Introduction to Markov Decision Processes

Martin L. Puterman and Timothy C. Y. Chan

July 8, 2024

Chapter 1

Model Foundations

> *One must first have a strong foundation.*
>
> *Sri Auribindo, Indian Philosopher 1872-1950*

Effective use of Markov decision processes requires understanding its fundamentals. This chapter starts with a description of the basic components of a Markov decision process, namely, decision epochs, states, actions, transition probabilities, and rewards. From these components we derive decision rules, policies, stochastic processes and reward processes. These are not part of the model definition, but arise naturally and inherit structure from the basic model components. From this the next step is to describe optimality criteria, which provide a basis for comparing the quality of decisions. We use them to define the concept of an optimal policy, value functions, state-action value functions and the Bellman equation.

The book primarily focuses on discrete time models with discrete state and action spaces. It briefly discusses generalizations where appropriate such as the case of partially observable Markov decision processes (Chapter **??**) in which continuous state spaces arise naturally.

1.1 Basic Model Components

A Markov decision process model describes the fundamental elements of a recurring decision problem in which today's decision impacts future options. Decisions take

place over a *planning horizon*. *Decision epochs* represent specific points of time when a decision can be made (Figure 1.1). Time is divided into *periods* or *stages*; a period begins at one decision epoch and ends at the next decision epoch.

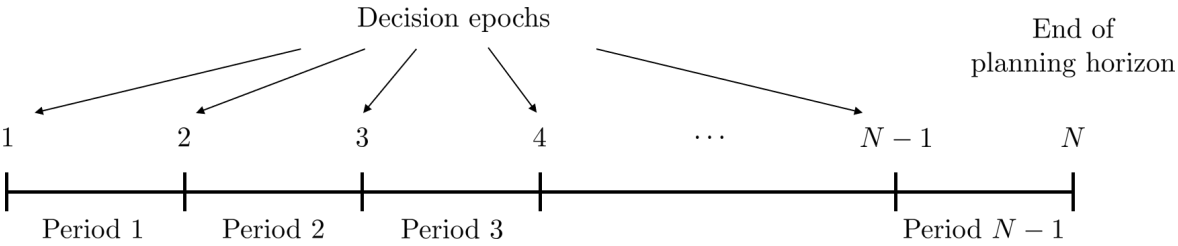


Figure 1.1: Decisions are made over a planning horizon divided into periods. The planning horizon may be finite (as shown) or infinite).

At a decision epoch, the decision maker[1] observes the state of the system, chooses an *action* and as a result of this choice, the system evolves to a new state according to a probability distribution that depends on the current state and the choice of action. After the system moves to the subsequent state, the decision maker receives a reward that depends on the starting state, action and possibly the subsequent state[2] These steps are repeated at each subsequent decision epoch. Figure 1.2 illustrates these events between two decision epochs. Such timelines are fundamental to developing a Markov decision process model. We formalize these notions in the following subsections.

1.1.1 Planning Horizons and Decision Epochs

The *planning horizon* is the time interval over which decisions are made. It may be *finite* or *infinite*. A *discrete-time*[3] Markov decision process model is either:

- A *Finite Horizon Model*, in which the planning horizon is a bounded interval divided into $N-1$ periods, or
- An *Infinite Horizon Model*, in which the planning horizon is unbounded and divided into an infinite number of periods.

Each period begins at a *decision epoch*, a point in time at which the decision maker observes the system state and chooses an action. A period ends immediately before the subsequent decision epoch. For example, each day at midnight, a retailer's inventory

[1]In this book, we use the expression *decision maker* to refer to a possibly animate entity who is making decisions. In the computer science literature, the decision maker is often referred to as an *agent* and in the engineering literature as a *controller* reflecting a reality in which an inanimate entity chooses and executes decisions.

[2]The reinforcement learning literature uses the acronym *SARSA* to refer to the sequence; *state, action, reward, state, action*.

[3]This book will not describe continuous time models.

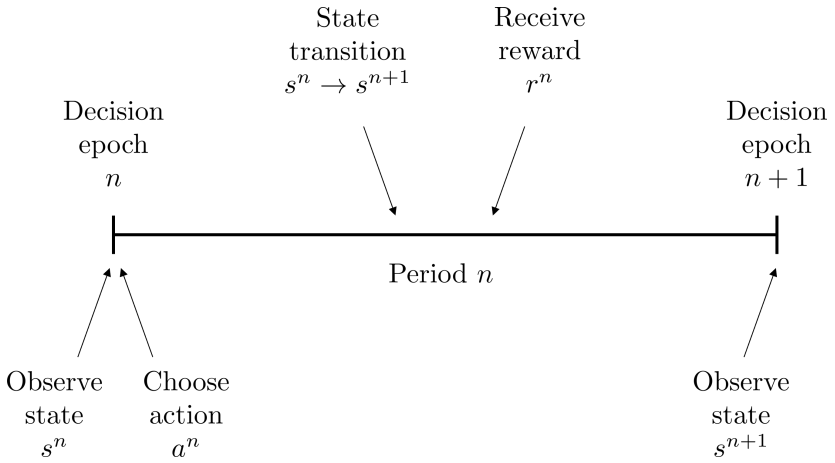


Figure 1.2: Timeline showing key events in period n. The decision maker observes state s^n at decision epoch n and chooses action a^n. The state transitions to state s^{n+1} according to the transition probabilities, the decision maker receives a reward r^n, and then the decision maker is then in the position to gets choose a new action at decision epoch $n+1$ after observing state s^{n+1}.

management system observes the stock level of all products and decides how many additional units of each to order from suppliers. In this case, the period is a day, and the decision epoch corresponds to midnight of that day.

When the planning horizon is of finite length, our convention is to divide it into $N-1$ periods with $N-1$ decision epochs. No decision is made at the end of the planning horizon, epoch N, which we refer to as the *terminal epoch*. It is included to evaluate the consequences of the decision at epoch $N-1$. Using $N-1$ also leads to cleaner formulas. Accordingly, our convention is to refer to finite horizon problems as $(N-1)$-period problems. In a limited number of finite horizon applications such as shortest path problems, decision epochs may index the order in which decisions are made, rather than pre-defined points in time. Chapter **??** focuses on finite horizon models, while Chapters **??** – **??** focus on infinite horizon models.

Throughout the book, when indexing "realizations", that is, states visited, actions implemented or rewards received in a particular epoch, epochs will be represented by superscripts. In contrast, specific states and actions within the set of possible states and actions will be represented using subscripts[4]. For transition probability functions and reward functions, we will indicate the epoch in the subscript (see Figure 1.1).

1.1.2 States

The state of the system contains all *relevant* information available to the decision maker at a decision epoch. By relevant, we mean that by knowing this information

[4]For example a^n will denote the action chosen at decision epoch n and the expression a_m will denote the m-th action from a set of possible actions.

and choosing an action, the transition probabilities and rewards are fully specified. Let S denote the set of all states and s denote a specific state in S. We refer to S as the *state space* and an $s \in S$ as a *state*. States are necessarily *exhaustive* and *mutually exclusive*. At each decision epoch the system must be in one state in S[5].

Elements of S may be scalar (e.g., the inventory level of a single product) or vector-valued (e.g., queue lengths of each priority class in a multi-class queuing system). They may be ordered (e.g., the number of people in a queue) or abstract (e.g., the configuration of a Tetris screen and the shape awaiting placement in the wall).

Although our primary focus in this book is on models with S being discrete and finite, there are many possible generalizations, including choosing S to be countably infinite or a subset (either bounded or unbounded) of finite-dimensional Euclidean space. Also, specifically in network or decision analysis settings, S may vary with the decision epoch, in which case one may use an epoch-specific state space S_n at epoch n. An alternative is simply to define S as the union of S_n over all decision epochs n, though this approach may unnecessarily expand the state space at each decision epoch (because some states may not be reachable at certain epochs). We will assume that S is independent of the decision epoch.

1.1.3 Actions

Denote by A_s the set of all actions available to the decision maker in state $s \in S$. We refer to A_s as an *action set* and each $a \in A_s$ as an *action*. Actions are mutually exclusive; the decision maker cannot choose two actions at the same time, although, as we will see below, the decision maker may specify a distribution over the set of actions.

Like the state space, each A_s may be a finite set, a countably infinite set, a subset of finite-dimensional Euclidean space, or an abstract set. The action set may be independent of s, as in an infinite capacity queuing control model in which the action is to decide whether to admit or not admit a job. Our development will focus on A_s being discrete and finite. Assuming the states and actions are ordered, we will denote by $a_{s,j}$ the j-th action in the s-th state.

Note that in some states, A_s may contain a single element corresponding to "no action". We refer to such states as *non-actionable*. These situations occur most often when the system reaches an absorbing state state Δ from which it cannot exit. In that case, $A_\Delta = \{\text{Do nothing}\}$. This occurs most often in episodic models (such as controlling a robot to move from an origin to a destination in a maze or in problems in which the decision maker can stop the system at any time).

1.1.4 Transition Probabilities

Let $p_n(j|s,a)$ denote the probability that the system state becomes j at decision epoch $n+1$ when the decision maker chooses action a in state s at decision epoch n for

[5]Unlike Schrödingers's cat, the system cannot occupy two different states at the same time.

$n = 1, 2, \ldots, N-1$. Note that several random events may occur throughout the period between decision epochs; the transition probability does not explicitly represent each one, but instead represents the net change in state. As a probability, $p_n(j|s,a)$ has the following properties:

$$\begin{aligned} p_n(j|s,a) &\geq 0 \quad \text{for all } j \in S, a \in A_s, s \in S \\ \sum_{j \in S} p_n(j|s,a) &= 1 \quad \text{for all } a \in A_s, s \in S. \end{aligned}$$

Note that in a non-actionable state, Δ, with a_Δ representing the "Do nothing" action, $p_n(\Delta|\Delta, a_\Delta) = 1$.

When S represents an uncountable subset of finite-dimensional Euclidean space, the transition probability may be represented by a probability density function. We will require this level of generality in Chapter **??** on partially observable models.

When the transition probabilities do not vary over decision epochs, we refer to them as *stationary*. For infinite horizon models presented in this book, we assume transition probabilities are stationary, and thus drop the subscript n.

1.1.5 Rewards

We consider two representations for a reward: $r_n(s,a,j)$ and $r_n(s,a)$. The quantity $r_n(s,a,j)$ denotes the reward received in period n when the decision maker chooses action a in state s at decision epoch $n = 1, 2, \ldots, N-1$ and the system transitions to state j at decision epoch $n+1$. The latter quantity, $r_n(s,a)$, has a similar interpretation, but is independent of the subsequent state. The choice of reward function depends on the application – usually one of these two forms better describes a specific context. The examples in Chapter **??** will illustrate both.

We view rewards as intrinsic parts of the model that arise naturally from the application. However, in reinforcement learning models, the modeler may need to construct an *artificial* reward function that conforms with the objectives of the task.

We assume $r_n(s,a,j)$ and $r_n(s,a)$ are scalar and real-valued. Generalizations include vector-valued or abstract rewards. For example, as the result of an action in a fantasy game, the decision maker may receive a sword, a shield and a vial of magic potion. Instead of keeping track of this bundle of goods in the reward function, the decision maker will assign a numerical value to each item and be indifferent between collections of items with the same total value.

We refer to these quantities as *rewards* because we model a decision maker seeking to *maximize* rewards. We represent *costs* as negative rewards to allow for a decision maker who seeks to minimize costs. Consequently, minimizing costs corresponds to maximizing rewards. To avoid awkward minus signs in expressions, we will on some occasions when minimizing costs, instead write $c_n(s,a)$ to represent the cost of being state s and choosing action a or $c_n(s,a,j)$ to include the possibility that the cost depends on the state at the next decision epoch j.

