Notes Reinforcement Learning: An Introduction

Contents

1 Introduction

Reinforcement learning is about how an agent can learn to interact with its environment. Reinforcement learning uses the formal framework of Markov decision processes to define the interaction between a learning agent and its environment in terms of states, actions, and rewards.

1.3 Elements of Reinforcement Learning

Policy defines the way that an agent acts, it is a mapping from perceived states of the world to actions. It may be stochastic.

Reward defines the goal of the problem. A number given to the agent as a (possibly stochastic) function of the state of the environment and the action taken.

Value function specifies what is good in the long run, essentially to maximise the expected reward. The central role of value estimation is arguably the most important thing that has been learned about reinforcement learning over the last six decades.

Model mimics the environment to facilitate planning. Not all reinforcement learning algorithms have a model (if they don't then they can't plan, i.e. must use trial and error, and are called model free).

2 Multi-armed Bandits

Reinforcement learning involves evaluative feedback rather than instructive feedback. We get told whether our actions are good ones or not, rather than what the single best action to take is. This is a key distinction between reinforcement learning and supervised learning.

2.1 A k-armed Bandit Problem

In the k-armed bandit problem there are k possible actions, each of which yields a numerical reward drawn from a stationary probability distribution for that action. We want to maximise the expected total reward, taking an action at each *time step*. Some notation:

- Index timesteps by t
- Action A_t
- Corresponding reward R_t
- *Value* of action a is $q_*(a) = \mathbb{E}[R_t|A_t = a]$
- Estimate of value of action a at t is denoted $Q_t(a)$

We therefore want to choose $\{a_1, \ldots, a_T\}$ to maximise $\sum_{t=1}^{T} q_*(a_t)$.

At each timestep, the actions with the highest estimated reward are called the *greedy* actions. If we take this action, we say that we are *exploiting* our understanding of the values of actions. The other actions are known as *non-greedy* actions, sometimes we might want to take one of these to improve our estimate of their value. This is called *exploration*. The balance between exploration and exploitation is a key concept in reinforcement learning.

2.2 Action-value Methods

We may like to form estimates of the values of possible actions and then choose actions according to these estimates. Methods such as this are known as *action-value methods*. There are, of course, many ways of generating the estimates $Q_t(a)$.

An ε-greedy method is one in which with probability ε we take a random draw from all of the actions (choosing each action with equal probability), providing some exploration.

2.5 Tracking a Non-stationary Problem

If we decide to implement the sample average method, then at each iteration that we choose the given action we update our estimate by

$$Q_{n+1} = Q_n + \frac{1}{n}[R_n - Q_n] \tag{1}$$

Note that this has the (soon to be familiar) form

$$\text{NewEstimate} \leftarrow \text{OldEstimate} + \text{StepSize} \times [\text{Target} - \text{OldEstimate}]. \tag{2}$$

If the problem was non-stationary, we might like to use a time weighted exponential average for our estimates (*exponential recency-weighted average*). This corresponds to a constant step-size $\alpha \in (0, 1]$ (you can check).

$$Q_{n+1} = Q_n + \alpha[R_n - Q_n]. \tag{3}$$

We might like to vary the step-size parameter. Write $\alpha_n(a)$ for the step-size after the n^{th} reward from action a. Of course, not all choices of $\alpha_n(a)$ will give convergent estimates of the values of a. To converge with probability 1 we must have

$$\sum_n \alpha_n(a) = \infty \qquad \text{and} \qquad \sum_n \alpha_n(a)^2 < \infty. \tag{4}$$

Meaning that the coefficients must be large enough to recover from initial fluctuations, but not so large that they don't converge in the long run. Although these conditions are used in theoretical work, they are seldom used in empirical work or applications. (Most reinforcement learning problems have non-stationary rewards, in which case convergence is undesirable.)

2.6 Optimistic Initial Values

The exponential recency weighted method is biased by the initial value one gives. If we like, we may set initial value estimates artificially high to encourage exploration in the short run – this is called *optimistic initial values*. This is a useful trick for stationary problems, but does not apply so well to non-stationary problems as the added exploration is only temporary.

2.7 Upper-Confidence Bound Action Selection

We might like to discriminate between potential explorative actions. Note that ε-greedy does not do this. We define the *upper-confidence bound* action at t as follows

$$A_t \doteq \operatorname*{argmax}_a \left[Q_t(a) + c\sqrt{\frac{\ln(t)}{N_t(a)}} \right] \tag{5}$$

where $Q_t(a)$ is the value estimate for the action a at time t, $c > 0$ is a parameter that controls the degree of exploration and $N_t(a)$ is the number of times that a has been selected by time t. If $N_t(a) = 0$ then we consider a a maximal action.

This approach favours actions with a higher estimated rewards but also favours actions with uncertain estimates (more precisely, actions that have been chosen few times).

