Approximate Q-Learning

2/24/17

State Value *V(s)* vs. Action Value *Q(s,a)*

$$V = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$$

Either way, value is the sum of future discounted rewards (assuming the agent behaves optimally):

- *V(s)*: after being in state *s*
  - Value Iteration
- *Q(s,a)*: after being in state *s* and taking action *a*
  - Q-Learning

State Value vs. Action Value

- These concepts are closely tied.
- Both algorithms implicitly compute the other value.

Value Iteration update:

$$V(s) \leftarrow R(s) + \gamma \max_a \sum_{s'} P(s' \mid s, a) V(s')$$

Q-Learning update:

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

Converting Between V(s) and Q(s,a)

If you know *V(s)* <u>and the transition probabilities</u>, you can calculate *Q(s,a)* by taking an expected value:

$$Q(s,a) = \sum_{s'} P(s' \mid s,a) V(s')$$

If you know *Q(s,a)*, you can calculate *V(s)* directly by taking a max:

$$V(s) = \max_a Q(s,a)$$

On-Policy Learning (SARSA)

Instead of updating based on the ***<u>best action</u>*** from the next state, update based on the ***<u>action your current policy actually takes</u>*** from the next state.

SARSA update:

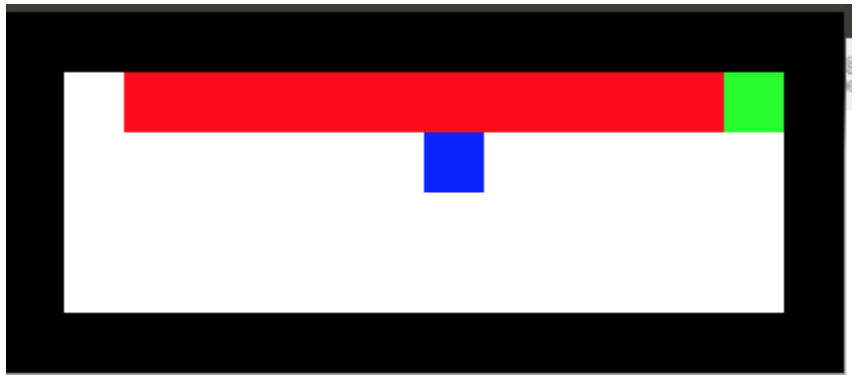
$$Q(s,a) \;=\; \alpha\Big[R(s) + \gamma Q(s',a')\Big] \;+\; (1-\alpha)Q(s,a)$$