When using optimality criteria based on expected rewards[6], the quantity $r_n(s,a)$ may represent the *expected* reward in period n where the expectation is taken over the possible states at decision epoch $n+1$:

$$r_n(s,a) = \sum_{j \in S} r_n(s,a,j) p_n(j|s,a). \tag{1.1}$$

For discrete time models, the model formulation does not account for how the reward is accumulated throughout a period between two decision epochs. For example, in an inventory control model with weekly decision epochs, the inventory levels may change during the week resulting in varying holding costs between decision epochs. A discrete time formulation accumulates all of these costs and summarizes them in the reward function for that one period.

In finite horizon models, we specify a *terminal reward* or *scrap value* $r_N(s)$ to represent the consequence of ending the planning horizon in state s. In infinite horizon models, we omit the terminal reward. Furthermore, we delete the subscript n from the reward function in the infinite horizon setting, since our focus in that case is on *stationary* rewards.

1.1.6 Markov Decision Processes and Decision Trees

A decision tree provides a visual display of the basic model components. In Figure 1.3 square boxes represent states, arcs from boxes to circles represent possible actions in each state, arcs from circles to subsequent states denote transitions that occur according to a transition probability and result in a reward.

It should be evident from the decision tree representation that, in general, there is an exponential explosion in the possible trajectories through the tree as we increase the length of the planning horizon.

1.2 Derived Objects

A Markov decision process is fully specified by the five model components described in the previous section:

- the planning horizon N (which may be infinite),
- the set of states S,
- the sets of actions A_s for each $s \in S$,
- the transition probabilities $p_n(j|s,a)$, and

[6] In some applications, particularly in economic contexts, a decision maker seek to maximize expected *utility* or some other risk sensitive criterion.

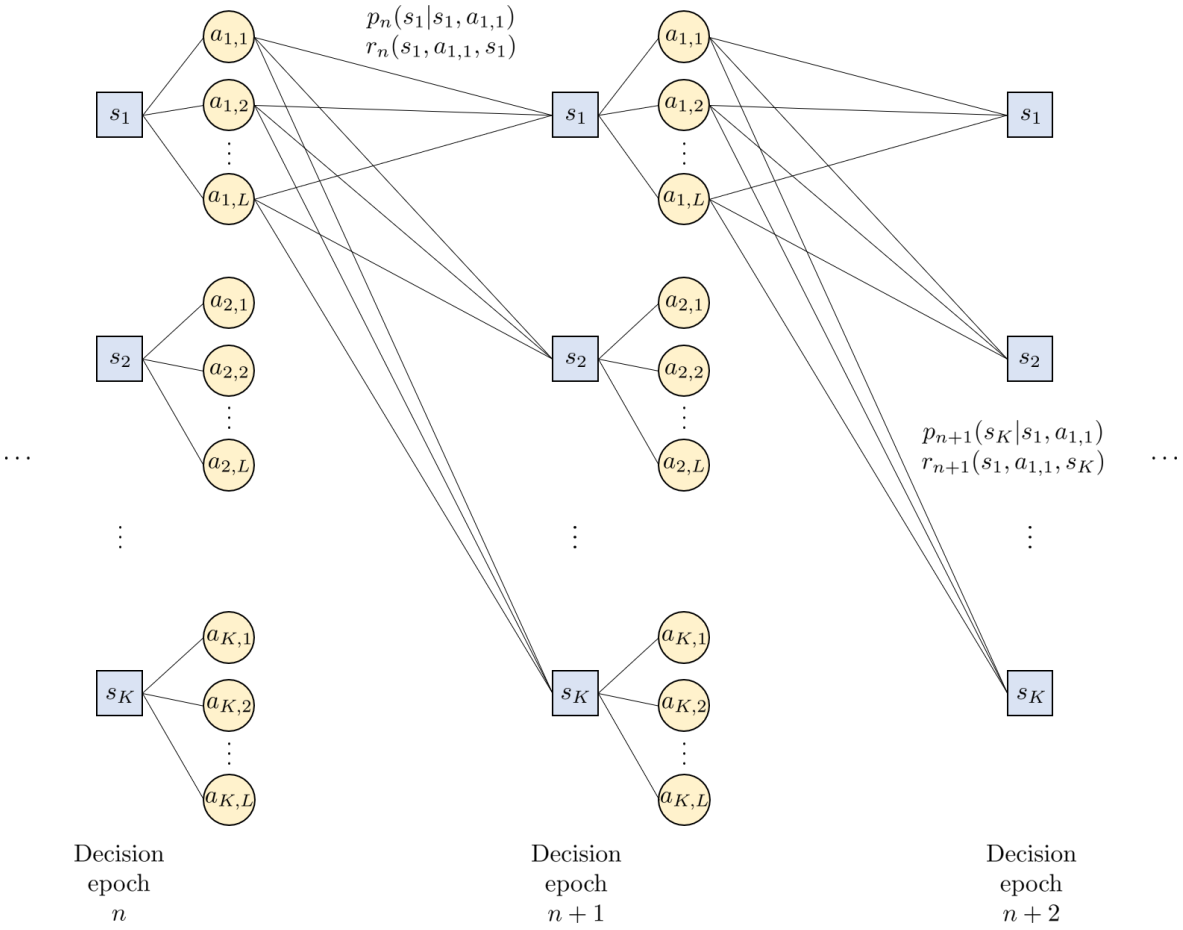


Figure 1.3: Representing a Markov decision process by a decision tree.

- the rewards $r_n(s, a, j)$ or $r_n(s, a)$.

In this section, we describe decision rules, policies, derived stochastic processes and reward processes, which are not part of the basic formulation, but are fundamental concepts derived from the basic model components.

1.2.1 Decision Rules

A *decision rule* describes both the information and mechanism a decision maker uses to select an action in a given state at a single, specific decision epoch. Decision rules can be classified on the basis of two independent dimensions:

- Information: Markovian or history-dependent
- Mechanism: Deterministic or randomized

Information

The Markovian vs. history-dependent dichotomy describes the *information* used by the decision maker when choosing an action. A *Markovian* decision rule uses only the state at the current decision epoch to select actions, while a *history-dependent* decision rule considers some or all of the previous states and actions up to and including the current state when choosing an action. That is, at decision epoch n, a Markovian decision rule is a function of s^n and a history-dependent decision rule is a function of a *trajectory* $(s^1, a^1, s^2, \ldots, a^{n-1}, s^n)$[7]. Thus, a Markovian decision rule is a special case of a history-dependent decision rule in which the history is summarized in a single state. We write

$$H_n = \{h^n = (s^1, a^1, s^2, \ldots, a^{n-1}, s^n) \mid s^1 \in S, a^1 \in A_{s^1}, s^2 \in S, \ldots, a^{n-1} \in A_{s^{n-1}}, s^n \in S\} \tag{1.2}$$

as the set of all histories, h^n, leading up to decision epoch n. Note that $H_1 = S$. While the past sequence of rewards could also be explicitly included in the history, we view its inclusion as redundant since the history of states and actions alone allows us to reconstruct the past rewards through the reward functions[8]. Note that $h^1 = s^1$ and for $n = 2, 3, \ldots, N$,

$$h^n = (h^{n-1}, a^{n-1}, s^n). \tag{1.3}$$

We will use this recursion explicitly in Section **??**.

Mechanism

The deterministic vs. randomized dichotomy describes the *mechanism* used to select an action at a given decision epoch. A *deterministic* decision rule selects an action with certainty, while a *randomized* decision rule selects an action according to a specified probability distribution. A deterministic decision rule is a special case of a randomized decision rule corresponding to a degenerate probability distribution[9].

In the reinforcement learning literature, randomized decision rules arise naturally in simulation-based algorithms which use -"exploration". We will develop this concept in Chapters **??** and 2.

Types of decision rules

The two dimensions described above lead to four classes of decision rules.

[7]Recall our convention that superscripts refer to the state and/or action chosen at decision epoch n. Also, history-dependent decision rules need not consider the entire history, but a subset of past actions and states.

[8]In model-free reinforcement learning models, a functional form for $r_n(s, a, j)$ may not be known. In this case the history might need to be augmented to include the reward.

[9]A degenerate probability distribution places all of the probability on one action.

- A *Markovian deterministic* decision rule (MD), d_n, is a function from states to actions. More formally, $d_n(s^n) = a^n$ denotes the decision rule that at decision epoch n chooses action $a^n \in A_{s^n}$ when the system is in state $s^n \in S$.

- A *history-dependent deterministic* decision rule (HD), d_n, is a function from the set of histories to actions. More formally, $d_n(h^n) = a^n$ denotes the decision rule that chooses action $a^n \in A_{s^n}$ when the system history is $h^n \in H_n$ and s^n is the state at decision epoch n. The subtlety here is that although the decision rule's action choice may vary with history, it can only choose actions from the set A_{s^n} at epoch n, which depends only on the state at decision epoch n. This means that given two different histories h^n and $\hat{h}^n$ that both arrive at state s^n, $d_n(h^n)$ and $d_n(\hat{h}^n)$ may choose different actions, but each action will be chosen from the same action set A_{s^n}.

- A *Markovian randomized* decision rule (MR), d_n, is a mapping from the set of states to the set of probability distributions over the action set. More formally, it specifies a probability distribution

  $$w^n_{d_n}(a^n|s^n) := P[Y_n = a^n],$$

  where the random variable Y_n denotes the action $a^n \in A_{s^n}$, chosen at decision epoch n, when the system state is s^n.

- A *history-dependent randomized* decision rule (HR), d_n, is a mapping from the set of histories to the set of probability distributions over the action set. It specifies a probability distribution

  $$w^n_{d_n}(a^n|h^n) := P[Y_n = a^n], \tag{1.4}$$

  where the random variable Y_n denotes the action $a^n \in A_{s^n}$, chosen at decision epoch n, when the system history is $h^n \in H_n$ and the system state is s^n.

We denote these classes of decision rules as D^{MD}, D^{HD}, D^{MR} and D^{HR}, respectively.

It is important to note that because of the ability to construct history-dependent decision rules, the Markov decision process formulation gives rise to a system that might not evolve in a Markovian fashion. The expression "Markov decision process" refers to the fact that rewards and transition probabilities depend on the past only through the state and action at the present decision epoch. It does not, however, mean that every stochastic process generated by this model is a Markov chain. See Section 1.2.3 below for more details on this point.

Subsequent chapters will establish the optimality of Markovian deterministic decision rules so it will be unnecessary to consider randomized decision rules when seeking optimal policies. However, randomized decision rules are fundamental in linear programming formulations of the infinite-horizon Markov decision process model (Chapters **??**-**??**), exploration-based simulation algorithms (Chapter **??**) and policy-based reinforcement learning algorithms (Chapter 2).

1.2.2 Policies

A *policy* π, also referred to as a *contingency plan* or *strategy*, is a sequence of decision rules, one for each decision epoch. In finite horizon models, we write $\pi = (d_1, d_2, \ldots, d_{N-1})$. In infinite horizon models, $\pi = (d_1, d_2, \ldots)$. In the classical Markov decision process setting, once an optimality criterion has been specified, the decision maker's goal is to find an optimal policy – one that maximizes or minimizes the particular criterion.

The four classes of decision rules defined in Section 1.2.1 result in four classes of policies, Π^{MD}, Π^{HD}, Π^{MR} and Π^{HR}. In addition, we define stationary deterministic (SD) and stationary randomized (SR) policies, denoted by the classes Π^{SD} and Π^{SR}, respectively. A *stationary policy* chooses the same decision rule at every decision epoch and will be represented as $\pi = (d, d, \ldots)$[10].

Some comments about policies follow:

1. The class Π^{HR} denotes the most general class of policies; all policies we consider in the book are in this class. We require this level of generality to define optimality, but we will often not require this level of generality to find optimal policies. For example, in finite horizon models it will be sufficient to restrict one's search for an optimal policy to the class of Markovian deterministic policies (see Section 1.4).

2. Policies in Π^{HR} generally cannot be implemented in long finite-horizon or infinite horizon models because storage requirements increase exponentially with respect to the planning horizon length. Fortunately, in finite horizon models, there exist optimal policies in this class that are Markovian and deterministic, and in infinite horizon models, there exist optimal policies that are stationary and deterministic.