2.8 Gradient Bandit Algorithms

Suppose that we choose actions probabilistically based on a preference for each action, $H_t(a)$. Let the action at t be denoted by A_t. We then define the probability of choosing action a via the softmax

$$\pi_t(a) \doteq \mathbb{P}(A_t = a) = \frac{e^{H_t(a)}}{\sum_i e^{H_t(i)}}. \tag{6}$$

We then iteratively perform updates according to

$$H_{t+1}(a) = H_t(a) + (R_t - \bar{R}_t)(\mathbb{1}_{A_t=a} - \pi_t(a)), \tag{7}$$

where $\bar{R}_t$ is the mean of previous rewards. The box in the notes shows that this is an instance of stochastic gradient ascent since the expected value of the update is equal to the update when doing gradient ascent on the (total) expected reward.

3 Finite Markov Decision Processes

We say that a system has the *Markov property* if each state includes all information about the previous states and actions that makes a difference to the future.

The MDP provides an abstraction of the problem of goal-directed learning from interaction by modelling the whole thing as three signals: action, state, reward.

Together, the MDP and agent give rise to the *trajectory* $S_0,\ A_0,\ R_1,\ S_1,\ A_1,\ S_2,\ R_2,\ \ldots$. The action choice in a state gives rise (stochastically) to a state and corresponding reward.

3.1 The Agent–Environment Interface

We consider finite Markov Decision Processes (MDPs). The word finite refers to the fact that the states, rewards and actions form a finite set. This framework is useful for many reinforcement learning problems.

We call the learner or decision making component of a system the *agent*. Everything else is the *environment*. General rule is that anything that the agent does not have absolute control over forms part of the environment. For a robot the environment would include it's physical machinery. The boundary is the limit of absolute control of the agent, not of its knowledge.

The MDP formulation is as follows. Index time-steps by $t \in \mathbb{N}$. Then actions, rewards, states at t represented by $A_t \in \mathcal{A}(s)$, $R_t \in \mathcal{R} \subset \mathbb{R}$, $S_t \in \mathcal{S}$. Note that the set of available actions is dependent on the current state.

A key quantity in an MDP is the following function, which defines the *dynamics* of the system.

$$p(s', r|s, a) \doteq \mathbb{P}(S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a) \tag{8}$$

From this quantity we can get other useful functions. In particular we have the following:

state-transition probabilities

$$p(s'|s, a) \doteq \mathbb{P}(S_t = s'|S_{t-1} = s, A_{t-1} = A) = \sum_{r\in\mathcal{R}} p(s', r|s, a) \tag{9}$$

note the abuse of notation using p again; and,

expected reward

$$r(s, a) = \mathbb{E}[R_t|S_{t-1} = s, A_{t-1} = a] = \sum_{r\in\mathcal{R}} r \sum_{s'\in\mathcal{S}} p(s', r|s, a). \tag{10}$$

3.2 Goals and rewards

We have the *reward hypothesis*, which is a central assumption in reinforcement learning:

> All of what we mean by goals and purposes can be well thought of as the maximisation of the expected value of the cumulative sum of a received scalar signal (called reward).

3.3 Returns and Episodes

Denote the sequence of rewards from time t as R_{t+1}, R_{t+2}, R_{t+3}, ... We seek to maximise the *expected return* G_t which is some function of the rewards. The simplest case is where $G_t = \sum_{\tau>t} R_\tau$.

In some applications there is a natural final time-step which we denote T. The final time-step corresponds to a *terminal state* that breaks the agent-environment interaction into subsequences called *episodes*. Each episode ends in the same terminal state, possibly with a different reward. Each starts independently of the last, with some distribution of starting states. We denote the set of states including the terminal state as $\mathcal{S}^+$

Sequences of interaction without a terminal state are called *continuing tasks*.

We define G_t using the notion of *discounting*, incorporating the *discount rate* $0 \leq \gamma \leq 1$. In this approach the agent chooses A_t to maximise

$$G_t \doteq \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}. \tag{11}$$

This sum converges wherever the sequence R_t is bounded. If $\gamma = 0$ the agent is said to be myopic. We define $G_T = 0$. Note that

$$G_t = R_{t+1} + \gamma G_{t+1}. \tag{12}$$

Note that in the case of finite time steps or an episodic problem, then the return for each episode is just the sum (or whatever function) of the returns in that episode.

3.4 Unified Notation for Episodic and Continuing Tasks

We want to unify the notation for episodic and continuing learning.

We introduce the concept of an *absorbing state*. This state transitions only to itself and gives reward of zero.

To incorporate the (disjoint) possibilites that $T = \infty$ or $\gamma = 1$ in our formulation of the return, we might like to write

$$G_t \doteq \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k. \tag{13}$$

3.5 Policies & Value Functions

Policy

A *policy* $\pi(a|s)$ is a mapping from states to the probability of selecting actions in that state. If an agent is following policy π and at time t is in state S_t, then the probability of taking action A_t is $\pi(a|s)$. Reinforcement learning is about altering the policy from experience.

Value Functions

As we have seen, a central notion is the value of a state. The *state-value function* of state s under policy π is the expected return starting in s and following π thereafter. For MDPs this is

$$v_\pi \doteq \mathbb{E}_\pi[G_t | S_t = s], \tag{14}$$

where the subscript π denotes that this is an expectation taken conditional on the agent following policy π.

