Expectation Values and the Bellman Equation

Phil Tabor

Reinforcement learning deals with agents acting upon an environment, causing it to transition from one state to another, and receiving a reward in the process. The ultimate goal of the agent is to maximize this total reward over time. Fortunately, we have the machinery of the Markov Decision Process at our disposal, which makes the calculation of the optimal trajectory through the environment a tractable, though often difficult, problem to solve. The details of how this is accomplished will depend on the nature of the system, and what is known about it in advance.[1]

When we are dealing with well understood deterministic systems, the way in which the agent's actions affect the system is precisely known. This makes predicting the future state of the system simple, since the agent just has to know the current state and what action it's taking. It can predict how the system will evolve with absolute certainty, and this makes planning for the future an easy matter. It simply observes the state of the system and then takes the action that will yield the best reward. By applying this heuristic at every move, the agent will naturally maximize its total reward over time. Unfortunately, we rarely have this luxury, due to a lack of sufficient knowledge of the system dynamics.

In most applications, the system may very well be deterministic, but its dynamics are not known in advance, thus making it impossible to predict the future evolution of the system with certainty. Therefore, it's not possible for the agent to select the action that yields the most profitable reward, at least at the outset. In this case, we have to make use of probabilistic calculations that tell us the likelihood of ending up in some state and receiving some reward. The issue of planning then boils down to taking actions that are the most likely to lead to states that give the agent the largest possible rewards. With some clever mathematics, we can sidestep our lack of knowledge of the dynamics of the system and employ an iterative process to maximize the agent's total reward over time.

1 The Mathematics of the Markov Decision Process

Recall that the Markov Decision Process describes a system in which the set of states, actions, and rewards have a bounded (meaning not infinite) number of elements. This set may be arbitrarily

[1]In this document I am borrowing notation and content structure from the excellent textbook on reinforcement learning, by Sutton and Barto. It's free, please check it out for more discussion.

large, but it is still finite. Since the dynamics of the system are unknown, from the perspective of the agent, the states it encounters and rewards it receives along the way are all *random* variables. This means the state and reward pairs the agent observes are described by some probability distribution. We denote this probability distribution:

$$p(s', r|s, a)$$

which reads as: 'the probability of ending up in state s prime and receiving reward r, given the agent is in state s and takes action a'.

This probability distribution has the property that:

$$\sum_{s' \in S} \sum_{r \in R} p(s', r|s, a) = 1$$

which basically means that this probability distribution describes the whole dynamics of our system. When we sum over all possible combinations, we have to end up in some other state of the system, described by that probability distribution.

What's cool about this is that we can use it to calculate the state transition probabilities, meaning, given we're in some state s and take action a, what is the probability we will end up in some other state of interest s'

$$p(s'|s, a) = \sum_{r \in R} p(s', r|s, a)$$

We have to sum over the rewards because for each state s' there is some set of possible rewards. Each of those rewards corresponds to a possible way of ending up in the state s' given state s and action a, and each of those has some probability. Probabilities are additive, so we have to take all of them into account.

This brings us to one quantity of particular interest, the expected reward for a state-action pair.

$$r(s, a) = E[R_t|S_{t-1} = s, A_{t-1} = a] = \sum_{r \in R} r \sum_{s' \in S} p(s', r|s, a)$$

This is the definition of the expectation value. In general terms, an expectation value is the sum of probabilities of an outcome multiplied by what you get in each outcome. Let's take a look at a simple example.

Suppose we're playing a simple coin toss game. If we flip the coin and it comes up heads, then you get 1 point. If it comes up tails, you get -1 point. If we flip the coin 2 times, then what is your expected number of points?

$$E[r] = p(heads)(1point) + p(tails)(-1point) = 0.5(1point) - (0.5)(1point) = 0$$

Intuitively, we should expect to get 0 total points because the coin is just a two state system, and since it's fairly weighted, each state (i.e. heads or tails) is equally likely. To get a non-zero expected reward, we have to create some kind of imbalance in the system, either through weighting the coin or changing the rewards in the event of one of the states.

This trivial example is precisely what we are doing in reinforcement learning, but for more complex systems. The added complexity means that we can't think simply in terms of single rewards for single actions, rather we have to consider the agent's trajectory through the state space of the environment, and the sequence of rewards it receives in the process.

2 Returns and Reward Discounting

When we turn our attention to the idea that we have to maximize the agent's total reward over the entirety of the episode, then we naturally arrive at the concept of returns. Simply put, a return is the total sum of rewards the agent will receive over the course of the episode.

$$G_t = r_{t+1} + r_{t+2} + \cdots + r_T$$

Where T is the terminal time step of the episode.

