CSE 473: Artificial Intelligence

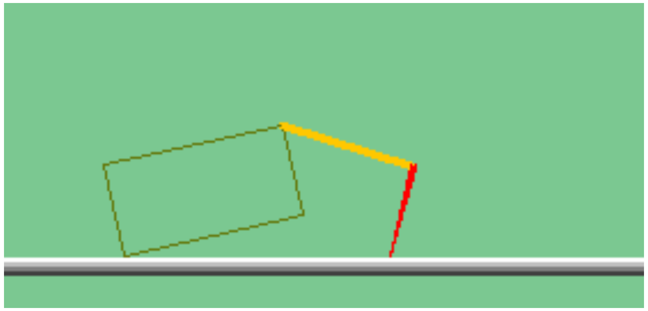
Reinforcement Learning

Hanna Hajishirzi

Many slides over the course adapted from either Luke Zettlemoyer, Pieter Abbeel, Dan Klein, Stuart Russell or Andrew Moore

Reinforcement Learning

- Reinforcement learning:
  - Still have an MDP:
    - A set of states $s \in S$
    - A set of actions (per state) A
    - A model T(s,a,s')
    - A reward function R(s,a,s')
  - Still looking for a policy $\pi(s)$

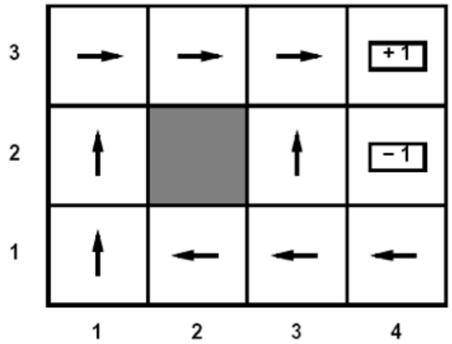


  - New twist: don't know T or R
    - I.e. don't know which states are good or what the actions do
    - Must actually try actions and states out to learn

Key Ideas for Learning

- Online vs. Batch
  - Learn while exploring the world, or learn from fixed batch of data
- Active vs. Passive
  - Does the learner actively choose actions to gather experience? or, is a fixed policy provided?
- Model based vs. Model free
  - Do we estimate T(s,a,s') and R(s,a,s'), or just learn values/policy directly

Passive Learning



- Simplified task
  - You don't know the transitions T(s,a,s')
  - You don't know the rewards R(s,a,s')
  - You are given a policy $\pi(s)$
  - Goal: learn the state values (and maybe the model)
  - I.e., policy evaluation

- In this case:
  - Learner "along for the ride"
  - No choice about what actions to take
  - Just execute the policy and learn from experience
  - We'll get to the active case soon
  - This is NOT offline planning!

Model-Based Learning

- Idea:
  - Learn the model empirically (rather than values)
  - Solve the MDP as if the learned model were correct

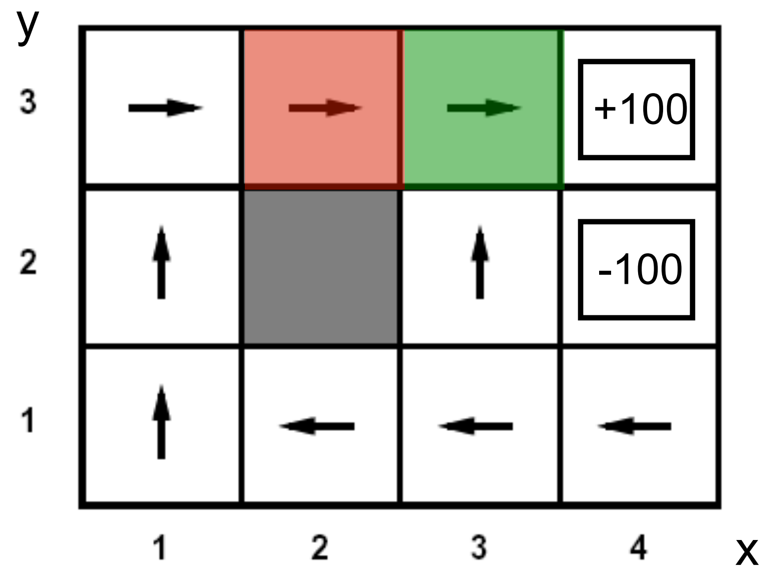
- Empirical model learning
  - Simplest case:
    - Count outcomes for each s,a
    - Normalize to give estimate of T(s,a,s')
    - Discover R(s,a,s') the first time we experience (s,a,s')
  - More complex learners are possible (e.g. if we know that all squares have related action outcomes, e.g. "stationary noise")

Example: Model-Based Learning

- Episodes:

(1,1) up -1
(1,2) up -1
(1,2) up -1
(1,3) right -1
(2,3) right -1
(3,3) right -1
(3,2) up -1
(3,3) right -1
(4,3) exit +100
(done)

(1,1) up -1
(1,2) up -1
(1,3) right -1
(2,3) right -1
(3,3) right -1
(3,2) up -1
(4,2) exit -100
(done)



