EECS 496: Sequential Decision Making

Soumya Ray

sray@case.edu

Office: Olin 516

Office hours : T 4-5:30 or by appointment

Recap

- Every policy has a ____ ____ ____ satisfying the ____ equation.
- The Bellman optimality criterion says: ___= ___ (____ + ___ *sum over ____ * ____)
- The BOC is ___ and ___.
- We can find the optimal policy using the ___ ____ algorithm.
- What are the steps of this algorithm?
- The rate of convergence of this procedure is proportional to the ___ ____.
- In general the policy converges ___ the value function. Why?
- This can be exploited by the ___ ___ algorithm.
- What are the steps of this algorithm?
- How do the above algorithms solve the credit assignment problem? The exploration-exploitation tradeoff?

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 (how?)

- We can use these approximations to solve the Bellman equations and find 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

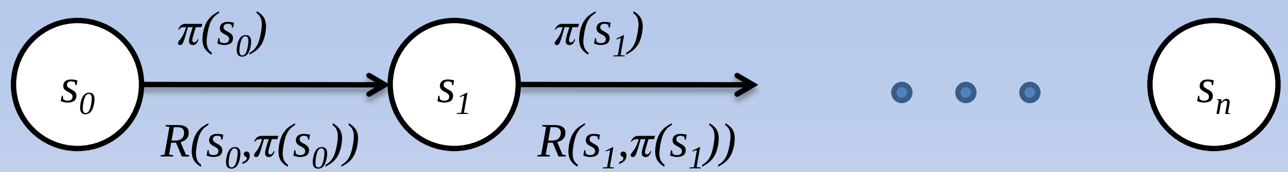
- But in fact we don’t really need this; we can compute the value function without ever explicitly estimating T and R
  - These are “model-free” methods

Passive RL with Monte Carlo

- We have a fixed policy and want to compute its value, but we don't want to store/estimate T and R

- Simple method: run the policy and keep track of the rewards seen after each state

Picture



$$V^{\pi}(s_0) = \frac{1}{N}\sum_k \left[R(s_0, \pi(s_0)) + \gamma R(s_1^k, \pi(s_1^k)) + \ldots + \gamma^{n_k} R(s_{n_k}^k, \pi(s_{n_k}^k)) \right]$$

Properties of MC methods

- Clearly this converges to the correct solution in the limit, without storing T and R
- Estimates for each state are performed independently of estimates of other states
  - Bellman equation is not used
    - Can we do better?

RL with MC/DP combination

- It seems wasteful to perform policy evaluations with MC methods, DP seems better suited to this
  - But we don't want to learn/keep the models…

- A family of methods that achieves this seemingly paradoxical goal is called "temporal difference learning"

Temporal Difference Learning

- Start with an arbitrary value function V_0
- Run the given policy
- *For each observed* $(s,\pi(s),s')$, do

Temporal difference error

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

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

Learning rate

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

Key Idea

- Policy evaluation update:

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

- TD-learning update

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

Key Idea

- 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" sampling

“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 arbitrary 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, but *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

- A reasonable exploration strategy is ε-greedy exploration

ε-greedy Exploration

- In each iteration, the agent will:
  - Follow the action recommended by the current policy (the "greedy action") 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

Boltzmann Exploration

- Put a probability distribution on all actions based on their value, then sample from that distribution:

$$\Pr(a) = \frac{e^{Q(a)/T}}{\sum_{a'} e^{Q(a')/T}}$$

Temperature parameter

GLIE Exploration

- All 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 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