Study Notes of Reinforcement Learning: Tabular Solution Methods

Elynn Chen[1]

[1]ORFE, Princeton University
[1]OpenAI

2019-03-01

Abstract

This article summarizes important concepts in the first part (Chapter 1 to 8) of Sutton and Barto (2018). This part mainly deals with tabular solution methods. All of the methods have three key ideas in common: (1) they all seek to estimate value functions; (2) they all follow the general strategy of generalized policy iteration (GPI), meaning that they maintain an approximate value function and an approximate policy, and they continually try to improve each on the basis of each other. The methods are different with respect to: (1) whether they are sample updates need only a sample model, or can be done from a distribution model; (2) the degree of bootstrapping; (3) on-policy where the agent learns the value function for the policy it is currently following, or off-policy where the agent learns the value function for the policy from a different policy.

As a preview, the second part of the book (next week) will explore function approximation, which can be viewed as an orthogonal spectrum of possibilities ranging from tabular methods at one extreme through state aggregation, a variety of linear methods, and then a diverse set of nonlinear methods.

Key words: Reinforcement learning, Multi-Armed Bandits (MAB), Finite Markov Decision Processes (MDP), dynamic programming, Monte Carlo methods, Temporal-Difference (TD) learning, value function, optimal policy, Generalized Policy Iteration (GPI), sample updates, expected updates, bootstrapping, on-policy and off-policy, .

1 Multi-Armed Bandits

Multi-armed bandits are the simplest forms of reinforcement learning in that there is only a single state and a finite (or infinite) action space. In this case, the learning is *non-associative*, meaning that learning to act in only one situation or the action is not associated with multiple situations.

1.1 Summary of Multi-Armed Bandits

- A K-armed Bandit Problem: Multiple actions with only one state.
- Action-value methods
  - Estimate action value $Q_t(a)$. Sample-average method. Estimation mean and variance.
  - Action selection methods.

    $A_t \doteq \arg\max_a f\left(Q_t(a)\right)$, where $f(\cdot)$ is some function of action value $Q_t(a)$.

    1. Greedy method. Only use estimates mean. No exploration. $f(\cdot)$ is identity function.
    2. ε-Greedy method. Only use estimates mean. With exploration at probability ε, but treat non-greedy actions indiscriminately. Introduce randomness in $f(\cdot)$.
    3. Upper-Confidence-Bound (UCB) Action Selection. Use both mean and variance of estimates of action-values. Especially, explore which action has the most upper hands. $f(\cdot)$ takes into account the variance of $Q_t(a)$.
- Gradient Bandit Algorithms
- Contextual Bandits (Associative Search)

1.2 A K-Armed Bandit Problem

In K-armed bandit problem, an agent is faced repeatedly with a choice among k different options, or actions. At time $1 \leq t \leq T$, the action space is $\mathcal{A}_t = \{a_{t,1}, \cdots, a_{t,K}\}$. In the case when $\mathcal{A}_t$ is constant over t, we depress subscript t and write the action space as $\mathcal{A}$. The agent chooses some action A_t at time t and receive a reward $R_t(A_t)$. This repeated for T time steps and the objective of the agent is to maximize the expected total rewards over the entire time period.

$$\max_{\substack{\{A_1,\cdots,A_T\} \\ A_t \in \mathcal{A}}} \mathbb{E}\left[\frac{1}{T}\sum_{t=1}^{T} R_t(A_t)\right] \tag{1.1}$$

In the simplest case, the environment is noiseless and deterministic, the reward get at time t is generated exactly from some unknown function $Q^\star(A)$ for each $A \in \mathcal{A}$. If K is finite, then the agent could simply try K times, get the true function $Q^\star(A)$ and stick with the action that maximize $Q^\star(A)$ in all $T - K$ periods. As $T \longrightarrow \infty$, this strategy is optimal. If you have many time steps ahead on which to make action selections, then it may be better to explore the nongreedy actions and discover which of them are better than the greedy action.

