Active Learning

- We saw how to learn $V_\pi(s)$ with TD Learning.
- That was good for passive Learning.
  - In passive Learning, we are given samples generated by policy π
  - Our task is to find $V_\pi(s)$. TD Learning is a model-free approach to learn $V_\pi(s)$'s.
- Active Learning: We want to be able to do more than that?

We want to be able to update policy π (i.e., the sampling policy) so that not only we collect samples and learn the model, but also make decision that give us good rewards.

- Active Reinforcement Learning:
  - Actively Collecting data, while we Learn the model
- Problem model:
  - We don't know T and R, and We choose the actions.
  - Goal: Learn the optimal policy / actions.

Active Learning

- In active reinforcement learning, Learner makes the choices.
- Fundamental tradeoff:
  - exploration v.s. exploitation

Recall: To Estimate an Expectation, use Running Average of Samples

- Let's first write the Bellman equation in terms of Values (i.e. $V^*(s)$):
  - $V^*(s) = \max_a \sum_{s'} T(s,a,s') [R(s,a,s') + \gamma V^*(s')]$
  - Value iteration finds $V^*(s)$ with dynamic programming, i.e.

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') [R(s,a,s') + \gamma V_k(s')]$$

  - In reinforcement Learning, we don't have T and R.

★ That's why we usually use samples to estimate expectations.

- e.g., in TD we could estimate $V_\pi(s)$ w/ running average of samples

$$V_\pi(s) = \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V_\pi(s')] \xrightarrow[\text{running average of samples}]{\text{So, we estimated it with}} V_{\pi,k+1}(s) = (1-\alpha) V_{\pi,k}(s) + \alpha [R(s,a,s') + \gamma V_{\pi,k}(s')]$$

Can We Use Running Average to Estimate Optimal Values?

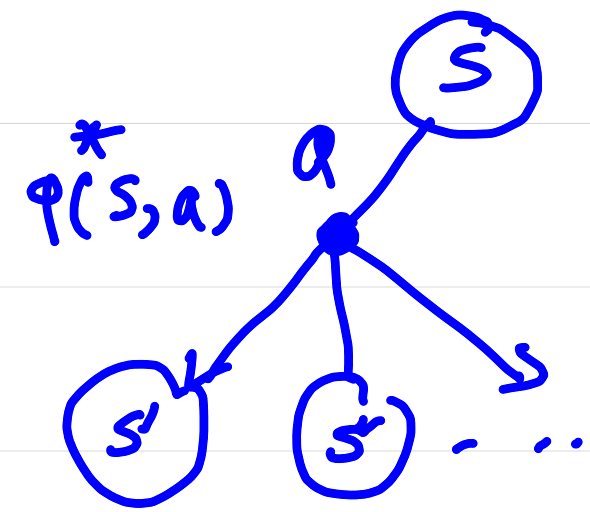
- Now, we want to do active learning,
  - i.e., Learning the optimal policy / value / q-value
- Can we use the same idea as in TD to find $V^*(s)$?
  - In other words, Can we use running averaging to find $V^*(s)$?
- Let's revisit the recursion for optimal values, i.e. $V^*(s)$:
  - $V^*(s) = \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$
- Main question: Is $V^*(s)$ written in the above recursion an expectation? Yes ☐ No ☑

  why? Because of the max.

Can We Use Running Average to Estimate q-values?

- Now, let's write the recursion for $Q^*(s,a)$ in terms of Q^*

$$\boxed{}\; Q^*(s,a) = \underbrace{\sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma \max_{a'} Q^*(s',a')\right]}_{\text{This is an expectation}}$$



- Nice! We can use the running average:

$$\text{Sample} = R(s,a,s') + \gamma \max_{a'} Q(s',a')$$

$$Q(s,a) \longleftarrow (1-\alpha)\, Q(s,a) + \alpha\ \text{Sample}$$

- Q-Learning: Sample-based q-value iteration.