3. Stationary policies are most relevant to infinite horizon models. Proofs and methods in Chapters **??** – **??** will use the result that there exist stationary deterministic policies that are optimal in the class of history-dependent randomized policies under several different optimality criteria.

4. In finite horizon models, a Markovian deterministic policy may be characterized by a *lookup table*. A lookup table is an array in which rows correspond to states, columns correspond to decision epochs and an entry indicates which action the policy selects in that state at that decision epoch. Although a useful conceptual framework, a lookup table may be an impractical method of encoding a policy in models with a large number of states. Chapter **??** on approximate dynamic programming provides some approaches for addressing this challenge.

5. Figure 1.4 summarizes the relationship between policy classes. Although not represented in the figure, there may exist policies that are stationary and history-dependent (i.e., non-Markovian). An example is a policy that makes decisions

[10] In most cases, a stationary policy is necessarily Markovian.

based on the current state as well as the starting state. However, these examples represent "edge cases" and are uncommon. Thus, in this book, we only consider stationary policies within the Markovian class.

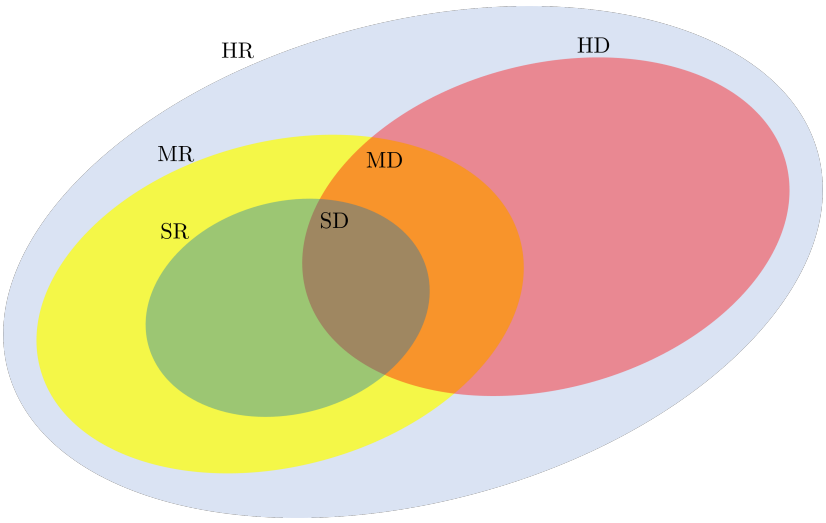


Figure 1.4: Relationships between policy classes. Labels immediately outside of an oval denote the entire oval, whereas labels inside the intersection of two ovals denote the entire intersection between them. For example, SD is the intersection of SR and HD. MD is the intersection of MR and HD, which includes SD (i.e., any region with orange tint). **(grayscale?)**

1.2.3 Derived stochastic processes

Once a policy is chosen and a probability distribution over the starting state of the process is specified, the probabilistic evolution of a Markov decision process is completely determined. Let the random variable X_1 denote the initial state of the Markov decision process and let $\gamma(s) := P[X_1 = s]$ denote the initial state probability distribution, which we assume is independent of policy choice. When, as in most applications, the system starts in a specific state at decision epoch 1, say s^1, we write $\gamma(s) = 1$ for $s = s^1$ and $\gamma(s) = 0$ for $s \neq s^1$.

Let π denote a policy for either a finite or infinite horizon problem. Let the random variable Y_1 denote the action chosen by d_1 at decision epoch 1. There are two potential sources of variability that affect Y_1:

1. Variability in X_1, which occurs when X_1 is random, and
2. Variability in action choice in X_1, which occurs if d_1 is a randomized decision rule.

Once Y_1 is chosen, the next state, X_2, is random with a distribution determined by the transition probabilities. This is true even if X_1 and Y_1 are not random, i.e., when the system starts in a specific state s^1 and d_1 is deterministic, respectively. Consequently, the second action Y_2 will be random as well, and so on. Thus, a policy and initial state distribution generate the stochastic process

$$(X_1, Y_1, X_2, Y_2, \ldots, X_{N-1}, Y_{N-1}, X_N) \tag{1.5}$$

in a finite horizon model and

$$(X_1, Y_1, X_2, Y_2, \ldots) \tag{1.6}$$

in an infinite horizon model.

We introduce the following important terminology:

A realization of (1.5) and (1.6) is referred to as a *trajectory*.

Policies induce probability distributions over trajectories.

Some comments about policies with respect to derived stochastic processes follow:

1. When $\pi \in \Pi^{\text{MR}}$, the stochastic process is a discrete time Markov chain. See Exercise 4.

2. When π is history-dependent, the stochastic process need not be a Markov chain. That is, at each decision epoch, given a state and action, the transition probabilities may depend on all or some of the past. See Exercise 4.

3. A policy $\pi = (d_1, d_2, \ldots)$[11] induces a probability distribution p^π on trajectories that correspond to realizations of $(X_1, Y_1, X_2, Y_2, \ldots)$. To determine it requires enumerating realizations and multiplying the transition probabilities and action randomization distributions corresponding to each realization as follows:

   (a) For $\pi \in \Pi^{\text{HR}}$,

   $$\begin{aligned} &p^\pi(s^1, a^1, s^2, a^2, s^3, \ldots) \\ &\quad = \gamma(s^1) w^1_{d_1}(a^1|s^1) p_1(s^2|s^1, a^1) w^2_{d_2}(a^2|h^2) p_2(s^3|s^2, a^2) w^3_{d_3}(a^3|h^3) \ldots \end{aligned} \tag{1.7}$$

   where $h^1 = s^1$, $h^2 = (s^1, a^1, s^2)$, and so on. Recall that $w^n_{d_n}(a|h)$, defined in Section 1.2.1, denotes the probability that decision rule d_n chooses action a in state s at epoch n given history h.

   Note that the entire history first enters into this expression through the randomization distribution at decision epoch 2 and subsequently, but not through the transition probabilities, which only depend on the state and

[11] The quantity $d_n(\cdot)$ denotes the decision rule chosen by π at decision epoch n.

action. If a deterministic policy was used, the history would also only enter through the decision rule as in (1.7), but the decision rule would explicitly choose the action in each transition probability.

(b) For $\pi \in \Pi^{\text{MR}}$,

$$\begin{aligned} &p^\pi(s^1, a^1, s^2, a^2, s^3, \ldots) \\ &\quad = \gamma(s^1) w^1_{d_1}(a^1|s^1) p_1(s^2|s^1, a^1) w^2_{d_2}(a^2|s^2) p_2(s^3|s^2, a^2) w^3_{d_3}(a^3|s^3) \ldots \end{aligned} \tag{1.8}$$

This expression is similar to (1.7), except that histories are replaced with states. Stationary versions of this expression provide the basis for policy gradient methods, which we explore in Chapter 2.

(c) For $\pi \in \Pi^{\text{MD}}$

$$p^\pi(s^1, a^1, s^2, a^2, s^3, \ldots) = \gamma(s^1) p_1(s^2|s^1, a^1) p_2(s^3|s^2, a^2) \ldots \tag{1.9}$$

since $d_1(s^1) = a^1$, $d_2(s^2) = a^2$, and so on. In this case, we need only multiply transition probabilities with each other (and the initial state distribution) to find the probability distribution of the stochastic process generated by π.

1.2.4 Reward Processes

Section 1.2.3 showed that a policy π generates a stochastic process of states and actions represented by (1.5) in finite horizon models and (1.6) for infinite horizon models with distribution $p^\pi(\cdot)$ defined in (1.7) in the most general case. These processes generate corresponding stochastic processes of rewards

$$(r_1(X_1, Y_1, X_2), r_2(X_2, Y_2, X_3) \ldots, r_{N-1}(X_{N-1}, Y_{N-1}, X_N), r_N(X_N)) \tag{1.10}$$

in finite horizon models, and

$$(r_1(X_1, Y_1, X_2), r_2(X_2, Y_2, X_3), \ldots) \tag{1.11}$$

in infinite horizon models.

In greater generality we represent these stochastic processes by either

$$\mathcal{R}_N := (R_1, R_2, \ldots, R_N) \tag{1.12}$$

or

$$\mathcal{R}_\infty := (R_1, R_2, \ldots) \tag{1.13}$$

where the random variable R_n represents the reward received in period n.

Letting r^n denote a realization of R_n, we write expressions of the form

$$P[\mathcal{R}_N = (r^1, r^2, \ldots, r^n)] \quad \text{and} \quad P[\mathcal{R}_\infty = (r^1, r^2, \ldots)].$$

When reward processes arise in a Markov decision process, the policy dependent probability $p^\pi(\cdot)$ replaces the generic probability $P[\cdot]$.

Simulation or observing repeated experiments in the real world will generate many different sequences of rewards. The probability distribution of $\mathcal{R}_N$ and $\mathcal{R}^\infty$ describes the likelihood of observing each sequence. In this book, we will have little occasion to explicitly compute such probability distributions, but they will provide the basis for interpreting several expressions. Most calculations will avoid this onerous task by evaluating distributions and expectations sequentially. Example **??** shows how to derive the distribution of $\mathcal{R}_N$ in an Markov decision process by determining which states and actions map into the same reward sequence.

In our discussion of values in the next section, it is sufficient to work directly with the reward process described above but in the simulation methods in Chapters **??** and 2 we consider realizations (trajectories) that include states, actions and rewards of the form[12]

$$(X_1, Y_1, R_1, X_2, Y_2, \ldots, R_N) \quad \text{and} \quad (X_1, Y_1, R_1, X_2, Y_2, \ldots).$$

Thus a realization of the process will be written as either

$$(s^1, a^1, r^1, s^2, \ldots, r^N) \quad \text{or} \quad (s^1, a^1, r^1, s^2, \ldots).$$

Following convention, we refer to a stochastic process together with a reward function as a *reward process*. When the stochastic process is a Markov chain, we refer to it as a *Markov reward process*. Markov reward processes most often arise as the sequence of rewards of a Markovian policy in a Markov decision process.

1.2.5 Assigning a value to a reward process

Next, we show how reward processes lead to decision making criteria. As noted in Section 1.1.5, we will assume that reward functions $r_n(s, a, j)$ are real-valued so that each R_n, as defined implicitly through (1.12) or (1.13), is real-valued. Thus, the random reward sequence $\mathcal{R}_N = (R_1, R_2, \ldots, R_N)$ takes values in $\Re^N$ and $\mathcal{R}_\infty = (R_1, R_2, \ldots)$ takes values in $\Re^\infty$.

Expected utility

To assign a value to a sequence of rewards, a decision maker may use *expected utility*. Let $u(\cdot)$ denote a real-valued function on $\Re^N$ or $\Re^\infty$, and let $E[\cdot]$ denote expectation with respect to the probability distribution of $\mathcal{R}_N$ or $\mathcal{R}_\infty$. The expected utility of these reward sequences equals $E[u(R_1, R_2, \ldots, R_N)]$ and $E[u(R_1, R_2, \ldots)]$, respectively.

[12]Technically speaking when $r_n(\cdot, \cdot, \cdot)$ is a function of the current state, the current action and the next state, we need to observe X_{n+1} before we can evaluate the reward. So technically, at least, we should write the stochastic process $(X_1, Y_1, X_2, R_2, Y_2, \ldots, X_N, R_N)$.