Similarly, we define the *action-value function* for policy π to be the expected return from taking action a in state s and following π thereafter

$$q_\pi(s,a) \doteq \mathbb{E}_\pi[G_t|S_t = s, A_t = a]. \tag{15}$$

The value functions v_π and q_π can be estimated from experience.

Bellman Equation

The Bellman equations express the value of a state in terms of the value of its successor states. They are a consistency condition on the value of states.

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[G_t|S_t = s] && (16)\\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s] && (17)\\ &= \sum_{a\in\mathcal{A}(s)} \pi(a|s) \sum_{s',r} p(s',r|s,a)\left[r + \gamma \mathbb{E}_\pi[G_{t+1}|S_{t+1} = s']\right] && (18)\\ &= \sum_{a\in\mathcal{A}(s)} \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')] && (19) \end{aligned}$$

The value function v_π is the unique solution to its Bellman equation.

3.6 Optimal Policies & Optimal Value Functions

We say that $\pi \geq \pi'$ iff $v_\pi(s) \geq v_{\pi'}(s) \quad \forall s \in \mathcal{S}$. The policies that are optimal in this sense are called optimal policies. There may be multiple optimal policies. We denote all of them by π_*.

The optimal policies share the same optimal value function $v_*(s)$

$$v_*(s) \doteq \max_\pi v_\pi(s) \quad \forall s \in \mathcal{S}. \tag{20}$$

They also share the same optimal action-value function $q_*(s,a)$

$$q_*(s,a) = \max_\pi q_\pi(s,a) \quad \forall s \in \mathcal{S}, a \in \mathcal{A}(s), \tag{21}$$

this is the expected return from taking action a in state s and thereafter following the optimal policy.

$$q_*(s,a) = \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1})|S_t = s, A_t = a]. \tag{22}$$

Since v_* is a value function, it must satisfy a Bellman equation (since it is simply a consistency condition). However, v_* corresponds to a policy that always selects the maximal action. Hence

$$v_*(s) = \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma v_*(s')]. \tag{23}$$

Similarly,

$$\begin{aligned} q_*(s,a) &= \mathbb{E}[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a')|S_t = s, A_t = a] && (24)\\ &= \sum_{s',r} p(s',r|s,a)[r + \gamma \max_{a'} q_*(s',a')]. && (25) \end{aligned}$$

Note that once one identifies an optimal value function v_*, then it is simple to find an optimal policy. All that is needed is for the policy to act greedily with respect to v_*. Since v_* encodes all information on future rewards, we can act greedily and still make the long term optimal decision (according to our definition of returns).

Having q_* is even better since we don't need to check $v_*(s')$ in the succeeding states s', we just find $a_* = \text{argmax}_a \, q_*(s, a)$ when in state s.

4 Dynamic Programming

The term Dynamic Programming (DP) refers to a collection of algorithms that can be used to compute optimal policies given perfect model of the environment as a Markov Decision Process (MDP). DP methods tend to be computationally expensive and we often don't have a perfect model of the environment, so they aren't used in practice. However, they provide useful theoretical basis for the rest of reinforcement learning.

Unless stated otherwise, will assume that the environment is a finite MDP. If the state or action space is continuous, then we will generally discretise it and apply finite MDP methods to the approximated problem.

The key idea of DP, and of reinforcement learning generally, is the use of value functions to organize and structure the search for good policies. We use DP and the Bellman equations to find optimal value functions.

4.1 Policy Evaluation (Prediction)

We can use the Bellman equation for the state-value function v_π to construct an iterative updating procedure.

Iterative Policy Evaluation

Consider a sequence of approximate value functions $v_0, v_1, v_2, \ldots$ each mapping $\mathcal{S}^+$ to $\mathbb{R}$. The initial approximation, v_0, is chosen arbitrarily (except that the terminal state, if any, must be given value 0), and each successive approximation is obtained by using the Bellman equation for v_π as an update rule:

$$v_{k+1} \doteq \mathbb{E}_\pi[R_{t+1} + \gamma v_k(S_{t+1}) | S_t = s] \tag{26}$$

$$= \sum_a \pi(s|a) \sum_{s',r} p(s', r|s, a) \left[r + \gamma v_k(s') \right] \tag{27}$$

Clearly, $v_k = v_\pi$ is a fixed point. The sequence $\{v_k\}$ can be shown in general to converge to v_π as $k \to \infty$ under the same conditions that guarantee the existence of v_π. This algorithm is called *iterative policy evaluation*. This update rule is an instance of an *expected update* because it performs the updates by taking an expectation over all possible next states rather than by taking a sample next state.

4.2 Policy Improvement

Policy Improvement Theorem

Let π, π' be any pair of deterministic policies, such that

$$q_\pi(s, \pi(s)) \geq v_\pi(s) \quad \forall s \in \mathcal{S}. \tag{28}$$

That is, π' is as least as good as π. Then we have (shown below)

$$v_{\pi'}(s) \geq v_\pi(s) \quad \forall s \in \mathcal{S} \tag{29}$$

so π' gives at least as good (expected) return as π.

