CS229 Lecture notes

原作者：[Andrew Ng](http://cs229.stanford.edu/)（[吴恩达](http://open.163.com/movie/2008/1/M/C/M6SGF6VB4_M6SGHFBMC.html)）

翻译：[CycleUser](https://zhuanlan.zhihu.com/python-kivy)

Part XIII

Reinforcement Learning and Control

We now begin our study of reinforcement learning and adaptive control. In supervised learning, we saw algorithms that tried to make their outputs mimic the labels y given in the training set. In that setting, the labels gave an unambiguous “right answer” for each of the inputs x. In contrast, for many sequential decisions making and control problems, it is very difficult to provide this type of explicit supervision to a learning algorithm. For example, if we have just built a four-legged robot and are trying to program it to walk, then initially we have no idea what the “correct” actions to take are to make it walk, and so do not know how to provide explicit supervision for a learning algorithm to try to mimic. In the reinforcement learning framework, we will instead provide our algorithms only a reward function, which indicates to the learning agent when it is doing well, and when it is doing poorly. In the four-legged walking ex- ample, the reward function might give the robot positive rewards for moving forwards, and negative rewards for either moving backwards or falling over. It will then be the learning algorithm’s job to figure out how to choose actions over time so as to obtain large rewards.

Reinforcement learning has been successful in applications as diverse as autonomous helicopter flight, robot legged locomotion, cell-phone network routing, marketing strategy selection, factory control, and efficient web-page indexing. Our study of reinforcement learning will begin with a definition of the Markov decision processes (MDP), which provides the formalism in which RL problems are usually posed.

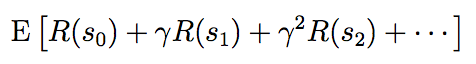
# 1 Markov decision processes

A Markov decision process is a tuple (S, A, {Psa}, γ, R), where:

1. S is a set of states. (For example, in autonomous helicopter flight, S might be the set of all possible positions and orientations of the helicopter.)
2. A is a set of actions. (For example, the set of all possible directions in which you can push the helicopter’s control sticks.)
3. Psa are the state transition probabilities. For each state s ∈ S and action a ∈ A, Psa is a distribution over the state space. We’ll say more about this later, but briefly, Psa gives the distribution over what states we will transition to if we take action a in state s.
4. γ ∈ [0, 1] is called the discount factor.
5. R : S × A → R is the reward function. (Rewards are sometimes also written as a function of a state S only, in which case we would have R : S → R).

The dynamics of an MDP proceeds as follows: We start in some state s0, and get to choose some action a0 ∈ A to take in the MDP. As a result of our choice, the state of the MDP randomly transitions to some successor state s1, drawn according to s1 ∼ Ps0a0. Then, we get to pick another action a1. As a result of this action, the state transitions again, now to some s2 ∼ Ps1a1 . We then pick a2, and so on. . . . Pictorially, we can represent this process as follows:  a0 a1 a2 a3 s0 −→s1 −→s2 −→s3 −→...  Upon visiting the sequence of states s0, s1, . . . with actions a0, a1, . . ., our total payoff is given by  R(s0,a0) + γR(s1,a1) + γ2R(s2,a2) + ··· . Or, when we are writing rewards as a function of the states only, this becomes R(s0) + γR(s1) + γ2R(s2) + ··· .  For most of our development, we will use the simpler state-rewards R(s), though the generalization to state-action rewards R(s,a) offers no special difficulties.

Our goal in reinforcement learning is to choose actions over time so as to maximize the expected value of the total payoff:



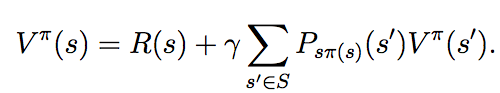
Note that the reward at timestep t is discounted by a factor of γt. Thus, to make this expectation large, we would like to accrue positive rewards as soon as possible (and postpone negative rewards as long as possible). In economic applications where R(·) is the amount of money made, γ also has a natural interpretation in terms of the interest rate (where a dollar today is worth more than a dollar tomorrow).

A policy is any function π : S → A mapping from the states to the actions. We say that we are executing some policy π if, whenever we are in state s, we take action a = π(s). We also define the value function for a policy π according to



Vπ(s) is simply the expected sum of discounted rewards upon starting in state s, and taking actions according to π.1

Given a fixed policy π, its value function V π satisfies the Bellman equations:



This says that the expected sum of discounted rewards Vπ(s) for starting in s consists of two terms: First, the immediate reward R(s) that we get rightaway simply for starting in state s, and second, the expected sum of future discounted rewards. Examining the second term in more detail, we see that the summation term above can be rewritten Es′∼Psπ(s) [V π(s′)]. This is the expected sum of discounted rewards for starting in state s′, where s′ is distributed according Psπ(s), which is the distribution over where we will end up after taking the first action π(s) in the MDP from state s. Thus, the second term above gives the expected sum of discounted rewards obtained after the first step in the MDP.

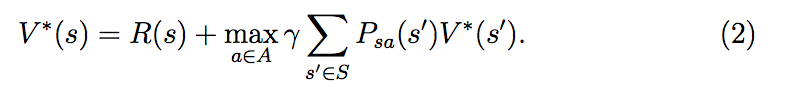
Bellman’s equations can be used to efficiently solve for Vπ. Specifically, in a finite-state MDP (|S| < ∞), we can write down one such equation for V π (s) for every state s. This gives us a set of |S | linear equations in |S | variables (the unknown Vπ(s)’s, one for each state), which can be efficiently solved for the V π (s)’s.

1This notation in which we condition on π isn’t technically correct because π isn’t a random variable, but this is quite standard in the literature.

We also define the **optimal value function** according to

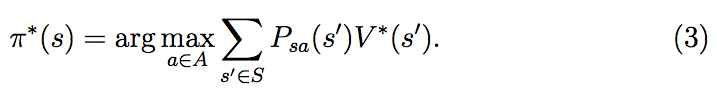


In other words, this is the best possible expected sum of discounted rewards that can be attained using any policy. There is also a version of Bellman’s equations for the optimal value function:



The first term above is the immediate reward as before. The second term is the maximum over all actions a of the expected future sum of discounted rewards we’ll get upon after action a. You should make sure you understand this equation and see why it makes sense.

We also define a policy π∗ : S → A as follows:



Note that π∗(s) gives the action a that attains the maximum in the “max” in Equation (2).

It is a fact that for every state s and every policy π, we have



The first equality says that the V π∗ , the value function for π∗, is equal to the optimal value function V ∗ for every state s. Further, the inequality above says that π∗’s value is at least a large as the value of any other other policy. In other words, π∗ as defined in Equation (3) is the optimal policy.

Note that π∗ has the interesting property that it is the optimal policy for all states s. Specifically, it is not the case that if we were starting in some state s then there’d be some optimal policy for that state, and if we were starting in some other state s′ then there’d be some other policy that’s optimal policy for s′. Specifically, the same policy π∗ attains the maximum in Equation (1) for all states s. This means that we can use the same policy π∗ no matter what the initial state of our MDP is.