Approximate Q-Learning

3-25-16

Exploration policy vs. optimal policy

Where do the exploration traces come from?

- We need some policy for acting in the environment before we understand it.
- We'd like to get decent rewards while exploring.
  - Explore/exploit tradeoff.

In lab, we're using an epsilon-greedy exploration policy.

After exploration, taking random bad moves doesn't make much sense.

- If Q-value estimates are correct a greedy policy is optimal.

On-policy learning

Instead of updating based on the best action from the next state, update based on the action your current policy actually takes from the next state. SARSA update:

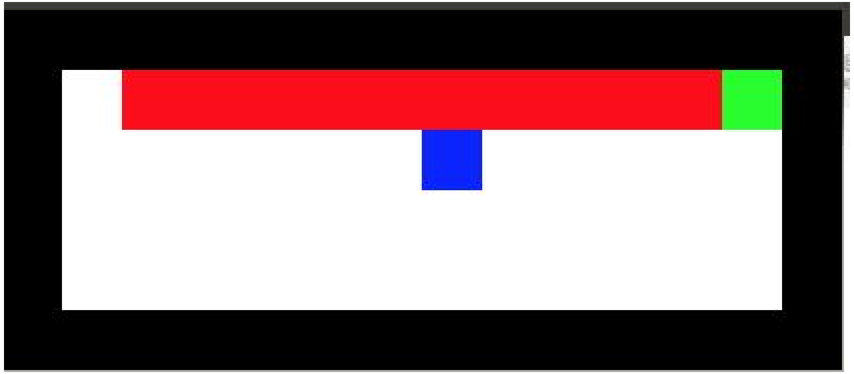
$$Q(s,a) \;=\; \alpha\Big[R(s) + \gamma Q(s',a')\Big] \;+\; (1-\alpha)Q(s,a)$$

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

When would this be better or worse than Q-learning?