Often utility functions are additive, so

$$u(R_1, R_2, \ldots, R_N) = \sum_{n=1}^{N} u_n(R_n) \tag{1.14}$$

for N finite or infinite, where $u_n(\cdot)$ is a real-valued function for each n. When N is finite, the expected utility becomes

$$E[u(\mathcal{R}_N)] = E[u(R_1, R_2, \ldots, R_N)] = E\left[\sum_{n=1}^{N} u_n(R_n)\right]. \tag{1.15}$$

When N is infinite, we define[13].

$$E[u(\mathcal{R}_\infty)] = E[u(R_1, R_2, \ldots)] := \lim_{N\to\infty} E\left[\sum_{n=1}^{N} u_n(R_n)\right]. \tag{1.16}$$

Interpreting utilities

Utilities $u_n(\cdot)$ encode a decision maker's attitude towards both risk (variability) and timing of rewards. The most common choice for $u_n(\cdot)$ is

$$u_n(r) = \lambda^{n-1} r$$

where $0 \leq \lambda \leq 1$ so that

$$E[u_n(R_n)] = \lambda^{n-1} E[u_n(R_n)].$$

We refer to the quantity λ as the *discount factor*. When $\lambda = 1$ the decision maker is indifferent to the timing of rewards.hen $\lambda < 1$ the decision maker prefers receiving rewards sooner than later. For example, if $\lambda = 0.95$, a decision maker would be indifferent between receiving one unit next period and 0.95 units now.

When using $E[R_n]$ to make decisions, a decision maker is said to be *risk neutral*. For example, a risk neutral decision maker would be indifferent between the following two gambles since they have the same expected value:

- Gamble A: win \$100 with probability 1, or
- Gamble B: win \$0 with probability 0.5 and \$200 with probability 0.5.

A *risk seeking* decision maker would prefer Gamble B and a *risk averse* decision maker would prefer Gamble A. Convex increasing utility functions such as r^2, correspond

[13] In (1.16), we would prefer to take the limit inside the expectation but that limit need not exist. An example below explores this point further

to risk seeking decision making while concave increasing utility functions such as $\sqrt{r}$ correspond to risk averse decision making[14].

The Markov decision process models we consider in this book will focus on decision making by risk neutral decision makers, so utility is replaced by rewards or discounted rewards directly. However, it is important to be aware that other utility functions apply, and optimal policies with respect to one utility function will not necessarily be optimal for a different utility function.

As an example, consider coaching decisions in a football game. If a team is behind near the end of the game, its coach may be risk seeking in an attempt to catch up. Alternatively, if a team is ahead, its coach may choose to make more conservative decisions, based on a risk averse utility function, in order to hold on to the lead.

1.3 Optimality Criteria: Transforming a Markov decision process into a Markov decision problem

Up to this point, we have formulated the Markov decision process without considering the decision maker's preferences for reward sequences generated by different policies. In the following sections we define and comment on the following three optimality criteria that are commonly used to evaluate and compare policies:

- expected total reward,
- expected discounted reward, and
- long-run average reward.

The expected total reward criterion applies to both finite and infinite horizon models, while the latter two are most often used in infinite horizon models. A policy is said to be *optimal* when it maximizes the appropriate criterion over the set of all history-dependent randomized policies. To be precise, we use the expression *Markov decision problem* to correspond to a Markov decision process *together* with a optimality criterion[15]. However, as with most of the published literature, we will continue to use the general phrase *Markov decision process* to refer to both cases, whether an optimality criterion is included or not.

1.3.1 Expected Total Reward

We define the expected total reward of a reward sequence and policy. Because of technical considerations we distinguish finite horizon from infinite horizon models.

[14]Which would better reflect your attitude towards gambles? Why?

[15]We emphasize that a Markov decision process is defined independent of the optimality criterion.

Finite horizon

Define the *expected total reward* of a finite sequence of random rewards $(R_1, R_2, \ldots, R_N)$ by

$$E\left[\sum_{n=1}^{N} R_n\right] \tag{1.17}$$

where the expectation is over the distribution of reward sequences. Since many reward sequences might map into the same total reward, in theory, when $R - n$ is discrete, we can compute this quantity by first enumerating all possible values for the sum and and then accumulating the probabilities of the trajectories with the same sum.

As noted in Section 1.2.3, a policy π generates a random sequence

$$(X_1, Y_1, X_2, Y_2, \ldots, X_{N-1}, Y_{N-1}, X_N)$$

of states and actions and consequently generates a random sequence of rewards by setting $R_n = r_n(X_n, Y_n, X_{n+1})$ or $R_n = r_n(X_n, Y_n)$ for $n = 1, 2, \ldots, N-1$ and $R_N = r_N(X_N)$.

Thus we can define the expected total reward of a policy $\pi \in \Pi^{\text{HR}}$ by

Finite horizon expected total reward:

$$v^\pi(s) := E^\pi\left[\sum_{n=1}^{N} R_n \,\middle|\, X_1 = s\right], \tag{1.18}$$

where the expectation is with respect to the probability distribution of random rewards that result from different realizations of the stochastic process generated by π with $X_1 = s$ (Section 1.2.4).

We refer to the expected total reward of policy π, $v^\pi(s)$, as its *value*. Implicit in this notation for finite horizon models is that $v^\pi(s)$ gives the value for a model with $N - 1$ decision epochs and terminal epoch N[16]. We will call the function $v^\pi(\cdot)$ the *value function*. Note that we will most often see $v^\pi(s)$ written as either

$$v^\pi(s) = E^\pi\left[\sum_{n=1}^{N-1} r_n(X_n, Y_n, X_{n+1}) + r_N(X_n) \,\middle|\, X_1 = s\right] \tag{1.19}$$

or

$$v^\pi(s) = E^\pi\left[\sum_{n=1}^{N-1} r_n(X_n, Y_n) + r_N(X_n) \,\middle|\, X_1 = s\right] \tag{1.20}$$

depending on the form of r_n. To simplify notation, we will often write $E_s^\pi[\cdot]$ instead of $E^\pi[\cdot|X_1 = s]$.

[16]Some authors use the notation $v_N^\pi(s)$ where N denotes the planning horizon.

Simulating the expected total reward

The following "algorithm" describes how to use simulation to estimate $v^\pi(s)$. It generates a single replicate for a single starting state s^1. Since we often need $v^\pi(s)$ for all $s \in S$, we would repeat this for all states. *Monte Carlo estimates* (Chapter **??**) are obtained averaging $v^\pi(s)$ over many replicates.

Algorithm 1.1. Simulating a single replicate of a finite horizon value function for a deterministic policy.

1. **Initialization:**
   (a) Specify $\pi = (d_1, d_2, \ldots, d_{N-1})$.
   (b) Choose $s^1 \in S$.
   (c) Set $n = 1$ and $v = 0$.
2. **Iterate:** While $n \leq N - 1$:
   (a) Set $a^n = d(s^n)$,
   (b) Sample s^{n+1} from $p_n(\cdot|s^n, a^n)$.
   (c) Set $r^n = r_n(s^n, a^n, s^{n+1})$.
   (d) $v \leftarrow v + r^n$
   (e) $n \leftarrow n + 1$
3. $v(s^1) \leftarrow v + r_N(s^N)$

Some comments about simulating the expected total reward follow:

1. Although you might not intend to simulate this process, this algorithm describes how the process would evolve in an implementation.
2. We explicitly use the functions $p_n(\cdot|\cdot, \cdot)$ and $r_n(\cdot, \cdot, \cdot)$ in the above procedure. When this is done, the approach is said to be *model-based*. Alternatively if s^{n+1} in step 2(b) and r^n in step 2(c) were outputs of a simulation or real-time process, the approach is said to be *model-free*. This distinction will be most significant in Chapters **??** and 2.
3. If we wished to instead simulate a randomized policy, we would replace step 2(a) by "sample a^n from $w^n_{d_n}(\cdot|s^n)$".
4. Obvious modifications would be required to evaluate a history-dependent policy. This would be impractical if N is large and the decision rules depended on the whole past.

Chapter **??** will develop a straightforward approach to compute $v^\pi(s)$ numerically or analytically for any $s \in S$ and any π. However, in problems with large state spaces, simulation and approximation may be preferable.

Infinite horizon models

We define the expected total reward (value) of a policy π in an infinite horizon model for all $s \in S$ by

Infinite horizon expected total reward:

$$v^\pi(s) := \lim_{N\to\infty} E_s^\pi \left[\sum_{n=1}^{N} R_n \right]. \tag{1.21}$$

Note that we define the expected total reward of policy as the limit of the expected value of it finite sums. This is because

$$\lim_{n\to\infty} \sum_{n=1}^{N} R_n \tag{1.22}$$

need not exist so that its expectation may be undefined. We can write (1.21) in terms of the reward function directly as

$$v^\pi(s) = \lim_{N\to\infty} E_s^\pi \left[\sum_{n=1}^{N} r_n(X_n, Y_n, X_{n+1}) \right] \quad \text{or} \quad v^\pi(s) = \lim_{N\to\infty} E_s^\pi \left[\sum_{n=1}^{N} r_n(X_n, Y_n). \right]$$

In contrast to the finite horizon setting, we now have to be concerned with whether the above limit exists. Some possible limiting behaviors for this (or any) sequence include:

- **Convergence:** occurs when $E[R_n]$ decreases sufficiently quickly or becomes zero eventually,
- **Divergence:** occurs when $E[R_n]$ remains sufficiently positive or negative,
- **Oscillation:** occurs when $E[R_n]$ alternates between positive and negative values and does not die out.

When using the expected total reward criterion, the infinite horizon Markov decision process literature has focused on models in which the limit in (1.21) exists. Most examples of this kind are either *episodic models* or *optimal stopping problems*, which we examine in Chapters **??**, **??** and 2.

An illustrative example.

The following example illustrates the above definition of the expected total reward in infinite horizon models.

Example 1.1. Consider the Markov reward process depicted in Figure 1.5. It contains three states $S = \{s_1, s_2, s_3\}$. Rewards and transition probabilities are represented in brackets below the arcs. The first number in the bracket represents the reward received if that transition takes place and the second number represents the probability of that transition. Thus, from state s_1 the system transitions to state s_2 or s_3 with probability 0.5 and generates a reward of 0. From state s_2 the Markov chain transitions to state s_3 with certainty and generates a reward of 1 and from state s_3 the system transitions to state s_2 with certainty and generates a reward of -1. No other transitions are possible.

We now consider the calculation of $v(s_1)$. There are two possible realizations of the Markov chain describing the state at epoch n, namely

$$(s_1, s_2, s_3, s_2, s_3, \ldots) \quad \text{or} \quad (s_1, s_3, s_2, s_3, s_2, \ldots).$$

Each occurs with probability 0.5. The corresponding Markov reward process generates sequences

$$\rho_1 := (0, -1, 1, -1, 1, \ldots) \quad \text{and} \quad \rho_2 := (0, 1, -1, 1, -1, \ldots).$$

each occurring with probability 0.5. Thus we can write

$$P[\mathcal{R}_\infty = \rho_1 | X_1 = s_1] = P[(\mathcal{R}_\infty = \rho_2 | X_1 = s_1] = 0.5.$$

The sequence of **total rewards** corresponding to each realization are

$$\sigma_1 := (0, -1, 0, -1, 0, \ldots) \quad \text{and} \quad \sigma_2 := (0, 1, 0, 1, 0, \ldots).$$

For finite N even, the total reward

$$P\left[\sum_{n=1}^{N} R_n = -1 \;\middle|\; X_1 = s_1\right] = P\left[\sum_{n=1}^{N} R_n = 1 \;\middle|\; X_1 = s_1\right] = 0.5$$

while for N odd

$$P\left[\sum_{n=1}^{N} R_n = 0 \;\middle|\; X_1 = s_1\right] = 1.$$

Next we let $N \to \infty$. Since both possible total reward series oscillate, neither σ_1 or σ_2 has a limit. Hence the quantity $\lim_{N\to\infty} \sum_{n=1}^{N} R_n$ does not exist, so that the expression

$$E\left[\lim_{N\to\infty} \sum_{n=1}^{N} R_n \;\middle|\; X_1 = s_1\right]$$

is not well defined. However for any N,

$$E\left[\sum_{n=1}^{N} R_n \;\middle|\; X_1 = s_1\right] = 0,$$

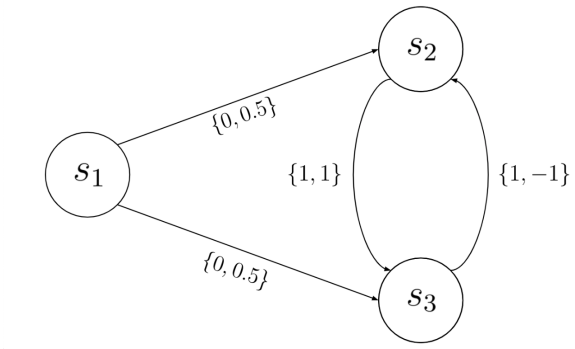


Figure 1.5: Markov reward process analyzed in Example 1.1

so

$$\lim_{N\to\infty} E\left[\sum_{n=1}^{N} R_n \;\middle|\; X_1 = s_1\right] = 0$$

so that the limit in (1.21) exists.