The argument below also shows that if $q_\pi(s, \pi(s)) > v_\pi(s)$ at any s, then there is at least one s for which $v_{\pi'}(s) > v_\pi(s)$.

proof:

$$\begin{aligned}
v_\pi(s) &\leq q_\pi(s, \pi'(s)) \\
&= \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = \pi'(s)] \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s] \\
&\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots | S_t = s] \\
&= v_{\pi'}(s)
\end{aligned}$$

Policy Improvement Algorithm

Now consider a policy that is greedy with respect to $q_\pi(s, a)$. Define

$$\pi'(s) = \operatorname*{argmax}_a q_\pi(s, a) \tag{30}$$

$$= \operatorname*{argmax}_a \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = a] \tag{31}$$

$$\operatorname*{argmax}_a \sum_{s', r} p(s', r | s, a)[r + \gamma v_\pi(s')]. \tag{32}$$

Now we can use v_π to get $\pi' \geq \pi$, then use $v_{\pi'}$ to get *another* policy. (In the above, ties are broken arbitrarily when the policy is deterministic. If the policy is stochastic, we accept any policy that assigns zero probability to sub-optimal actions.)

Note that by construction

$$q_\pi(s, \pi'(s)) \geq v_\pi(s)$$

therefore

$$v_{\pi'} \geq v_\pi$$

so we get from this process a monotonically increasing sequence of policies.

Note also that is π' is as good as π then $v_{\pi'} = v_\pi$ and $\forall s \in \mathcal{S}$

$$\begin{aligned}
v_\pi &= \max_a \mathbb{E}[R_{t+1} + \gamma v_{\pi'(S_{t+1})} | S_t = s, A_t = a] \\
&= \max_a \sum_{s', r} p(s', r | s, a)(r + \gamma v_{\pi'}(s'))
\end{aligned}$$

which is the Bellman optimality condition for v_*, so both π and π' are optimal. This means that policy improvement gives a strictly better policy unless the policy is already optimal.

The policy improvement theorem holds for stochastic policies too, but we don't go into that here.

4.3 Policy Iteration

We can exploit policy improvement iteratively to get the policy iteration algorithm.

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Loop:
      $\Delta \leftarrow 0$
      Loop for each $s \in \mathcal{S}$:
         $v \leftarrow V(s)$
         $V(s) \leftarrow \sum_{s',r} p(s', r \mid s, \pi(s)) \big[r + \gamma V(s')\big]$
         $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
      old-action $\leftarrow \pi(s)$
      $\pi(s) \leftarrow \operatorname{argmax}_a \sum_{s',r} p(s', r \mid s, a) \big[r + \gamma V(s')\big]$
      If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

A finite MDP has only a finite number of policies (as long as they are deterministic, of course) so this process is guaranteed to converge.

4.4 Value Iteration

Policy iteration can be slow because each iteration involves running the entire policy evaluation until convergence.

It turns out that one can truncate the policy evaluation step of policy iteration in many ways without losing convergence guarantees. One special case of this is *value iteration*, where we truncate policy evaluation after only one update of each state. This algorithm converges to v_* under the same conditions that guarantee the existence of v_*.

Value Iteration, for estimating $\pi \approx \pi_*$

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:
| $\Delta \leftarrow 0$
| Loop for each $s \in \mathcal{S}$:
| $v \leftarrow V(s)$
| $V(s) \leftarrow \max_a \sum_{s',r} p(s', r \mid s, a) \big[r + \gamma V(s')\big]$
| $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Output a deterministic policy, $\pi \approx \pi_*$, such that
   $\pi(s) = \operatorname{argmax}_a \sum_{s',r} p(s', r \mid s, a) \big[r + \gamma V(s')\big]$

Note the $\max_a$ in the assignment of $V(s)$, since we only one sweep of the state space and then choose the greedy policy.

It may be more efficient to interpose multiple policy evaluation steps in between policy improvement iterations, all of these algorithms converge to an optimal policy for discounted finite MDPs.

4.5 Asynchronous Dynamic Programming

The DP methods that we have described so far all involve a full sweep of the state space on each iteration. This is potentially a very costly procedure.

Asynchronous DP algorithms update the values in-place and cover states in any order whatsoever. The values of some states may be updated several times before the values of others are updated once. To converge correctly, however, an asynchronous algorithm must continue to update the values of all the states: it can't ignore any state after some point in the computation.

Asynchronous DPs give a great increase in flexibility, meaning that we can choose the updates we want to make (even stochastically) based on the interaction of the agent with the environment. This procedure might not reduce computation time in total if the algorithm is run to convergence, but it could allow for a better rate of progress for the agent.

4.6 Generalised Policy Iteration

We use the term *generalised policy iteration* (GPI) to refer to the general idea of letting policy evaluation and policy improvement processes interact, independent of the granularity and other details of the two processes. Almost all reinforcement learning methods are well described as GPI, including the policy iteration algorithms we have discussed in this section. GPI works via the competing but complementary nature of the two processes. In some cases it can be guaranteed to converge.