- We have: $Q^*(s,a) = \sum T(s,a,s')\left[R(s,a,s') + \gamma \max_{a'} Q^*(s',a')\right]$
- Thus, we can estimate the above expectation by
  - Whenever you receive a new sample from a transition like $(s, a, s', R(s,a,s'))$,
  - By using running averaging you can update your estimate of $Q^*(s,a)$: $Q_{k+1}(s,a) \leftarrow (1-\alpha_t)\, Q_k(s,a) + \alpha_t \underbrace{\left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a')\right]}_{\text{new sample}}$

Q-Learning Properties

- Q-learning converges to optimal q-values, even if you're acting suboptimally!
- This is called off-policy learning.
- what it means is that in the limit it doesn't matter how you select the action. Regardless of action selection, you will converge to the optimal q-values (hence optimal policy) in the limit.
- But,
  - You have to explore enough (Needs to visit all states often enough)
  - You have to eventually make the learning rate small enough, but not decrease too quickly. $\sum \alpha_t = \infty$, $\sum \alpha_t^2 < \infty$

Exploration vs. Exploitation

- It's not a good idea to always exploit the action that you consider to be the best action you've seen so far.

- You have to try many actions. You must explore.
  - They may not be good for you. But you won't know unless you try them.

How to Explore

- Several scheme for forcing exploration:
  - random actions (ε-greedy) $0 < \varepsilon \ll 1$
    - Every step, flip a coin.
    - with (small) ε probability, act randomly
    - with $(1-\varepsilon)$ probability, act based on the current policy (i.e. estimated q-values)
  - with ε-greedy, we eventually explore the space $\rightarrow$ hence, find the optimal values

    But, it keeps randomly choosing actions even when the learning is done
  - Solutions: i) lower ε over time ii) use exploration function

i) Lower ε Over Time

- One option is to set $\varepsilon = 1/t$.
- i.e., at time step t, with probability $1/t$, act randomly
- with probability $(1-1/t)$, act on optimal policy.

- It does eventually converge, but
  - It can be slow
  - It explore all states with similar probability

ii) Exploration Functions

- Idea: to explore areas whose badness is not (yet) established, and eventually stop exploring

- We can implement this idea by modifying our estimate of the q-values.

→ increase the estimate q-values based on how unexplored they are:

  - previously, new sample was: $R(s,a,s') + \gamma \max_{a'} Q(s',a')$
  - with exploration function, we modify the new sample to

$$R(s,a,s') + \gamma \max_{a'} f\big(Q(s',a'), N(s',a')\big)$$

Exploration Functions, cont'd

- $N(s',a')$: # times we had seen the pair (s',a') before
- $f\left(Q(s',a'), N(s',a')\right) = Q(s',a') + \dfrac{K}{N(s',a')}$ ← Some Constant

  ↳ exploration function

- So, the update rule with exploration function will be

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

- Note: This propagates the "bonus" back to states that lead to unknown states, as well.

Regret (not on exam)

- Regret is a measure of your total mistake cost
  - The difference between:
    - Your (expected) reward, including youthful suboptimality
    - and, optimal (expected) reward
- To minimize regret, you need to go beyond learning to be optimal.
  - It requires optimally learning to be optimal
- E.g.: random exploration and exploration function both converge to optimal solution
  - But random exploration has higher regret.

