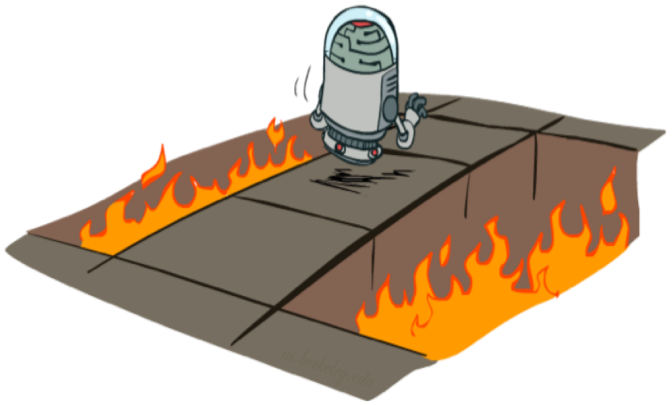
Reinforcement Learning: scaling up

With many slides from Dan Klein and Pieter Abbeel and Stuart Russel

Review: “Active” Reinforcement Learning

- Want: optimal policy
  - You don’t know the transitions T(s,a,s’)
  - You don’t know the rewards R(s,a,s’)
  - You choose the actions now
  - Goal: learn the optimal policy / values



- In this case:
  - Learner makes choices!
  - Fundamental tradeoff: **exploration** vs. **exploitation**
  - This is NOT offline planning! **take actions** in the world and get rewards