$$Q(s,a) \;+=\; \alpha\Big[R(s) + \gamma Q(s',a') - Q(s,a)\Big]$$

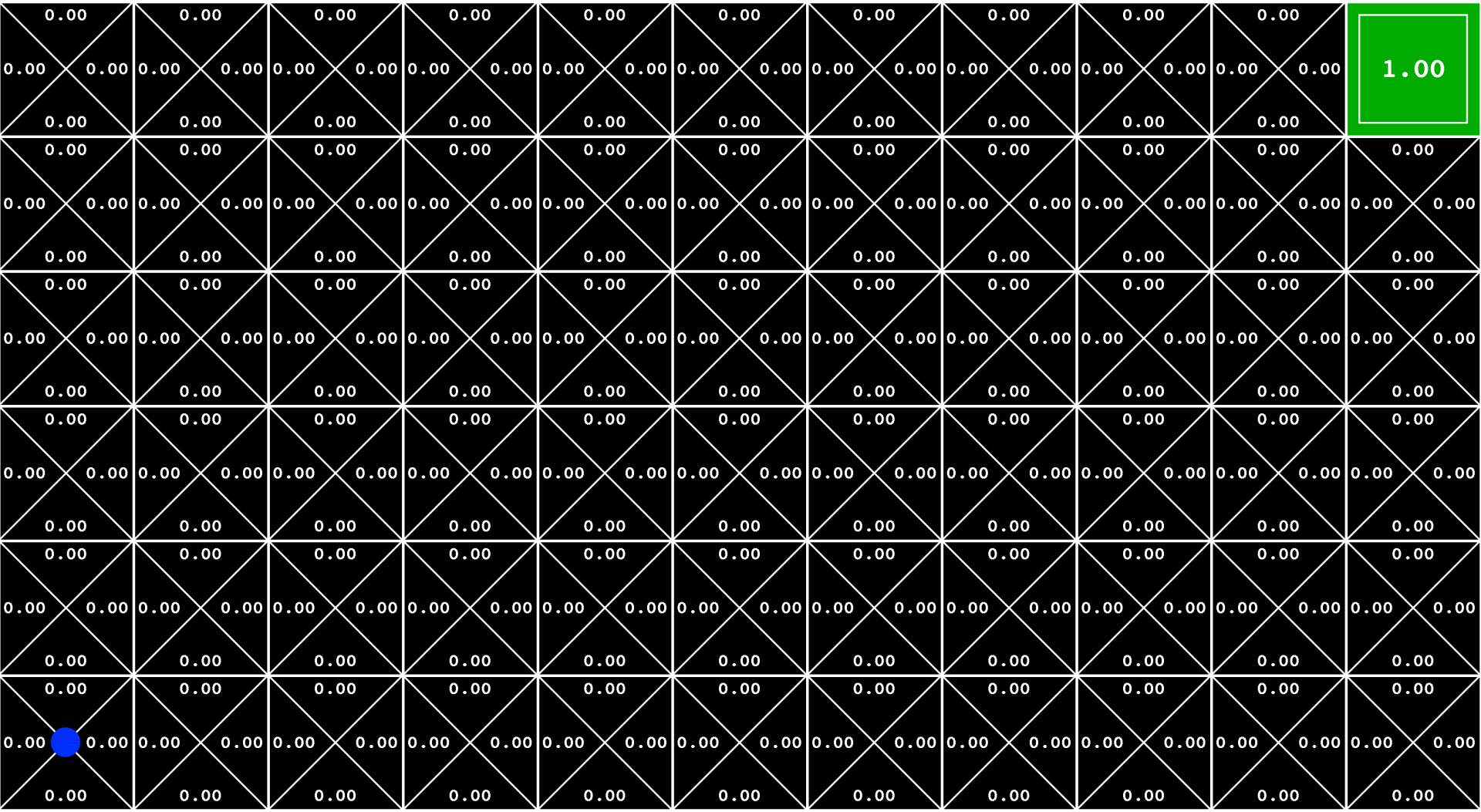
<u>When would this be better or worse than Q-learning</u>?

Demo: Q-learning vs SARSA

https://studywolf.wordpress.com/2013/07/01/reinforcement-learning-sarsa-vs-q-learning/



What will Q-learning do here?



Problem: Large State Spaces

If the state space is large, several problems arise.

- The table of Q-value estimates becomes enormous.
- Q-value updates can be slow to propagate.
- High-reward states can be hard to find.

The state space grows exponentially with the number of relevant features in the environment.

Reward Shaping

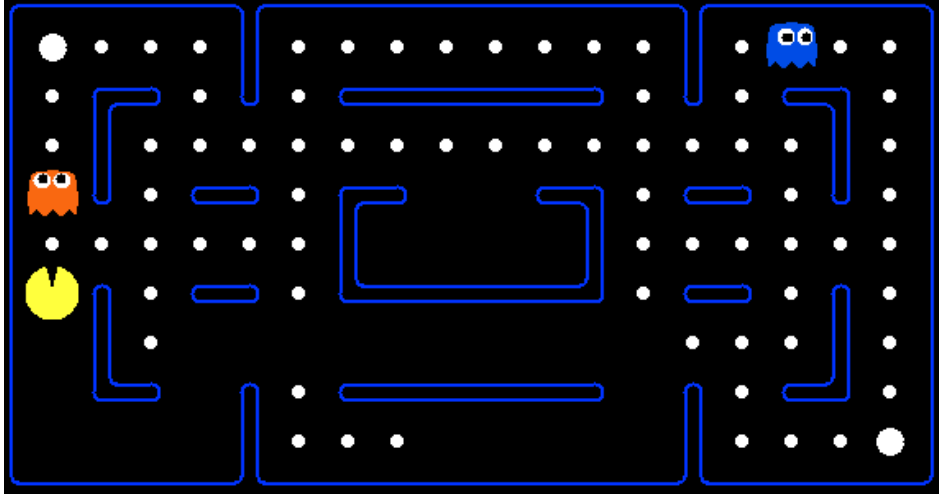
Idea: give some small intermediate rewards that help the agent learn.

