Reinforcement Learning

Charles Elkan
elkan@cs.ucsd.edu

December 6, 2012

Reinforcement learning is the type of learning done by an agent who is trying to figure out a good policy for interacting with an environment. The framework is that time is discrete; at each time step the agent perceives the current state of the environment and selects a single action. After performing the action, the agent earns a real-valued reward or penalty, time moves forward, and the environment shifts into a new state. The goal of the agent is to learn a policy for choosing actions that leads to the best possible long-term sum of rewards.

Reinforcement learning is fundamentally different from supervised learning because correct labels are never provided explicitly to the agent. That is, the agent is never told what the right action is in a given state. Nor is the agent ever told what future rewards will be. However, at each time step the agent does find out what the current short-term reward is.

Reinforcement learning has real-world applications in domains where there is an intrinsic tradeoff between rewards and penalties in the present and in the future, and where there is an environment that can be modeled as unknown and random. Successful application areas include controlling robots, scheduling elevators, managing relationships with customers, and treating patients. In the latter three examples, the environment consists of humans, which are modeled as acting probabilistically. Each person is a separate instance of the environment. All persons are described by the same probability distributions, but the specific trajectory of each person is a different random outcome.

Reinforcement learning is appropriate for domains where the agent faces a stochastic but not adversarial environment. It is not suitable for domains where the agent faces an intelligent opponent, because in adversarial domains the opponent does not change state probabilistically. For example, reinforcement learning methods are not appropriate for learning to play chess. However, reinforcement

learning can be successful for learning to play games that involve both strategy and chance, such as backgammon, if it is reasonable to model the opponent as following a randomized strategy. Game theory is the appropriate formal framework for modeling agents interacting with intelligent opponents. Of course, finding good algorithms is even harder in game theory than in reinforcement learning.

1 Markov decision processes (MDPs)

The basic reinforcement learning scenario is formalized using three concepts:

1. a set S of alternative states that the environment may be in
2. a set A of alternative actions that the agent may take, and
3. rewards that are real numbers.

At each time t, the agent perceives its state $s_t \in S$ and the available actions A. The agent chooses one action $a \in A$ and receives from the environment the new state s_{t+1} and the reward r_{t+1}.

Formally, the environment is represented as a so-called Markov decision process (MDP). An MDP is a four-tuple (S, A, P, R) where

- S is the set of states, which is also called the state space,
- A is the set of actions,
- P is the conditional probability distribution $p(S_{t+1} = s' | S_t = s, A_t = a)$ that describes the probability that doing action a in state s at time t will lead to state s' at time $t + 1$, and
- R is the probability distribution $p(r|s, a)$ of the immediate reward r received after doing action a in state s.

The states of an MDP possess the Markov property. This means that if the current state of the MDP at time t is known, then transitions to a new state at time $t + 1$ are independent of all previous states. In other words, "history is irrelevant."

Some extensions of the MDP framework are possible without changing the nature of the problem fundamentally. In particular, the reward may depend on the next state as well as on the current state and action, that is the reward can be given by a distribution $p(r|s, a, s')$. Or, for maximal mathematical simplicity, the

reward can be a deterministic function just of the current state, $R : S \to \mathbb{R}$. It is crucial that state transitions and rewards are stationary: although they are random, their probability distributions are the same at all times. Also, we assume that the reward at each time step is bounded: $|r_t| < b$ for some known constant b, for all time steps t.

The goal of the agent is to maximize a cumulative function of the rewards. If the maximum time step is $t = T$ where T is finite, then the goal is to maximize $\sum t = 1^T r_t$. (The initial state may called $t = 0$ or $t = 1$.) If the maximum time horizon is infinite, then the goal is to maximize a discounted sum that involves a discounting factor γ, which is usually a number just under 1:

$$R = \sum_{t=1}^{\infty} \gamma^t r(s_t, a_t).$$

The factor γ makes the sum above finite even though it is a sum over an infinite number of time steps.

In order to maximize its total reward, the agent must acquire a policy. A deterministic and stationary policy is a mapping $\pi : S \to A$ that specifies which action to take for every state. An optimal policy $\pi*$ is one that maximizes the expected value of R, where the expectation is based on the transition probability distribution $p(S_{t+1} = s' | S_t = s, A_t = \pi^*(s_t))$.

Given an MDP, an optimal policy is sometimes called a solution of the MDP. Optimal policies always have the Markov property: the best action to choose depends on the current state only, If the time horizon is infinite, then optimal policies are also stationary: the best action is the same regardless of the current time. With a finite time horizon, the best action may be a function $\pi^*(s, t)$ of the current time t as well as of the current state t. Optimal policies may be not unique.

2 Policy iteration

Assume for now that the agent knows the transition probabilities $p(s'|s, a)$, and that the agent also knows the immediate reward function, which is a deterministic function $S \times A \to \mathbb{R}$. Then the agent can compute an optimal policy without needing any data collected from interacting with the MDP.

End of the lecture on Tuesday November 27.

Let $\pi : S \rightarrow A$ be any deterministic, stationary policy. If the state space S is a discrete finite set, as is the action space A, then the number of alternative policies is finite. Specifically, this number is m^n where $m = |A|$ and $n = |S|$. The function $V^\pi : S \rightarrow \mathbb{R}$ gives the expected total reward achieved by starting in state s and acting according to the policy π. This function, called the value function of π, satisfies the equation

$$V^\pi(s) = r(s, \pi(s)) + \gamma \sum_{s'} p(s'|s, \pi(s)) V^\pi(s'). \tag{1}$$

Note that this equation is linear. An algorithm called policy iteration, due to Howard (1960), uses value functions to find an optimal policy. The policy iteration method begins with an arbitrary policy π and repeats two steps until convergence:

1. Compute V^π given π (policy evaluation).

2. Find a better π' based on V^π (policy improvement).

Computing V^π in Step 1 is formulated and solved as a set of simultaneous linear equations. For each state s in the finite state space S, Equation (1) gives one equation, so there are n equations in total. There are also n unknowns, namely $V^\pi(s)$ for each s. Step 2 is performed using the update rule

$$\pi'(s) = \text{argmax}_a \left[r(s, a) + \gamma \sum_{s'} p(s'|s, a) V^\pi(s') \right]. \tag{2}$$

Policy iteration has the advantage that there is a clear stopping condition: when the function V^π does not change after performing Step 1, then the algorithm terminates. Note that the optimal value function is unique, but the optimal policy is not always unique.

An agent can execute the policy iteration algorithm only if it knows the transition probabilities of the environment. If these probabilities are unknown, then the agent must discover them at the same time it is trying to discover a good policy. This is a learning problem as opposed to an inference problem. It is called reinforcement learning.