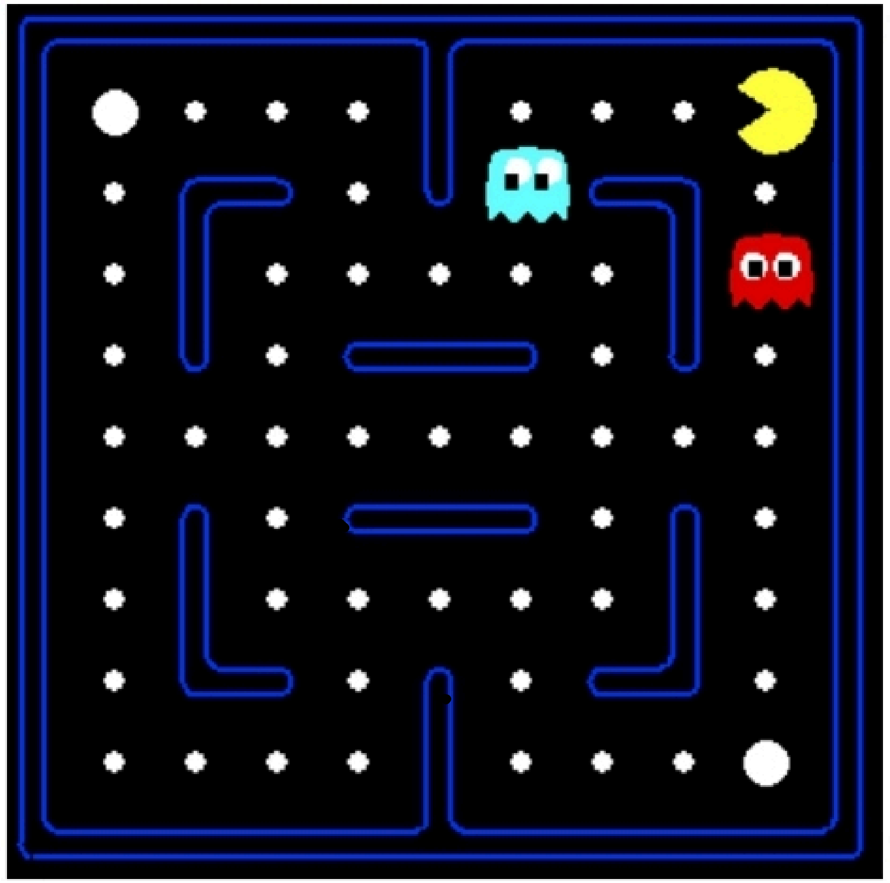
Approximat Q-Learning

- Q-Learning keeps a table of all q-values
- In real life problem, we cannot possibly learn about every single state
  - Too many state to visit them all
  - Too many states to hold the q-tables in memory
- We should generalize
  - Learn about the small number of training states you encounter during training.
  - Generalize that knowledge to new similar states (similar to what we did in supervised learning)

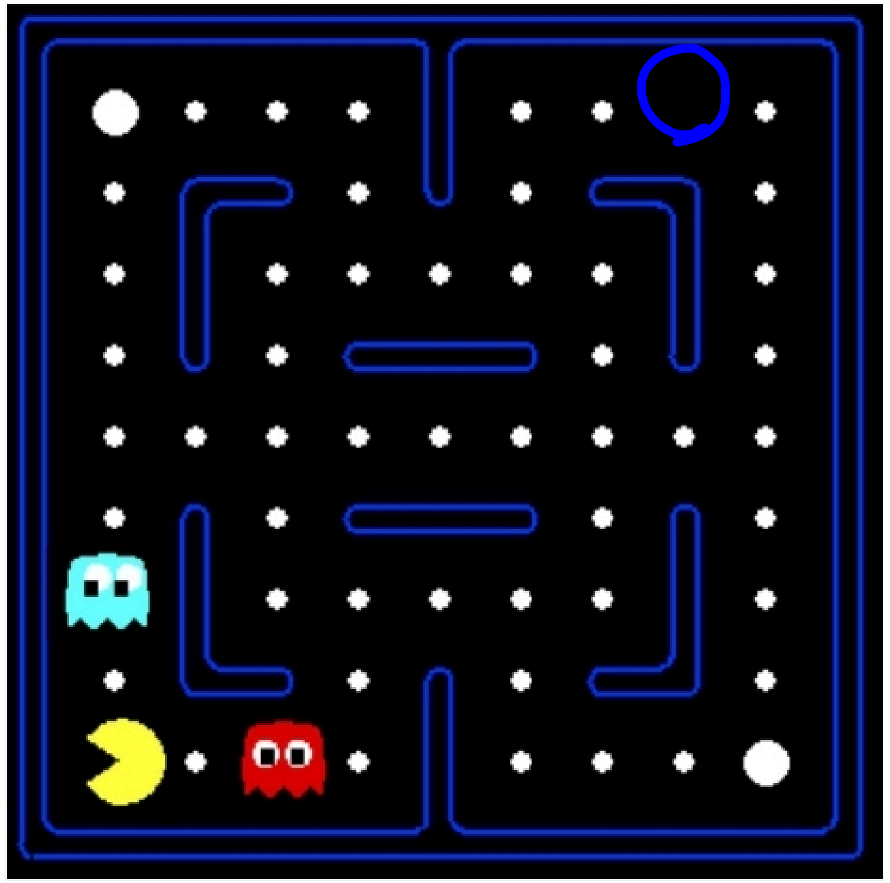
Approximate Q-Learning

Let's say we discover through experience that this state is bad:

In naïve q-learning, we know nothing about this state:



Or even this one!