Demo: Q-learning vs SARSA

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

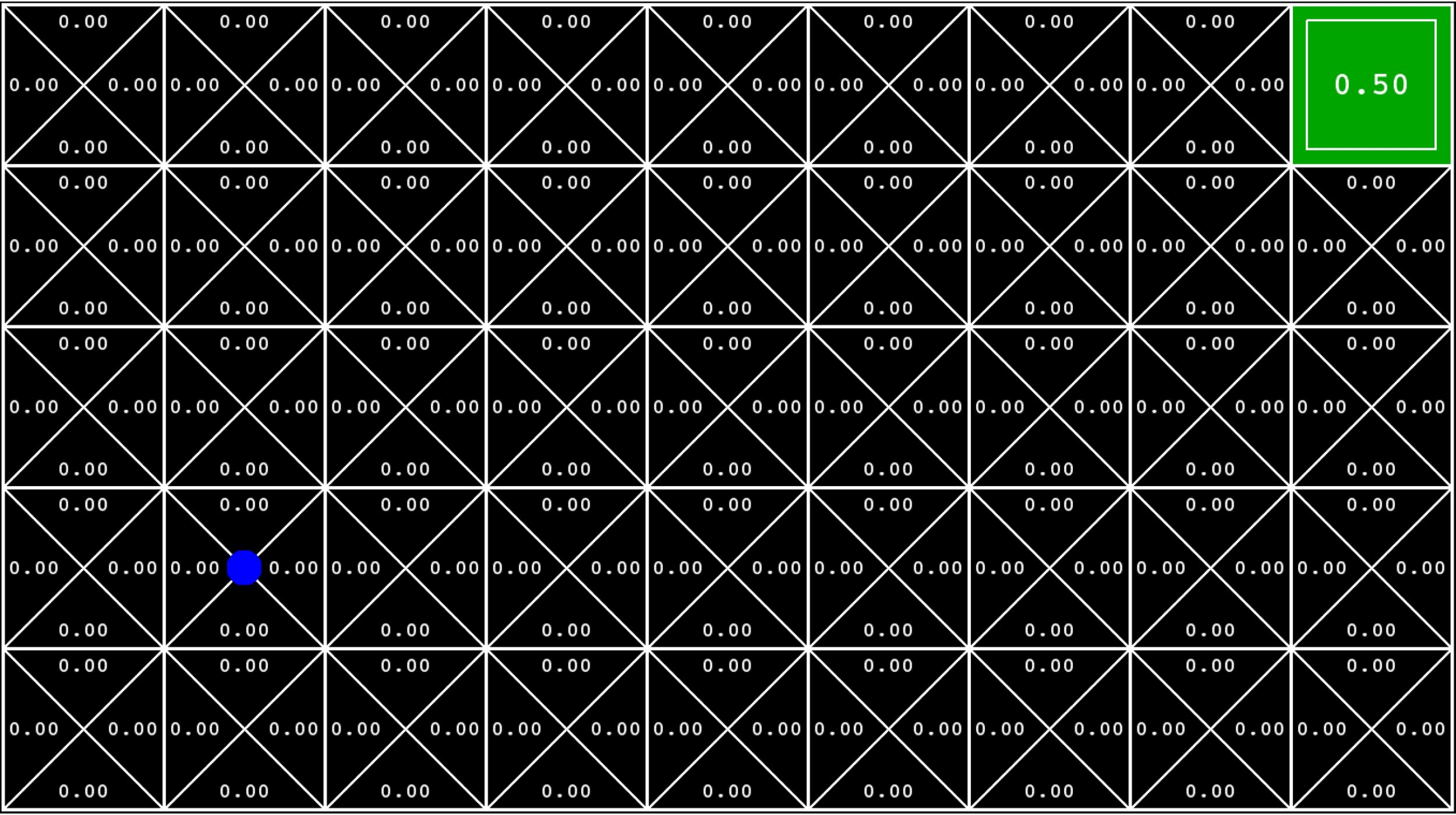


Problem: large state spaces

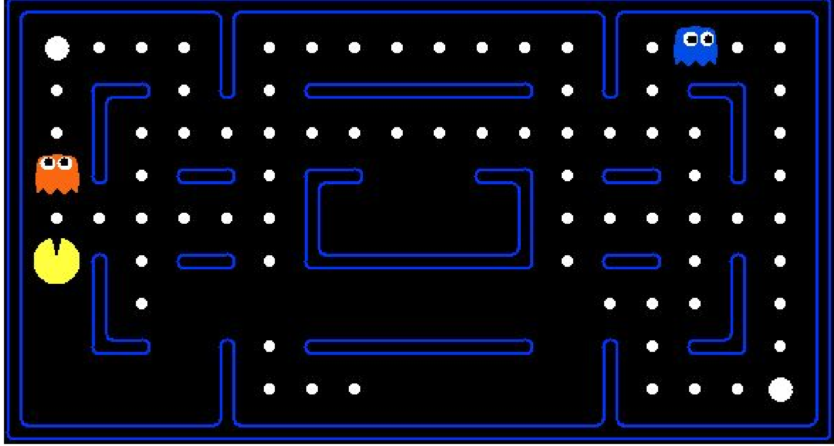
If the state space is large, several problems arise.

- The table of Q-value estimates can get extremely large.
- Q-value updates can be slow to propagate.
- High-reward states can be hard to find.

State space grows exponentially with feature dimension.


0.00 0.00 0.00 0.00 | 0.00 0.00 0.00 0.00 | 0.00 0.00 0.00 0.00 | 0.00 0.00 0.00 0.00 | 0.00 0.00 0.00 0.00 | 0.00 0.00 0.00 0.00 | 0.00 0.00 0.00 0.00 | 0.00 0.00 0.00 0.00 | 0.50
0.00 0.00 0.00 0.00 | 0.00 0.00 0.00 0.00 | 0.00 0.00 0.00 0.00 | 0.00 0.00 0.00 0.00 | 0.00 0.00 0.00 0.00 | 0.00 0.00 0.00 0.00 | 0.00 0.00 0.00 0.00 | 0.00 0.00 0.00 0.00 | 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 | 0.00 0.00 0.00 0.00 | 0.00 0.00 0.00 0.00 | 0.00 0.00 0.00 0.00 | 0.00 0.00 0.00 0.00 | 0.00 0.00 0.00 0.00 | 0.00 0.00 0.00 0.00 | 0.00 0.00 0.00 0.00 | 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 | 0.00 0.00 0.00 0.00 | 0.00 0.00 0.00 0.00 | 0.00 0.00 0.00 0.00 | 0.00 0.00 0.00 0.00 | 0.00 0.00 0.00 0.00 | 0.00 0.00 0.00 0.00 | 0.00 0.00 0.00 0.00 | 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 | 0.00 0.00 0.00 0.00 | 0.00 0.00 0.00 0.00 | 0.00 0.00 0.00 0.00 | 0.00 0.00 0.00 0.00 | 0.00 0.00 0.00 0.00 | 0.00 0.00 0.00 0.00 | 0.00 0.00 0.00 0.00 | 0.00 0.00 0.00 0.00

PacMan state space



- PacMan's location (107 possibilities).
- Location of each ghost (107^2).
- Locations still containing food.
    - 2^{104} combinations.
    - Not all feasible because PacMan can't jump.
- Pills remaining (4 possibilities).
- Whether each ghost is scared (4 possibilities … ignoring the timer).

$107^3 * 4^2 = 19{,}600{,}688$ … ignoring the food!

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.

Function Approximation

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

- We can think of feature extraction as a change of basis.

