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, $\dots$. 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 $R_{t+1}, R_{t+2}, R_{t+3}, \dots$. 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.

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

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,

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

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 $a_* = \mathrm{argmax}_a\, q_*(s, a)$ 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 $v_0, v_1, v_2, \dots$ 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:

$$\begin{aligned} v_{k+1} &\doteq \mathbb{E}_\pi[R_{t+1} + \gamma v_k(S_{t+1}) | S_t = s] &(26)\\ &= \sum_a \pi(s|a) \sum_{s',r} p(s', r|s, a) \left[r + \gamma v_k(s')\right] &(27)\end{aligned}$$

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

$$
\begin{aligned}
\pi'(s) &= \underset{a}{\operatorname{argmax}}\, q_\pi(s, a) && (30) \\
&= \underset{a}{\operatorname{argmax}}\, \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = a] && (31) \\
& \underset{a}{\operatorname{argmax}} \sum_{s', r} p(s', r | s, a)[r + \gamma v_\pi(s')]. && (32)
\end{aligned}
$$

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

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

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 $\forall s \in \mathcal{S}$

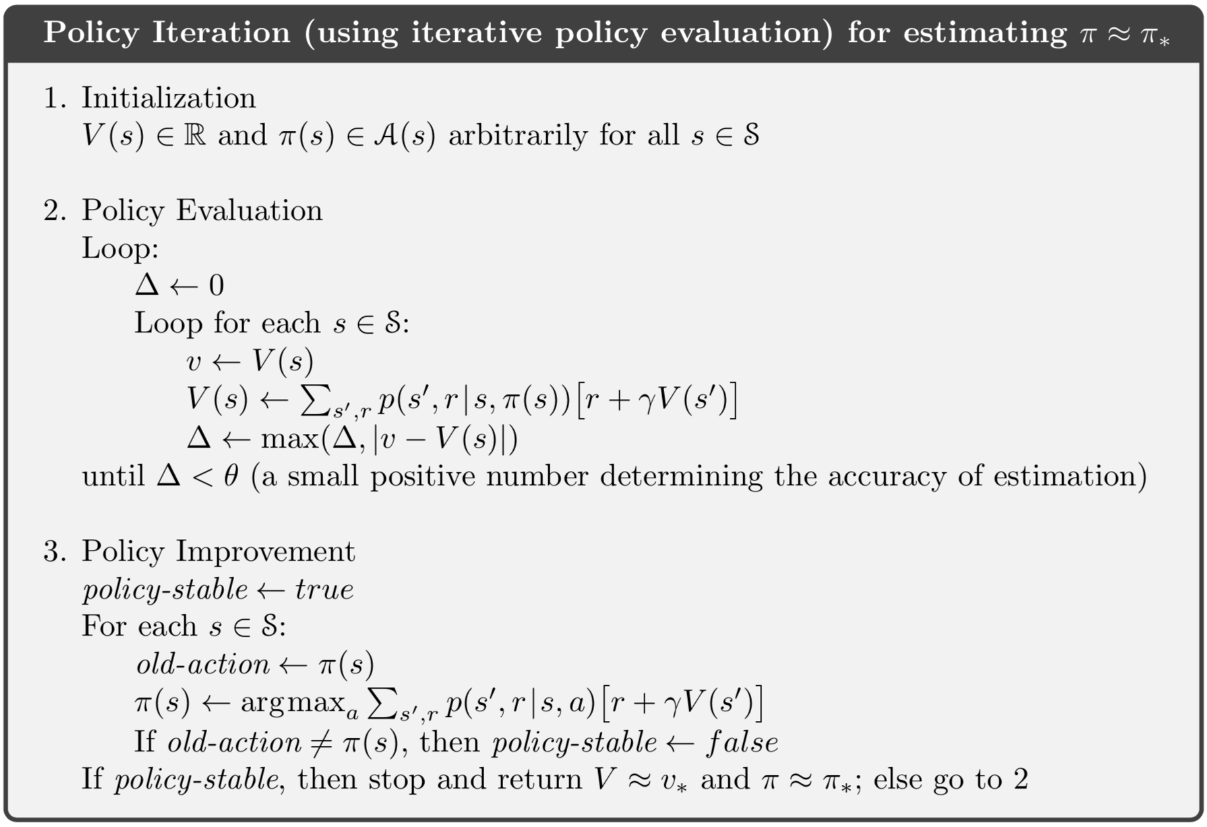
$$
\begin{aligned}
v_\pi &= \max_a \mathbb{E}[R_{t+1} + \gamma v_{\pi'(S_{t+1})} | S_t = s, A_t = a] \\
&= \max_a \sum_{s', r} p(s', r | s, a)(r + \gamma v_{\pi'}(s'))
\end{aligned}
$$

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.