Thus we defined the expected total reward in an infinite horizon model as the "limit of expected partial sums" as opposed to the "expectation of the limit of partial sums".

Note we can pass the limit inside the expectation by considering the *lim inf* and *lim sup* of the sequence of partial sums. These quantities always exist and are well defined. Thus, in this example

$$P\left[\limsup_{N\to\infty} \sum_{n=1}^{N} R_n = 0 \;\middle|\; X_1 = s_1\right] = P\left[\limsup_{N\to\infty} \sum_{n=1}^{N} R_n = 1 \;\middle|\; X_1 = s_1\right] = 0.5$$

so that

$$E\left[\limsup_{N\to\infty} \sum_{n=1}^{N} R_n \;\middle|\; X_1 = s_1\right] = 0.5.$$

Moreover

$$E\left[\liminf_{N\to\infty} \sum_{n=1}^{N} R_n \;\middle|\; X_1 = s_1\right] = -0.5.$$

Optimal policies

We say that a policy π^* is *optimal* if

$$v^{\pi^*}(s) \geq v^{\pi}(s) \tag{1.23}$$

for all $s \in S$ and $\pi \in \Pi^{\text{HR}}$. Chapter **??** analyzes finite horizon Markov decision process models and Chapter **??** considers infinite horizon models under this optimality

criterion.

1.3.2 Expected Total Discounted Reward

In contrast to the expected total reward criterion, the expected total discounted reward criterion accounts for the "time value of money" – that is, receiving a reward at some future epoch is worth less than receiving an identical reward now. This is the most widely used criterion in infinite horizon models.

Finite horizon models

We define the *expected total discounted reward* of the reward sequence $\mathcal{R}_\infty = (R_1, R_2, \ldots)$ by

$$E\left[\sum_{n=1}^{N} \lambda^{n-1} R_n\right] \tag{1.24}$$

where the expectation is respect to the distribution of the sequence of rewards and the quantity $0 \leq \lambda < 1$ is the *discount factor*[17]. In finite horizon models, discounting has no impact on theory or algorithms, but may affect a decision maker's preference for policies. For example, with a discount factor close to 0, a decision maker will prefer actions that lead to larger immediate rewards, in contrast to a situation with a discount factor close to 1 when future rewards would be of greater significance.

Analogous to the expected total reward case, the expected total discounted reward of policy π, for each $s \in S$, is written

$$v_\lambda^\pi(s) := E_s^\pi\left[\sum_{n=1}^{N} \lambda^{n-1} R_n\right], \tag{1.25}$$

where the expectation is respect to the probability distribution of random rewards that result under different realizations of the stochastic process determined by π as described in Section 1.2.4.

Infinite horizon models

Discounting plays a fundamental role in the application and analysis of infinite horizon models and is the most studied optimality criterion. In infinite horizon models, the expected total discounted reward of policy π for each $s \in S$ is given by

[17]Recall the discount factor λ explicitly captures the present value of one unit of reward received one time period in the future.

Expected discounted reward:

$$v_\lambda^\pi(s) := \lim_{N\to\infty} E_s^\pi \left[\sum_{n=1}^{N} \lambda^{n-1} R_n\right]. \tag{1.26}$$

Unlike the expected total reward case, we do not require special model structure for the infinite sum (1.26) to converge, only that $E[\lambda^{n-1}R_n]$ decays sufficiently quickly. To ensure this we assume throughout the book that rewards are bounded. This means that there exists a finite M for which

Bounded rewards:

$$|r(s,a,j)| \leq M \text{ for all } a \in A_s,\ s \in S \text{ and } j \in S. \tag{1.27}$$

Since the sum of a geometric series satisfies $\sum_{n=1}^{\infty} \lambda^{n-1} = 1/(1-\lambda)$, under the boundedness assumption it follows that

$$\frac{-M}{1-\lambda} \leq v_\lambda^\pi(s) \leq \frac{M}{1-\lambda} \tag{1.28}$$

for all $\pi \in \Pi^{\text{HR}}$. Equation (1.28) shows the key role of the assumption that $\lambda < 1$. As noted previously when $\lambda = 1$ the series may not converge even when rewards are bounded.

Note that in contrast to the infinite horizon expected total reward, the quantity

$$\lim_{N\to\infty} \sum_{n=1}^{N} \lambda^{n-1} R_n$$

exists in a discounted model[18].

Hence it follows[19] that

$$\lim_{N\to\infty} E_s^\pi \left[\sum_{n=1}^{N} \lambda^{n-1} R_n\right] = E_s^\pi \left[\lim_{N\to\infty} \sum_{n=1}^{N} \lambda^{n-1} R_n\right] = E_s^\pi \left[\sum_{n=1}^{\infty} \lambda^{n-1} R_n\right]. \tag{1.29}$$

where the expression on the right is a shorthand for the middle expression.

[18]This is true because when rewards are bounded, the "tail" of the sum can be made arbitrarily small for every trajectory

[19]The result follows from the bounded convergence theorem.

Example 1.1 (ctd.) Returning to Example 1.1, we show that when $\lambda < 1$, the limit of the sequence of partial sums for each of the discounted sequences of rewards exists and is finite.

For ρ_1, the sequence of its discounted partial sums σ_1' is

$$\sigma_1' = (0, -\lambda, -\lambda + \lambda^2, -\lambda + \lambda^2 - \lambda^3, \ldots)$$

so that its limit exists and is given by

$$\sum_{n=1}^{\infty} \lambda^{n-1} R_n = -\lambda \sum_{n=1}^{\infty} \lambda^{2(n-1)} + \lambda^2 \sum_{n=1}^{\infty} \lambda^{2(n-1)} = -\frac{\lambda(1-\lambda)}{1-\lambda^2} = -\frac{\lambda}{1+\lambda}$$

Similarly, the discounted reward of sequence ρ_2 equals $\lambda/(1-\lambda)$. Thus when $\lambda < 1$

$$E\left[\lim_{N\to\infty} \sum_{n=1}^{N} \lambda^{n-1} R_n \,\middle|\, X_1 = s_1\right] = 0.5\frac{\lambda}{1+\lambda} - 0.5\frac{\lambda}{1+\lambda} = 0.$$

It is easy to see that

$$\lim_{N\to\infty} E\left[\sum_{n=1}^{N} \lambda^{(n-1)} R_n \,\middle|\, X_1 = s_1\right] = 0,$$

since the expectation equals 0 for any finite N. Hence (1.29) holds in this model.

An alternative and important interpretation of discounting

Discounting arises naturally in models where rewards are units of currency. Due to inflation, near-term rewards are preferable to future rewards. Discounting also arises in another, rather surprising way which extends its applicability.

Suppose the planning horizon length is not fixed but represented by a random variable, T, distributed according to a geometric distribution[20] with parameter λ, independent of $(R_1, R_2, \ldots)$ or $(X_1, Y_1, X_2, Y_2, \ldots)$.

We write the expected total reward over a random horizon of length T as

$$E_T\left[E\left[\sum_{n=1}^{T} R_n\right]\right], \tag{1.31}$$

[20] A random variable T follows a *geometric distribution* with parameter λ if

$$P(T = k) = (1-\lambda)\lambda^{k-1} \tag{1.30}$$

for $k = 1, 2, \ldots$.

where E_T denotes the expectation with respect to T. Assuming R_n is bounded and $0 \leq \lambda < 1$,

$$E_T\left[E\left[\sum_{n=1}^{T} R_n\right]\right] = E\left[\sum_{k=1}^{\infty}(1-\lambda)\lambda^{k-1}\sum_{n=1}^{k} R_n\right]$$
$$= E\left[\sum_{n=1}^{\infty}\sum_{k=n}^{\infty}(1-\lambda)\lambda^{k-1}R_n\right] = E\left[\sum_{n=1}^{\infty}\lambda^{n-1}R_n\right]$$

where the last inequality follows from the relationship $\sum_{k=n}^{\infty}\lambda^{k-1} = \lambda^{n-1}/(1-\lambda)$. The assumptions on R_n and λ allow interchange of the order of summation. Thus

$$E_T\left[E\left[\sum_{n=1}^{T} R_n\right]\right] = E\left[\sum_{n=1}^{\infty}\lambda^{n-1}R_n\right]. \tag{1.32}$$

In the case when the rewards are generated by policy π starting from state s,

Alternative representation for the expected discounted reward:

$$v_\lambda^\pi(s) = E_T\left[E_s^\pi\left[\sum_{n=1}^{T} R_n\right]\right]. \tag{1.33}$$

This result provides an alternative interpretation for discounting. Since the random variable T may be regarded as the time until the first "failure" in an independent, identically distributed series of Bernoulli trials with "success" probability λ, it means that when a decision maker uses expected total reward to evaluate policies in a system that terminates at the time of a random failure independent of the decision maker's policy it is equivalent to using the expected total discounted reward. This is particularly appropriate in applications where the process can terminate suddenly for exogenous reasons. Examples include:

- **animal behavior modeling** when the animal might die of unanticipated causes (e.g., predation) independent of its decision making. See Section **??**.
- **clinical decision making** in which a patient may die as a consequence of some event independent of the disease being treated. See Section **??**.

Thus, discounting makes sense even in models that do not use economic values for rewards.

Optimal policies

We say that a policy π^* is *discount optimal* if

$$v_\lambda^{\pi^*}(s) \geq v_\lambda^\pi(s) \tag{1.34}$$

for all $s \in S$ and $\pi \in \Pi^{\mathrm{HR}}$. Chapter **??** analyzes infinite horizon Markov decision process models under this optimality criterion.

1.3.3 Long-run average reward for an infinite horizon model

In contrast to discounting, which emphasizes short term behavior, the long-run average reward criterion focuses on steady state or limiting behavior of derived stochastic processes. For that reason, the long-run average reward is most appropriate for infinite horizon, non-terminating models with frequent decision epochs. Note that in finite horizon models, average reward is equivalent to expected total reward (why?).

As an example, consider a queuing system in which the decision maker inspects the system state very frequently, such as every second, and decides which service rate to use. Section **??** provides a rigorous formulation of such a problem. For such a problem:

- The **expected total reward** criterion would not be able to distinguish between policies because expected total costs or rewards would be unbounded.
- The **expected discounted reward** criterion would not be appropriate because the decision maker is interested in long-term system performance. Given the time scale of decision making, rewards at future decision epochs should **not** be less valuable than current rewards. However, if random failure of the system could occur discounting may be appropriate but it would require discount rates very close to 1.

We define the *average reward*[21] or more accurately the *long-run average reward* of the reward sequence $(R_1, R_2, \ldots)$ by

$$\lim_{N\to\infty} E\left[\frac{1}{N}\sum_{n=1}^{N} R_n\right].$$

When rewards are generated by a policy π we define the *average reward*, $g^\pi(s)$ of policy π for $s \in S$ by

Average reward:

$$g^\pi(s) := \lim_{N\to\infty} E_s^\pi\left[\frac{1}{N}\sum_{n=1}^{N} R_n\right]. \tag{1.35}$$

Note that the limit in (1.35) exists for all stationary policies when S is finite, as will be shown in Chapter **??**. It need not exist when S is countable or policies are history-dependent. In such cases we replace the limit above by the "lim inf" or the "lim sup" (see Chapter **??**). The quantity g^π is sometimes referred to as the *gain* because the expected total reward in state s grows at rate $g^\pi(s)$ per epoch in the limit.