$\gamma = 1$

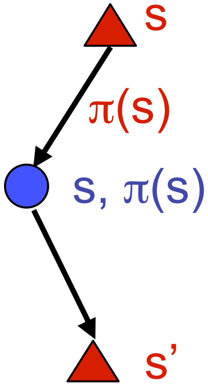
T(<3,3>, right, <4,3>) = 1 / 3

T(<2,3>, right, <3,3>) = 2 / 2

Model-free Learning

$$V^\pi(s) \leftarrow \sum_{s'} T(s,\pi(s),s')[R(s,\pi(s),s') + \gamma V^\pi(s')]$$

- Big idea: why bother learning T?
- Question: how can we compute V if we don't know T?
  - Use direct estimation to sample complete trials, average rewards at end
  - Use sampling to approximate the Bellman updates, compute new values during each learning step

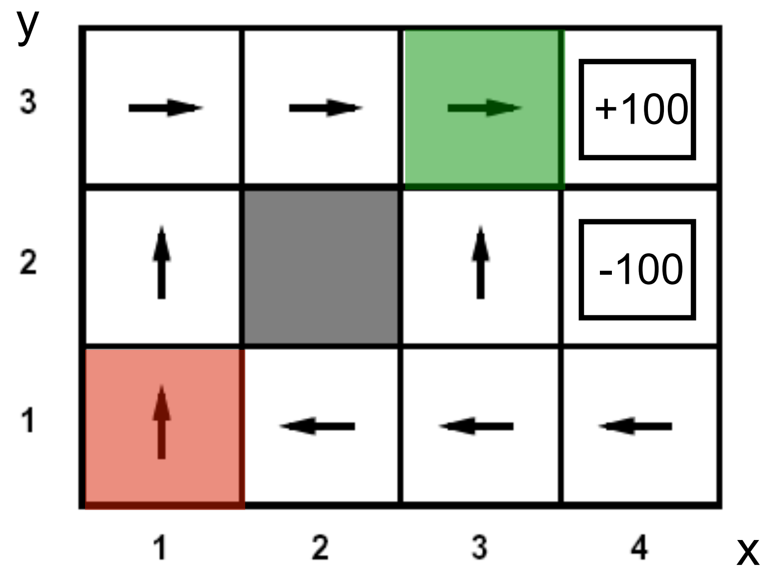


Simple Case: Direct Estimation

- Average the total reward for every trial that visits a state:

(1,1) up -1
(1,2) up -1
(1,2) up -1
(1,3) right -1
(2,3) right -1
(3,3) right -1
(3,2) up -1
(3,3) right -1
(4,3) exit +100
(done)

(1,1) up -1
(1,2) up -1
(1,3) right -1
(2,3) right -1
(3,3) right -1
(3,2) up -1
(4,2) exit -100
(done)



$\gamma = 1$, $R = -1$

V(1,1) ~ (92 + -106) / 2 = -7

V(3,3) ~ (99 + 97 + -102) / 3 = 31.3

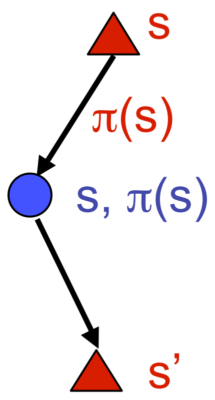
Problems with Direct Evaluation

- What's good about direct evaluation?
  - It is easy to understand
  - It doesn't require any knowledge of T and R
  - It eventually computes the correct average value using just sample transitions
- What's bad about direct evaluation?
  - It wastes information about state connections
  - Each state must be learned separately
  - So, it takes long time to learn

Temporal Difference Learning

$$V^{\pi}(s) \leftarrow \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V^{\pi}(s')]$$

- Big idea: why bother learning T?
  - Update V each time we experience a transition
- Temporal difference learning (TD)
  - Policy still fixed!
  - Move values toward value of whatever successor occurs: running average!



$$sample = R(s, \pi(s), s') + \gamma V^{\pi}(s')$$

$$V^{\pi}(s) \leftarrow (1 - \alpha)V^{\pi}(s) + (\alpha)sample$$

$$V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha(sample - V^{\pi}(s))$$

TD Policy Evaluation

$$V^\pi(s) \leftarrow (1-\alpha)V^\pi(s) + \alpha\left[R(s,\pi(s),s') + \gamma V^\pi(s')\right]$$

(1,1) up -1

(1,2) up -1

(1,2) up -1

(1,3) right -1

(2,3) right -1

(3,3) right -1

(3,2) up -1

(3,3) right -1

(4,3) exit +100

(done)