4.7 Efficiency of Dynamic Programming

If we ignore a few technical details, then the (worst case) time DP methods take to find an optimal policy is polynomial in the number of states and actions. Compare this to the searching the states directly, which is exponential.

5 Monte Carlo Methods

Monte Carlo methods learn state and action values by sampling and averaging returns (i.e. not from dynamics like DP). These methods learn from experience (real or simulated) and require no prior knowledge of the environments dynamics.

Monte Carlo methods tus require well defined returns, so we will consider them only for episodic tasks. Only on completion of an episode do values and policies change.

We still use the generalised policy iteration framework, but we adapt it so that we learn the value function from experience rather than compute it *a priori*.

5.1 Monte Carlo Prediction

The idea is to average the returns following each state to get an estimate of the state value

$$v_\pi(s) = \mathbb{E}_\pi[G_{t+1}|S_t = s].$$

Given enough observations, the sample average converges to the true state value under the policy π.

Given a policy π and a set of episodes, here are two ways in which we might estimate state values

First Visit MC average returns from first visit to state s in order to estimate $v_\pi(s)$

Every Visit MC average returns following every visit to state s.

First visit MC generates iid estimates of $v_\pi(s)$ with finite variance, so the sequence of estimates converges to the expected value by the law of large numbers as visits to s tend to ∞. Every visit MC does not generate independent estimates, but still converges.

An algorithm for first visit MS (what we will focus on) is below. Every visit is the same, just without the check for S_k occurring earlier in the episode.

First-visit MC prediction, for estimating $V \approx v_\pi$

Input: a policy π to be evaluated
Initialize:
    $V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$
    $Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Loop forever (for each episode):
    Generate an episode following π: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
        $G \leftarrow G + R_{t+1}$
        Unless S_t appears in $S_0, S_1, \ldots, S_{t-1}$:
            Append G to $Returns(S_t)$
            $V(S_t) \leftarrow$ average$(Returns(S_t))$

Monte Carlo methods are often used even when the dynamics of the environment are knowable, e.g. in Blackjack. It is often much easier to create sample games than it is to calculate environment dynamics directly.

MC estimates for different states are independent (unlike bootstrapping in DP). This means that we can use MC to calculate the value function for a subset of the states, rather than the whole state space as with DP. Along with the ability to learn from experience and simulation, this is the another advantage that MC has over DP.

5.2 Monte Carlo Estimation of Action Values

If we don't have a model for the environment, then it is more useful to estimate action-values. With a model we can use state values to find a policy by searching possible actions, as with DP (value iteration, etc.). We can't do this without knowledge of the dynamics, so one of the primary goals of MC is to estimate q_*. We start with policy evaluation for action-values.

Policy Evaluation for Action-Values

The policy evaluation problem for action-values is to estimate $q_\pi(s, a)$ for some π. This is essentially the same as for state values, only we now talk about state-action pairs being visited, i.e. taking action a in state s, rather than just states being visited.

If π is deterministic, then we will only estimate the values of actions that π dictates. We therefore need to incorporate some exploration in order to have useful action-values (since, after all, we want to use them to make informed decisions).

One consideration is to make π stochastic, e.g. ε-soft. Another is the assumption of *exploring starts*, which specifies that ever state-action pair has non-zero probability of being selected as the starting state. Of course, this is not always posible in practice.

For now we assume exploring start. Later we will come back to the issue of *maintaining exploration*

5.3 Monte Carlo Control

We make use of the GPI framework for action-values. Policy evaluation is done as described. Policy improvement is done by making the policy greedy with respect to the action-value function, so no model is needed for this step

$$\pi(s) \doteq \operatorname*{argmax}_a q_(s, a).$$

We generate a sequence of policies π_k each greedy with respect to $q_{\pi_{k-1}}(s, a)$. The policy improvement theorem applies: for all $s \in \mathcal{S}$

$$\begin{aligned} q_{\pi_k}(s, a) &= q_{\pi_k}(s, \operatorname*{argmax}_a q_\pi(s, a)) && (33)\\ &= \max_a q_{\pi_k}(s, a) && (34)\\ \geq q_{\pi_k}(s, \pi_k(s)) && (35)\\ = v_{\pi_k}(s) && (36) \end{aligned}$$

So π_{k+1} uniformly better than π_k or it is optimal.

The above procedure's convergence depends on assumptions of exploring starts and infinitely many episodes. We will relax the first later, but we will address the second now.

Two approaches to avoid infinitely many episodes:

1. Stop the algorithm once the q_{π_k} stop moving within a certain error. (In practice this is only useful on the smallest problems.)

2. Stop policy evaluation after a certain number of episodes, moving the action value towards q_{π_k}, then go to policy improvement.

For MC policy evaluation, it is natural to alternate policy evaluation and improvement on a episode by episode basis. We give such an algorithm below (with the assumption of exploring starts).