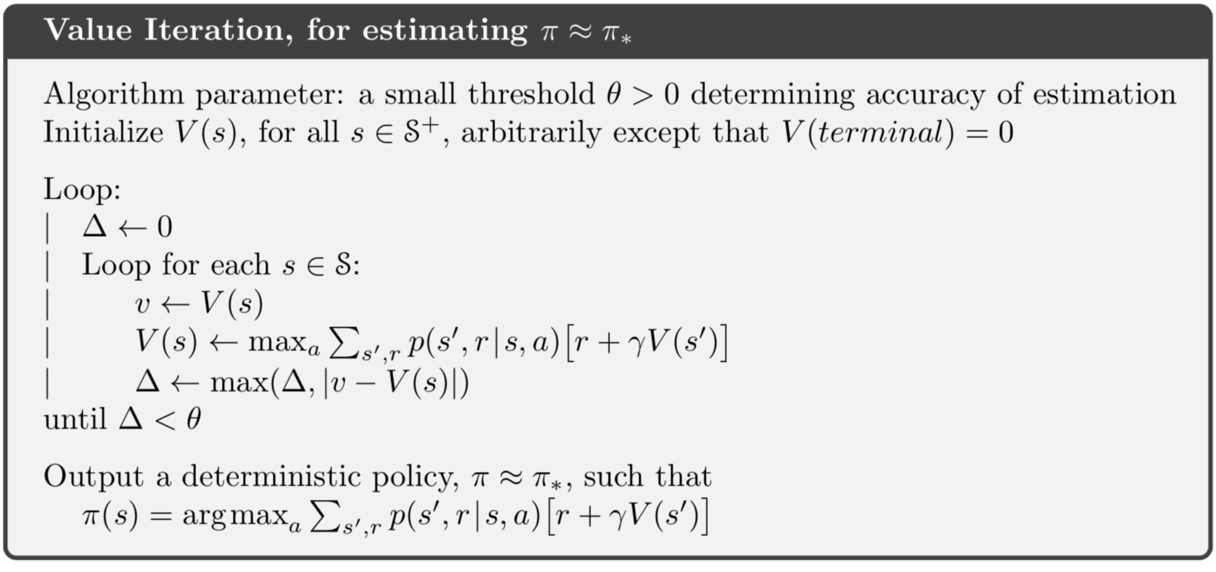
```
Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

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

3. Policy Improvement
   $policy\text{-}stable \leftarrow true$
   For each $s \in \mathcal{S}$:
       $old\text{-}action \leftarrow \pi(s)$
       $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r \,|\, s, a) \big[ r + \gamma V(s') \big]$
       If $old\text{-}action \neq \pi(s)$, then $policy\text{-}stable \leftarrow false$
   If $policy\text{-}stable$, 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 $s \in \mathcal{S}$:
|       $v \leftarrow V(s)$
|       $V(s) \leftarrow \max_a \sum_{s',r} p(s', r \,|\, s, a) \big[ r + \gamma V(s') \big]$
|       $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Output a deterministic policy, $\pi \approx \pi_*$, such that
    $\pi(s) = \arg\max_a \sum_{s',r} p(s', r \,|\, s, a) \big[ r + \gamma V(s') \big]$
```

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

5.1 Monte Carlo Prediction

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

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

Loop forever (for each episode):
    Generate an episode following $\pi$: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    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 $\textit{Returns}(S_t)$
            $V(S_t) \leftarrow \text{average}(\textit{Returns}(S_t))$
```

5.2 Monte Carlo Estimation

5.3 Monte Carlo Control

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 $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$
    $\textit{Returns}(s,a) \leftarrow$ 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 $\pi$: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    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 $\textit{Returns}(S_t, A_t)$
            $Q(S_t, A_t) \leftarrow \text{average}(\textit{Returns}(S_t, A_t))$
            $\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$
```

5.4 Monte Carlo Control without Exploring Starts

On-policy first-visit MC control (for ε-soft policies), estimates $\pi \approx \pi_*$

```
Algorithm parameter: small $\varepsilon > 0$
Initialize:
    $\pi \leftarrow$ an arbitrary $\varepsilon$-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 $\pi$: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    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 $Returns(S_t, A_t)$
            $Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$
            $A^* \leftarrow \arg\max_a Q(S_t, a)$                    (with ties broken arbitrarily)
            For all $a \in \mathcal{A}(S_t)$:
                $\pi(a|S_t) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(S_t)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(S_t)| & \text{if } a \neq A^* \end{cases}$
```