Reinforcement Learning: An Introduction

Rishabh Sambare

May 6, 2021

Contents

Chapter 1

Introduction

Reinforcement learning is learning an action to maximize numeric reward. The exploitation vs exploration problem is equivalent to asking when you should stick to your guns, or when you should take a risk.

RL agents are assumed to be capable of interaction with their environments. SOme RL methods can learn with approximators which can address the curse of dimensionality on supervised learning methods.

The general elements of an RL system are as follows:

- An agent
- An environment
- A policy
- A reward signal
- A value function
- (optional) An internal environment model

A policy defines an agent's way of behaving at a certain time. Policies are functions from an environment perception to actions to be taken. Policies may be stochastic.

A reward signal defines the goal for a reinforcement learning problem. On each time step, the environment provides (the notion of the environment providing is a convention, this signal could be considered internal to the agent, but not editable by the agent) the agent with a reward. The agent attempts to maximize this reward over the long run. Reward functions may be stochastic.

The value function defines how useful a state is in the long-term. A state may net a high reward signal, but force subsequent states to yield low reward, or vice versa.

Models of the environment are the primary means by which a reinforcement learning agent can plan. Model-free methods are pure trial-and-error, while

model-based methods use model to try and infer the resultant environmental states of certain actions/policies.

I don't know why I added this next paragraph... Sutton says it's not relevant but includes it anyway. Evolutionary reinforcement learning methods apply multiple 'static' policies that interact with separate environment instances. The policies with the highest reward, and some random variations (exploration), are carried over to the 'next generation' of policies. This cycle repeats with some dynamic programming algorithm.

Chapter 2

Multi-armed Bandits

Reinforcement learning uses training information that evaluates actions; it does not and cannot instruct the correct action. The evaluative feedback of an action then obviously will depend on the action itself, while instructive feedback is independent of the action an agent it takes; it just provides the right one.

2.1 A k-armed Bandit Problem

We define the k-armed bandit problem to be when an agent is faced repeatedly with a choice among k options. After the choice, the agent receives a reward from a probability distribution unique to the chosen action. The goal is to maximize the expected total reward over t time steps. We can model this expected reward as $q_*(a)$, where

$$q_*(a) = \mathbb{E}[R_t \mid A_t = a].$$

We do not know q_* exactly for any action a. We can make estimates $Q_t(a)$, where t is the time step of that particular estimate. Taking an action a for which $Q_t(a)$ is the maximum estimated expected reward is a 'greedy' action exploitation. Selecting a 'non-greedy' estimate is an exploration action, as now you can improve your estimate of the non-greedy action and 'explore' your action values.

2.2 Action-value Methods

These methods refer to the class of methods by which we can estimate the value of an action and how to select an action. The sample-average method to estimate the expected reward of an action is to take the mean of all previous rewards from the action.

$$Q_t(a) = \frac{\sum_{i=1}^{t-1}(R_i)(1_{A_i=a})}{\sum_{i=1}^{t-1} 1_{A_i=a}}$$

where $1_{A_i=a}$ is the predicate function for when the action $A_i = a$. We can see that as $t \to \infty$, $Q_t(a) \to q_*(a)$.

Greedy action selection is written as

$$A_t = \arg\max_a Q_t(a).$$

We define ε-greedy action selection methods to be methods that randomly choose non-greedy actions with probability ε, then choosing between greedy actions randomly $1 - \varepsilon$ of the time.

2.3 The 10-armed Testbed

An assessment of greedy and ε-greedy action selections was done on 2000 randomly generated 10-armed bandit problems. The $q_(a)$ value for $a = 1, 2, 3...10$ was sampled from a normal distribution with variance 1 and mean 0. When an agent picked A_t at time t, the reward R_t was selected from a normal distribution with mean $q_(A_t)$ and variance 1. Basically, the greedy version sucked because it never was greedy on the right action. The 0.01-greedy action was alright initially, but did the best later as it would be non-greedy 1% of the time, so once it found the optimal action it stuck with it longer than the 0.1-greedy action, which, despite finding the optimal action sooner, didn't stick with it as much of the time.

We efficiently compute sample averages for an action's estimated reward through a recursive formula that only requires $O(1)$ computations at each update. LET Q_n denote the n-th estimate of an action's reward. Then

$$Q_{n+1} = \frac{1}{n}\sum_{i=1}^{n} R_i = Q_n + (1/n)[R_n - Q_n]$$

A general computation in reinforcement learning algorithms is similar

$$newEstimate = oldEstimate + stepSize[Target - Old]$$

where the newest sample, though possibly noisy, is considered the 'target' value.