Monte Carlo ES (Exploring Starts), for estimating $\pi \approx \pi_*$

Initialize:
$\pi(s) \in \mathcal{A}(s)$ (arbitrarily), for all $s \in \mathcal{S}$
$Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$
$Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Loop forever (for each episode):
Choose $S_0 \in \mathcal{S}$ and $A_0 \in \mathcal{A}(S_0)$ such that all pairs have probability > 0
Generate an episode from S_0, A_0, following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
$G \leftarrow 0$
Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
$G \leftarrow G + R_{t+1}$
Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:
Append G to $Returns(S_t, A_t)$
$Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$
$\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$

It is easy to see that optimal policies are a fixed point of this algorithm. Whether this algorithm converges in general is still, however, an open question.

5.4 Monte Carlo Control without Exploring Starts

On Policy vs. Off Policy

On-policy methods evaluate or improve the policy that is used to make decisions, whereas off-policy methods evaluate or improve one that is different than the one used to generate the data.

On-Policy Techniques without Exploring Starts

We consider ε-greedy policies that put probability $1 - \varepsilon + \frac{\varepsilon}{|\mathcal{A}(s)|}$ on the maximal action and $\frac{\varepsilon}{|\mathcal{A}(s)|}$ on each of the others. These are examples of ε-soft policies in which $\pi(a|s) \geq \frac{\varepsilon}{|\mathcal{A}(s)|}$.

On-policy first-visit MC control (for ε-soft policies), esti

Algorithm parameter: small $\varepsilon > 0$

Initialize:

$\pi \leftarrow$ an arbitrary ε-soft policy

$Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

$Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Repeat forever (for each episode):

Generate an episode following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1},$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:

$G \leftarrow G + R_{t+1}$

Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A$

Append G to $Returns(S_t, A_t)$

$Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$

$A^* \leftarrow \arg\max_a Q(S_t, a)$ (with ties

For all $a \in \mathcal{A}(S_t)$:

$$\pi(a|S_t) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(S_t)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(S_t)| & \text{if } a \neq A^* \end{cases}$$

We use this idea in the GPI framework:

We now show that an ε-greedy policy with respect to q_π, π', is an improvement over any ε-soft policy π. For any $s \in \mathcal{S}$

$$q_\pi(s, \pi'(s)) = \sum_a \pi'(a|s) q_\pi(s,a) \tag{37}$$

$$= \frac{\varepsilon}{|\mathcal{A}(s)|} \sum_a q_\pi(s,a) + (1-\varepsilon) \max_a q_\pi(s,a) \tag{38}$$

$$\geq \frac{\varepsilon}{|\mathcal{A}(s)|} \sum_a q_\pi(s,a) + (1-\varepsilon) \sum_a \frac{\pi(a|s) - \frac{\varepsilon}{|\mathcal{A}(s)|}}{1-\varepsilon} q_\pi(s,a) \tag{39}$$

$$= \sum_a \pi(a|s) q_\pi(s,a) \tag{40}$$

$$= v_\pi(s) \tag{41}$$

(where line 3 follows because a weighted average with weights $w_i \geq 0$ and $\sum_i w_i = 1$ is $\leq$ the max term).

This satisfies the condition of the policy improvement theorem so we now know that $\pi' \geq \pi$.

Previously, with deterministic greedy policies, we would get automatically that fixed points of policy iteration are optimal policies since

$$v_*(s) \doteq max_\pi v_\pi(s) \quad \forall s \in \mathcal{S}.$$

Now our policies are not deterministically greedy, our value updates do not take this form. We note, however, that we can consider an equivalent problem where we change the environment to select state and reward transitions at random with probability ε and do what our agent asks with probability $1 - \varepsilon$. We have moved the stochasticity of the policy into the environment, creating an equivalent

problem. The optimal value function in the new problem satisfies its Bellman equation

$$\tilde{v}_\pi(s) = (1-\varepsilon)\max_a \tilde{q}_\pi(s,a) + \frac{\varepsilon}{|\mathcal{A}(s)|}\sum_a \tilde{q}_\pi(s,a) \tag{42}$$

$$= (1-\varepsilon)\max_a \sum_{s',r} p(s',r|s,a)[r+\gamma\tilde{v}_\pi(s')] + \frac{\varepsilon}{|\mathcal{A}(s)|}\sum_a\sum_{s',r} p(s',r|s,a)[r+\gamma\tilde{v}_\pi(s')]. \tag{43}$$

We also know that at fixed points of our algorithm

$$v_\pi(s) = (1-\varepsilon)\max_a q_\pi(s,a) + \frac{\varepsilon}{|\mathcal{A}(s)|}\sum_a q_\pi(s,a) \tag{44}$$

$$= (1-\varepsilon)\max_a \sum_{s',r} p(s',r|s,a)[r+\gamma v_\pi(s')] + \frac{\varepsilon}{|\mathcal{A}(s)|}\sum_a\sum_{s',r} p(s',r|s,a)[r+\gamma v_\pi(s')]. \tag{45}$$

This is the same equation as above, so by uniqueness of solutions to the Bellman equation we have that $v_\pi = \tilde{v}_\pi$ and so π is optimal.

