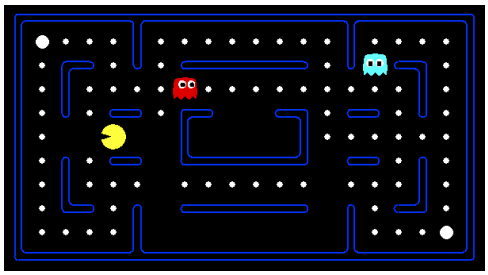
CS188 Fall 2013 Section 5: Reinforcement Learning

1 Learning with Feature-based Representations

We would like to use a Q-learning agent for Pacman, but the state size for a large grid is too massive to hold in memory (just like at the end of Project 3). To solve this, we will switch to feature-based representation of Pacman's state. Here's a Pacman board to refresh your memory:



1. What features would you extract from a Pacman board to judge the expected outcome of the game?
   The usual ones, for example as in Project 2.

2. Say our two minimal features are the number of ghosts within 1 step of Pacman (F_g) and the number of food pellets within 1 step of Pacman (F_p). For this pacman board:

   Extract the two features (calculate their values).
   $f_g = 2, f_p = 1$

3. With Q Learning, we train off of a few episodes, so our weights begin to take on values. Right now $w_g = 100$ and $w_p = -10$. Calculate the Q value for the state above.
   First of all, the Q value will not depend on what action is taken, because the features we extract do not depend on the action, only the state.

$$Q(s,a) = w_g * f_g + w_p * f_p = 100 * 2 + -10 * 1 = 190$$

4. We receive an episode, so now we need to update our values. An episode consists of a start state s, an action a, an end state s', and a reward $R(s, a, s')$. The start state of the episode is the state above (where you already calculated the feature values and the expected Q value). The next state has feature values $F_g = 0$ and $F_p = 2$ and the reward is 50. Assuming a discount of 0.5, calculate the new estimate of the Q value for s based on this episode.

$$\begin{aligned} Q_{new}(s, a) &= R(s, a, s') + \gamma * \max_{a'} Q(s', a') \\ &= 50 + 0.5 * (100 * 0 + -10 * 2) \\ &= 40 \end{aligned}$$

5. With this new estimate and a learning rate (α) of 0.5, update the weights for each feature.

$$w_g = w_g + \alpha * (Q_{new}(s, a) - Q(s, a)) * f_g(s, a) = 100 + 0.5 * (40 - 190) * 2 = -50$$
$$w_p = w_p + \alpha * (Q_{new}(s, a) - Q(s, a)) * f_p(s, a) = -10 + 0.5 * (40 - 190) * 1 = -85$$

   Note that now the weight on ghosts is negative, which makes sense (ghosts should indeed be avoided). Although the weight on food pellets is now also negative, the difference between the two weights is now much lower.

6. Good job on updating the weights. Now let's think about this entire process one step back. What values do we learn in this process (assuming features are defined)? When we have completed learning, how do we tell if Pacman does a good job?
   The values we learn are the feature weights. Once Pacman completes its learning, we will evaluate its performance by running the game and looking at the win/lose outcomes and the reward accrued from the start state.

7. In some sense, we can think about this entire process, on a meta level, as an input we control that produces an output that we would like to maximize. If you have a magical function ($F(input)$) that maps an input to an output you would like to maximize, what techniques (from math, CS, etc) can we use to search for the best inputs? Keep in mind that the magical function is a black box.
   Of course, you can use random search (changing values indiscriminately), or try some evenly distributed values in the possible range (which is sometimes called beam search).

   Also, remember that earlier in the course we talked about local search, which includes techniques such as hill climbing (going in the direction of maximum positive change), simulated annealing (where the rate of random exploration is lowered as time goes on), and genetic algorithms. These are all possible solutions.

8. Now say we can calculate the derivative of the magical function, $F'(input)$, giving us a gradient or slope. What techniques can we use now?
   Gradient descent, or many techniques from calculus and optimization developed for such problems. Although these techniques will be very useful to you as an artificial intelligence researcher, don't worry: we will not expect you to know how to use any of them on the exams!

2 Odds and Ends

1. When using features to represent the Q-function is it guaranteed that the feature-based Q-learning finds the same optimal $Q*$ as would be found when using a tabular representation for the Q-function?
   No, if the optimal Q-function $Q*$ cannot be represented as a weighted combination of features, then the feature-based representation would not have the expressive power to find it.

2. Why is temporal difference (TD) learning of Q-values (Q-learning) superior to TD learning of values?
   Because if you use temporal difference learning on the values, it is hard to extract a policy from the learned values. Specifically, you would need to know the transition model T. For TD learning of Q-values, the policy can be extracted directly by taking $\pi(s) = \arg\max_a Q(s, a)$.

3. Can all MDPs be solved using expectimax search? Justify your answer.
   No, MDPs with self loops lead to infinite expectimax trees. Unlike search problems, this issue cannot be addressed with a graph-search variant.

4. When learning with ϵ-greedy action selection, is it a good idea to decrease ϵ to 0 with time? Why or why not?
   Yes, especially when using on-policy learning methods. The reason is that as the agent learns the actual optimal policy for the world, it should switch from a mix of exploration and exploitation to mostly exploitation (unless the world is changing, in which case it should always keep exploring).