However, a more practical setting is that the actual reward at time t is noisy and stochastic, specifically,

$$R_t(A) = Q^\star(A) + E_t, \tag{1.2}$$

where E_t is a random noise. In most research, it is assumed to be Gaussian $E_t \sim \mathcal{N}(0, \sigma^2)$ or Sub-Gaussian. Maybe there are literature exploring heavy tail distributions.

Under the assumption that $E_t \sim \mathcal{N}(0, \sigma^2)$,

$$Q^\star(a) = \mathbb{E}\left[R_t(A) | A = a\right]. \tag{1.3}$$

By Law of Large Number (LLN), $1/T \sum_{t=1}^{T} R_t(a)$ is a consistent estimator of $Q^\star(a)$.

1.3 Action-value Methods

Action-value methods refer to estimating the values of actions and using the estimates to make action selection decision. Greedy action refers to choosing the action whose estimated value is greatest at any time step. It *exploits* the current knowledge of the values of the actions. If instead one of the non-greedy actions is chosen, the agents *explores* the non-greedy action's value to discover which of them are better than the greedy action.

In any specific case, whether it is better to explore or exploit depends in a complex way on the precise values of the estimates, uncertainties, and the number of remaining steps. There are many sophisticated methods for balancing exploration and exploitation for particular mathematical formulations of the k-armed bandit and related problems. However, most of these methods make strong assumptions about stationarity and prior knowledge that are either violated or impossible to verify in applications and in the full reinforcement learning problem. The guarantees of optimality or bounded loss for these methods are of little comfort when the assumptions of their theory do not apply.

1.3.1 Action Value Estimation: Sample-Average

One natural way to estimate is averaging the rewards actually received:

$$Q_t(a) \doteq \frac{\sum_{i=1}^{t-1} R_i \mathbb{1}_{A_i=a}}{\sum_{i=1}^{t-1} \mathbb{1}_{A_i=a}} \tag{1.4}$$

Incremental Implementation. The sample average estimation can be computed in a computationally efficient manner, in particular, with constant memory and per-time-step computation. See Section 2.4 of Sutton and Barto (2018) for more detail.

Non-stationary Problem. The averaging (1.4) is appropriate for stationary bandit problems. For non-stationary problems, it makes sense to give more weight to recent rewards than to long-past rewards. See Section 2.5 of Sutton and Barto (2018) for more detail.

Initial Values $Q_1(a)$. The Greedy and ε-Greedy methods are dependent to some extent on the initial action-value estimates $Q_1(a)$, i.e. biased by their initial estimates. For sample-average methods, the bias disappears once all actions have been selected at least once. But for methods with constant α, the bias may be permanent. The downside is that the initial estimates become, in effect, a set of parameters that must be picked by the user. The upside is that they provide an easy way to supply some prior knowledge about what level of rewards can be expected.

Initial action values can also be used as a simple way to encourage exploration. For example, set initial action values $Q_1(a)$ really high. The result is that all actions are tried several times before the value estimates converge. This technique for encouraging exploration is called *optimistic initial values*. It is quite effective on stationary problems, but it is far from being a general useful approach to encouraging exploration because its drive for exploration is inherently temporary. See Section 2.6 of Sutton and Barto (2018) for more detail.

1.3.2 Greedy Method

Greedy method always exploits current knowledge to maximize immediate reward; it spends no time at all sampling apparently inferior actions to see if they might really be better.

$$A_i \doteq \arg\max_a Q_t(a) \tag{1.5}$$

1.3.3 ε-Greedy Method

ε-Greedy behaves greedily most of the time, but every once in a while, say with small probability ε, instead select randomly from among all the actions with equal probability, independent of the

action-value estimates.

An advantage of these methods is that, in the limit as the number of steps increases, every action will be sampled an infinite number of times, thus ensuring that all the $Q_t(a)$ converge to $q_\star(a)$. This of course implies that the probability of selecting the optimal action converges to greater than $1-\varepsilon$, that is, to near certainty.

1.3.4 Upper-Confidence-Bound (UCB) Action Selection

Exploration is needed because of the uncertainty associated with the accuracy of the action-value estimates. ε-greedy action selection forces the non-greedy actions to be tried, but indiscriminately, with no preference for those that are nearly greedy or particular uncertain. Intuitively, a better algorithm would take into account both how close estimates of the values of different actions (means), as well as the uncertainties in those estimates (variances).