5.5 Off-Policy Prediction via Importance Sampling

Off-policy learning uses information gained by sampling the *behaviour policy* b to learn the *target policy* π. The behaviour policy explores the environment for us during training and we update the target policy accordingly.

In this section we consider the prediction problem: estimating v_π or q_π for a fixed and known π using returns from b. In order to do this we need the assumption of coverage:

$$\pi(a|s) \geq 0 \implies b(a|s) \geq 0. \tag{46}$$

This implies that b must be stochastic wherever it is not identical to π. The target policy π may itself be deterministic, e.g. greedy with respect to action-value estimates.

Importance Sampling

We use *importance sampling* to evaluate expected returns from π given returns from b.

Define the importance sampling ratio as the relative probability of a certain trajectory from S_t

$$\rho_{t:T-1} = \frac{\mathbb{P}(A_t, S_{t+1}, A_{t+1}, \dots)|S_t, A_{t:T-1}\sim\pi}{\mathbb{P}(A_t, S_{t+1}, A_{t+1}, \dots)|S_t, A_{t:T-1}\sim b} \tag{47}$$

$$= \frac{\prod_{k=t}^{T-1}\pi(A_k|S_k)\mathbb{P}(S_{k+1}|S_k,A_k)}{\prod_{k=t}^{T-1} b(A_k|S_k)\mathbb{P}(S_{k+1}|S_k,A_k)} \tag{48}$$

$$= \prod_{k=t}^{T-1}\frac{\pi(A_k|S_k)}{b(A_k|S_k)} \tag{49}$$

where the state transition dynamics $\mathbb{P}$ cancel out.

If we have returns G_t from evaluating policy b, so $v_b(s) = \mathbb{E}[G_t|S_t = s]$, then we can calculate

$$v_\pi(s) = \mathbb{E}[\rho_{t:T-1}G_t|S_t = s]$$

Estimation

Introduce new notation:

- Label all time steps in a single scheme. So maybe episode 1 is $t = 1, \dots, 100$ and episode 2 is $t = 101, \dots, 200$, etc.
- Denote the set times of first/every visit to s by $\mathcal{T}(s)$ (spanning episodes).
- Let $T(t)$ be the first termination after t
- Let G_t be the returns from t to $T(t)$

We can now give two methods of values for π from returns from b: **Ordinary Importance Sampling**

$$V(s) \doteq \frac{\sum_{t \in \mathcal{T}(s)} \rho_{t:T-1} G_t}{|\mathcal{T}(s)|} \tag{50}$$

Weighted Importance Sampling

$$V(s) \doteq \frac{\sum_{t \in \mathcal{T}(s)} \rho_{t:T-1} G_t}{\sum_{t \in \mathcal{T}(s)} \rho_{t:T-1}} \tag{51}$$

or 0 if the denominator is 0.

Weighted importance sampling is biased (e.g. it's expectation is $v_b(s)$ after 1 episode) but has bounded variance. The ordinary importance sampling ratio is unbiased, but has possibly infinite variance, because the variance of the importance sampling ratios themselves is unbounded.

Assuming bounded returns, the variance of the weighted importance sampling estimator converges to 0 even if the variance of the importance sampling ratios is infinite. In practice, this estimator usually has dramatically lower variance and is strongly preferred.

5.6 Incremental Implementation

We look for incremental calculations of the averages that make up the estimates, as in Chapter 2.

For on-policy methods the incremental averaging is the same as in Chapter 2. For off-policy methods, but with ordinary importance sampling, we only need to multiply the returns by the importance sampling ratio and then we can average as before.

We will now consider weighted importance sampling. We have a sequence of returns G_i, all starting in the same state s and each with a random weight W_i (e.g. $W_i = \rho_{i:T(i)-1}$). We want to iteratively calculate (for $n \geq 2$)

$$V_n = \frac{\sum_{k=1}^{n-1} W_k G_k}{\sum_{k=1}^{n-1} W_k}.$$

We can do this with the following update rules

$$V_{n+1} = V_n + \frac{W_n}{C_n}[G_n - V_n] \tag{52}$$

$$C_{n+1} = C_n + W_{n+1} \tag{53}$$

where $C_0 = 0$ and V_1 is arbitrary (notice that it cancels out as $V_2 = G_1$).

Below is an algorithm for off-policy weighted importance sampling (set $b = \pi$ for on policy). The estimator Q converges to q_π for all encountered state-action pairs.

Off-policy MC prediction (policy evaluation) for estimating $Q \approx q_\pi$

Input: an arbitrary target policy π
Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
    $Q(s,a) \in \mathbb{R}$ (arbitrarily)
    $C(s,a) \leftarrow 0$

Loop forever (for each episode):
    $b \leftarrow$ any policy with coverage of π
    Generate an episode following b: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    $W \leftarrow 1$
    Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
        $G \leftarrow \gamma G + R_{t+1}$
        $C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
        $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$
        $W \leftarrow W \frac{\pi(A_t|S_t)}{b(A_t|S_t)}$
        If $W = 0$ then exit For loop

5.7 Off-Policy Monte Carlo Control

Below is an algorithm for estimating π_* and q_* in the GPI framework. The target policy π is the greedy policy with respect to Q, which is an estimate of q_π. This algorithm converges to q_π as long as an infinite number of returns are observed for each state-action pair. This can be achieved by making b ε-soft. The policy π converges to π_* at all encountered states even if b changes (to another ε-soft policy) between or within episodes.