(1,1) up -1

(1,2) up -1

(1,3) right -1

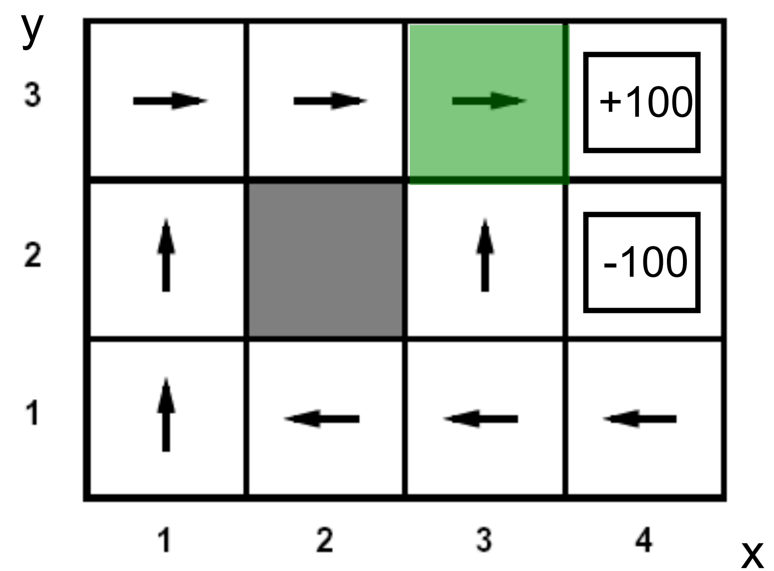
(2,3) right -1

(3,3) right -1

(3,2) up -1

(4,2) exit -100

(done)



Updates for V(<3,3>):

V(<3,3>) = 0.5*0 + 0.5*[-1 + 1*0] = -0.5

V(<3,3>) = 0.5*-0.5 + 0.5*[-1+1*100] = 49.475

V(<3,3>) = 0.5*49.475 + 0.5*[-1 + 1*-0.75]

Take γ = 1, α = 0.5, V_0(<4,3>)=100, V_0(<4,2>)=-100, V_0 = 0 otherwise

Detour: Exp. Moving Average

- Exponential moving average
  - Makes recent samples more important

$$\bar{x}_n = \frac{x_n + (1-\alpha)\cdot x_{n-1} + (1-\alpha)^2 \cdot x_{n-2} + \dots}{1 + (1-\alpha) + (1-\alpha)^2 + \dots}$$

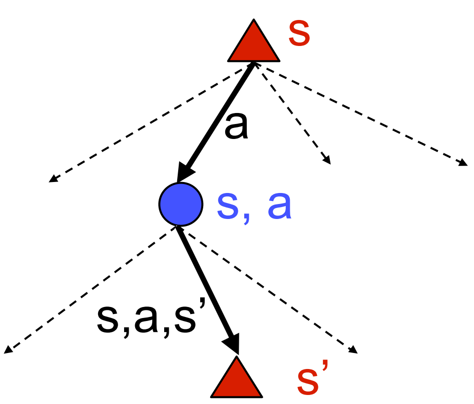
  - Forgets about the past (distant past values were wrong anyway)
  - Easy to compute from the running average

$$\bar{x}_n = (1-\alpha)\cdot \bar{x}_{n-1} + \alpha \cdot x_n$$

- Decreasing learning rate can give converging averages

Problems with TD Value Learning

- TD value leaning is model-free for policy evaluation (passive learning)
- However, if we want to turn our value estimates into a policy, we're sunk:



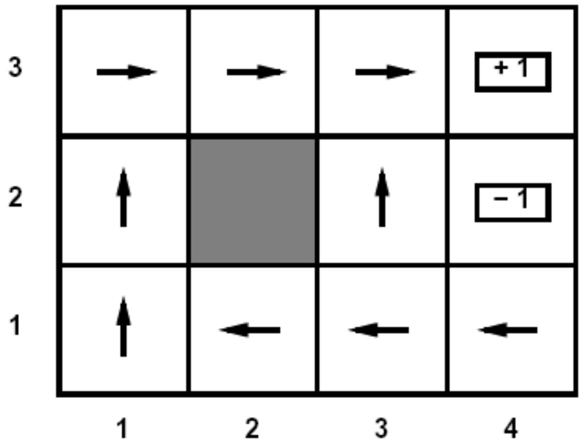
$$\pi(s) = \arg\max_a Q^*(s,a)$$

$$Q^*(s,a) = \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s') \right]$$

- Idea: learn Q-values directly
- Makes action selection model-free too!

Active Learning

- Full reinforcement learning
  - You don’t know the transitions T(s,a,s’)
  - You don’t know the rewards R(s,a,s’)
  - You can choose any actions you like
  - Goal: learn the optimal policy
  - … what value iteration did!



- In this case:
  - Learner makes choices!
  - Fundamental tradeoff: exploration vs. exploitation
  - This is NOT offline planning! You actually take actions in the world and find out what happens…

Q-Learning Update

- Q-Learning: sample-based Q-value iteration

$$Q^*(s,a) = \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \max_{a'} Q^*(s',a') \right]$$