The Upper Confidence Bound (UCB) action selection is one effective way of doing this by selecting actions according to

$$A_t \doteq \arg\max_a \left[Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}}, \right] \tag{1.6}$$

where $N_t(a)$ denotes the number of times that action a has been selected prior to time t, and the number $c > 0$ controls the degree of exploration. If $N_t(a) = 0$, then a is consider to be a maximizing action. The square root term $\sqrt{\frac{\ln t}{N_t(a)}}$ is a measurement of the uncertainty or variance in the estimate of a's value. The quantity being max'ed over is thus a sort of upper bound on the possible true value of action a, wich c determine the confidence level.

Research challenges. Dealing with non-stationary problems calls for more complex methods. Another difficulty is dealing with large state spaces, particularly when using function approximation.

1.4 Gradient Bandit Algorithms

Action value action selection estimate action values and use those estimates to select actions. Instead, we could learn for each action a a (relative) numerical *preference* instead of the *value*. Only the relative preference of one action, denoted as $H_t(a)$, over another is important. The action probabilities are determined according to a *soft-max distribution* (i.e., Gibbs or Boltzmann distribution) as follows:

$$\pi_t(a_i) \doteq Pr\{A_t = a_i\} \doteq \frac{e^{H_t(a_i)}}{\sum_{j=1}^{k} e^{H_t(a_j)}}. \tag{1.7}$$

Initially, all action preferences are the same (e.g. $H_1(a) = 0$ for all a) so that all actions have an equal probability of being selected. In the case of two actions, the soft-max distribution is the same as that given by the logistic, or sigmod, function often used in statistics and artificial neural networks (*Excercise 2.9* on page 37 of Sutton and Barto (2018)).

1.4.1 Learning algorithm for Gradient Bandit

The algorithm starts with all action preferences being the same (e.g. $H_1(a) = 0$ for all a). One each step t, after selecting action A_t and receiving the reward R_t, the action preferences are updated by:

$$H_{t+1}(A_t) \doteq H_t(A_t) + \alpha\,(R_t - \bar{R}_t)(1 - \pi_t(A - t)), \tag{1.8}$$

$$H_{t+1}(a) \doteq H_t(a) - \alpha\,(R_t - \bar{R}_t)\pi_t(a), \quad \text{for all} \quad a \neq A_t, \tag{1.9}$$

where $\alpha > 0$ is a step-size parameter, and $\bar{R}_t$ is the average of all the rewards up through and including time t, which can be computed incrementally as described in Section 2.4 and 2.5 in Sutton and Barto (2018). The $\bar{R}_t$ term serves as a baseline with which the reward is compared against at each step.

The Bandit Gradient Algorithm as Stochastic Gradient Ascent. See page 38 of Sutton and Barto (2018) for more details.

1.5 Associative Search: Contextual Bandits

To be completed. See page 41 of Sutton and Barto (2018) for more details.

Research directions and overviews. To be completed later.

Watch RL lecture by David Silver on Youtube.

1.6 Thompson Sampling

For more on Thompson Sampling. See the tutorial.

1.7 To Be Done

Explore research topics in Contextual Bandits.

2 Finite Markov Decision Processes

Finite Markov decision processes (MDPs) are a classical formalization of sequential decision making, where actions influence not just immediate rewards, but also subsequent situations, or states, and through those future rewards. Whereas in bandit problems we estimated the value $q_\star(a)$ of each action a, in MDPs we estimate the value $q_\star(s, a)$ of each action a in each state s.

Direction for research. MDPs are a mathematically idealized form of the reinforcement learning problem for which precise theoretical statements can be made. As in all of artificial intelligence, there is a tension between breadth of applicability and mathematically tractability.

2.1 General MDP Framework

The general MDP framework contains two part: the agent-environment interface and the goals and rewards.

2.1.1 The Agent-Environment Interface

The framework of the problem of learning from interaction to achieve a goal. The *agent* is the learning or decision maker. The *environment* comprises of everything outside the agent that he interact continually with.