A simple bandit algorithm half-converted to Python:

```
import random
def simple_bandit(k, eps):
    Q = []
    N = []
    for a in range(k):
        Q[a] = 0
        N[a] = 0
    while True:
        greedy = findIndex(max(Q))
        rand = random.uniform(0,1)
        if rand <= eps:
            A = # random non-max member of Q
        else:
            A = Q[greedy]

        R = bandit(A) # samples a reward from action's probability distribution
        N[A] = N[A] + 1
        Q[A] = Q[A] + 1/N[A](R-Q[A])
```

Nonstationary or stochastic environments basically eliminate the feasibility of greedy methods since the rewards are constantly changing, so enter...

2.4 Tracking a Non-stationary Problem

In non-stationary environments, it makes more sense to prefer newer rewards to older ones. Modifying the incremental update rules:

$$Q_{n+1} = Q_n + \alpha(R_n - Q_n)$$

where $\alpha \in [0, 1]$ is constant.

$$Q_{n+1} = Q_n + \alpha(R_n - Q_n)$$

$$Q_{n+1} = (1 - \alpha)Q_n + \alpha R_n$$

$$Q_{n+1} = \alpha R_n + (1 - \alpha)[\alpha R_{n-1} + (1 - \alpha)Q_{n-1}]$$

This next step kinda big, if you're invested you should probably just look up the section in the book,

$$Q_{n+1} = (1 - \alpha)^n Q_1 + \alpha \sum_{i=1}^{n} (1 - \alpha)^{n-i} R_i$$

The first term is the original estimate of Q. This estimate matters less as $n \to \infty$. The second term with the summation sums all rewards with each time step lowering a reward's weight on the new estimate by a multiplicative factor $1 - \alpha$. The weights overall sum to 1, use a geometric series on the second term if you wanna check. This is called an exponential-recency weighted average.

For stationary environments, converge on true action values require your step-size parameter sequences to meet the following conditions

$$(1) \quad \sum_{n=1}^{\infty} \alpha_n(a) = \infty$$

$$(2) \quad \sum_{n=1}^{\infty} \alpha_n^2(a) < \infty$$

This is a result of Blum and is actually a kind of interesting and accessible to an extent paper. I recommend looking at it for a bit if you find the two conditions kind of arbitrary like I did. Just black box the weird measure theory parts if you don't know any, it still kinda reads well. Intuitively, the first condition guarantees that fluctuations do not affect the final expected value (if the sum of the step parameters converge on a real number, then fluctuating reward values will make up a non-trivial percentage of the final expected value approximation). The second step guarantees that the steps become small enough such that the sequence of estimates converges.

2.5 Optimistic Initial Values

All current methods discussed are biased on initial estimates

- For sample-average, bias disappears after one sample.
- For constant-α methods, the initial bias is permanent though its weight converges to 0 over repeated sampling.

Initial values can be set to initial expectations of the variable, or you can use them to explore.

> Suppose you have all actions with mean 0 and variance 1. If you set the initial estimate of the rewards to be +5, and the learning agent repeatedly gets less, this lowers the overall estimate of the reward, and the learner tries another random action with estimated reward +5. This process repeats until the initial +5 estimate's weighting on the estimated reward becomes negligible. This leads to lots of initial exploration despite the agent choosing greedily. Over time, the values converge on their actual estimates (insofar as the sequence of step size parameters fulfill the aforementioned properties).

This technique is called 'optimistic initial values'. Obviously, this is ineffective for non-stationary problems since the exploration happens temporarily at the onset of the algorithm, and will not renew with the reward values.

2.6 Upper Confidence Bound Action Selection

ε-greedy actions select exploratory actions randomly from non-greedy actions. Some notion of 'potential' should exist to denote that some actions are more

probably optimal than others. Sutton introduces the following in his book,

$$A_t = \arg\max_a [Q_t(a) + c\sqrt{\frac{\ln(t)}{N_t(a)}}]$$

$c > 0$ is denoted the 'degree of exploration' as if c is larger, then the term with the least number of attempts has larger value (notice that smaller values for $N_t(a)$ increase the overall value).

U.C.B works well for stationary environments, but is hard for non-stationary ones since each action's reward is, in a sense, imprinted permanently by its past 'viability'.