But, this raises a question. What if the episode doesn't end? What if it's a continual task? In that case, the returns grow without limit, and it makes no sense to talk about maximizing something that is going to grow to infinity anyway. So, we introduce the concept of *discounting.* We define a hyperparameter of our agent, denoted by the Greek letter gamma (γ), such that $0 \leq \gamma \leq 1$. Gamma serves as a multiplicative factor of our rewards, and follows a power law.

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

If gamma is less than 1, every successive power of gamma decreases the weight of the reward at that time step. It is thus a mechanism for discounting the amount of value the agent places on the future reward, in the present. As gamma approaches 1, the agent becomes more and more farsighted, placing greater emphasis on future rewards. As gamma approaches 0, the agent becomes more and more myopic, placing greater emphasis on more immediate rewards.

Beyond being needed for ensuring that our sum of rewards stays finite in continual tasks, discounting has a pretty strong basis in first principles. Since our environment dynamics are unknown, and indeed may be stochastic, the certainty of receiving the same sequence of rewards is less than guaranteed. The further out in the future, the less certainty around the sequence of rewards, and so it makes more sense to discount rewards that are further away in time.

This form for our returns has a pretty neat property that we can take advantage of:

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \cdots$$

$$G_t = r_{t+1} + \gamma(r_{t+2} + \gamma r_{t+3} + \gamma^2 r_{t+4} + \cdots)$$

$$G_t = r_{t+1} + \gamma G_{t+1}$$

This is a recursive relationship between returns at successive time steps. It does require that we introduce an additional definition: $G_T = 0$. The return for the terminal state is precisely 0, which makes sense because the episode terminates in the final step. Hence, the agent receives no future rewards and so the terminal state has no value. We will use this shortly, when we derive the final form of the Bellman equation.

3 Policies and Value Functions

While the discounted returns are a handy mathematical apparatus for defining what it is the agent needs to maximize, they aren't expressed in terms of anything the agent can use to predict the future. In other words, merely keeping track of the rewards received doesn't tell the agent anything if it doesn't also associate those rewards with the states it encountered along the way. When the agent associates states with the returns that follow visits to that state, it's able to learn that certain states are more valuable, and hence it can attempt to reach those states again.

The correlation between states and returns is what we call the *value function.* It's a quantitative way to measure the goodness of particular states, such that the agent can learn to access those states in the future. Of course, the states the agent encounters will depend on the actions it takes, so we have to implement a method for choosing actions for the agent. This is accomplished by something called the *policy*, denoted by the greek letter π. It is a function of a given state, and it is a mapping between states and probabilities of selecting each action in the agent's action space. We'll often see it written as $\pi(a|s)$ which reads as the probability of selecting action a, given the state s of the environment. Learning then means modifying the agent's policy over time, in such a way that the probability of selecting actions with large future returns is maximal, while minimizing the probability of selecting unprofitable actions.

The value function is defined as the expectation value of the returns, assuming that the agent follows its policy π. It's given mathematically by the following:

$$v_\pi(s) = E_\pi[G_t|S_t = s] = E_\pi\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| S_t = s\right] \text{ for all } s \in S$$

It's also useful to think in terms of the values of state and action pairs, while following the agent's policy π. This is denoted by $q_\pi(s,a)$ and it is called the *action value function*, which has a very similar form to the value function:

$$q_\pi(s) = E_\pi[G_t|S_t = s, A_t = a] = E_\pi\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| S_t = s, A_t = a\right]$$

If this is a little abstract, remember that expectation values are just sums of probabilities multiplied by outcomes. So in this case, we are taking a look at a given state s and evaluating the policy for that state. It gives us some probability of selecting each action in the agent's action space, and each action causes the environment to transition into some state s', and give the agent a reward in the process. So if we were to calculate this by hand, for a very simple system with known state transitions and rewards, we would put the state into the agent's policy (a function) and get back some probabilities. We would then see what the new state of the system would be for each of those actions, and the reward the agent would receive. Then we multiply the output of the agent's policy function by the reward received in each of those new states, and voila, we have our expectation value.

Now, as I stated way up at the top of this document, we don't usually know those dynamics. But, that's not really a problem. In that case, we can figure out the dynamics by playing the game and keeping track of the rewards received and states encountered. We could keep a running average of the returns that follow our visits to each state, and use that as an estimate of the return. We plug that into our value function and action value function, and boom we've solved the problem. Methods that try to learn the dynamics of the system by interacting with it are called *Monte Carlo methods*, and we will cover those in depth later in this course.