- Like a heuristic, this can guide the search in the right direction.
- Rewarding novelty can encourage exploration.

Disadvantages:

- Requires intervention by the designer to add domain-specific knowledge.
- If reward/discount are not balanced right, the agent might prefer accumulating the small rewards to actually solving the problem.
- Doesn’t reduce the size of the Q-table.

PacMan State Space



- PacMan's location
  - ~100 possibilities
- The ghosts' locations
  - ~100^2 possibilities
- Locations still containing food.
  - Enormous number of combinations
- Pills remaining
  - 4 possibilities
- Ghost scared timers
  - ~40^2 possibilities

The state space is the cross product of these feature sets.

- So there are ~100^3*4*40^2*(food configs) states.

Function Approximation

Key Idea: learn a value function as a linear combination of features.

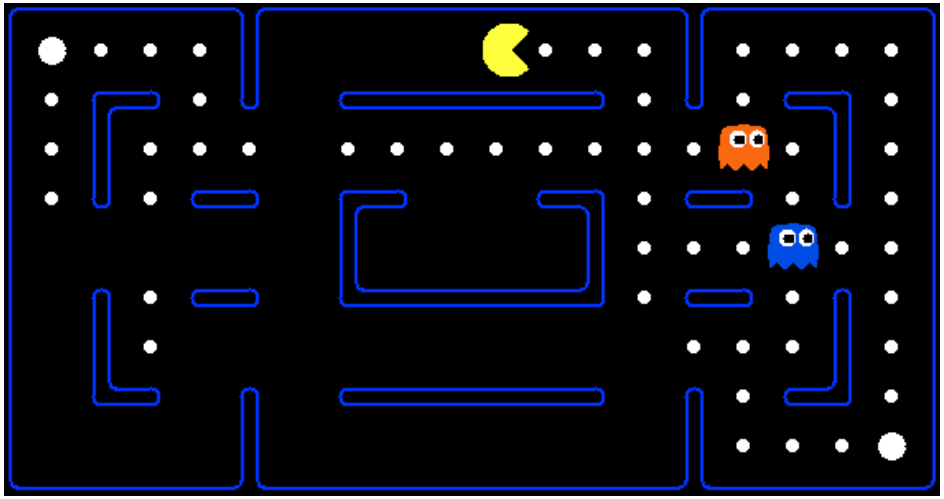
- For each state encountered, determine its representation in terms of features.
- Perform a Q-learning update on each feature.
- Value estimate is a sum over the state's features.

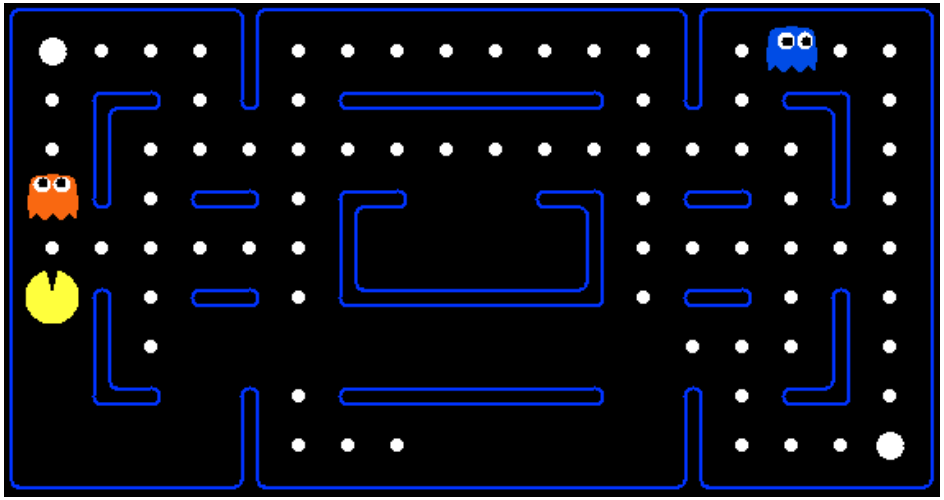
This is our first real foray into machine learning. Many methods we see later are related to this idea.