The agent and environment interact at each of a sequence of discrete time steps, $t = 0, 1, 2, 3, \cdots$. At each time step t, the agent receives some representation of the environment's *states*, $S_t \in \mathcal{S}$, and on that basis selects and *action*, $A_t \in \mathcal{A}(s)$ or $A_t \in \mathcal{A}$ if we assume the action set is the same in all states. One step later, at time step $t + 1$, the agent receives a numerical *reward*, $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$, and moves to a new state, S_{t+1}.

The MDP and agent together thereby give rise to a sequence or *trajectory* that begins like this:

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots \tag{2.1}$$

In *finite* MDP, the sets of states, actions, and rewards ($\mathcal{S}$, $\mathcal{A}$, and $\mathcal{R}$) all have a finite number of elements. The solution can be expressed in tabular form. The random variable R_t and S_t have well defined discrete probability distributions dependent only on the preceding state and action. Mathematically, the **dynamic function of MDP** is defined as:

$$\begin{aligned} &p(s', r|s, a) \doteq \Pr\{S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a\}, \\ &\textstyle\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r|s, a) = 1, \quad \text{for all} \quad s \in \mathcal{S}, a \in \mathcal{A}. \end{aligned} \tag{2.2}$$

for all $s', s \in \mathcal{S}$, $r \in \mathcal{R}$, and $a \in \mathcal{A}$. The ordinary deterministic function $p(\cdot) : \mathcal{S} \times \mathcal{R} \times \mathcal{S} \times \mathcal{A} \in [0, 1]$ defines the dynamics of the MDP. Note here that Markov property means that the probability of each possible value for S_t and R_t depends only on the immediately preceding state and action, S_{t-1} and A_{t-1}, and conditionally independent on earlier states and actions. This is a restriction on the state S_{t-1} that it must include all information about all past agent-environment interactions that make a different for the future. Later, *approximate methods* do not rely on this assumption. Also, a Markov state can be learned and constructed from non-Markov observations.

Dynamic function (2.2) is the basis and other useful functions can be defined from it. With a slight abuse of notation, we defined the following functions:

The *state-transition probabilities* is a three-argument function $p : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$,

$$p(s|s', a) \doteq \Pr\{S_t = s'|S_{t-1} = s, A_{t-1} = a\} \;=\; \sum_{r \in \mathcal{R}} p(s', r|s, a). \tag{2.3}$$

The expected rewards for state-action pair $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$:

$$r(s, a) \doteq \mathbb{E}\left[R_t|S_{t-1} = s, A_{t-1} = a\right] = \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} r \cdot p(s', r|s, a). \tag{2.4}$$

The expected rewards for state-action-next-state triples $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$

$$r(s', s, a) \doteq \mathbb{E}\left[R_t|s_t = s', S_{t-1} = s, A_{t-1} = a\right] = \sum_{r \in \mathcal{R}} r \cdot p(r|s', s, a) \;=\; \sum_{r \in \mathcal{R}} r \cdot \frac{p(r, s'|s, a)}{p(s'|s, a)} \tag{2.5}$$

2.1.2 Goals, Rewards, Returns and Episodes

The *reward hypothesis*: " That all of what we mean by goals and purposes can be well thought of as the maximization of the expected value of the cumulative sum of a received scalar signal (called reward). " The reward signal is your way of communicating to the robot *what* you want it to achieve, not *how* you want it achieved.

A sequence of rewards received after time step t is denoted as $R_{t+1}, R_{t+2}, R_{t+3}, \ldots$, iin general, we seek to maximize the *expectation* of the **total return** G_t defined as some specific function of the reward sequence after t. Undiscounted and discounted returns can be formulated as following,

$$G_t \;\doteq\; \sum_{k=0}^{T-1} R_{t+k+1}, \tag{2.6}$$

$$G_t \;\doteq\; \sum_{k=0}^{T-1} \gamma^k R_{t+k+1}. \tag{2.7}$$

where T is a final time step, $0 \leq \gamma \leq 1$ is the discount rate.