One other thing to consider is that I've made a hidden assumption in this discussion. I've assumed that the states of the environment are all discrete and distinct. Kind of like configurations of a tic tac toe board, or the position of an agent in a maze. At first glance, this seems to be a very very limiting assumption. Most environments, at least the ones that bear a resemblance to the real world, are not discrete. They are in fact *continuous.* But, not to worry. We have the magic of deep neural networks to save the day. We can use deep neural networks to act as function approximators for our policy as well as our value function. We'll deal with this more later when we get to policy gradient and actor critic methods. I just bring it up now so that you're aware of the hidden assumption here, and that you know it's not a problem.

Something else that may have occurred to you is that the value function is expressed in terms of the expectation value of the return for time t. Earlier in the lecture, we discussed the fact that there is a recursive relationship between returns at successive time steps. Let's see what happens if we plug that into the equation for the value function:

$$v_\pi(s) = E_\pi[G_t|S_t = s]$$

$$v_\pi(s) = E_\pi[r_{t+1} + \gamma G_{t+1}|S_t = s]$$

Ok, so we've expressed return at time step t as a function of the reward at time t + 1 and return for time t + 1. What's the big deal? How can we use this? Well, remember that we're calculating an expectation value, and the expectation value is just the probability of something happening multiplied by what we get when that something happens. So in this case, our probabilities are defined by three different quantities, the of selecting each action, the probability of ending up in some new state and the probability of receiving a reward (the very same r_{t+1}) in that state. So there are three different probabilities, and since we have to sum over those, we're going to have three different sums when we replace the symbol for the expectation value by its definition. Let's see what happens when we do that:

$$v_\pi = \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a) \Big[r + \gamma E_\pi[G_{t+1}|S_{t+1} = s'] \Big]$$

So there we have our three sums over the new states s', actions a, and rewards r. We have our policy $\pi(a|s)$ along with the state transition probabilities $p(s', r|s, a)$ and these are all multiplied by the sum of the reward we can expect to receive and the discounted expectation value of the returns that follow time $t+1$ given that we're in state s' at time $t+1$.

If this isn't clear, please remember that expectation values are averages and averages are additive quantities. That is to say that $Average(A+B) = Average(A) + Average(B)$. I exploited that to leave the expectation value for the state s', because it leads to this very elegant expression:

$$v_\pi = \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a) \Big[r + \gamma v_\pi(s')\Big], \text{ for all } s \in S.$$

So what this means is that not only do we have a recursive relationship between the returns at successive time steps, *but also for the value functions.* This equation is central to all of reinforcement learning, and it is called the *Bellman Equation.* Our entire mission is to solve this equation and use it to figure out the absolute best possible policy, or set of best possible policies.

Naturally, there is an analogous equation for the action value function. I'm going to write it down rather than derive it, since the steps are pretty similar and I think you get the idea by now.

$$q_\pi = \sum_{s',r} p(s', r|s, a)[r + \gamma \sum_{a'} \pi(a'|s') q_\pi(s', a')]$$

Here, I have condensed the two sums into one and indicated two indices s', r, but we have many of the same parameters. We sum over new states and rewards and multiply the state transition probabilities by the sum of the reward we receive and the resulting action value functions. We have to sum over all possible actions a' for a state s' because we have one action value function for each action, and the probabilities of getting those action value functions depend on the policy π.

3.1 Optimal Policies

Now that we have a way of evaluating the goodness of states (or state action pairs) for a given policy, we have a mechanism for comparing policies. We simply compare the value function of each policy and see which one gives us the largest value for a given state. If we can iterate this process over all the states in the state space, we arrive at a policy that has the largest possible value function for each state. This means it's the *optimal policy.* In fact, there can be more than one optimal policy, so it's really a set of policies rather than a single one. All the policies in the set of optimal policies are practically equivalent, so if you find one just be happy and consider the problem solved.

The remainder of this course deals with the question of exactly how we find the optimal policy, so I'm not going to address it here. Stay tuned for more details on finding the optimal policy and solving the Bellman equation.

4 Conclusion

So, that was a pretty large amount of ground to cover. The basic idea is that we are dealing with systems with unknown dynamics, or even stochastic dynamics (meaning random), and we therefore have to deal with expectation values. When we incorporate expectation values into the future rewards the agent expects to receive, we arrive at a recursive relationship between the values of successive states. This relationship allows us to do a couple things: compare the values of policies and to hop from one valuable state to another.

By iteratively comparing the values of policies and improving them, we can arrive at what is known as the optimal policy. Once we've done this, we've effectively solved the reinforcement learning problem. Of course, that is much easier said than done.