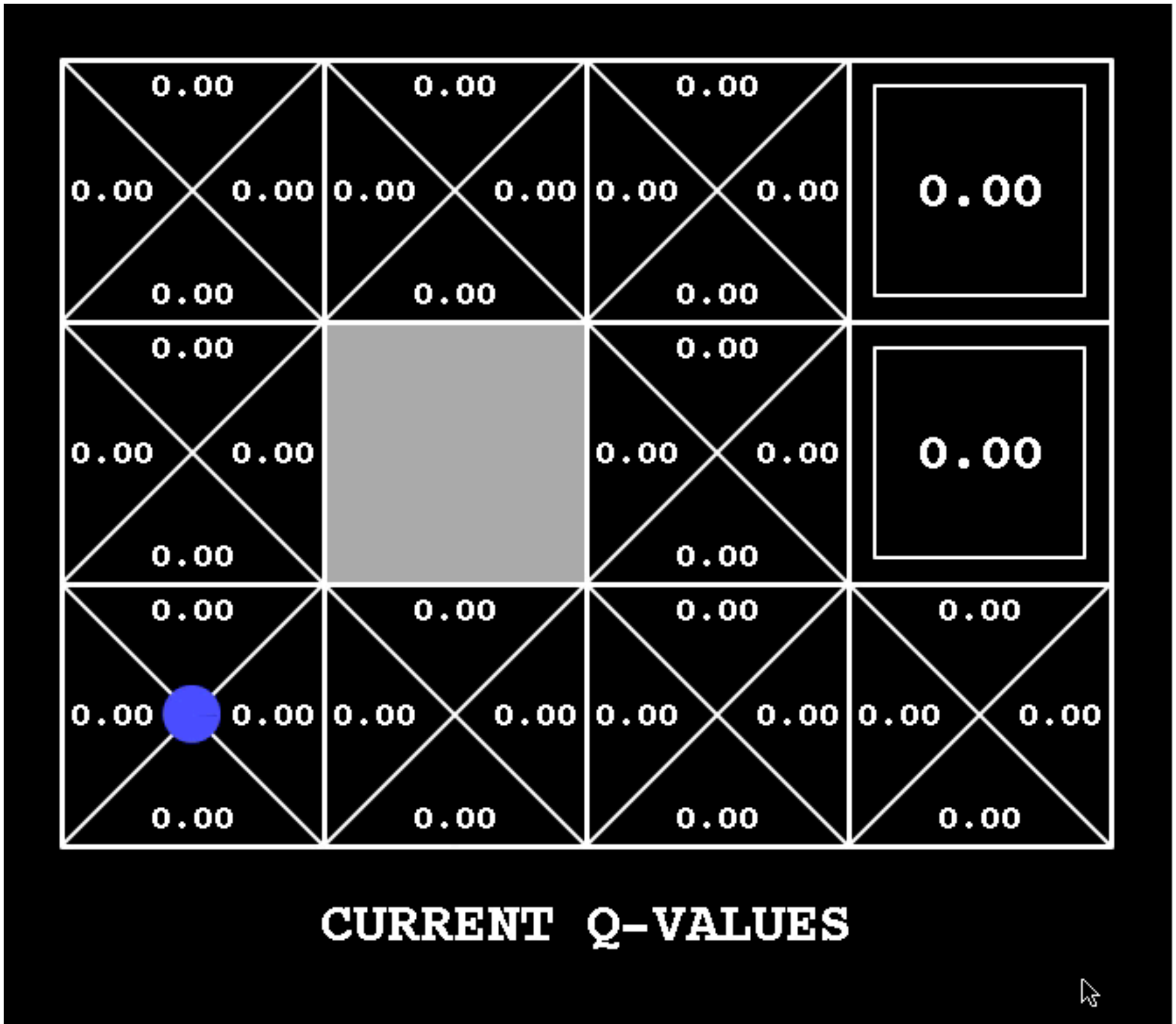
- Learn Q*(s,a) values
  - Receive a sample (s,a,s',r)
  - Consider your old estimate: $Q(s,a)$
  - Consider your new sample estimate:

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

  - Incorporate the new estimate into a running average:

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha)\left[sample\right]$$

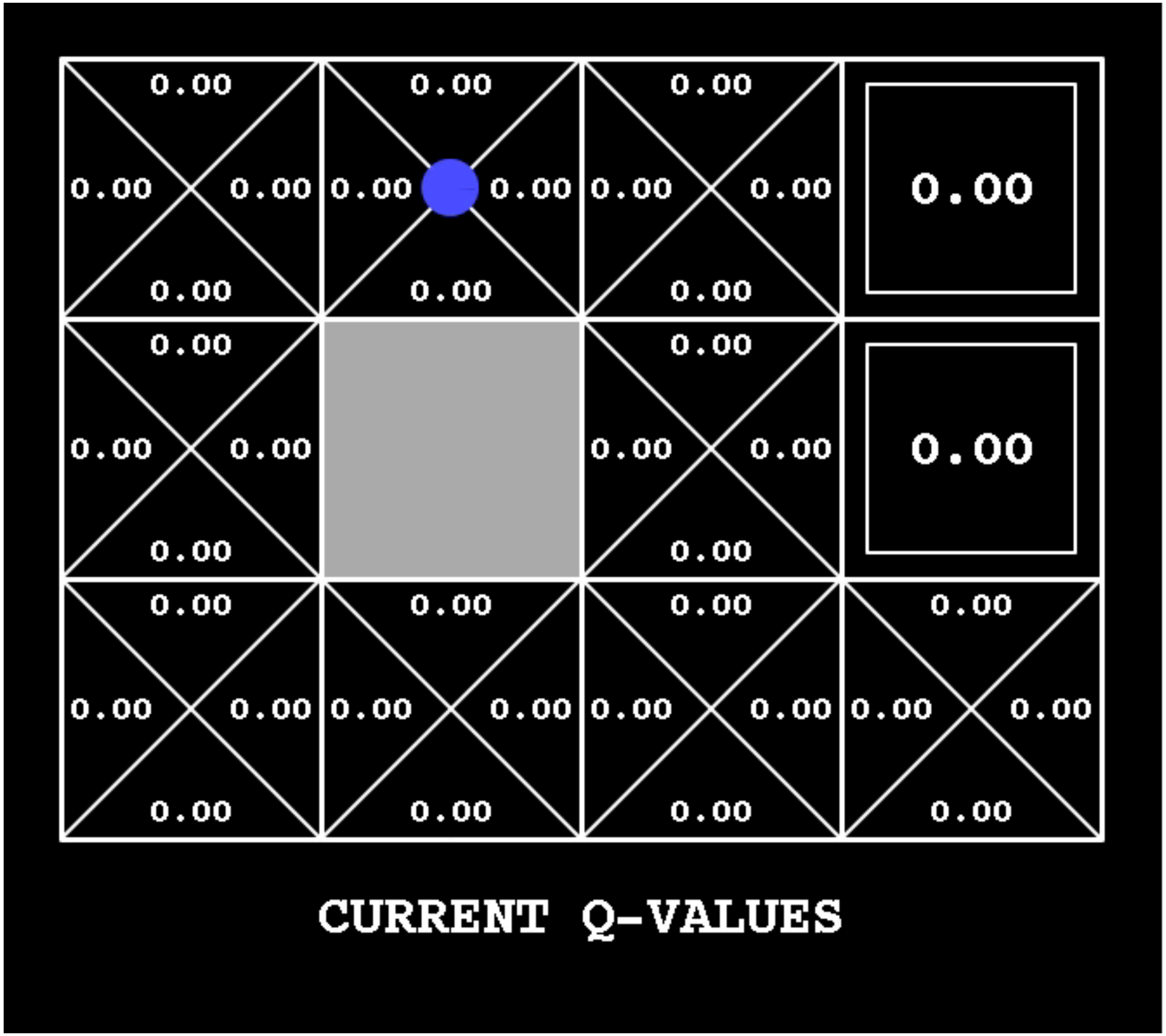
Q-Learning: Fixed Policy



Exploration / Exploitation

- Several schemes for action selection
  - Simplest: random actions (ε greedy)
    - Every time step, flip a coin
    - With probability ε, act randomly
    - With probability 1-ε, act according to current policy
  - Problems with random actions?
    - You do explore the space, but keep thrashing around once learning is done
    - One solution: lower ε over time
    - Another solution: exploration functions

Q-Learning: ε Greedy



Exploration Functions

- When to explore
  - Random actions: explore a fixed amount
  - Better idea: explore areas whose badness is not (yet) established

- Exploration function
  - Takes a value estimate and a count, and returns an optimistic utility, e.g. $f(u, n) = u + k/n$ (exact form not important)
  - Exploration policy $\pi(s')=$

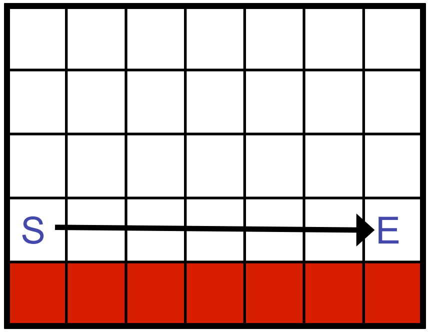
$$\max_{a'} Q_i(s', a') \quad \text{vs.} \quad \max_{a'} f(Q_i(s', a'), N(s', a'))$$

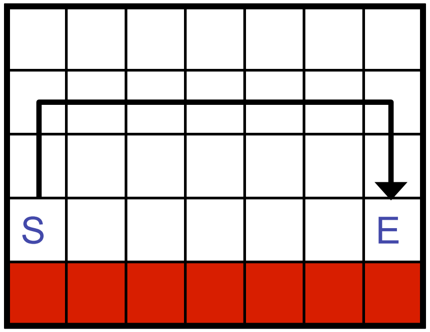
Q-Learning Final Solution

- 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!
  - Not too sensitive to how you select actions (!)
- Neat property: off-policy learning
  - learn optimal policy without following it (some caveats)





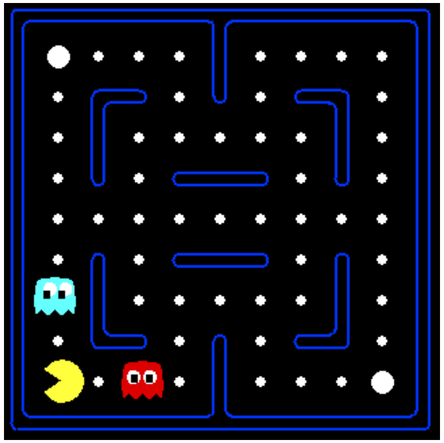
Q-Learning

- 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

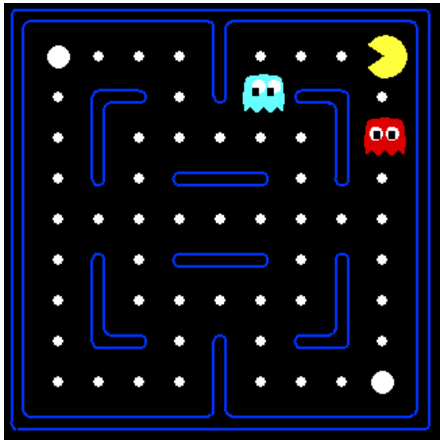
- Instead, we want to generalize:
  - Learn about some small number of training states from experience
  - Generalize that experience to new, similar states
  - This is a fundamental idea in machine learning, and we’ll see it over and over again