$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + w_3 f_3(s,a)$$

Feature Based Representations

- Solution: Describe a (state, action) using a vector of features
- E.g.:
  - Distance to closest ghost
  - Distance to closest food
  - Is pacman in a tunnel? (0/1)
  - etc.
- Similarly, we can represent q-states (s,a) with features.
  - E.g., action moves closer to food, etc.

Linear Value Function

- Idea: approximate q-value of (s,a) with a linear combination of the features

$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \dots + w_d f_d(s,a)$$

- Pros: our experince is summed up in a few powerful numbers.
- Cons: States may share features, but could be very different
  - you must have enogh features
- But wait. We are estimating some unknown target function as a linear combination of some feature
  - We saw this before:
    - That's the beauty of ECE421. The loop is closed.

your welcome!

Approximate Q-Learning

- Q_Learning : $Q(s,a) \leftarrow Q(s,a) - \alpha \Big[\underbrace{Q(s,a) - \big(R(s,a,s') + \gamma \max_{a'} Q(s',a')\big)}_{\text{difference}} \Big]$
  - We use a table to keep track of q-values of (s,a) pairs
- Q_Learning with linear value function:
  - We have features $(f_1(s,a), f_2(s,a), \dots)$ and weight $(w_1, w_2, \dots)$
  - We must learn the weight with which we can approximate the values "best".
    - What does "best" mean?
      - ★ It should minimize the error between prediction and observation / sample.

Approximate Q-Learning and Onlin Least Squares

- It's good that we studied linear regression. We know well how to find the best parameters

weight parameter learning model: Online least squares

Goal: Finds weights that minimize the mean squared error between predicted value ($Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \dots + w_d f_d(s,a)$) and target value ($R(s,a,s') + \gamma \max_{a'} Q(s',a')$)

Learning Algorithm: SGD

we updat weights after each single transition

$$Q(s',a') = w_1 f_1(s',a') + \dots + w_d f_d(s',a')$$

$$\sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma \max_{a'} Q^*(s',a')\right] = Q^*(s)$$

Approximate Q-Learning and Online Least Squares

- SGD: $\min_{\underline{w}} E_{in}$: Thus, $w_i \leftarrow w_i - \eta \nabla_{\underline{w}} e_n$

- $e_n(\underline{w}) = \frac{1}{2}(\text{predicted} - \text{target})^2 = \frac{1}{2}\left(\underline{w}^T \underline{f}(s,a) - \left(R(s,a,s') + \gamma \max_{a'} Q(s',a')\right)\right)^2$

- $\nabla_{\underline{w}} e_n = \left(\underline{w}^T \underline{f}(s,a) - \left(R(s,a,s') + \gamma \max_{a'} Q(s',a')\right)\right) \cdot \underline{f}(s,a)$

- SGD update: $\underline{w} \leftarrow \underline{w} - \eta \nabla e_n = \underline{w} - \eta \cdot (\text{diff}) \cdot \begin{bmatrix} f_1(s,a) \\ f_2(s,a) \\ \vdots \\ f_d(s,a) \end{bmatrix}$

  - Hence, $\forall i \in \{1, \dots, d\}$: $w_i \leftarrow w_i - \eta \cdot (\text{diff})\, f_i(s,a)$

Homework: Interpret Q-Learning as an Online Least Squares problem

- In Q-learning, we saw the update

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

Use online Least squares to justify this update rule.

[Hint: It's online Least squares. So, you have a linear regression model. Clearly, specify the linear function]

[Hint: one-hot-encoding]

Title