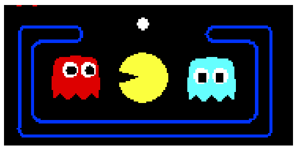
CS188 Fall 2017 Section 6: RL

1 Pacman with Feature-Based Q-Learning

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. To solve this, we will switch to feature-based representation of Pacman's state.

1. 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). You'll notice that these features depend only on the state, not the actions you take. Keep that in mind as you answer the next couple of questions. For this pacman board:



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

2. 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.

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

3. 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. 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 $\gamma = 0.5$, calculate the new estimate of the Q value for s based on this episode.

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

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

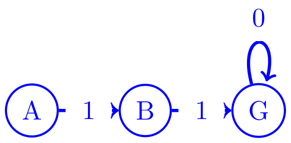
$$w_g = w_g + \alpha * (Q_{new}(s,a) - Q(s,a)) * f_g(s,a) = 100 + 0.5 * (40 - 190) * 2 = -50$$
$$w_p = w_p + \alpha * (Q_{new}(s,a) - Q(s,a)) * f_p(s,a) = -10 + 0.5 * (40 - 190) * 1 = -85$$

   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.

2 Odds and Ends

1. 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.

2. 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. For example, consider the following MDP with deterministic transitions:

   With discount $\gamma = 1$, the optimal Q values are $Q(A, right) = 2, Q(B, right) = 1, Q(G, stay) = 0$.

   Suppose we have just one feature f, which depends only on states, with value $f(A) = 1$, $f(B) = 2$, and $f(G) = 0$. There's no linear function that can map the feature values for the states to the optimal Q values above, so it's not possible for feature-based Q-learning to find the optimal values.

3. Why might Q-learning be superior to TD learning of values?

   (a) 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 and reward function R. For Q-learning, the policy can be extracted directly by taking $\pi(s) = \arg\max_a Q(s, a)$.

   (b) While TD learning learns values for a particular policy (online policy evaluation), Q-learning is *off-policy*: it will converge to Q-values that give you the optimal policy, even if the actions you used to explore were sub-optimal. The caveats to this are that you need to explore enough, and you need conditions on the learning rate α (has to eventually be small enough, but not decrease too fast)

4. When performing Q-learning with ϵ-greedy action selection, is it a good idea to decrease ϵ to 0 with time? Why or why not? Remember that ϵ is the (small) probability that you choose a *random* action, and $1 - \epsilon$ is the (large) probability you act on your current policy.
   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).