Return (2.6) is suitable for *episodic task* where the agent-environment interaction breaks naturally into episodes, such as plays of a game, trips through a maze, or any sort of repeated interactions. Return (2.7) is appropriate for *continuing* tasks, in which the interaction does not naturally break into episodes but continues without limit. We define a return function that can apply to both the episodic and continuing cases by adding a special *absorbing state*. Then a unified notation for both episodic and continuing tasks can be written as

$$G_t \doteq \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k, \tag{2.8}$$

including the possibility that $T = \infty$ and $\gamma = 1$ (but not both). It is important to notice that the return function can be formulated recursively,

$$G_t = R_{t+1} + \gamma G_{t+1}. \tag{2.9}$$

2.2 Policies and Value Functions

Value functions are functions of states (or of state-action pairs) that estimate *how good* it is for the agent to be in a given state (or how good it is to perform a given action in a given state). The notation of "how good" is defined in terms of *expected return*.

A **policy** $\pi(a|s)$ is a mapping from states to probabilities of selecting each possible action.

The **state-value function** $v_\pi(s)$ of a state s under a policy π is the expected return when starting in s and following policy π thereafter. For MDPs, $v_\pi(s)$ is formally defined by

$$v_\pi(s) = \mathbb{E}_\pi \left[G_t | S_t = s \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s \right], \text{ for all } s \in \mathcal{S}, \tag{2.10}$$

where $\mathbb{E}_\pi [\cdot]$ denotes the expected value of a random variable given that the agent follows policy π, and t is any time step. Note that the value of a terminal state, if any, is always zero.

The **action-value function** $q_\pi(s, a)$ of taking action a in state s under a policy π if the expected return starting from s, taking the action a, and thereafter following policy π. For MDPs, $q_\pi(s, a)$ is formally defined by

$$q_\pi(s, a) = \mathbb{E}_\pi \left[G_t | S_t = s, A_t = a \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a \right], \text{ for all } s \in \mathcal{S}, \quad a \in \mathcal{A}. \tag{2.11}$$

Note that $v_\pi(s)$ and $q_\pi(s, a)$ are defined as the expectation of some random variables. They

can be estimated from experiences (samples). *Monte Carlo metohds* estimate by averaging over many random samples of actual returns for each state s or for each state-action pair (s, a) without assuming any models. Alternatively, the agent may assume some parameterized function approximator (with few parameters than states s) and adjust the parameters to better match the observed returns (in Part II of the book).

2.3 Bellman equation

A fundamental property of value functions is that they satisfy recursive relationships. The following **Bellman equation** gives the *self-consistency condition* the value functions must satisfied.

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi\left[G_t|S_t=s\right] = \mathbb{E}_\pi\left[R_{t+1}+\gamma G_{t+1}|S_t=s\right] \\ &= \sum_a \pi(a|s)\sum_{s'}\sum_r p(s',r|s,a)\left[r+\gamma\mathbb{E}_\pi\left[G_{t+1}|S_{t+1}=s'\right]\right] \\ &= \sum_a \pi(a|s)\sum_{s'}\sum_r p(s',r|s,a)\left[r+\gamma v_\pi(s')\right] \end{aligned} \tag{2.12}$$

$$\begin{aligned} q_\pi(s,a) &= \mathbb{E}_\pi\left[G_t|S_t=s,A_t=a\right] = \mathbb{E}_\pi\left[R_{t+1}+\gamma G_{t+1}|S_t=s,A_t=a\right] \\ &= \sum_a \pi(a|s)\sum_{s'}\sum_r p(s',r|s,a)\left[r+\gamma\mathbb{E}_\pi\left[G_{t+1}|S_{t+1}=s'\right]\right] \\ &= \sum_a \pi(a|s)\sum_{s'}\sum_r p(s',r|s,a)\left[r+\gamma\sum_{a'}\pi_(a'|s')\mathbb{E}_\pi\left[G_{t+1}|S_{t+1}=s',A_{t+1}=a'\right]\right] \\ &= \sum_a \pi(a|s)\sum_{s'}\sum_r p(s',r|s,a)\left[r+\gamma\sum_{a'}\pi_(a'|s')q_\pi(s',a')\right]. \end{aligned} \tag{2.13}$$

