Reinforcement Learning

Jacob Denson

September 19, 2016

Table Of Contents

Chapter 1

Markov Decision Processes

Reinforcement learning is an attempt to design intelligences that can both learn from and interact with the environment. In its most general form, an agent must select action in a world that yield the greatest reward for the agent. We begin with a basic problem which provides an introduction to the techniques we use in later chapters, and then we move on to the most general specification of the problem.

The main idea of reinforcement learning is the reward hypothesis – that all goals can be thought of as the maximization of the expected value of some reward signal responding to actions that an agent performs. Given this, the main process by which all agents can become intelligent is to find ways to maximize this reward signal. This is a hypothesis which we will touch on throughout the class. Once concern with this view is that we are not really building intelligence through this method, only building machines that attempt to imitate intelligence. We provide endorphisms in the human brain guiding behaviour as a counter example.

AI takes the intentional stance on this issue, treating all systems as having intensions, regardless of what they do. The intention of evolution is to attempt to build animals with the perfect survival skills. Evolution is a random process which does not really think about its intentions, yet we can see it as having the intentions by the result of the action. This abstracts the problem, and is a useful way of thinking about problems.

Chapter 2

Bandit Problems

Imagine you are at a casino, which has a healthy selection of slot machines to play on. You of course, being a strategic game player, want to find the slot machine that maximizes your profit from gambling[1]. You hope that you will end up with a profit, but you will also be satisfied, if this is not possible, to find the machine that results in you losing the least amount of money. From the start, you have no idea how any slot machines work, so the only way to determine its reward is to get results from pulling its lever. This is the one-armed bandit problem. It forms a good problem to begin the attack on reinforcement learning, because it is the simplest type of problem which is mathematically tractable, yet still possesses problems involved in 'real' machine learning tasks.

Mathematically, the problem of the multi armed bandits is given by a set A of actions, and, for each action a, a probability distribution of 'rewards' R_a over the real numbers. The goal is to choose a sequence of actions to maximize our reward, by learning which actions gives us the most reward. In the simplest case which we will begin by addressing, the action space A is discrete.

After a bandit problem has been specified, how do we make a decision on actions we want to take? The key is a policy or decision function, which, given a memory of past actions, decides (possibly randomly) on the next action to take. First, define a history of length k to be an element of $(A \times \mathbf{R})^k$. The decision function is actually a sequence of functions $\pi_1, \pi_2, \dots,$

[1]Casino's do actually adjust the odds of certain slot machines to give certain players the 'feel' of being ahead of the casino, to promote further play.

such that π_k maps histories of length $k-1$ to probability distributions over action space. The policy induces a sequence of actions $A_1, A_2, \dots$ and rewards $X_1, X_2, \dots$, such that if $H_k = [(A_1, X_1), \dots, (A_k, X_k)]$, then

$$A_k \sim \pi_k(H_{k-1}) \qquad (X_k | A_k) \sim R_{A_k}$$

The goal is to choose a policy which maximizes the expected reward

$$\mathbf{E}\left[\sum_{k=1}^{n} X_k\right]$$

for some value $n \in \mathbf{N}$ (the 'finite horizon problem') or for $n = \infty$ (the infinite horizon problem), in which case we try to optimize the order of growth of the expected reward as $n \to \infty$.

For each action a, let

$$\mu_a = \mathbf{E}(X_k | A_k = a) = \int_{\mathbf{R}} dR_a$$

In the finite case, we may define $\mu^* = \max \mu_a$, which is the reward taken for choosing the 'best' action. If a user knew the distributions of each action, he would obviously try and pick the action corresponding to μ^* every time. The real problem is learning the distributions of each action, which still optimizing the reward obtained. We see therefore that this problem possesses the *explore-exploit tradeoff*; if we wish to maximize reward, we must attempt to use the arms which have the highest estimated mean reward. But if we only exploit, then we will never learn the optimum reward, because we need to sample all the arms to obtain good estimates of the mean reward, so we need to 'explore' as well. Good general-use policies must balance exploring and exploiting to achieve maximal rewards.

Another way to visualize the problem is that we do not want to maximize 'good' decisions, but to minimize the number of 'bad' decisions we make. Define the regret function

$$R_n = \mathbf{E}[\sum_{k=1}^{n} (\mu^* - X_k)]$$

We wish to minimize the growth of regret over time. Using the tower rule, defining the random variable

$$T_a(n) = \#\{1 \leqslant k \leqslant n : A_k = a\}$$

we find that

$$\begin{aligned}
\mathbf{E}[\sum_{k=1}^{n}(\mu^* - X_k)] &= n\mu^* - \mathbf{E}[\sum_{k=1}^{n} X_k] \\
&= n\mu^* - \sum_{k=1}^{n} \mathbf{E}[\mathbf{E}[X_k|A_k]] \\
&= n\mu^* - \sum_{a\in A}\sum_{k=1}^{n} \mathbf{P}(A_k = a)\mu_a \\
&= n\mu^* - \sum_{a\in A} \mu_a \mathbf{E}(T_a(n)) \\
&= \sum_{a\in A} \Delta_a \mathbf{E}(T_a(n))
\end{aligned}$$

where $\Delta_a = \mu^* - \mu_a$ is the action gap between a and the optimal action. In a good policy, the regret should have a low order of growth. An achievable order is sublinear.