Similarly to the discounted model, when rewards are bounded,

$$\lim_{N\to\infty} E_s^\pi\left[\frac{1}{N}\sum_{n=1}^{N} R_n\right] = E_s^\pi\left[\lim_{N\to\infty}\frac{1}{N}\sum_{n=1}^{N} R_n\right]. \tag{1.36}$$

[21] We interchangeably refer to this quantity as the *long-run average reward* or *gain*.

Example 1.1 (ctd.) Returning to Example 1.1, we show that when rewards are bounded, the limit of the sequence of the average partial sums for each of the sequences of rewards exists and is finite.

For ρ_1, the sequence of average partial sums σ_1'' is

$$\sigma_1'' = \left(0, -\frac{1}{2}, 0, -\frac{1}{4}, \ldots\right)$$

so that its limit exists and equals 0. Similarly, the limit of average of partial sums for ρ_2 equals 0. Thus

$$E\left[\lim_{N\to\infty} \frac{1}{N}\sum_{n=1}^{N} R_n \middle| X_1 = s_1\right] = 0.$$

since the expectation equals 0 for any finite N. Hence it is easy to see that (1.36) holds in this model.

Optimal policies

A policy π^* is *average reward optimal* or *gain optimal* if

$$g^{\pi^*}(s) \geq g^{\pi}(s) \tag{1.37}$$

for all $\pi \in \Pi^{\mathrm{HR}}$ and $s \in S$. Chapter **??** analyzes infinite horizon Markov decision process models under this optimality criterion.

1.4 The one-period problem: a fundamental building block

One-period models are the building blocks of Markov decision processes. Most of the algorithms we present later in the book are based on decomposing a multi-period model into a series of interlinked one-period models.

A one-period model begins with a decision epoch, evolves for one period of time and ends at the terminal non-decision epoch. Thus, it is the simplest representation of a finite horizon Markov decision process, namely the case of $N = 2$. In it, a policy consists of a single Markovian decision rule[22], and the derived stochastic process is (X_1, Y_1, X_2). Figure 1.6 illustrates the timing of events and the nomenclature of the one-period problem.

[22]In a one-period model, there is no need for history-dependent policies because the only information available to the decision maker is the state at decision epoch 1.

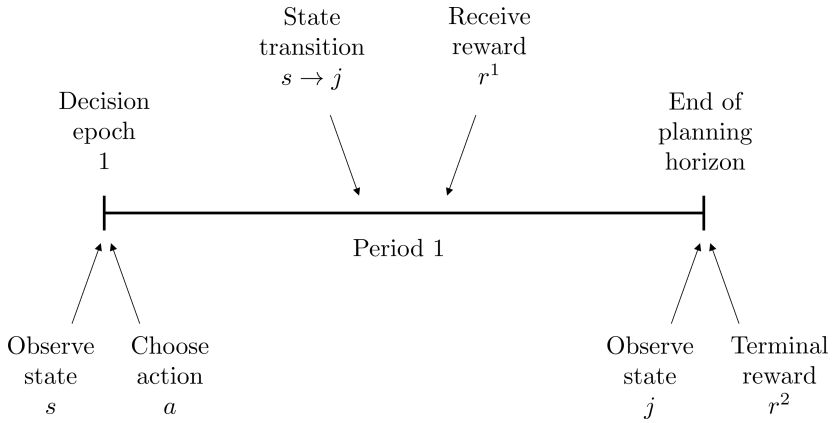


Figure 1.6: Illustration of timing of events and notation for the one-period problem. Note that action a may be chosen deterministically or probabilistically.

Let S denote the state space and A_s denote the sets of actions defined for each $s \in S$. We assume that S and each A_s are discrete and finite. Let $r_1(s, a, j)$ denote the reward function in period 1, $p_1(j|s, a)$ denote the transition probability function in period 1, and $r_2(j)$ denote the terminal reward function. We use this simple model to help build intuition for more complex models.

For a policy π , let $v^\pi(s)$ be the expected total reward when the process starts in state s at decision epoch 1. It is given by

$$v^\pi(s) := E_s^\pi[r_1(X_1, Y_1, X_2) + r_2(X_2)], \tag{1.38}$$

where the expectation is with respect to the probability distribution of (X_1, Y_1, X_2) induced by policy π when the system starts in state $s \in S$. The assumption that the system starts in state s with certainty implies that $X_1 = s$.

Suppose $\pi = (d_1)$ is deterministic and $d_1(s) = a'$. Since s and $d_1(s) = a'$, the only random quantity in (1.38) is X_2. Under this assumption, (1.38) is equivalent to

$$v^\pi(s) = \sum_{j \in S} p_1(j|s, a')(r_1(s, a', j) + r_2(j)). \tag{1.39}$$

Therefore, to compute $v^\pi(s)$, requires only $p_1(j|s, a') = P[X_2 = j]$; we do not need to derive the distribution of $r_1(X_1, Y_1, X_2) + r_2(X_2)$. Consequently, this avoids an enumeration of all of the latter's possible values as described in Section 1.2.4. Expressions of the form (1.39) appear frequently in the book, so it is important to understand why (1.39) is equivalent to (1.38).

When d_1 is randomized, Y_1 would also be random and $P[Y_1 = a|X_1 = s] = w_{d_1}(a|s)$. In this case

$$v^\pi(s) = \sum_{a \in A_s} w_{d_1}(a|s) \sum_{j \in S} p_1(j|s, a)(r_1(s, a, j) + r_2(j)). \tag{1.40}$$

1.4.1 Optimal values

We now show how to find optimal policies and values. It easiest to summarize the key result for a one period model in the form of a theorem with a proof[23].

Theorem 1.1. Suppose in a finite state and action one-period Markov decision problem that the policy $\pi^* = (d_1^*)$ satisfies

$$v^{\pi^*}(s) = \max_{a \in A_s} \left\{ \sum_{j \in S} p_1(j|s,a)(r_1(s,a,j) + r_2(j)) \right\}. \tag{1.41}$$

for all $s \in S$ and $a \in A_s$. Then

$$v^{\pi^*}(s) \geq v^{\pi}(s) \tag{1.42}$$

for all $s \in S$ and any policy π.

Proof. Let π be a deterministic policy. Then from (1.39) its easy to see that $v^{\pi}(s)$ is bounded above by the largest value possible[24] of the right hand side of (1.39) over all actions $a \in A_s$, that is,

$$v^{\pi}(s) \leq \max_{a \in A_s} \left\{ \sum_{j \in S} p_1(j|s,a)(r_1(s,a,j) + r_2(j)) \right\} = v^{\pi^*}(s). \tag{1.43}$$

Now let π be a randomized policy, that is $\pi \in \Pi^{\mathrm{MR}}$. From (1.40) and the definition of $v^{\pi^*}(s)$

$$\begin{aligned} v^{\pi}(s) &= \sum_{a \in A_s} w_{d_1}(a|s) \sum_{j \in S} p_1(j|s,a)(r_1(s,a,j) + r_2(j)) \\ &\leq \sum_{a \in A_s} w_{d_1}(a|s) \left[\max_{a' \in A_s} \sum_{j \in S} p_1(j|s,a')(r_1(s,a',j) + r_2(j)) \right] \qquad (1.44) \\ &= \sum_{a \in A_s} w_{d_1}(a|s) v^{\pi^*}(s) = v^{\pi^*}(s) \qquad (1.45) \end{aligned}$$

which establishes the result for all randomized policies. Since in a one-period model, $\Pi^{\mathrm{HR}} = \Pi^{\mathrm{MR}}$, the result applies to all policies. □

We define the real-valued function on S, $v^*(s)$, to be the *value* of the one-period problem by:

[23] For those of you not familiar with this form of presentation, a theorem describes an important result and a proof demonstrates why it is so.

[24] If A_s were non-finite, a supremum ("sup") over all actions would replace the max in (1.43)

$$v^*(s) := \sup_{\pi \in \Pi^{\text{MR}}} v^\pi(s). \tag{1.46}$$

Note in the one-period problem, Π^{MR} represents the set of **all** policies. On account of randomization, there are an uncountably infinite number of such policies[25]. Since there are infinitely many such policies and we don't know in advance whether there is one with largest value, we write "sup" instead of "max".

Thus the result of this theorem is: *There exists a policy with value specified by* (1.46) *that equals the value of the one-period Markov decision process.* Since we have shown that there exists a policy that exceeds the value of all other policies, the "sup" is attained and we have established the following result.

Corollary 1.1. In a finite state and action one-period Markov decision process

$$v^*(s) = \max_{\pi \in \Pi^{\text{MR}}} v^\pi(s) = v^{\pi^*}(s). \tag{1.47}$$

Randomized policies are not necessary

Buried in the proof of Theorem 1.1 was the result that maximizing over deterministic policies was equivalent to maximizing over randomized policies. Since this is a fundamental concept, we formalize the result below.

Let $\mathscr{P}(A_s)$ denote the set of probability distributions on A_s. Each $w \in \mathscr{P}(A_s)$ corresponds to a different randomized decision rule. Since $\mathscr{P}(A_s)$ is not a finite set, we write *sup* instead of *max* in (1.48) below[26]. What we wish to show is

$$\sup_{u \in \mathscr{P}(A_s)} \left\{ \sum_{a \in A_s} u(a) \sum_{j \in S} p_1(j|s,a)(r_1(s,a,j) + r_2(j)) \right\} \tag{1.48}$$

$$= \max_{a \in A_s} \left\{ \sum_{j \in S} p_1(j|s,a)(r_1(s,a,j) + r_2(j)) \right\}. \tag{1.49}$$

To see this, first note that (1.48) is greater than or equal to (1.49). This is because any maximizing action in (1.49) corresponds to a degenerate probability distribution[27] in (1.48) with all the probability mass placed on the maximizing action. To see that these two expressions are in fact equal, consider an optimal probability distribution from (1.48) that puts positive probability mass on two or more actions. All of these

[25]For each state, the set of randomized decision rules in that state can be represented by an $|A_s|$-dimensional unit simplex corresponding to *all* probability distribution on A_s. Π^{MR} is equivalent to the Cartesian product of these simplices.

[26]When maximizing over a non-finite set, we write "sup" even when the "sup" is attained to emphasize that the set is not finite.

[27]A *degenerate* probability distribution places all of its mass on a single element.

actions must have the same expected total reward. Thus, a degenerate probability distribution that puts all mass on any of those actions would also have the same expected total reward. Put more simply, the weighted average of a set of values cannot be larger than every one of the individual values.

Lemma 1.1 which will be used often in the rest of the book, formalizes the above argument in greater generality.

Lemma 1.1. Let U be an arbitrary finite set, let $f(\cdot)$ be a real-valued function on U, and $w(\cdot)$ be a probability distribution on U. Then,

$$\max_{u\in U} f(u) \geq \sum_{u\in U} w(u)f(u) \tag{1.50}$$

and

$$\max_{u\in U} f(u) = \sup_{w\in\mathscr{P}(U)} \sum_{u\in U} w(u)f(u). \tag{1.51}$$

Proof. To prove (1.50), let $\bar{f} := \max_{u\in U} f(u)$. Then

$$\bar{f} = \sum_{u\in U} w(u)\bar{f} \ \geq \max_{u\in U} f(u).$$

To prove (1.51), choose $u^* \in \arg\max_{u\in U} f(u)$ and define $w^* \in \mathscr{P}(U)$ by $w^*(u) = 1$ if $u = u^*$ and $w^*(u) = 0$, if $u \neq u^*$. Then, by the definition of w^* and (1.50), for any $w \in \mathscr{P}(U)$

$$\sum_{u\in U} w^*(u)f(u) = \max_{u\in U} f(u) \geq \sum_{u\in U} w(u)f(u)$$

from which the result follows. □

What the equivalence between (1.48) and (1.49) means is that a decision maker cannot receive a larger expected reward by randomizing over actions than by just using a deterministic decision rule determined by (1.53). This argument establishes for the one-period model that if d_1^* satisfies (1.53) and $\pi^* = (d_1^*)$, then the *optimal expected total reward*, denoted $v^*(s)$, satisfies

$$v^*(s) = \sup_{\pi\in\Pi^{\mathrm{MR}}} v^\pi(s) = \max_{\pi\in\Pi^{\mathrm{MD}}} v^\pi(s) = v^{\pi^*}(s) \tag{1.52}$$

for all $s \in S$. (Be sure you can identify each expression in (1.52) and distinguish what it represents.) Note that we do not have to consider history-dependent policies here because for a one-period problem there is no distinction between history-dependent and Markovian policies.