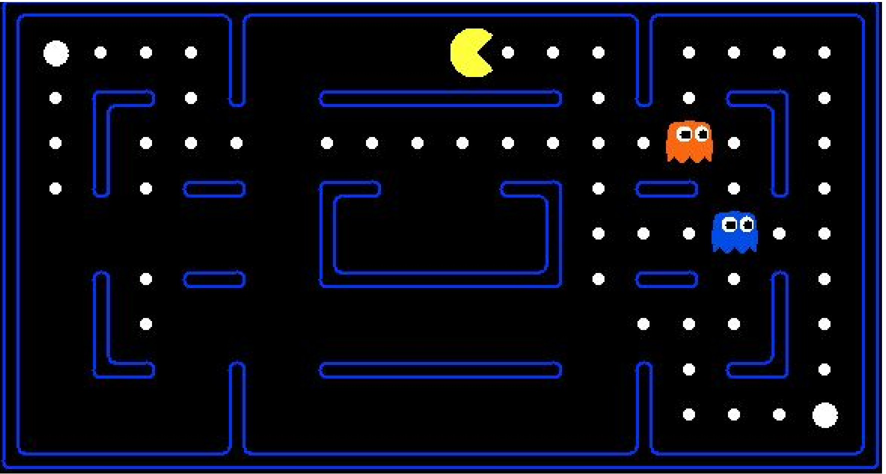
- For each state encountered, determine its representation in terms of features.
- Perform a Q-learning update on each feature.
- Value estimate is a sum over the state's features.

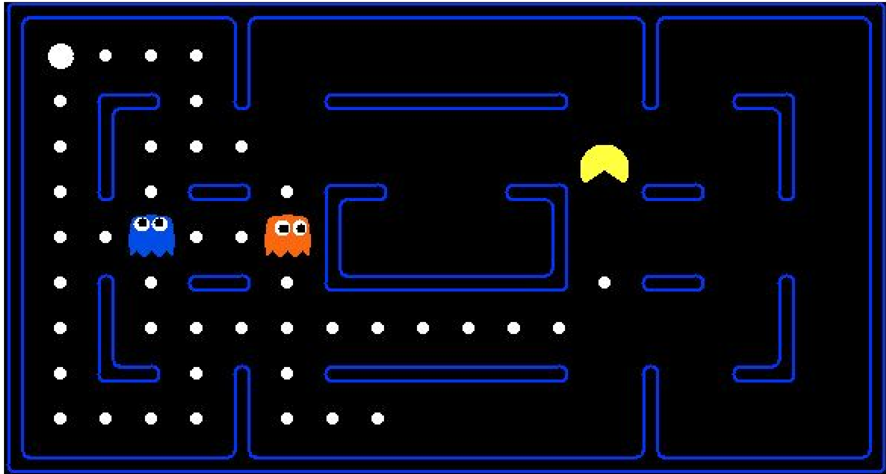
PacMan features from lab

- "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

Exercise: extract features from these states

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





Approximate Q-learning update

Initialize weight for each feature to 0.

$$Q(s,a) = \sum_{i}^{n} f_i(s,a) w_i$$

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

Note: this is performing gradient descent; derivation in the reading.

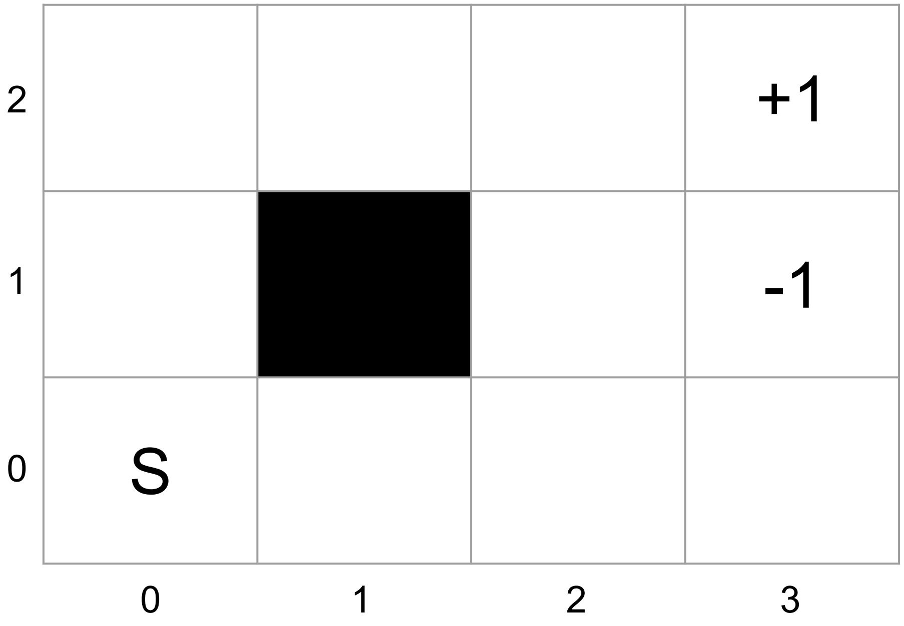
Advantages and disadvantages of approximation

- + 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.

Exercise: approximate Q-learning

Features:
COL ∈ {0, ⅓, ⅔, 1},
R0 ∈ {0, 1},   R1 ∈ {0, 1},   R2 ∈ {0, 1}



discount: 0.9   learning rate: 0.2

Use these exploration traces:

(0,0)→(1,0)→(2,0)→(2,1)→(3,1)

(0,0)→(0,1)→(0,2)→(1,2)→(2,2)→(3,2)

(0,0)→(0,1)→(0,2)→(1,2)→(2,2)→(2,1)→(3,1)

(0,0)→(0,1)→(0,2)→(1,2)→(2,2)→(3,2)

$$Q(s,a) = \sum_{i}^{n} f_i(s,a) w_i$$

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