CS 188 Introduction to Artificial Intelligence
Spring 2023 Note 14

These lecture notes are heavily based on notes originally written by Nikhil Sharma.

Last updated: January 15, 2023

Exploration and Exploitation

We've now covered several different methods for an agent to learn an optimal policy, and harped on the fact that "sufficient exploration" is necessary for this without really elaborating on what's really meant by "sufficient". In the upcoming two sections, we'll discuss two methods for distributing time between exploration and exploitation: ε-greedy policies and exploration functions.

ε-Greedy Policies

Agents following an **ε-greedy policy** define some probability $0 \leq \varepsilon \leq 1$, and act randomly and explore with probability ε. Accordingly, they follow their current established policy and exploit with probability $(1-\varepsilon)$. This is a very simple policy to implement, yet can still be quite difficult to handle. If a large value for ε is selected, then even after learning the optimal policy, the agent will still behave mostly randomly. Similarly, selecting a small value for ε means the agent will explore infrequently, leading Q-learning (or any other selected learning algorithm) to learn the optimal policy very slowly. To get around this, ε must be manually tuned and lowered over time to see results.

Exploration Functions

This issue of manually tuning ε is avoided by **exploration functions**, which use a modified Q-value iteration update to give some preference to visiting less-visited states. The modified update is as follows:

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha \cdot [R(s,a,s') + \gamma \max_{a'} f(s',a')]$$

where f denotes an exploration function. There exists some degree of flexibility in designing an exploration function, but a common choice is to use

$$f(s,a) = Q(s,a) + \frac{k}{N(s,a)}$$

with k being some predetermined value, and $N(s,a)$ denoting the number of times Q-state (s,a) has been visited. Agents in a state s always select the action that has the highest $f(s,a)$ from each state, and hence never have to make a probabilistic decision between exploration and exploitation. Instead, exploration is automatically encoded by the exploration function, since the term $\frac{k}{N(s,a)}$ can give enough of a "bonus" to some infrequently-taken action such that it is selected over actions with higher Q-values. As time goes on and states are visited more frequently, this bonus decreases towards 0 for each state and $f(s,a)$ regresses towards $Q(s,a)$, making exploitation more and more exclusive.

Summary

It's very important to remember that reinforcement learning has an underlying MDP, and the goal of reinforcement learning is to solve this MDP by deriving an optimal policy. The difference between using reinforcement learning and using methods like value iteration and policy iteration is the lack of knowledge of the transition function T and the reward function R for the underlying MDP. As a result, agents must *learn* the optimal policy through online trial-by-error rather than pure offline computation. There are many ways to do this:

- Model-based learning - Runs computations to estimate the values of the transition function T and the reward function R and uses MDP-solving methods like value or policy iteration with these estimates.
- Model-free learning - Avoids estimation of T and R, instead using other methods to directly estimate the values or Q-values of states.
  - Direct evaluation - follows a policy π and simply counts total rewards reaped from each state and the total number of times each state is visited. If enough samples are taken, this converges to the true values of states under π, albeit being slow and wasting information about the transitions between states.
  - Temporal difference learning - follows a policy π and uses an exponential moving average with sampled values until convergence to the true values of states under π. TD learning and direct evaluation are examples of on-policy learning, which learn the values for a specific policy before deciding whether that policy is suboptimal and needs to be updated.
  - Q-Learning - learns the optimal policy directly through trial and error with Q-value iteration updates. This an example of off-policy learning, which learns an optimal policy even when taking suboptimal actions.
  - Approximate Q-Learning - does the same thing as Q-learning but uses a feature-based representation for states to generalize learning.
- To quantify the performance of different reinforcement learning algorithms we use the notion of **regret**. Regret captures the difference between the total reward accumulated if we acted optimally in the environment from the beginning and the total reward we accumulated by running the learning algorithm.