2.7 A Gradient Bandit Algorithm

You can consider action-selection algorithms that use some point notion of preference $H_t(a)$ to rank each action. Use a soft-max to decide an action

$$\Pr A_t = a = \frac{e^{H_t(a)}}{\sum_{b=1}^{k} e^{H_t(b)}} = \pi_t(a)$$

Action preferences are decided by stochastic gradient ascent.

$$(1) \quad H_{t+1}(A_t) = H_t(A_t) + \alpha(R_t - \hat{R}_t)(1 - \pi_t(A_t))$$

$$(2) \quad H_{t+1}(a) = H_t(a) - \alpha(R_t - \hat{R}_t)(\pi_t(a) \quad \text{for all } a \neq A_t$$

Here $\alpha > 0$ is the step-size parameter. Notice in (1) that, if the reward is higher, exploitation is incentivized. Notice that (1) is small when π_t is high, detracting from exploitation if one action is clearly better than all the others. In (2), the more an action is not selected, the less it will be selected in the future, as the $\pi_t(a)$ term denotes.

Chapter 3

Finite Markov Decision Processes

3.1 Agent-Environment Interface

At a time step t, the agent receives S_t and selects A_t. At time step $t+1$, the agenct receives R_{t+1} and S_{t+1}. This continuation for an indefinite number of time steps is called a trajectory.

$$S_0, A_0, R_1, S_2, A_1, R_2, S_2, A_2, \cdots .$$

A finite MDP has finite sets $\mathcal{S}, \mathcal{A}, \mathcal{R}$ for the possible states, actions, and rewards respectively. For a finite MDP, the reward and state distributions are discrete, dependent only on the preceding state and action.

$$p(s', r \mid s, a) = Pr\{S_t = s, R_t = r \mid S_{t-1} = s, A_{t-1} = a\}.$$

This is just notational expansion. p defines the *dynamics* of the MDP.

A state-transition probability is

$$\begin{aligned} p(s' \mid s, a) &= Pr\{S_t = s \mid S_{t-1} = s, A_{t-1} = a\}. \\ &= \sum_{r \in \mathcal{R}} p(s', r \mid s, a). \end{aligned}$$

The expected reward for a state-action pair could be

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

which if you notice is just the definition of an expected value from a marginal probability distribution (given certain parameters, of course).

The expected rewards of a state-action-state triplet is

$$r(s, a, s') = \mathbb{E}[R_t \mid S_{t-1} = s, A_{t-1} = a, S_t = s'] \sum_{r \in \mathcal{R}} r \frac{p(s', r \mid s, a)}{p(s' \mid s, a)}.$$

This is the probability of a reward r given that a state s' arises from performing action a in state s (remember that all of these variables are likely stochastic at some level).

3.2 Goals and Rewards

Reward signals should strictly be what you wanted achieved, not how you want it achieved (unless your method is provably the best, in some game-theoretic sense?).

3.3 Returns and Episodes

We seek to maximize the expected reward of some function of the reward sequence. If the agent-environment interaction breaks naturally (whatever this means, I think it usually just implies repetition) into sub-sequences, they are called *episodes*.

An epsiode ends in a terminal state $\mathcal{S}^+$, being reset to a sampled starting state. The time of a terminal state is a random variable (presumably dependent on previous s, a pairs?).

If the agent-environment interaction does not break naturally, this is a *continuous* task (i.e. $T = \infty$ for the terminal step). Expected rewards probably have some sort of mechanism to make their infinite sum converge, usually a discounting function with $\gamma \in [0, 1]$. If $\gamma = 0$, the agent is *myopic* and cares only for the immediate reward (I'm myopic :( ).

3.4 Unified Notation for Episodic and continuing Tasks

An episodic task can be considered a continuous task where all states S_t for $t \geq T$ have reward 0 and map back to themselves for all actions.

3.5 Policies and Value Functions

A policy π is a mapping from states to *probabilities* of selecting actions. A value function of a state s under policy π is

$$v_\pi = \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma R_{t+k+1} \mid S_t = s].$$

where $\mathbb{E}_\pi$ is the expected value of a random variable given that the agent is following policy π. Any value function will map terminal states to 0. v_π is the state-value function for π.

An action-value function under policy π is slightly different.

$$v_\pi(s, a) = \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] = \mathbb{E}_\pi[\sum_{n=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a].$$

The key difference is that the action-value function assumes the action a was chosen, which the policy doesn't imply in a state-value function since the policy may be stochastic.