1.4.2 Optimal policies

Since we now know that an optimal policy exists, we need to know how to find it. First we note that the *argmax* of a function returns the set arguments that maximize the function[28]. Thus, in general, the argmax is a set that contains multiple elements. Only when there is a unique minimizer of a function does the argmax return a single value.

Theorem 1.2. In a finite state and action one-period Markov decision problem let $d_1^*(\cdot)$ denote a deterministic decision rule that for each $s \in S$ satisfies

$$d_1^*(s) \in \arg\max_{a \in A_s} \left\{ \sum_{j \in S} p_1(j|s,a)(r_1(s,a,j) + r_2(j)) \right\}. \tag{1.53}$$

Then the policy $\pi^* = (d_1^*)$ is optimal.

Proof. From (1.53) and (1.39) it follows that

$$v^*(s) = \max_{a \in A_s} \left\{ \sum_{j \in S} p_1(j|s,a)(r_1(s,a,j) + r_2(j)) \right\} \tag{1.54}$$

$$= \sum_{j \in S} p_1(j|s, d_1^*(s))(r_1(s, d_1^*(s), j) + r_2(j)) = v^{\pi^*}(s) \tag{1.55}$$

for all $s \in S$. Hence π^* is an optimal policy. □

Some observations about these results follow:

1. **Greedy Actions:** We refer to any action that achieves the maximum in the right hand side of (1.53) as a *greedy* action.

2. **Independent Problems:** To find an optimal policy in this model, one solves an independent problem (1.53) for each state $s \in S$.

3. **Trade-offs:** The expression

$$\max_{a \in A_s} \left\{ \sum_{j \in S} p_1(j|s,a)(r_1(s,a,j) + r_2(j)) \right\}$$

appears frequently in solution algorithms for Markov decision processes. This maximization highlights the trade-off between the immediate reward $r_1(s,a,j)$ and a future reward $r_2(j)$. This notion of balancing a current or *myopic* reward with future rewards is fundamental to Markov decision processes.

[28]Formally, let B denote an arbitrary set and $f(b)$ denote a real-valued function on B. Then

$$b^* \in \arg\max_{b \in B} f(b)$$

if $f(b^*) = \max_{b \in B} f(b)$. When b^* is the unique maximizer, we replace $\in$ by $=$ in the above expression.

4. **Future Values:** The future reward is encapsulated in the model component $r_2(X_2)$ in the one-period model. An important question in multi-period finite or infinite horizon models is: *Can the future reward be replaced with a single function that captures the cumulative reward associated with system evolution beyond the current decision epoch*? The answer is "yes". We elaborate on this key concept in subsequent chapters.

5. **Rollout and approximations:** In modern applications with very large state spaces, the terminal reward might represent an approximation to the "value" of being in state s that has been determined "offline". Then to determine the next action in state s "online", the decision maker solves the one-period problem and uses the greedy action. Such an approach has been used to determine good strategies in chess, go and backgammon[29].

1.4.3 State-action value functions

Reinforcement learning introduces another fundamental quantity, the *state-action value function*. While not essential in Chapters **??**-**??**, they play a key role in Chapters **??** and 2.

A state-action value function gives the expected total reward (or discounted reward) of a specified action chosen in a particular state. In the one-period problem[30] it is defined for all $a \in A_s$ and $s \in S$ by

$$q(s,a) := \sum_{j \in S} p_1(j|s,a)(r_1(s,a,j) + r_2(j)). \tag{1.56}$$

These are often referred to as q-functions.

The state-action value function provides the decision maker with all the information necessary to find the value of a policy, optimal policies and (optimal) values. Let $\pi = (d_1)$ denote an arbitrary deterministic policy. Then from (1.39), its value is given by

$$v^\pi(s) = q(s, d_1(s)). \tag{1.57}$$

Moreover, as a direct result of Theorem 1.1,

$$v^*(s) = \max_{a \in A_s} q(s,a) \tag{1.58}$$

and from Theorem 1.2

$$d_1^*(s) \in \arg\max_{a \in A_s} q(s,a). \tag{1.59}$$

[29] See Bertsekas [2022] for additional details.

[30] In models with longer planning horizons, they are distinguished according to whether the decision maker follows a specific policy or the optimal policy after choosing an (s,a)-pair. In the one-period model, there are no decisions after the first decision epoch so this distinction is immaterial.

We will see that in Part III of the book that one might choose to estimate $q(s,a)$ by simulation. In such a situation, (1.57) can be used to find the value of a policy and (1.59) and (1.58) to find an optimal policy and its value, respectively.

1.4.4 Summary of results for a one-period problem

Putting this all together, we have demonstrated the following for a one-period model:

1. There exists a Markovian deterministic policy $\pi^* = (d_1^*)$ that is optimal in the class of all policies.

2. To "solve" the one-period problem under the expected total reward criterion, we compute $v^*(s)$ for all $s \in S$ as follows

$$v^*(s) = \max_{a \in A_s} \left\{ \sum_{j \in S} p_1(j|s,a)(r_1(s,a,j) + r_2(j)) \right\} \tag{1.60}$$

   and choose $d^*(s)$ so that

$$d^*(s) \in \arg\max_{a \in A_s} \left\{ \sum_{j \in S} p_1(j|s,a)(r_1(s,a,j) + r_2(j)) \right\}. \tag{1.61}$$

3. The expected total reward of the optimal policy satisfies

$$\begin{aligned} v^{\pi^*}(s) &= \sum_{j \in S} p_1(j|s, d_1^*(s))(r_1(s, d_1^*(s), j) + r_2(j)) \\ &= \max_{a \in A_s} \left\{ \sum_{j \in S} p_1(j|s,a)(r_1(s,a,j) + r_2(j)) \right\} = v^*(s). \end{aligned}$$

Our goal in later parts of the book will be to generalize these ideas to more complex models.

1.5 A Two-State Model

In this section, we provide a simple and concrete example that illustrates basic model components and notation, and previews calculations that will be seen later in the book. We formulate this model as a finite horizon Markov decision process; we analyze an infinite horizon version in later chapters. We assume the transition probabilities and rewards are stationary, that is, they do not vary from decision epoch to decision epoch. Moreover, we assume the reward is a function of the current state, the chosen action, and the subsequent state.

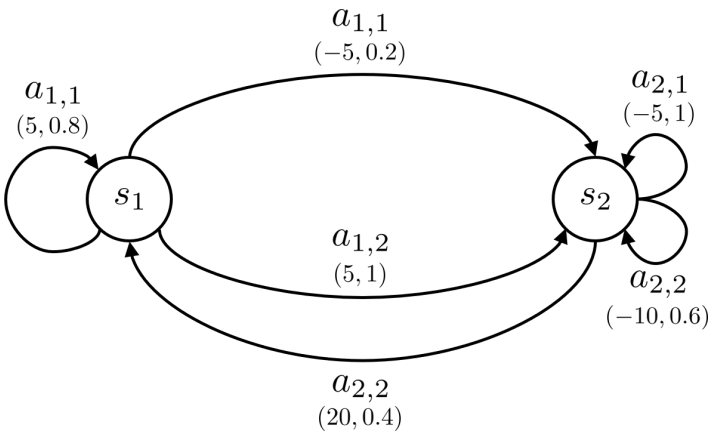


Figure 1.7: Graphical representation of two-state model. Circles denote states, arcs denote actions and transitions between states, and the expressions in parentheses denote rewards and transition probabilities, respectively. Zero probability transitions have been omitted.

Example 1.2. Consider the two-state model illustrated in Figure 1.7. In this model, there are two states, and two actions to choose from in each state. Actions $a_{1,2}$ and $a_{2,1}$ result in deterministic outcomes – when the decision maker chooses these actions, transitions occur to a specified state with certainty. Consequently, rewards $r_n(s_1, a_{1,2}, s_1)$ and $r_n(s_2, a_{2,1}, s_1)$ are superfluous since they correspond to outcomes that cannot occur. We assume terminal rewards of 0. The formal formulation follows.

Decision Epochs:

$$\{1, 2, \ldots, N\}, \quad N < \infty$$

States:

$$S = \{s_1, s_2\}$$

Actions:

$$A_{s_1} = \{a_{1,1}, a_{1,2}\}, \quad A_{s_2} = \{a_{2,1}, a_{2,2}\}$$

Rewards: For $n = 1, 2, \ldots, N$

$$r_n(s_1, a_{1,1}, s_1) = 5, \quad r_n(s_1, a_{1,1}, s_2) = -5$$

$$r_n(s_1, a_{1,2}, s_1) = 0, \quad r_n(s_1, a_{1,2}, s_2) = 5$$

$$r_n(s_2, a_{2,1}, s_1) = 0, \quad r_n(s_2, a_{2,1}, s_2) = -5$$

$$r_n(s_2, a_{2,2}, s_1) = 20, \quad r_n(s_2, a_{2,2}, s_2) = -10$$
$$r_N(s_1) = 0, \quad r_N(s_2) = 0$$

Transition Probabilities: For $n = 1, 2, \ldots, N$

$$p_n(s_1|s_1, a_{1,1}) = 0.8, \quad p_n(s_2|s_1, a_{1,1}) = 0.2$$
$$p_n(s_1|s_1, a_{1,2}) = 0, \quad p_n(s_2|s_1, a_{1,2}) = 1$$
$$p_n(s_1|s_2, a_{2,1}) = 0, \quad p_n(s_2|s_2, a_{2,1}) = 1$$
$$p_n(s_1|s_2, a_{2,2}) = 0.4, \quad p_n(s_2|s_2, a_{2,2}) = 0.6$$

1.5.1 A One-Period Model

In this section we use value functions and state-action value functions to compute optimal values and policies in Example 1.2. Recall that in a one-period model, $N = 2$.

Value functions

From (1.41), we compute $v^*(s)$ as

$$\begin{aligned} v^*(s_1) &= \max_{a \in \{a_{1,1}, a_{1,2}\}} \{p_1(s_1|s_1, a)(r_1(s_1, a, s_1) + r_2(s_1)) + p_1(s_2|s_1, a)(r_1(s_1, a, s_2) + r_2(s_2))\} \\ &= \max\{0.8 \times (5 + 0) + 0.2 \times (-5 + 0), 5 + 0\} = \max\{3, 5\} = 5. \end{aligned}$$

and

$$\begin{aligned} v^*(s_2) &= \max_{a \in \{a_{1,1}, a_{1,2}\}} \{p_1(s_1|s_1, a)(r_1(s_1, a, s_1) + r_2(s_1)) + p_1(s_2|s_1, a)(r_1(s_1, a, s_2) + r_2(s_2))\} \\ &= \max\{-5 + 0, 0.4 \times (20 + 0) + 0.6 \times (-10 + 0)\} = \max\{-5, 2\} = 2. \end{aligned}$$

The first term in the "max" corresponds to action $a_{i,1}$, $i = 1, 2$ and the second term in the "max" corresponds to $a_{i,2}$, $i = 1, 2$. Thus, from (1.53), $\pi^* = (d_1^*)$ where $d_1^*(s_1) = a_{1,2}$ and $d_1^*(s_2) = a_{2,2}$ and from (1.47), $v^{\pi^*}(s_1) = 5$ and $v^{\pi^*}(s_2) = -2$.

Observe that this calculation was particularly simple because the terminal reward in each state equals 0. Exercise 6 asks you to explore the sensitivity of the optimal decision to the terminal reward.

