D5: Reinforcement Learning

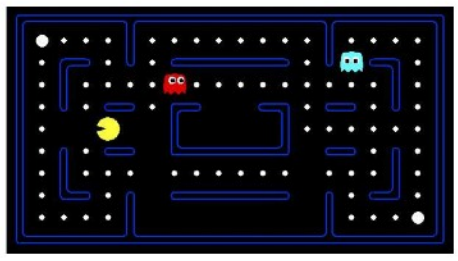
Tuesday, October 30, 2018 8:30 PM

section_5_Z4EbG4SSFwsVQHZSQfzzLLGCZqxqH2

CS188 Spring 2014 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?

   - Distance to food
   - Distance to ghost
   - Powerpellet on?

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).

   Fg=2
   Fp=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.

   generally Q states are associated with state action pairs

   However, let $Q = w^T f = (100*2-10*1)=190$

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.

(s,a,s') resulted in R(s,a,s') = 50
V(s') = $w^T f(s')$ = 100*0-2*10=-20
Expected V(s) = 190
Realized V(s) = 50+gamma*-20 = 40

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

$$w_g = w_g + \alpha \left[R(s) + \gamma \max_{a'} \hat{Q}_\theta(s', a') - \hat{Q}_\theta(s, a)\right] \frac{\delta \hat{Q}_\theta(s, a)}{\delta \theta_g}$$

=100+0.5*(50+0.5*-20-190)*2= -50
w_i = -10+0.5*(50+0.5*-20-190)*1=-85

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?

We learn w. Whicih lets us approximate Q(s, a)

Play a shit ton of games. Compare final utility from games against Q(s,a)

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.

Gradient descent using approximations of slope by applying perturbations around F(input)

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
Stochastic Gradient descent

w

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?

   feature representation Q-learning vs Q-learning
   save space!

   No guarantees. bad feature selections = GAME OVER MAN

2. Why is temporal difference (TD) learning of Q-values (Q-learning) superior to TD learning of values?

   Q-value is superior because we can use it for "active" policy learning without relying on a model!

3. Can all MDPs be solved using expectimax search? Justify your answer.

   No!

   Exploring an MDP may involve cycles, therefore, the expectimax tree may have infinite depth

   Explicit search is not possible in this case. one must rely on recursive equations or other "analytical" simplifications.

4. When learning with ϵ-greedy action selection, is it a good idea to decrease ϵ to 0 with time? Why or why not?

   Sure. In the limit we will have learned Q. Therefore, we will have discovered optimal policy.

   Of course, there may always be small probabilities of undiscovered regions of the graph.
   So the answer is subjective without absolute knowledge that there arent amazing hidden rewards

   on-policy: i can actively learn but I have to act on the policy
   off-policy: i can learn about the policy and improve the policy, even though I just do whatever random junk
   my actions dictate how mcuh confidence ill have in my Q(s,a) but it doesn't inherently change what acitons I take if i want to keep learning with high probability that firepits are awful.

Temporal difference both uses a data point by data point update method. Can be considered taking
a "running average", especially in terms of the limit
Bellman:

$$V(s) = argmax_a Q(s,a) = argmax_a \sum_{s\prime} (R(s,a,s') + \gamma V(s'))$$

#this is your one-layer search expectimax, at termination (one-layer) use evaluation function of previous guess
Consider both update functions
Given experience: s, a, s', r. Update function

$$V_k(s) = (1-\alpha)V_{k-1}(s) + \alpha(+\gamma V(s'))$$

over time, we'll aggregate V(s) with the different s' that we will get
the probabilistic average…. so that's fine

However, how can we determine a policy?
pi(s) = argmax(w.r.t. a in A, Q(s,a))
if we want to expand Q(s,a) we have to use model information (R, T)
GO FUCK YA SELF

Temporal difference both uses a data point by data point update method. Can be considered taking
a "running average", especially in terms of the limit
Bellman:

$$Q(s,a) = \sum_{s\prime} T(s,a,s')(R(s,a,s') + \gamma V(s'))$$

$$= \sum_{s\prime} T(s,a,s')(R(s,a,s') + \gamma\ argmax_{a'}($$

Consider both update functions
Given experience: s, a, s', r

$$Q(s,a) = (1-\alpha)Q_{k-1}(s,a) + \alpha($$
$$+ \gamma\ argmax_{a'} Q(s',a'))$$

NOTE: we make an interesting assumptiong that Q(s,a) always ends up in s'. With enough iterations this averages out to the right value for whatever Q(s,a), using a bunch of it's different s' over time