Example: Pacman

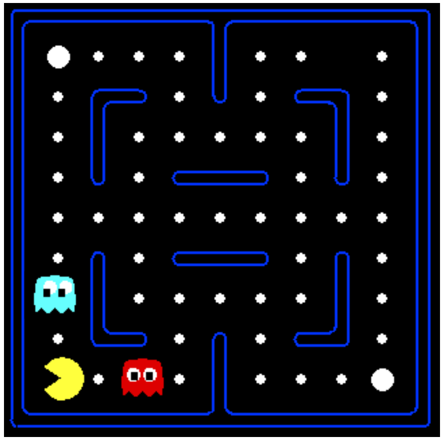
- Let's say we discover through experience that this state is bad:



- In naïve q learning, we know nothing about related states and their q values:

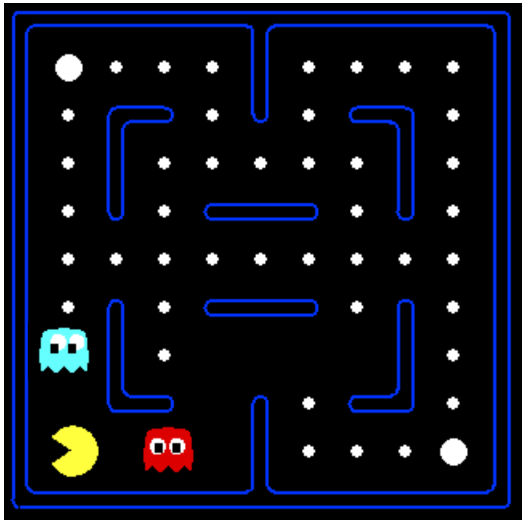


- Or even this third one!



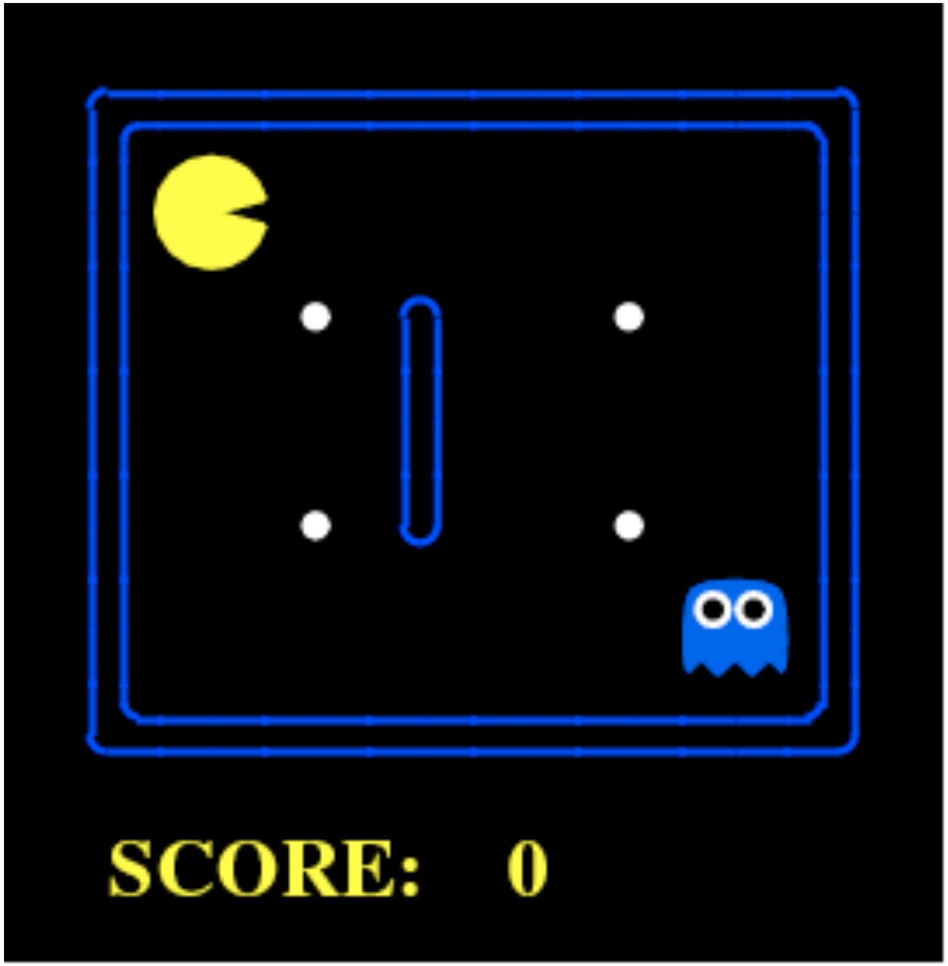
Feature-Based Representations

- 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)