We begin by attempting to estimate the expected reward of an action. The law of large numbers tells us that, provided we sample an action infinitely many times, the random variable

$$\widehat{\mu_m}(n) = \frac{1}{T_a(n)} \sum_{k=1}^{n} \mathbf{I}\{A_k = a\} X_k$$

converges to μ_m as $n \to \infty$ almost surely. There is a computational trick for keeping track of these estimates without having to store the indicator function. We have

$$\widehat{\mu_m}(n+1) = \frac{T_a(n)\widehat{\mu_m}(n) + \mathbf{I}\{A_{n+1} = a\}X_{n+1}}{T_a(n+1)}$$

and this gives a recurrence relation meaning we need only keep track of $T_a(n)$ and $\widehat{\mu_m}(n)$, for the rest of the data is given to us sequentially. This may seem to solve our problem completely, for our change in regret will then converge to zero once we begin selecting optimum actions. But, if we start selecting optimal actions, then we won't necessarily sample all actions infinitely often, which implies that our estimates will not converge. Thus we must form a balance of acting in the greedy way, choosing an action with the highest sample average, and acting in an exploratory way.

A naive policy, the greedy method, always takes the action with the highest sample average. The problem with this method is, if the reward for some action gets distorted low enough, it may never be taken again, even if the actual expected reward is much higher. This normally occurs if the variance of actions is too high. Given a low-variance reward function, the greedy policy may choose the right action, but no guarentees are set. One other helpful trick may be to set $Q_0(a)$ optimistically, that is, not the default value of zero but some value higher than all rewards for an action. Then all values are at least tried one before they begin to settle down, so some exploration is done. Again, there is not really a way of mathematically guarenteeing the success of this policy. A smarter policy, the ε greedy method, which attempts to fix this issue by guarenteeing convergence at the cost of complete optimality.

Fix some real number ε such that $0 \leqslant \varepsilon \leqslant 1$. The idea of the ε greedy method is to act greedily most of the time, whereas for some proportion of the time related to ϵ, we act randomly, ensuring we sample enough time to get convergence of our sample averages. The ε greedy policy π is defined for each action a by

$$\pi_k(H_{k-1})(a) = \begin{cases} \varepsilon + \frac{1-\varepsilon}{|A|} & : a = \text{argmax}_{a' \in A} \widehat{\mu_{a'}}(k) \\ \frac{1-\varepsilon}{|A|} & : \text{otherwise} \end{cases}$$

No action has non-zero probability, so $\widehat{\mu_a}(k)$ converges to μ_a for each a. This means that, in the limit, the ε-greedy strategy will act optimally $\varepsilon + (1-\varepsilon)/|A|$ percent of the time. If we decrease the value of ε, our policy will act more optimally in the limit, but it will take longer for $\widehat{\mu_a}(k)$ to converge to $\widehat{\mu_a}(k)$. The expected change in regret for π converges in the limit to

$$\frac{\varepsilon}{|A|} \sum_{a \in A} \Delta_a$$

as the length of the experience extends to infinity, and hence we see that $R_n \sim n\varepsilon \left(\sum \Delta_a\right)/|A|$, and we have linear regret.

ε-greedy policies are a simple method that converge to a close to optimal policy. However, if there is one action a with a very low expected value, the regret for π will be strongly affected (in other words, if the average value of the action gap is large). The problem is that the action with low value expectation is selected at equal probability with any other action. The softmax method attempts to fix this. We select an action more probably if it has a higher sample average. A softmax method takes as a parameter a temperature τ, and calculates a policy to be

$$\pi_k(H_{k-1})(a) = \frac{e^{\widehat{\mu_a}(k)/\tau}}{\sum_{a'} e^{\widehat{\mu_{a'}}(k)/\tau}}$$

In the limit, our regret converges to

$$\sum_a \frac{e^{\mu_a/\tau}}{\sum_{a'} e^{\mu_{a'}/\tau}} \Delta_a$$

As τ decreases in value, the disparity between the selection of actions with differing sample average increases, causing the policy to act more greedily. Besides the convergence of the policy, there are no mathematical guarentees on which τ to use that have been developed, which makes it hard to pick which τ is useful. This means unless we can find an optimal value of τ for softmax by experimentation, it is hard to recommend softmax over an ε greedy algorithm do to the more complicated definition of softmax.

A more advanced and recent solution to the problem is to compute confidence intervals for the sample averages collected. Then we act greedily, our policy selecting the action a such that the equation

$$Q_t(a) + c\sqrt{\frac{log(t)}{K_t(a)}}$$

is maximized, where $K_t(a)$ is the upper confidence bound for the sample average of the action a.

