Solving the inverted pendulum task via policy iteration

1 Reflections so far

Last chapter's solution to the inverted pendulum task was unsatisfactory. While the solution worked reasonably well in certain cases—for instance, it produced somewhat intelligent-looking pendulum-balancing behavior when the pendulum was nearly upright with low angular velocity—it certainly did not maintain balance generally, and could not recover once fallen over. The solution performed poorly due to two stiff limitations on our setting of the MDP: the state space was very coarse-grained, and our estimates of the transition probabilities were only a very rough approximation of "real experience".

In this chapter we work out an alternative method for solving the MDP which does not require a direct numerical solution. This new method—*policy iteration*—is generally less expensive than directly solving the Bellman equations. This will ease the first of the two aforementioned limitations, allowing us to set the MDP in a more fine-grained state space. In the next chapter we'll take on the second limitation, working out a method to estimate transition probabilities by sampling from real experience.

2 Policy iteration

At one point in the last chapter we stopped and noted the following. We can readily characterize a value function with respect to any given policy, and we can easily find an optimal policy with respect to any given value function:

$$\pi^*(s) = \arg\max_a \sum_{s' \in \mathcal{S}} P_a(s, s') V(s') \tag{1}$$

$$V(s) = \sum_{s' \in \mathcal{S}} P_{\pi(s)}(s, s') \Big(R_{\pi(s)}(s, s') + \gamma V(s') \Big). \tag{2}$$

In the end, we wrote down a set of Bellman equations which could be solved directly for the optimal policy. Recall, however, that we had to solve a set of $|\mathcal{S}|$ nonlinear recurrence relations in $|\mathcal{S}|$ variables, each one requiring a maximum over $|\mathcal{A}|$ expressions. This was akin to an exhaustive search over all possible actions that could be taken in all possible states in order to transition into all possible states.

Instead of solving for the optimal policy and optimal value function all at once, we can imagine a step-wise back-and-forth solution which goes something like this:

1. Select a random policy.
2. Compute the true value function under this policy.
3. With this true value function, we can see a better policy than the original one. Select that policy instead.
4. Compute the true value function under this new policy.
5. Using this new true value function, we can see an even better policy. So we choose that one instead... and so forth.
6. We repeat this process, back and forth, until we find that we no longer need to change our policy. At this point we are guaranteed to have an optimal policy.

This process is called *policy iteration.* Sutton & Barto (2017) offer an insightful distillation of the concept:

> [Policy iteration] is the general idea of two interacting processes revolving around an approximate policy and an approximate value function. One process takes the policy as given and perform some form of policy evaluation, changing the value function to be more like the true value function for the policy. The other process takes the value function as given and performs some form of policy improvement, changing the policy to make it better, assuming that the value function is its value function. Although each process changes the basis for the other, overall they work together to find a joint solution: a policy and value function that are unchanged by either process and, consequently, are optimal.

To implement this, we just need to work out the details for each of the two steps:

(i) finding the value function for a policy (policy evaluation)

(ii) finding the optimal policy for a value function (policy improvement).

We've already examined each of these steps in the past. Here we'll write out and explore their equations a refresher, starting with policy improvement.

Let's say that I'm handed a value function from on high. For the time being it doesn't matter under which policy is was computed, all that matters is that I have some function which accepts a state as an argument and returns the value of that state. Choosing an optimal policy is straightforward: we simply choose, at each state, the action which maximizes the expected value of the state which will follow.

$$\pi^*(s) = \arg\max_a \sum_{s' \in \mathcal{S}} P_a(s, s')V(s'). \tag{3}$$

In the last chapter we also derived the equation for the value function given a particular policy. It's the expected reward that results immediately from acting on our policy, plus the discounted expected value of the state which will follow:

$$V(s) = \sum_{s' \in \mathcal{S}} P_{\pi(s)}(s, s') \Big(R_{\pi(s)}(s, s') + \gamma V(s') \Big). \tag{4}$$

Of the two steps, this is the one which requires more computational effort. Finding $V(s)$ requires solving a set of $|\mathcal{S}|$ linear equations in $|\mathcal{S}|$ unknowns. Roughly, we have two options for solving this system: (i) directly, using methods from linear algebra, and (ii) via iterative numerical methods.

3 Computing the value function

Here we'll just consider the linear algebraic solution. Note that the function $V_\pi(s)$ can be represented by a lookup table holding the value for each individual state. If we have k states, this table is a vector of k entries where the i^{th} entry is the value of the i^{th} state in the set of all states:

$$\mathbf{v} \,|\, v_i = V(s_i) \,\text{for}\, i = \{1, ..., k\} \text{ where } |\mathcal{S}| = \{s_1, ..., s_k\}\,. \tag{5}$$

We'll denote the policy as a lookup table as well, a vector of k elements where the i^{th} element is an index in the range $\{1, ..., m\}$ for the action to be taken in state s_i, with m being the total number of actions available (that is, $\mathcal{A} = \{a_1, ..., a_m\}$):

$$\pi \,|\, \pi_i = \pi(s_i). \tag{6}$$

The transition probabilities can be represented by another lookup table, this time a $k \times k$ matrix where the entry i, j is the probability of transitioning from state s_i to s_j due to action π_i:

$$P \,|\, P_{i,j} = P_{\pi_i}(s = s_i, s' = s_j). \tag{7}$$

Like the transition probabilities, the reward function is a lookup table in the form of a matrix—a $k \times k$ matrix where the entry i, j is the immediate scalar reward received from transitioning from state s_i to s_j due to action π_i:

$$R \,|\, R_{i,j} = R_{\pi_i}(s = s_i, s' = s_j). \tag{8}$$

Now we have the machinery in place to re-write equation (4) with our vectors and matrices. For the state s_i, the scalar value v_i is given by

$$v_i = \sum_{j=1}^{k} P_{i,j}(R_{i,j} + \gamma v_j) \tag{9}$$

$$= \sum_{j=1}^{k} P_{i,j} R_{i,j} + \gamma \sum_{j=1}^{k} P_{i,j} v_j. \tag{10}$$

Let $\mathbf{p}_i$ be a vector representing the i^{th} row of P and $\mathbf{r}_i$ be a vector representing the i^{th} column of R^T (or, equivalently, the i^{th} row of R). We can collapse the sums to dot products and write the above equation as

$$v_i = \mathbf{p}_i \cdot \mathbf{r}_i + \gamma \mathbf{p}_i \cdot \mathbf{v}. \tag{11}$$

This can be expressed even more compactly by the matrix multiplication

$$\mathbf{v} = PR^T + \gamma P\mathbf{v}. \tag{12}$$

Subtracting $\mathbf{v}$ from the right hand side gives

$$0 = (\gamma P - 1)\mathbf{v} + PR^T, \tag{13}$$

and this is a nonhomogeneous linear equation of the form

$$A\mathbf{v} + \beta = 0 \,\text{where}\, A = \gamma P - 1 \,\text{and}\, \beta = PR^T \tag{14}$$

which is satisfied when

$$\begin{aligned} \mathbf{v} &= -A^{-1}\beta & (15)\\ &= -(\gamma P - 1)^{-1} PR^T & (16)\\ &= (1 - \gamma P)^{-1} PR^T. & (17) \end{aligned}$$

This is our algebraic solution for $V(s)$.

TO DO:

-conceptual re-casting of the explanation, severe revision

-consistency of language, decide which terms to use, and then define and use those only

-explain iterative policy evaluation?

-philosophical discussion of bootstrapping nature of reinforcement learning

-fix notation in matrix math!—should not be PR^T but a vector produced by dotting each row of P with each row of R