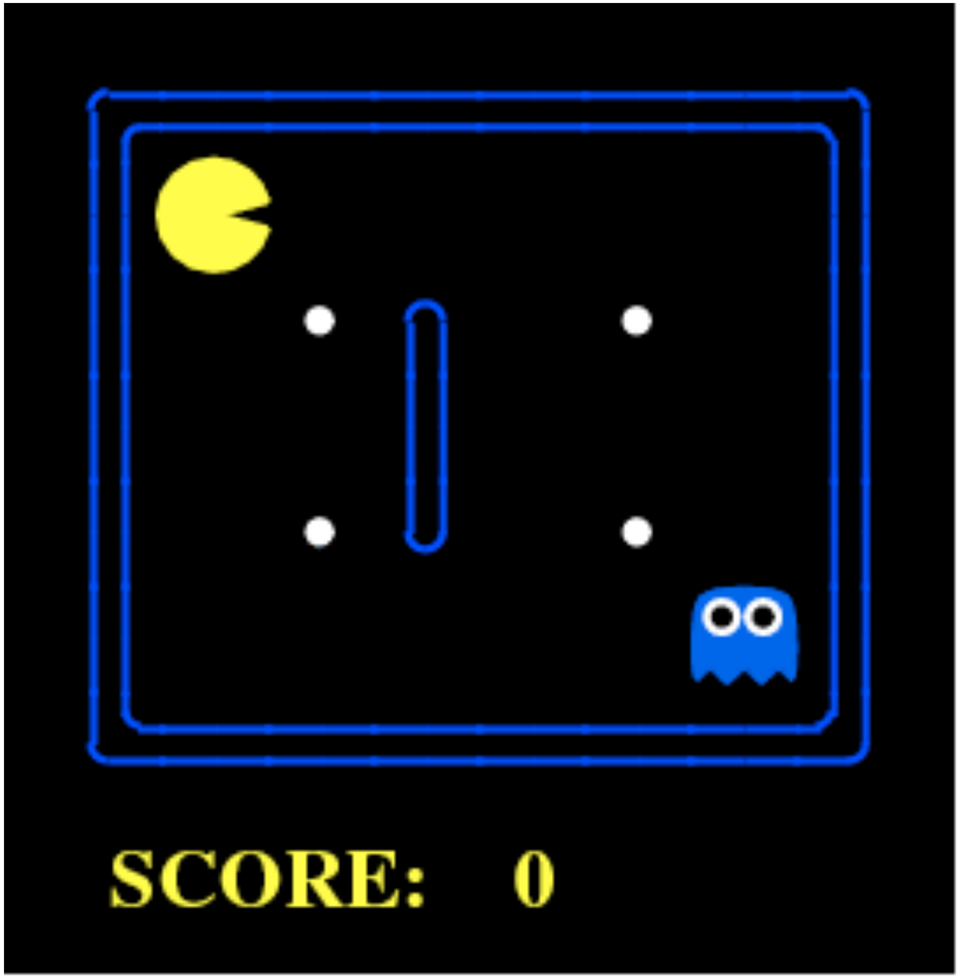
Which Algorithm?

Q-learning, no features, 50 learning trials:



Which Algorithm?

Q-learning, no features, 1000 learning trials:



Linear Feature Functions

- Using a feature representation, we can write a q function (or value 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 actually 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:

$$transition \ = (s,a,r,s')$$

$$\text{difference} = \left[r + \gamma \max_{a'} Q(s',a')\right] - Q(s,a)$$

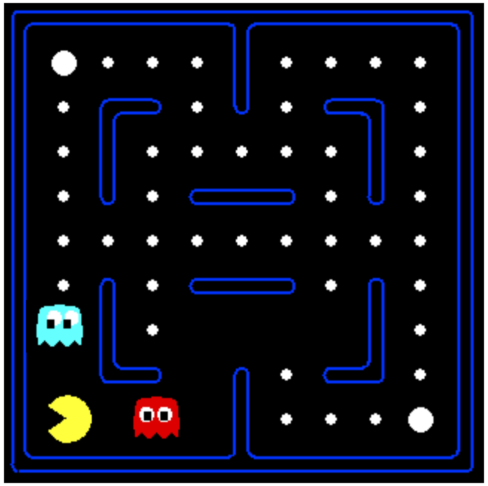
$$Q(s,a) \leftarrow Q(s,a) + \alpha \, [\text{difference}]$$ Exact Q's

$$w_i \leftarrow w_i + \alpha \, [\text{difference}] \, f_i(s,a)$$ Approximate Q's

- Intuitive interpretation:
  - Adjust weights of active features
  - E.g. if something unexpectedly bad happens, disprefer all states with that state's features
- Formal justification: online least squares