A final problem we will discuss with bandit problems is the problem of nonstationarity. Regardless of how long a policy interacts with a bandit, the distributions mapped from $\mathcal{R}(a)$ remain the same. However, what if these values do change according to some rule, that is, if there is instead a sequence of functions $(\mathcal{R}_0, \mathcal{R}_1, \dots)$, which the policy plays with at each interval. If these functions have no relation, there is no chance a policy could learn from them. Given that these chain in some systematic way, there may be a way to carry over experience from one reward kernal to the next. Our policy do not want convergence though, as q_* changes and thus there is no meaningful convergence we would want. Thus out normal sample average Q_t will not perform well. We take inspiration for a new sample average calculator from the equation

$$Q_t = Q_{t'} + \frac{1}{k}[R_k - Q_{t'}]$$

It is of the form

$$\text{‘new average’} = \text{‘old average’} + \text{‘step size’}[\text{‘new value’} - \text{‘old average’}]$$

If we adjust the step size to a non-zeroconstant value α, our sample average will not settle down to some value, but will still have some convergence to the mean. The full formula will then become

$$Q_{t+1} = (1-\alpha)^t Q_0 + \sum_{m=1}^{n} (1-\alpha)^{t-i} R_{k_m}$$

We call this the exponential recency weighted average. Things in the past become exponentially less inportant than things in the present. This allows us to get some convergence while not settling down completely.

We leave off this chapter with a complication in the problem. Suppose you are playing a one-armed bandit that changes $\mathcal{R}$ as in the non-stationary problem, but that we can identify which positions use the same $\mathcal{R}$ by some characteristic, say, the slot machine has the same colour when you pull it. We could then attempt to predict each individual colour's sample average, and from each colour learn what the best action would be. An additional challenge would be the added complexity in that which slot you pull may determine the next colour you see. Then the best action you would take may be impacted by which colour you see next. We will see that these complications form the bulk of the full ‘reinforcement learning’ problem.

Chapter 3

Reinforcement Learning: A Short Introduction to Markov Decision Processes

The next step to adding to the learning process is the introduction of state, the 'colours' of the bandit problem discussed at the end of the last chapter represent what we mean by the word state, a summary of experiences relevant enough to predict the present to the best of our abilities. Actions thus have further consequences than immediate reward. We call this associate learning problem the reinforcement learning problem.

The reinforcement learning task consists of an agent/learner/decision-maker interacting with its environment, everything that is outside of the agents direct control. The two interact continually through the environments presentation of rewards to the agents actions in the environment, providing 'reinforcement' that allows the agent to improve its behaviour over time, the agents primary motive. Over time, the environment changes, but the environment should give enough information at the current time or via previous interactions to allow an agent to form a state representation of the agents current standing with the markov property - the presense of enough relavant information to make intelligent decisions based on previous information about the same state. We define the reinforcement learning task rigorously through the introduction of a markov decision problem, or MDP.

A Markov Decision Problem, or MDP, is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P} \rangle$, where S is a non-empty set of states, A is a non-empty set of actions, R is a mapping from an initial state, an action, and a resultant state, $S \times A \times S$, to a distribution over the real numbers called the reward distribution (think of the reward distribution from the bandit problem), and P is a mapping from $S \times A$ to a probability distribution over states, called the transition probability kernel. We write

$$R(\cdot|s,a) \qquad\qquad P(\cdot|s,a)$$

for the reward and probability distributions mapped from state s and action a. Each state s has associated with it a subset of A called the admissible actions at state s. We write this as $\mathcal{A}_s$. The set of states, actions, and rewards are assumed to be countable in this course, though it should be said that this need not always be assumed. An experience in a markov decision process is a sequence with elements in S, A, and R, in that order, normally denoted

$$(S_0, A_0, R_1, S_1, A_1, R_2, \dots, S_n, A_n, R_{n+1})$$

We write R_{t+1} instead of R_t in the sequence to represent the fact that R_t is created leading into the next time period. A policy π is then a mapping from experience to a probability distribution over the admissible actions in the most recent state seen. The aim of a policy is to maximise its return, the expectation of reward the policy may see. We denote the expected reward after t steps by $\mathbf{E}_\pi[G_t]$. We define the expectation by the following equations based on the initial state S_0:

$$\mathbf{E}_\pi[G_t] = \sum_{k=0}^{\infty} \gamma^k R_k$$

$$R_0(S_0, A_0, R_1, \dots, S_n) = \sum_{a,s,r} \pi(a|S_0, A_0, \dots, S_n) P(s|S_n, a) \mathcal{R}(r|S_n, a, s) r$$

$$R_t(S_0, \dots, S_n) = \sum_{a,s,r} \pi(a|S_0, A_0, \dots, S_n) P(s|S_n, a) R_{t-1}(S_0, \dots, S_n, a, r, s)$$

Intuitively, given some initial state S_0, the policy picks an action, gets a reward from it, and moves to a new state. Given the new information, it picks an action, gets a reward, and the process continues. R_i then becomes the expected reward from the t'th action given we are using policy π. The optimal function π_* maximises this value.

Before we end this short chapter