PacMan Features from Lab

- "bias" always 1.0
- "#-of-ghosts-1-step-away" the number of ghosts (regardless of whether they are safe or dangerous) that are 1 step away from Pac-Man
- "closest-food" the distance in Pac-Man steps to the closest food pellet (does take into account walls that may be in the way)
- "eats-food" either 1 or 0 if Pac-Man will eat a pellet of food by taking the given action in the given state

Extract features from neighbor states:





- Each of these states has two legal actions.

Describe each (s,a) pair in terms of the basic features:

- bias
- #-of-ghosts-1-step-away
- closest-food
- eats-food

Approximate Q-Learning Update

Initialize weight for each feature to 0.

Every time we take an action, perform this update:

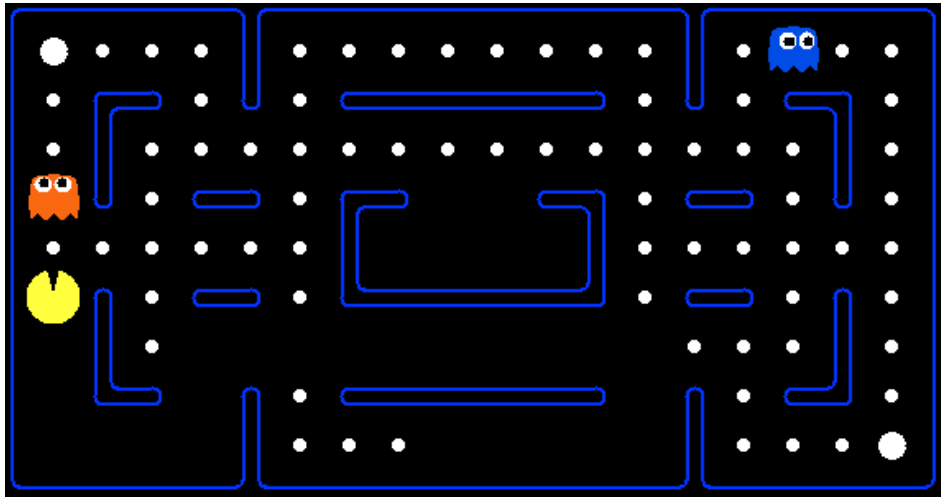
$$\begin{aligned} w_i &\leftarrow w_i + \alpha[correction] f_i(s,a) \\ correction &= (R(s,a) + \gamma V(s')) - Q(s,a) \end{aligned}$$

The Q-value estimate for *(s,a)* is the weighted sum of its features:

$$Q(s,a) = \sum_i^n f_i(s,a) w_i$$

Exercise: Feature Q-Update

discount: .95
learning rate: .3



Reward eating food: +10
Reward for losing: -500

- Suppose PacMan takes the *up* action.
  - The experienced next state is random, because the ghosts' movements are random.
  - Suppose  moves *right* and  moves *down*.

$$
\begin{aligned}
w_i &\leftarrow w_i + \alpha[correction] f_i(s,a) \\
correction &= (R(s,a) + \gamma V(s')) - Q(s,a)
\end{aligned}
$$

Old Feature Values:
$w_{bias} = 1$
$w_{ghosts} = -20$
$w_{food} = 2$
$w_{eats} = 4$

Notes on Approximate Q-Learning

- Learns weights for a tiny number of features.
- Every feature's value is update every step.
- No longer tracking values for individual *(s,a)* pairs.
  - *(s,a)* value estimates are calculated from features.

- The weight update is a form of gradient descent.
  - We've seen this before.

- We're performing a variant of linear regression.
- Feature extraction is a type of basis change.
  - We'll see these again.

Plusses and Minuses of Approximation

- + Dramatically reduces the size of the Q-table.
- + States will share many features.
  - + Allows generalization to unvisited states.
  - + Makes behavior more robust: making similar decisions in similar states.
- + Handles continuous state spaces!

- – Requires feature selection (often must be done by hand).
- – Restricts the accuracy of the learned rewards.
  - – The true reward function may not be linear in the features.