Off-policy MC control, for estimating $\pi \approx \pi_*$

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
    $Q(s,a) \in \mathbb{R}$ (arbitrarily)
    $C(s,a) \leftarrow 0$
    $\pi(s) \leftarrow \operatorname{argmax}_a Q(s,a)$ (with ties broken consistently)

Loop forever (for each episode):
    $b \leftarrow$ any soft policy
    Generate an episode using b: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    $W \leftarrow 1$
    Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
        $G \leftarrow \gamma G + R_{t+1}$
        $C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
        $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$
        $\pi(S_t) \leftarrow \operatorname{argmax}_a Q(S_t, a)$ (with ties broken consistently)
        If $A_t \neq \pi(S_t)$ then exit For loop
        $W \leftarrow W \frac{1}{b(A_t|S_t)}$

Notice that this policy only learns from episodes in which b selects only greedy actions after some

timestep. This can greatly slow learning.

5.8 *Discounting Aware Importance Sampling

We present a method of importance sampling that recognises the return as a discounted sum of rewards. This can help in estimation, since if an episode is of length 100 and $\gamma = 0$ then the final 99 terms of the importance sampling ration contribute nothing to the expected value of our estimator (they have expected value of 1) but can greatly increase its variance. We therefore construct a method of importance sampling that takes into account discounting.

Introduce the *flat partial returns*

$$\bar{G}_{t:h} \doteq \sum_{i=t+1}^{h} R_i \quad 0 \leq t \leq h \leq T$$

then it can be shown (by rearranging) that

$$G_t \doteq \gamma^{i-t} R_{i+1} \tag{54}$$

$$= (1-\gamma) \sum_{h=t+1}^{T-1} \gamma^{h-t-1} \bar{G}_{t:h} + \gamma^{T-t-1} \bar{G}_{t:T}. \tag{55}$$

Now we can scale each flat partial return by a truncated importance sampling ratio (hence reducing variance).

Ordinary Importance Sampling Ratio

$$V(s) \doteq \frac{\sum_{t \in \mathcal{T}(s)} \left[(1-\gamma) \sum_{h=t+1}^{T(t-1)} \gamma^{h-t-1} \rho_{t:h-1} \bar{G}_{t:h} + \gamma^{T(t)-t-1} \rho_{t:T(t)-1} \bar{G}_{t:T(t)} \right]}{|\mathcal{T}(s)|} \tag{56}$$

Weighted Importance Sampling Ratio

$$V(s) \doteq \frac{\sum_{t \in \mathcal{T}(s)} \left[(1-\gamma) \sum_{h=t+1}^{T(t-1)} \gamma^{h-t-1} \rho_{t:h-1} \bar{G}_{t:h} + \gamma^{T(t)-t-1} \rho_{t:T(t)-1} \bar{G}_{t:T(t)} \right]}{\sum_{t \in \mathcal{T}(s)} \left[(1-\gamma) \sum_{h=t+1}^{T(t-1)} \gamma^{h-t-1} \rho_{t:h-1} + \gamma^{T(t)-t-1} \rho_{t:T(t)-1} \right]} \tag{57}$$

5.9 *Per-Decision Importance Sampling

There is another way in which we may be able to reduce variance in off-policy importance sapling, even in the absence of discounting ($\gamma = 1$). Notice that the off-policy estimators are made up of terms like

$$\rho_{t:T-1} G_t = \rho_{t:T-1}(R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-t-1} R_T)$$

and that each of these terms is of the form

$$\rho_{t:T-1} R_{t+1} = \frac{\pi(A_t|S_t)}{b(A_t|S_t)} \cdots \frac{\pi(A_{T-1}|S_{T-1})}{b(A_{T-1}|S_{T-1})} R_{t+1}.$$

Now notice that only the first and last terms here are correlated, while all the others have expected value 1 (taken with respect to b). Clearly this is also the case at each t. This means that

$$\mathbb{E}[\rho_{t:T-1} R_{t+k}] = \mathbb{E}[\rho_{t:t+k-1} R_{t+k}]$$

therefore

$$\mathbb{E}[\rho t : T - 1 G_t] = \mathbb{E}[\tilde{G}_t]$$

where

$$\tilde{G}_t \doteq \sum_{i=t}^{T-1} \gamma^{i-t} \rho_{t:i} R_{i+1}.$$

Now we can write the ordinary importance sampling estimator as

$$V(s) \doteq \frac{\sum_{t \in \mathcal{T}(s)} \tilde{G}_t}{|\mathcal{T}(s)|}$$

possibly reducing variance in the estimator.

The weighted importance sampling estimators of this form that have so far been found have been shown to not be consistent (in the statistical sense). We don't know if a consistent weighted average form of this exists.

6 Temporal-Difference Learning