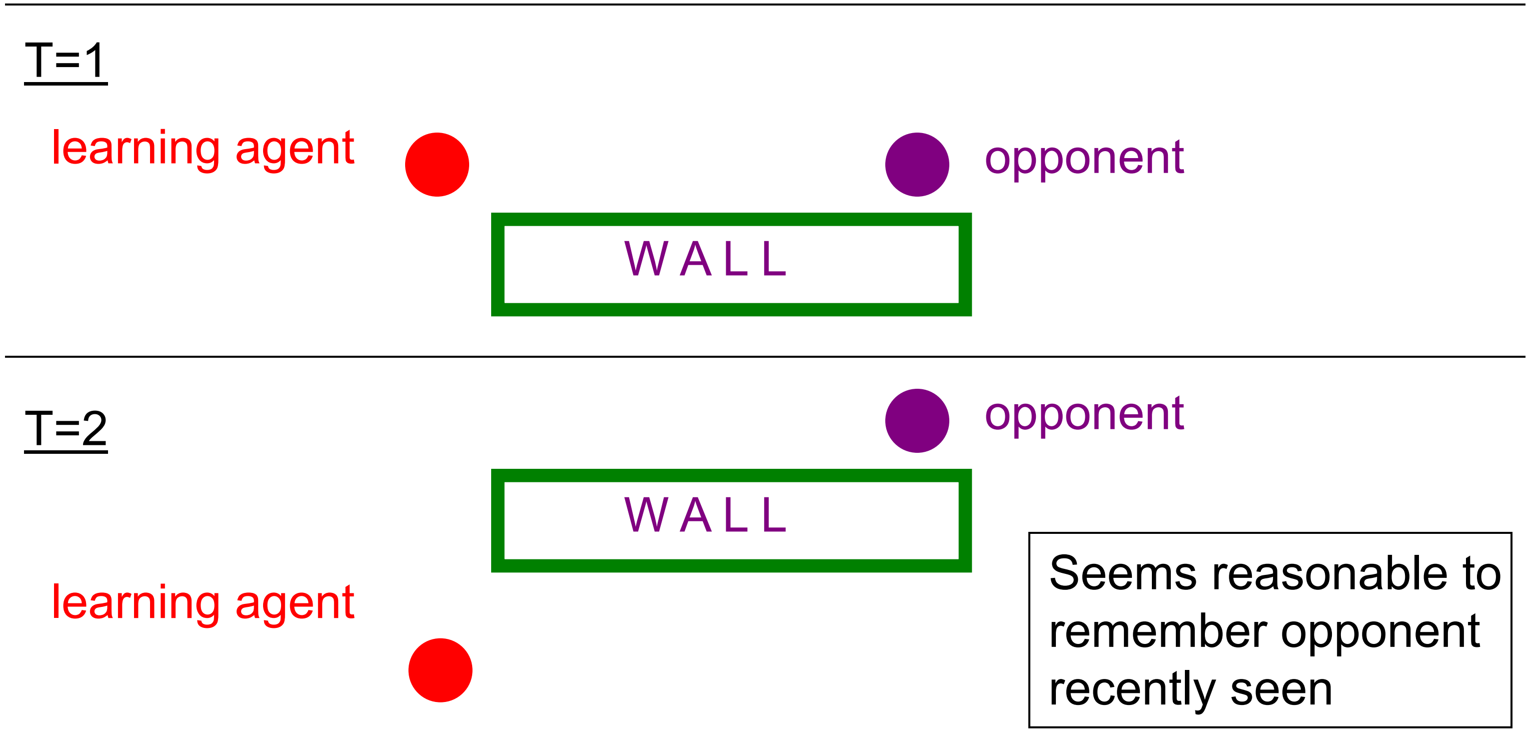
Common Confusion

State need not be solely the current sensor readings

- Markov Assumption
  - Value of state is independent of path taken to reach that state
- Can have memory of the past
  - Can always create Markovian task by remembering entire past history

Need for Memory: Simple Example

"out of sight, but not out of mind"



Seems reasonable to remember opponent recently seen

New Algorithm: **Q-Learning**

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

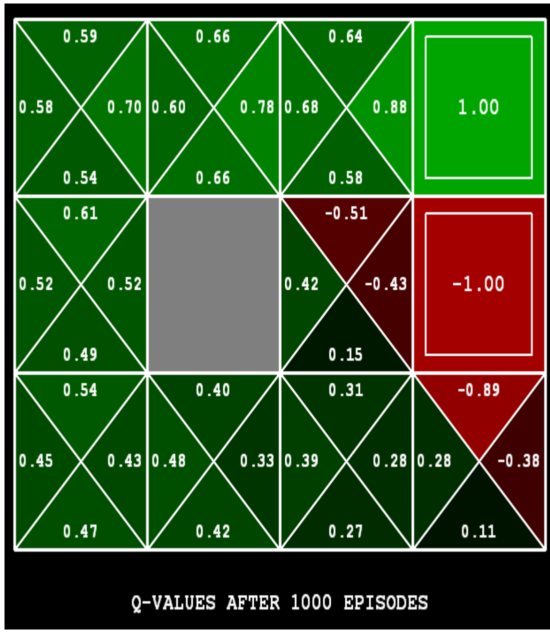
$$Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a')\right]$$

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

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

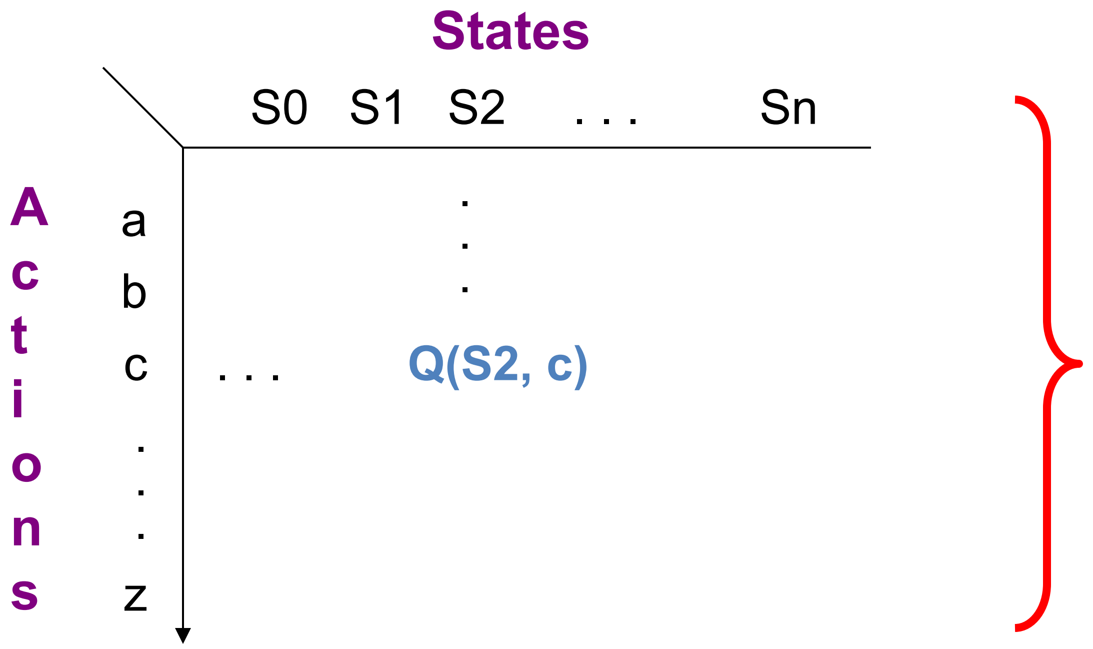
  - Incorporate new estimate in running average:

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



Q-Learning: Implementation Details

Remember, conceptually we are filling in a huge table



Tables are a very verbose representation of a function

Q-Learning: PseudoCode

$$sample = R(s,a,s') + \gamma \max_{a'} Q(s',a')$$
$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha)\,[sample]$$

```
initialize $Q[S, A]$ arbitrarily
observe current state $s$
**repeat forever:**
    select and carry out an action $a$
    observe reward $r$ and state $s'$
    $Q[s,a] \leftarrow Q[s,a] + \alpha\,(r + \gamma \max_{a'} Q[s',a'] - Q[s,a])$
    $s \leftarrow s'$
```

https://github.com/aimacode/aima-python/blob/master/rl.py

Why does Q-Learning Work?

Jude Shavlik, David Page, Wisconsin

- Intuition: Q-Learning performs iterative approximation
- Each round gets closer to “true” q-value
- If states visited infinitely often, will get infinitely close to true value

Q-Learning: Convergence Proof

- Applies to Q tables and deterministic, Markovian worlds. Initialize Q's 0 or random finite.
- **Theorem**: if every state-action pair visited infinitely often, 0≤γ<1, and |rewards| ≤ C (some constant), then

∀s, a

$$\lim_{t->\infty} \hat{Q}_t(s,a) = Q_{actual}(s,a)$$

the approx. Q table ($\hat{Q}$) the true Q table (Q)

Q-Learning Convergence Proof (cont.)

- Consider the max error in the approx. Q-table at step t:

$$\boxed{\Delta_t \equiv \max_{s,a} |\hat{Q}_t(s,a) - Q_{actual}(s,a)|}$$

- The max $Q_{actual}(s,a)$ is finite since |r| ≤ C, so max | Q_{actual} | $\leq \sum_{i=0}^{\infty} \gamma^i C = \frac{C}{1-\gamma}$

- Since. $|\hat{Q}_0|$ finite, we have. Δ_0 finite, i.e. <u>initial max error is finite</u>

Q-Learning Convergence Proof (cont.)

Let s' be the state that results from doing action a in state s. Consider what happens when we visit s and do a at step *t* + 1:

$$\left|\hat{Q}_{t+1}(s,a) - Q(s,a)\right| = \left|\left\{R + \gamma \max_{a'} \hat{Q}_t(s',a')\right\} - \left\{R + \gamma \max_{a''} Q(s',a'')\right\}\right|$$

Current state

Next state

By Q-learning rule (one step)

By def'n of Q (notice best a in s' might be different)

Q-Learning Convergence Proof (cont.)

$= \gamma \mid \max_{a'} \hat{Q}_t(s', a') - \max_{a''} Q(s', a'') \mid$

By algebra

$\leq \gamma \max_{a'''} \mid \hat{Q}_t(s', a''') - Q(s', a''') \mid$

Trickiest step, can prove by contradiction

Since $\left| \max_{a} f_1(a) - \max_{a'} f_2(a') \right| \leq \max_{a} \left| f_1(a) - f_2(a) \right|$

$\leq \gamma \max_{s'', a'''} \mid \hat{Q}_t(s'', a''') - Q(s'', a''') \mid$

Max at s' ≤ max at <u>any</u> s

$= \gamma \Delta_t$ Plugging in defn of Δ_t

Q-Learning Convergence Proof (cont.)