2.4 Optimal Policies and Optimal Value Functions

A policy π is defined to be better than or equal to a policy π' if its expected return is greater or equal to that of π' for all states. In other words, $\pi \geq \pi'$ if and only if $v_\pi(s) \geq v_{\pi'}(s)$ for all $s \in \mathcal{S}$. An **optimal policy** $\pi_\star$ is defined as a policy that is better than or equal to all other policies. The **optimal state-value function** $v_\pi(s)$, **optimal action-value function** $q_\star(s,a)$ and their relationship are formulated mathematically as following, for all $s \in \mathcal{S}$,

$$\begin{aligned} \pi_\star &= \arg\max_\pi v_\pi(s), \\ v_\star(s) &= \max_\pi v_\pi(s) = v_{\pi_\star}(s), \\ q_\star(s,a) &= \max_\pi q_\pi(s,a) = q_{\pi_\star}(s,a), \\ q_\star(s,a) &= \mathbb{E}_{\pi_\star}\left[R_{t+1}+\gamma v_\star(S_{t+1})|S_t=s,A_t=a\right]. \end{aligned}$$

Because $v_\star(s)$ is the value function for a policy, it must satisfy the self-consistency condition given by the Bellman equation for state values (2.12). Because it is the optimal value function, $v_\star(s)$'s consistency condition can be written in a special form without reference to any specific policy. The **Bellman optimality equation** express the fact that the value of a state under an optimal policy must equal the expected return for the best action from that state:

$$\begin{aligned} v_\star(s) &= \max_{a\in\mathcal{A}(s)} q_{\pi_\star}(s,a) \\ &= \max_a \mathbb{E}_{\pi_\star}\left[G_t|S_t=s,A_t=a\right] \\ &= \max_a \mathbb{E}_{\pi_\star}\left[R_t+\gamma G_{t+1}|S_t=s,A_t=a\right] \\ &= \max_a \mathbb{E}\left[R_t+\gamma v_\star(s')|S_t=s,A_t=a\right] \qquad (2.14) \\ &= \max_a \sum_{s',r} p(s',r|s,a)[r+\gamma v_\star(s')]. \qquad (2.15) \end{aligned}$$

The last two equations are two forms of the **Bellman optimality equation for** $v_\star$. The **Bellman optimality equation for** $q_\star$ is

$$\begin{aligned} q_\star(s,a) &= \mathbb{E}\left[R_t+\gamma G_{t+1}|S_t=s,A_t=a\right] \\ &= \mathbb{E}\left[R_t+\gamma \max_{a'} q_\star(S_{t+1},a')|S_t=s,A_t=a\right] \\ &= \sum_{s',r} p(s',r|s,a)[r+\gamma \max_{a'} q_\star(s',a')]. \qquad (2.16) \end{aligned}$$

For finite MDPs, the Bellman optimality equation for $v_\star(s)$ (2.14) has a unique solution. If the dynamics $p(s',r|s,a)$ of the environments are known, then in principle one can solve this system of equations for $v_\star$ using any one of a variety of methods for solving systems of nonlinear equations. One can also solve a related set of equations for $q_\star(s,a)$. Once $v_\star(s)$ or $q_\star(s,a)$ is known, it is relatively easy to determine the optimal policy. Actually, any policy that is *greedy* with respect to the optimal evaluation function is an optimal policy because $v_\star(s)$ and $q_\star(s,a)$ already take into account the long-term rewards.

2.5 Optimality and Approximation

Bellman optimality equations (2.14) and (2.16) defines a system of equations. If the dynamics $p(s',r|s,a)$ of the environments are known, the system is possible to be solved for $v_\star(s)$ and $q_\star(s,a)$. However, the reality is we rarely have a complete and accurate model of the environment's dynamics. Even they are known, the computational cost and memory available are import constraints. For

tasks with small, finite state sets (*tabular case*), it is possible to form approximates using arrays for tables with each entry for each state (or state-action pair), which is call *tabular methods*.

In general, our framing of the reinforcement learning problem forces us to settle for approximations. The online nature of reinforcement learning makes it possible to approximate optimal policies in ways that put more effort into learning to make good decisions for frequently encountered states, at the expense of less effort for infrequently encountered states. This is one key property that distinguishes reinforcement learning from other approaches to approximately solving MDPs.

