EECS 391: Introduction to AI

Soumya Ray

http://vorlon.case.edu/~sray/eecs391_sp12/index.html

Email: sray@case.edu

Office: Olin 516

Office hours: Tues 2:30-4:00 or by appointment

Example

Policy Iteration

- Notice that in order to determine the optimal policy, we don't really need the exact values of states
  - All we need is values that result in the correct ordering of actions
- In practice, we get this long before the values themselves converge
- Policy iteration is an algorithm that exploits this idea

Policy Iteration

- Start with an initial policy
- Calculate the value of this policy (*policy evaluation step*)

$$V^{\pi}(s) = R(s, \pi(s)) + \gamma \sum_{s'} T(s, \pi(s), s') V^{\pi}(s')$$

- Calculate a new policy (*policy improvement step*) using:

$$\pi_{i+1}(s) = \arg\max_{a} R(s, a) + \gamma \sum_{s'} T(s, a, s') V^{\pi_i}(s')$$

Is this a good idea?

- It might look like we need to call value iteration as a subroutine to do policy iteration, but not really
- For a *fixed policy*, the value function equations are linear and can be solved directly for small state spaces
- For large spaces, we may need to carry out some steps of value iteration-like computation
- Still, policy convergence is often faster, so we save computation costs overall

So are we done?

- We've looked at two algorithms to solve SDM problems
- But wait, what happened to credit assignment and the exploration-exploitation tradeoff?
  - The value function is how we solved the credit assignment problem
  - But we didn't solve the exploration-exploitation tradeoff---we avoided it by assuming that the agent knows the characteristics of the world

Reinforcement Learning

- Value Iteration and Policy Iteration both require the agent to know $R(s,a)$ and $T(s,a,s')$

- In general reinforcement learning (RL), the agent will have to learn these as well

“Passive” Reinforcement Learning

- Suppose *an agent is following a fixed policy* π, and we want to compute the value function, but we don’t have the transition and reward functions

- One strategy: “adaptive” dynamic programming

Adaptive Dynamic Programming

- As the agent executes its fixed policy, keep track of transitions and rewards
- At each step, we have approximations of T and R
- We can use these approximations to evaluate the value function:

$$V_{i+1}^{\pi}(s) \leftarrow \hat{R}(s, \pi(s)) + \gamma \sum_{s'} \hat{T}(s, \pi(s), s') V_i^{\pi}(s')$$

“Model-free” Passive RL

- Adaptive DP is a “model-based” algorithm
  - It requires us to estimate T and R in order to work
- But in fact we don’t really need this; there’s a way to compute the value function without ever explicitly estimating T and R
  - This method is called “Temporal Difference” Learning or TD-learning
  - This is a “model-free” method

Temporal Difference Learning

- Start with an arbitrary value function V_0
- *For each observed (s,a,s')*, do

$$V_{i+1}^{\pi}(s) \leftarrow V_i^{\pi}(s) + \alpha\left[R(s,a) + \gamma V_i^{\pi}(s') - V_i^{\pi}(s)\right]$$

- Until $\left|V_{i+1}^{\pi}(s) - V_i^{\pi}(s)\right|$ is very small

Learning rate

Temporal difference error

Notice: No explicit $T(s,a,s')$; $R(s,a)$ does not need to be stored

Key Idea

- Policy evaluation update:

$$V_{i+1}^{\pi}(s) \leftarrow R(s,\pi(s)) + \gamma \sum_{s'} T(s,\pi(s),s')V_i^{\pi}(s')$$

- In TD-learning, we'll adjust $V^{\pi}(s)$ and $V^{\pi}(s')$ *just for each transition we observe* to better approximate the Bellman equation

- This works because the frequency of the transition from s to s' via a is going to be proportional to $T(s,a,s')$ in the long term
  - "Monte Carlo" estimation

“Active” Reinforcement Learning

- So far, we were calculating the value function for a *fixed policy*, given no knowledge of R and T

- Now let’s look at the task of *computing the optimal policy* in the same situation
  - “Active” RL

Active Adaptive DP

- Start with random policy
- As the agent executes this policy, keep track of transitions and rewards
- At each step, we have approximations of T and R
- We can use these approximations in the *value iteration* algorithm

This doesn't work!

- Why?
  - Our initial model was just an estimate
  - We acted optimally *with respect to the wrong model, without attempting to fix the inaccuracies in the model*
  - This is the exploration problem
- Fix: sometimes we have to take actions *not* recommended by our current policy

Exploration Strategy

- We need to decide how to explore so we still converge to the optimal policy

- Two reasonable exploration strategies:
  - ε-greedy
  - Optimistic

ε-greedy Exploration

- In each iteration, the agent will:
  - Follow the action recommended by the current policy with probability (1- ε)
  - Execute a random action with probability ε
  - Decay ε slowly over time

- "Greedy"—pick best action according to current policy/value function

Optimistic Exploration

- When performing value iteration, for any $R(s,a)$ not seen yet, use a large positive quantity R_{max}
  - This encourages the agent to try unseen actions

GLIE Exploration

- Both the previous strategies are "GLIE" strategies: **G**reedy in the **L**imit of **I**nfinite **E**xploration

- It can be shown that any GLIE exploration strategy will eventually converge to the optimal policy, as required

Active ADP Redux

- Start with random policy
- As the agent executes this policy *with GLIE exploration*, keep track of transitions and rewards
- At each step, we have approximations of T and R
- We can use these approximations in the *value iteration* algorithm