Both q_π and v_π can be estimated from experience. If an agent follows π and maintains an average of the associated rewards for each state (or each action in each state), then repeated sampling can converge on true action-value functions or state-value functions. Such repeated sampling methods are called *Monte Carlo* fmethods. This is impractical for any sufficiently large value of $|\mathcal{S}|$.

If not, the agent maintains v_π and q_π as parametrized functions (with fewer parameters than $|\mathcal{S}|$) and adjusts the parameters to approximate expected returns.

Value functions are recursive,

$$\begin{aligned} v_\pi &= \mathbb{E}_\pi[G_t \mid S_t = s] \\ &= \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_\pi(s')] \end{aligned} \tag{3.1}$$

This last expression is called the *Bellman Equation* for the state-value function. Notice that it is an expectation over all actions, and also all rewards from all possible new states given an action. The state is supplied to the value function and so is a given to the policy function and joint probability of s', r.

The Bellman Equation for the action-value function is given as an exercise in this book, so I may have gotten it wrong.

$$q_\pi = \sum_{r,s'} p(s', r \mid s, a) \sum_{a'} \pi(a' \mid s')[r + \gamma q_\pi(s', a') \tag{3.2}$$

3.6 Optimal Policies and Value Functions

We can define a partial ordering on policies as $\pi \geq \pi'$ iff $v_\pi(s) \geq v_{\pi'}(s)$ for all $s \in \mathcal{S}$

(Why not use a metric on v_π that weights $v_\pi(s)$ by the probability of a state initially occurring? Like this

$$M_{v_\pi} = \sum_{s \in \mathcal{S}} p(S_0 = s) v_\pi(s).$$

and order based on $\pi \geq \pi'$ iff $M_{v_\pi} \geq M_{v_{\pi'}}$?)

In any case, the optimal policy π_* is the policy with

$$v_*(s) = \arg\max_\pi v_\pi(s).$$

for all $s \in \mathcal{S}$, and similarly the policy with

$$q_*(s, a) = \arg\max_\pi q_\pi(s, a).$$

for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$. Note that a here isn't necessarily the optimal a for s; it is a predefined choice. Since $q_*(s,a)$ is optimal, though, it is

$$q_*(s,a) = \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a].$$

Since v_* follows the best action in any specific situation, we do not need to sum over all probabilities of taking an action in a state since we always take the best one. So,

$$\begin{aligned} v_*(s) &= \arg\max_{a \in \mathcal{A}(s)} q_{\pi_*}(s,a) \\ &= \arg\max_{a \in \mathcal{A}(s)} \mathbb{E}_{\pi_*}[G_t \mid S_t = s, A_t = a] \\ &= \arg\max_{a \in \mathcal{A}(s)} \mathbb{E}_{\pi_*}[R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t = a] \\ &= \arg\max_{a \in \mathcal{A}(s)} \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_*(s'). \end{aligned} \tag{3.3}$$

This is the Bellman Optimality Equation for v_*. For q_*, we have

$$\begin{aligned} q_\pi(s,a) &= \mathbb{E}[R_{t+1} + \gamma \arg\max_{a'} q_*(S_{t+1}, a') \mid S_t = s, A_t = a] \\ &= \sum_{s',r} p(s', r \mid s, a)[r + \gamma \arg\max_{a'} q_*(s', a')]. \end{aligned} \tag{3.4}$$

Here the agent is supplied possibly non-optimal action a, but takes the optimal action a' afterward.

The Bellman optimality equations have unique solutions since they are a system of n equations in n variables (unclear). (Is this because we have to find the max a' for each s' in q, and find the max s' for each a' in v? Unclear on what the difference really is. I guess these two things are very related anyway so this is probably a semantic difference).

The q_* function is nicer than v_*. With v_* if you want to find the best action to take in state s, you have to take each action a and compute the (expected?) value of the resulting state $v_*(s)$. The best action is the one which maximizes this quantity. For q_*, you just run q_* on every action $a \in \mathcal{A}(s)$ and take the one which yields the maximum value.

3.7 Optimality and Approximation

Optimality is hard and computationally infeasible for practical problems if you want to solve the linear system of equations exactly. Instead, we can use approximate tabular methods (but even those fail after a certain $|\mathcal{S}|$) or function approximation.