3 Dynamic Programming

Dynamic programming (DP) refers to a collection of algorithms that can be used to compute optimal policies given a perfect model of the environment as a Markov decision process (MDP). 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 assume that the environment is a finite MDP. Its state, action, and reward sets, $\mathcal{S}$, $\mathcal{A}$ and $\mathcal{R}$, are finite, and that its dynamics are given by a set of probabilities $p(s', r|s, a)$, for all $s \in \mathcal{S}$, $a \in \mathcal{A}$, $r \in \mathcal{R}$, and $s' \in \mathcal{S}^+$ ($\mathcal{S}^+$ is $\mathcal{S}$ plus a terminal state if the problem is episodic).

The optimal state value function $v_\star(s)$ and optimal state-action value function $q_\star(s)$ satisfy the Bellman optimality equations:

$$\begin{aligned} v_\star(s) &= \max_a \ \mathbb{E}\left[R_t + \gamma v_\star(s')|S_t = s, A_t = a\right] \\ &= \max_a \ \sum_{s',r} p(s', r|s, a)[r + \gamma v_\star(s')], \end{aligned} \tag{3.1}$$

$$\begin{aligned} q_\star(s, a) &= \mathbb{E}\left[R_t + \gamma \max_{a'} \ q_\star(S_{t+1}, a')|S_t = s, A_t = a\right] \\ &= \sum_{s',r} p(s', r|s, a)[r + \gamma \max_{a'} \ q_\star(s', a')]. \end{aligned} \tag{3.2}$$

3.1 Policy Evaluation (Prediction)

Policy evaluation (prediction problem) compute the state-value function for an arbitrary policy π. For all $s \in \mathcal{S}$,

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi\left[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s\right] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\left[r + \gamma v_\pi(s')\right] \end{aligned} \tag{3.3}$$

If the environment's dynamics are completely known, then (3.3) is a system of $|\mathcal{S}|$ simultaneous linear equations in $|\mathcal{S}|$ unknowns (the $v_\pi(s), s \in \mathcal{S}$). The **existence and uniqueness** of $v_\pi(s)$ are guaranteed as long as either $\gamma < 1$ or eventually termination is guaranteed from all states under the policy π. **Iterative solution methods** are most suitable for our purpose here.

The **iterative policy evaluation** works as follow. Starting from the initial approximation v_0, each successive approximation is obtained by using the Bellman equation for $v_\pi(s)$ (3.3) as an update rule: for all $s \in \mathcal{S}$,

$$
\begin{aligned}
v_{k+1}(s) &= \mathbb{E}_\pi \left[R_{t+1} + \gamma v_k(S_{t+1}) | S_t = s \right] \\
&= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \left[r + \gamma v_k(s') \right] . \qquad (3.4)
\end{aligned}
$$

Convergence of the algorithm. Clearly, $v_k = v_\pi$ is a fix point for this update rule because of the Bellman equation. Indeed, the sequence $\{v_k\}$ can be shown in general to converge to v_π as $k \to \infty$ under the same condition that guarantee the existence of v_π.

Remark 1. All the updates done in DP algorithms are call **expected** updates because they are based on an expectation over all possible next states rather than on a sample next state.

Remark 2. Iterative policy evaluation is an example of a **classical successive approximation algorithm** for solving a system of linear equations. The **Jacobi-style** algorithm uses two arrays, one holding the old value while the other is updated. The **Gauss-Seidel-style** algorithm use one array and update the values "in place", that is, with each new value immediately overwriting the old one. This in-place algorithm converges to v_π and usually converges faster than the two-array version. We usually have the in-place version in mind when we think of DP algorithms.

3.2 Policy Improvement

3.3 Policy Iteration

3.4 Value Iteration

3.5 Generalized Policy Iteration

3.6 Efficiency of Dynamic Programming

4 Monte Carlo Methods

5 Temporal-Difference Learning

References

Sutton, R. S. and A. G. Barto (2018). *Reinforcement learning: An introduction*. MIT press.

Appendix A Proof

Appendix B Table and Plot