Example: Q-Pacman

$$Q(s,a) = 4.0 f_{DOT}(s,a) - 1.0 f_{GST}(s,a)$$



s

$$f_{DOT}(s, \text{NORTH}) = 0.5$$

$$f_{GST}(s, \text{NORTH}) = 1.0$$

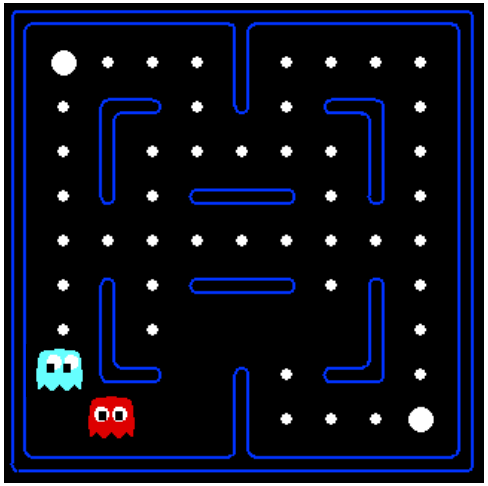
$$Q(s', \cdot) = 0 \quad Q(s,a) = +1$$

$$R(s,a,s') = -500$$

$a = \text{NORTH}$

$r = -500$

$$correction = -501$$



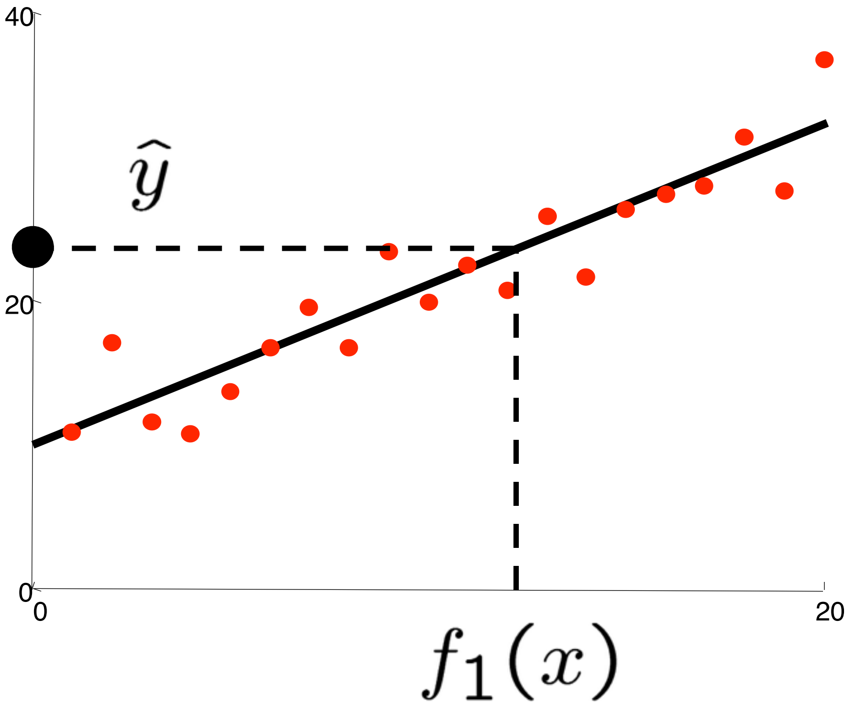
s'

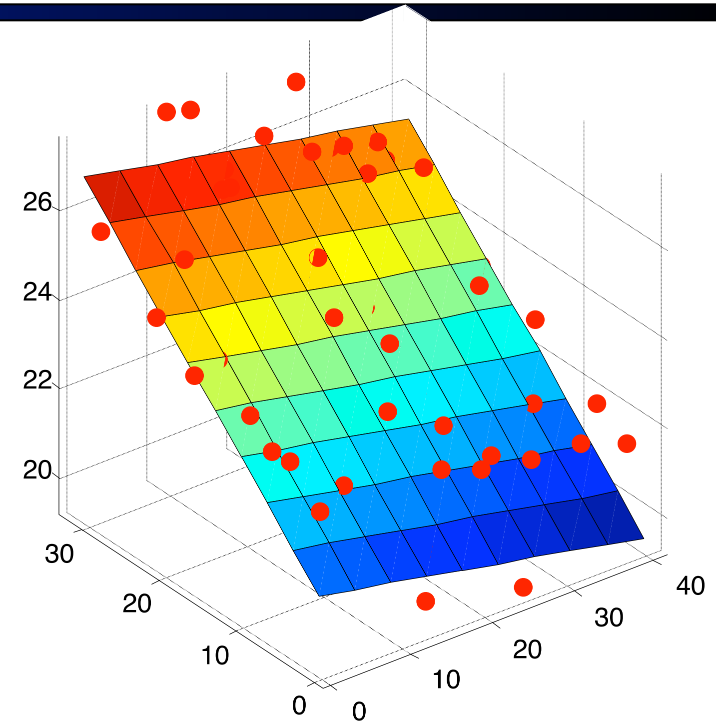
$$w_{DOT} \leftarrow 4.0 + \alpha\,[-501]\,0.5$$

$$w_{GST} \leftarrow -1.0 + \alpha\,[-501]\,1.0$$

$$Q(s,a) = 3.0 f_{DOT}(s,a) - 3.0 f_{GST}(s,a)$$

Linear Regression





Prediction

$$\widehat{y} = w_0 + w_1 f_1(x)$$

Prediction

$$\widehat{y}_i = w_0 + w_1 f_1(x) + w_2 f_2(x)$$