CS 486 — Lecture 20: Reinforcement Learning

1 Policy Iteration

- Deriving the optimal policy does not require accurate estimates of the utility function, $V^*(s)$.
- Policy iteration alternates between two steps:
  1. Policy evaluation — given a policy π_i, calculate $V^{\pi_i}(s)$ which is the utility of each state if the policy were to be executed.
  2. Policy improvement — calculate a new policy π_{i+1} using V^{π_i}.

  and terminate when there is no change in the policy.
- Policy improvement can be defined as:

$$\pi_{i+1}(s) = \arg\max_a \sum_{s'} P(s'|s,a)V^{\pi_i}(s')$$

- Policy evaluation can be defined as:

$$V_i(s) = R(s) + \gamma \sum_{s'} P(s'|s,\pi_i(s))V_i(s')$$

- Solving the policy evaluation step looks similar to the Bellman equations — however, since it does not rely on the max, it is linear and thus much simpler to solve!
- For n states, it would take $O(n^3)$ time to solve exactly via linear algebra, though we can also solve faster using an iterative method to estimate the value.

2 Passive Reinforcement Learning

- Recall RL is one of three types of ML, and it's kind of in-between supervised and unsupervised learning.
- We don't provide feedback for all the examples; and we deal with rewards and punishments once in a while.
- We want to maximize our total expected reward.

2.1 Direct Utility Estimation

- Let us first consider a passive learning agent:
  - Fix a policy π.
  - The goal is to learn $V^\pi(s)$, how good the policy is, similar to policy evaluation.
  - Note the agent knows neither the transition model $P(s'|s,a)$, nor the reward function $R(s)$. This is the key difference between policy evaluation and our passive learning agent.
- $V^\pi(s)$ is the expected total discounted reward from state s onward, and is the average reward in all samples for s. Each trial provides a sample of $V^\pi(s)$ for each state visited on the trial.
- This is very simple and reduces reinforcement learning to supervised learning, but it misses the fact that the $V^\pi(s)$ values for different states are not independent, but actually related by the Bellman equations.

2.2 Adaptive Dynamic Programming

- So we want to learn the $V^\pi(s)$ via Bellman Equations. But we know neither the reward function or the transition probabilities!
- We can instead use the *observed* rewards for our reward function.
- And we can use supervised learning to deal with the transition probabilities! We just have to count our observed transitions to build our transition probabilities!

2.3 Temporal Difference Learning

- Also tries to consider relationship between the different utility values in different states, similar to adaptive dynamic.
- Recall that $V^\pi(s) = R(s) + \gamma V^\pi(s')$ should be satisfied.
- Let us state in this case that the LHS is the current estimate and the RHS is the target estimate.
- We update $V^\pi(s)$ with $V^\pi(s) + \alpha(R(s) + \gamma V^\pi(s') - V^\pi(s))$, where α is the learning rate.
- Note we don't need the transition probabilities any more! We directly use observed transitions within our update rules rather than having to estimate them.