- Hence, every time, after *t*, we visit an <s, a>, its Q value differs from the correct answer by no more than $\gamma \Delta_t$
- Let $T_o=t_o$ (i.e. the start) and T_N be the first time since T_{N-1} where every <s, a> visited at least once
- Call the time between T_{N-1} and T_N, a complete interval

Clearly $\Delta_{T_N} \leq \gamma \Delta_{T_{N-1}}$

Q-Learning Convergence Proof (concluded)

- That is, every complete interval, Δ_t is reduced by at least γ
- Since we assumed every <s, a> pair visited infinitely often, we will have an infinite number of complete intervals

$$\text{Hence, } \lim_{t \to ?} \Delta_t = 0$$

Q-Learning Properties

- Amazing result: Q-learning converges to optimal policy -- even if you're acting **suboptimally**!
- This is called off-policy learning
- Caveats:
  - You have to explore enough
  - You have to eventually make the learning rate small enough
  - … but not decrease it too quickly
  - Basically, in the limit, it doesn't matter how you select actions (!)

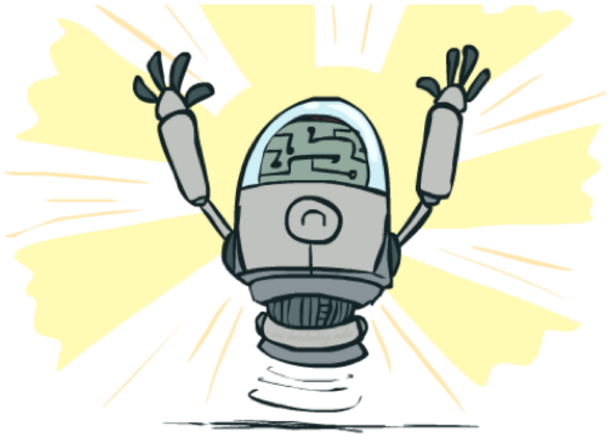
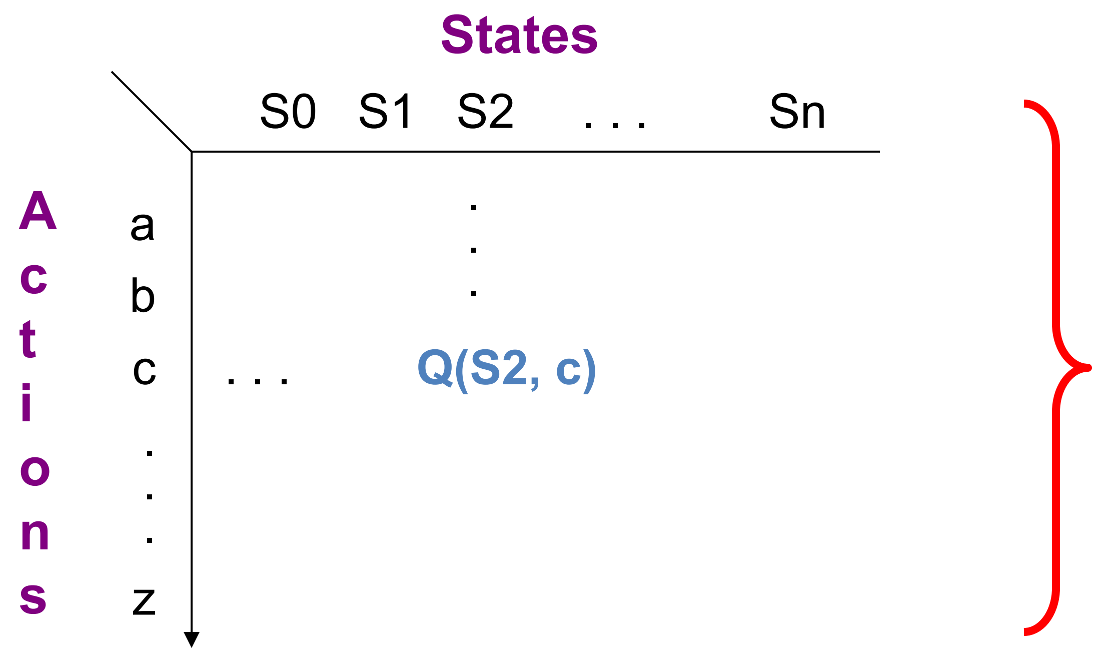


