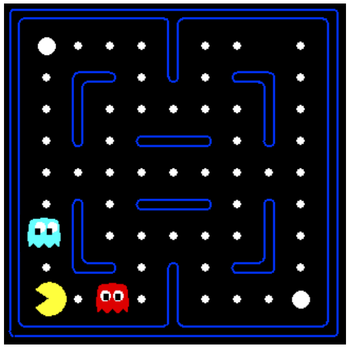
- Let’s say we discover through experience that this state is bad:



- In naïve Q learning, we know nothing about this state or its Q states:

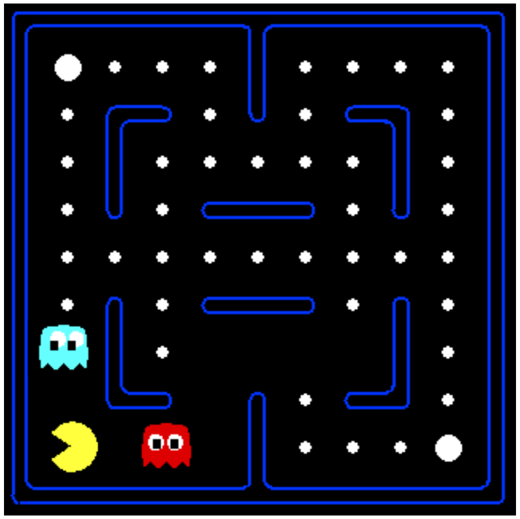


- Or even this one!



Feature-Based Representations

- Solution: describe a state using a vector of features
  - 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.
  - Can also describe a Q-state (s, a) with features (e.g. action moves closer to food)



Linear Feature Functions

- Using a feature representation, we can write a Q function (or Utility function) for any state using a few weights:

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

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

- Advantage: our experience is summed up in a few powerful numbers
- Disadvantage: states may share features but be very different in value!

Function Approximation

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

- Q-learning with linear Q-functions:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \, [error]$$

$$w_i \leftarrow w_i + \alpha \, [error] \, f_i(s,a)$$

- Intuitive interpretation:
  - Adjust weights of active features
  - E.g. if something unexpectedly bad happens, disprefer all states with that state's features (prudence or superstition?)

Summary: MDPs and RL

Things we know how to do:

- If we know the MDP
  - Compute U*, Q*, π* exactly
  - Evaluate a fixed policy π
- If we don't know the MDP
  - We can estimate the MDP then solve
  - We can estimate U for a fixed policy π
  - We can estimate Q*(s,a) for the optimal policy while executing an exploration policy

Techniques:

- Model-based DP
  - Value Iteration
  - Policy iteration
- Model-based RL
- Model-free RL:
  - Value learning
  - Q-learning