Table-Based (Dictionary) Q-Learning

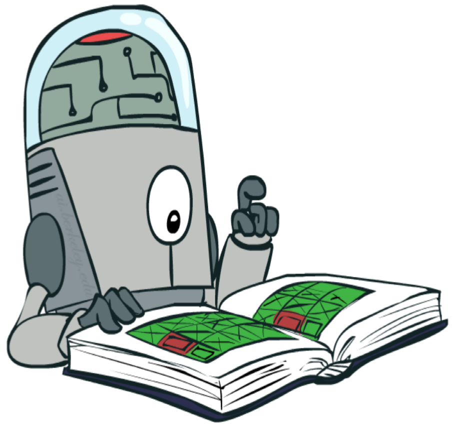
Remember, conceptually we are filling in a <u>huge table</u>

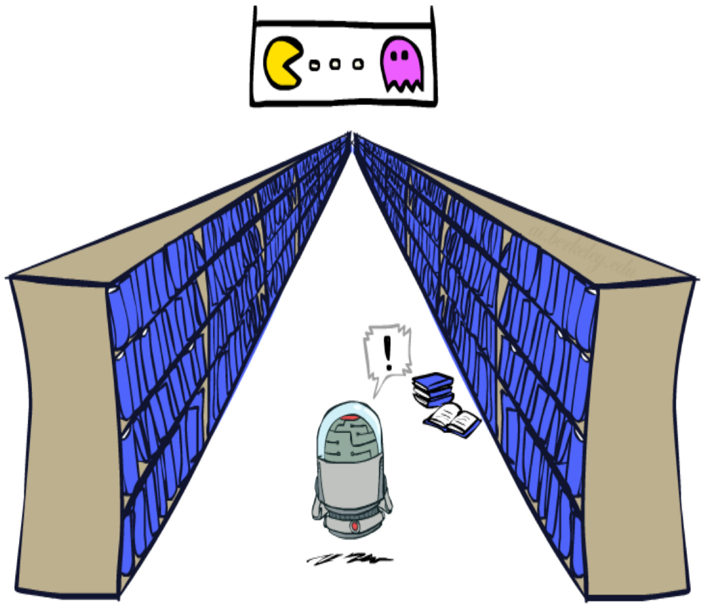


Tables are a very <u>verbose</u> representation of a function

Generalizing Across States

- Basic Q-Learning keeps a **table** of all q-values
- 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 situations
  - This is a fundamental idea in machine learning, and we'll see it over and over again





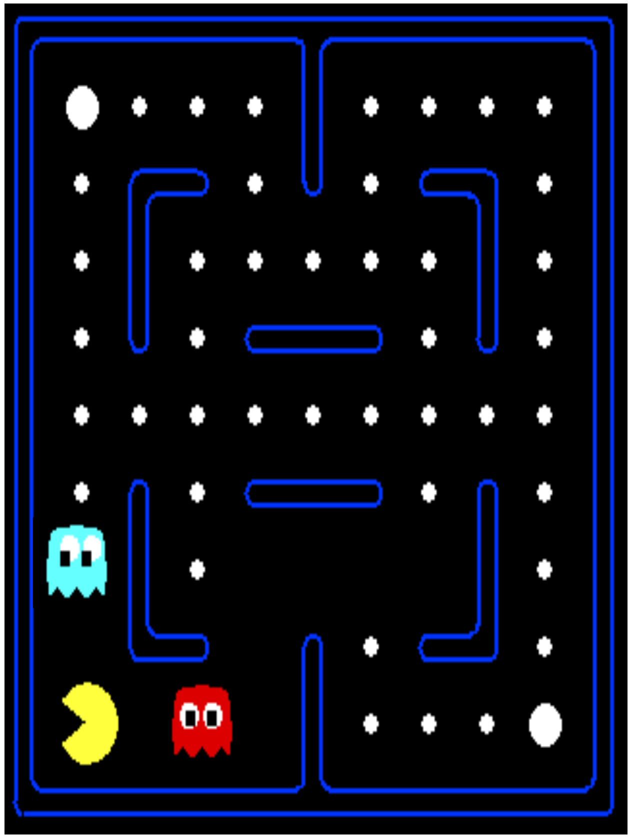
[demo – RL

RL and Function Approximation

- Exact Q-learning infeasible for many real applications due to curse of dimensionality: |S*A| table too big.
- Function Approximation (FA) is a way to "lift the curse:"
  - complexity D of FA needed to capture regularity in environment may be << |S|.
  - no need to sweep thru entire state space: train on N "plausible" samples and then generalize to similar samples drawn from the same distribution.

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:

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

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

- Advantage: our experience is summed up in a few powerful numbers

- Disadvantage: states may share features but actually be very different in value!

Estimating Values 1: Gradient Descent

To find a (local) minimum of a real-valued function $f(x)$:

- assign an arbitrary value to x
- repeat

$$x \leftarrow x - \eta \frac{df}{dx}$$

  where η is the step size

To find a local minimum of real-valued function $f(x_1, \ldots, x_n)$:

- assign arbitrary values to $x_1, \ldots, x_n$
- repeat:

    for each x_i

$$x_i \leftarrow x_i - \eta \frac{\partial f}{\partial x_i}$$

Estimating Values 1: Linear Regression

- A linear function of variables $x_1, \ldots, x_n$ is of the form

$$f^{\overline{w}}(x_1, \ldots, x_n) = w_0 + w_1 x_1 + \cdots + w_n x_n$$

  $\overline{w} = \langle w_0, w_1, \ldots, w_n \rangle$ are weights. (Let $x_0 = 1$).
- Given a set E of examples.
  Example e has input $x_i = e_i$ for each i and observed value, o_e:

$$Error_E(\overline{w}) = \sum_{e \in E} (o_e - f^{\overline{w}}(e_1, \ldots, e_n))^2$$

- Minimizing the error using gradient descent, each example should update w_i using:

$$w_i \leftarrow w_i - \eta \frac{\partial Error_E(\overline{w})}{\partial w_i}$$

Approximate Q-Learning

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

- Q-learning with linear Q-functions:

Error

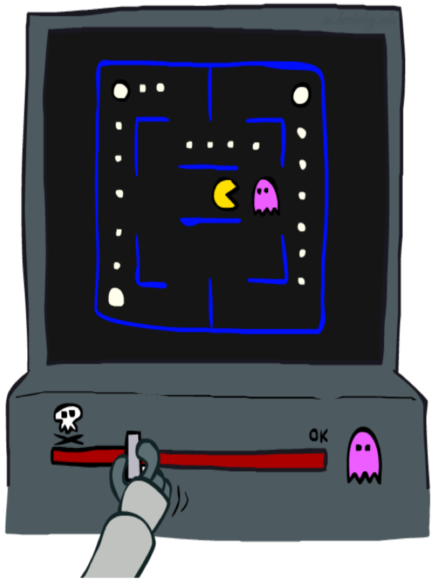
$$\text{transition } = (s, a, r, s')$$

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

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

Gradient for w_i

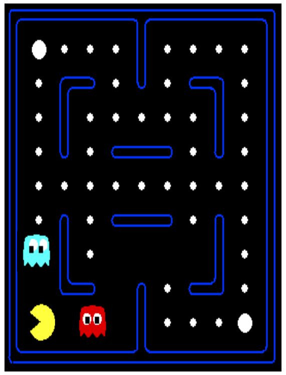
$$w_i \leftarrow w_i + \alpha\,[\text{difference}]\, f_i(s,a)$$ Approximate Q's



- Intuitive interpretation:
  - Adjust weights of active features
  - E.g., if something unexpectedly bad happens, *blame the features that were on*: dislike all states with that state's features

Example: Q-Pacman

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

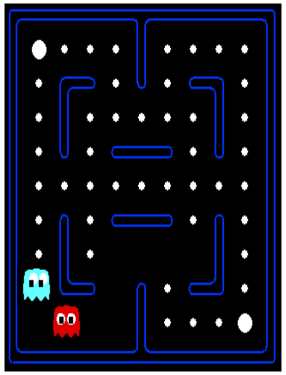
s 

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

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

$a = \text{NORTH}$

$r = -500$

s' 

$Q(s, \text{NORTH}) = +1$

$Q(s', \cdot) = 0$

$$r + \gamma \max_{a'} Q(s', a') = -500 + 0$$

difference $= -501$

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

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

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

Linear Combination of Features (Proj 4)

- Estimate Q(S,a) as weighted sum of features (e.g., for Pacman, can use exactly same features as in Proj 2):

  Q(S,a) = a1*f1 + a2*f2 + …. + ak*fk
  Q(S,b) = b1*f1 + b2*f2 + …. + bk*fk

- Use linear regression to estimate w's:
- For each update of Q(S,a):
  – Update a1…ak s.t. min(MSE)