State-action value functions

We now repeat these calculations using state-action value functions. From (1.56),

$$\begin{aligned}
q(s_1, a_{1,1}) &= p(s_1|s_1, a_{1,1})r(s_1, a_{1,1}, s_1) + p(s_2|s_1, a_{1,1})r(s_1, a_{1,1}, s_2) = 3 \\
q(s_1, a_{1,2}) &= p(s_1|s_1, a_{1,2})r(s_1, a_{1,2}, s_1) + p(s_2|s_1, a_{1,2})r(s_1, a_{1,2}, s_2) = 5 \\
q(s_2, a_{2,1}) &= p(s_1|s_2, a_{2,1})r(s_1, a_{2,1}, s_1) + p(s_2|s_,a_{2,1})r(s_1, a_{2,1}, s_2) = -5 \\
q(s_2, a_{2,2}) &= p(s_1|s_2, a_{2,2})r(s_2, a_{2,1}, s_1) + p(s_2|s_2, a_{2,2})r(s_2, a_{2,2}, s_2) = 2.
\end{aligned}$$

Hence from (1.59)

$$\begin{aligned}
d_1^*(s_1) &= \underset{a\in\{a_{1,1},a_{1,2}\}}{\arg\max}\ q(s_1, a) = a_{1,2} \\
d_1^*(s_2) &= \underset{a\in\{a_{1,1},a_{1,2}\}}{\arg\max}\ q(s_2, a) = a_{2,2}
\end{aligned}$$

and from (1.58)

$$\begin{aligned}
v^*(s_1) &= \max_{a\in\{a_{1,1},a_{1,2}\}} q(s_1, a) = 5 \\
v^*(s_2) &= \max_{a\in\{a_{1,1},a_{1,2}\}} q(s_2, a) = 2
\end{aligned}$$

in agreement with the calculations using value functions.

As an aside, we found that when coding algorithms, it was easier to organize calculations by first specifying $q(s, a)$.

1.5.2 A Two-Period Model

In a one-period model, history-dependent and Markovian decision rules (and policies) are equivalent since the history at decision epoch 1 is the same as the state at decision epoch 1. Thus, to illustrate all the different types of policies that are possible in our two-state model, we now consider a two-period version of it.

A Markovian deterministic policy $\pi \in \Pi^{\mathbf{MD}}$

Suppose $\pi = (d_1, d_2)$, where

$$d_1(s) = \begin{cases} a_{1,1}, & s = s_1 \\ a_{2,1}, & s = s_2 \end{cases} \quad \text{and} \quad d_2(s) = \begin{cases} a_{1,2}, & s = s_1 \\ a_{2,1}, & s = s_2. \end{cases} \tag{1.62}$$

In state s_1, action $a_{1,1}$ is chosen at the first decision epoch and $a_{1,2}$ is chosen at the second decision epoch. In state s_2, action $a_{2,1}$ is chosen at both decision epochs.

Next, we compute the expected total reward generated by this policy π. Suppose the process starts in state s_1. Then $a_{1,1}$ is chosen, which results in a self-transition to state s_1 with probability 0.8 and a transition to state s_2 with probability 0.2. The

corresponding rewards for these transitions are 5 and -5, respectively. At epoch 2, if the state is s_1, then $a_{1,2}$ is chosen and the system transitions to state s_2 with certainty and a reward of 5. This first sample path ($s_1 \to s_1 \to s_2$), which occurs with probability 0.8, has a total reward of $5 + 5 = 10$. If the state is s_2 at epoch 2, then action $a_{2,1}$ is chosen, which results in a self-transition and a reward of -5. This second sample path ($s_1 \to s_2 \to s_2$), which occurs with probability 0.2, has a total reward of $-5-5 = -10$. Since the terminal rewards are 0, the expected total reward of this particular Markov deterministic policy π is $0.8(10) + 0.2(-10) = 6$.

A similar set of calculations will show that if the process starts in state s_2, the expected total reward of policy π is -10. Finally, an initial probability distribution that specifies the starting state as s_1 with probability p and s_2 with probability $1-p$ would have an expected total reward of $16p - 10$. Calculations similar to this will be fundamental to analyzing partially observable MDP models in Chapter **??**.

A Markovian randomized policy $\pi \in \Pi^{\mathbf{MR}}$

Suppose that at decision epoch $n = 1, 2$, in state s_1, d_n chooses action $a_{1,1}$ with probability $q_{n,1}$ and action $a_{1,2}$ with probability $1 - q_{n,1}$ and in state s_2, d_n chooses action $a_{2,1}$ with probability $q_{n,2}$ and action $a_{2,2}$ with probability $1 - q_{n,2}$. Then, in the notation of Section 1.2.1, we have that for $n = 1, 2$,

$$w^n_{d_n}(a|s) = \begin{cases} q_{n,1}, & a = a_{1,1},\ s = s_1 \\ 1 - q_{n,1}, & a = a_{1,2},\ s = s_1 \\ q_{n,2}, & a = a_{2,1},\ s = s_2 \\ 1 - q_{n,2}, & a = a_{2,2},\ s = s_2. \end{cases} \tag{1.63}$$

Computing the expected total reward of a Markovian randomized policy is similar to the Markovian deterministic case, except that an additional expectation has to be taken with respect to the probability distribution over action selection specified by $w^n_{d_n}(a|s)$.

A history-dependent deterministic policy $\pi \in \Pi^{\mathbf{HD}}$

Suppose $\pi = (d_1, d_2)$, where

$$d_1(h) = \begin{cases} a_{1,1},\ h = s_1 \\ a_{2,1},\ h = s_2 \end{cases} \quad \text{and} \quad d_2(h) = \begin{cases} a_{1,2},\ h = (s_1, a_{1,1}, s_1) \\ a_{2,1},\ h = (s_1, a_{1,1}, s_2) \\ a_{2,2},\ h = (s_2, a_{2,1}, s_2) \end{cases} \tag{1.64}$$

For this policy, the chosen action depends on the entire history. In epoch 1, the history is simply the initial state. But in epoch 2, the history includes the initial state, the first action chosen, and then the subsequent state. If the state is s_1 at epoch 2, there is only one way for this to happen, given d_1 (initial state s_1 with action $a_{1,1}$).

However, there are two histories that lead to s_2 at epoch 2. Thus, the action chosen in state s_2 at epoch 2 depends on whether the process started in s_1 or s_2: starting in s_1 leads to choosing action $a_{2,1}$ in the second decision epoch, otherwise $a_{2,2}$. Notice that if the policy selects action $a_{2,1}$ for the history $(s_2, a_{2,1}, s_2)$, then this policy coincides with the Markovian deterministic policy described previously.

A history-dependent randomized policy $\pi \in \Pi^{\mathbf{HR}}$

Suppose at the first decision epoch, the randomized decision rule is described by the following probability distribution:

$$w_{d_1}^1(a|h) = \begin{cases} q_{1,1}, & a = a_{1,1},\ h = s_1 \\ 1 - q_{1,1}, & a = a_{1,2},\ h = s_1 \\ q_{1,2}, & a = a_{2,1},\ h = s_2 \\ 1 - q_{1,2}, & a = a_{2,2},\ h = s_2. \end{cases} \tag{1.65}$$

In state s_1, action $a_{1,1}$ is chosen with probability $q_{1,1}$ and action $a_{1,2}$ is chosen with probability $1 - q_{1,1}$. In state s_2, action $a_{2,1}$ is chosen with probability $q_{1,2}$ and action $a_{2,2}$ is chosen with probability $1 - q_{1,2}$.

At the second decision epoch, there are up to eight histories to consider, since there are two states and two actions in each state. However, $a_{1,2}$ and $a_{2,1}$ result in deterministic transitions to state s_2, so two of the eight histories are impossible. The remaining six are listed below:

$$\begin{aligned} h_1 &:= (s_1, a_{1,1}, s_1) \\ h_2 &:= (s_1, a_{1,1}, s_2) \\ h_3 &:= (s_1, a_{1,2}, s_2) \\ h_4 &:= (s_2, a_{2,1}, s_2) \\ h_5 &:= (s_2, a_{2,2}, s_1) \\ h_6 &:= (s_2, a_{2,2}, s_2). \end{aligned}$$

Given these possible histories at the second decision epoch, a randomized decision rule can be written as

$$w_{d_2}^2(a|h) = \begin{cases} q_{2,i}, & a = a_{1,1},\ h = h_i \\ 1 - q_{2,i}, & a = a_{1,2},\ h = h_i \end{cases} \tag{1.66}$$

for $i \in \{1, 5\}$ and

$$w_{d_2}^2(a|h) = \begin{cases} q_{2,i}, & a = a_{2,1},\ h = h_i \\ 1 - q_{2,i}, & a = a_{2,2},\ h = h_i \end{cases} \tag{1.67}$$

for $i \in \{2, 3, 4, 6\}$.

A stationary deterministic policy $\pi \in \Pi^{\mathbf{SD}}$

Suppose $\pi = (d, d)$, where

$$d(s) = \begin{cases} a_{1,1}, & s = s_1 \\ a_{2,1}, & s = s_2 \end{cases} \tag{1.68}$$

This policy uses the same decision rule in both decision epochs. It is similar to the Markovian deterministic policy above, except that if the system is in state s_1 at epoch 2, the SD policy chooses action $a_{1,1}$, whereas the MD policy chooses $a_{1,2}$.

A stationary randomized policy $\pi \in \Pi^{\mathbf{SR}}$

The Markovian randomized policy above can be transformed into a stationary randomized policy $\pi = (d, d)$ by simply omitting its dependence on n, for example:

$$w_d(a|s) = \begin{cases} q_1, & a = a_{1,1},\ s = s_1 \\ 1 - q_1, & a = a_{1,2},\ s = s_1 \\ q_2, & a = a_{2,1},\ s = s_2 \\ 1 - q_2, & a = a_{2,2},\ s = s_2. \end{cases} \tag{1.69}$$

This policy has a stationary probability distribution of choosing action $a_{1,1}$ versus $a_{1,2}$ when the system is in state s_1, and similarly for when the state is s_2.

1.6 Bibliographic Remarks

The expression "Markov decision process" originates with Bellman [1957]. His comprehensive book formulates a wide range of problems as Markov decision processes and introduces many of the basic concepts including states, actions, transition probabilities and optimality equations. Antecedents include Wald [1947] on statistical decision models, Massé [1946] and Gessford and Karlin [1958] on reservoir management and Shapley [1953] on stochastic games.

Subsequently Howard [1960], which was based on his MIT doctoral dissertation, discussed the average and discounted reward model, developed the policy improvement algorithm (see Chapter **??**) for both and provided a range of colorful applications.

Inspired by Howard's book, Blackwell [1962] provides a rigorous analysis of infinite horizon, finite state and action, discounted and undiscounted Markov decision process. He formulates the model using matrix notation and establishes the optimality of stationary policies in a discounted model by showing that policy improvement converges. However, the most significant results in this paper concern the existence and computation of optimal policies as the discount factor approaches 1 in which case the limit in (1.26) diverges. To do so, he uses Markov chain results from Kemeny and Snell [1960] concerning the fundamental matrix to relate the average and discounted cases

through a partial Laurent expansion. This chain of arguments was subsequently refined in A.F. Veinott [1969] where he introduces the concepts of n-discount optimality.

Several noteworthy books include Derman [1970], Sennott [1999], Bertseksas [2012], Powell [2007], and Sutton and Barto [2018]. Puterman [1994] provides a more detailed historical perspective.

1.7 Exercises

1. **(check this gives nice numbers)** Consider the following three-state model with $S = \{s_1, s_2, s_3\}$, actions $A_{s_i} = \{a_{i,1}, a_{i,2}\}$ for $i = 1, 2, 3$, rewards $r_n(s_i, a_{i,1}, s_j) = i - j, r_n(s_i, a_{i,2}, s_j) = j - i$ for $n = 1, 2, \ldots, N-1$ and $r_N(s_i) = -i^3$. Transition probabilities are given by $p_n(s_i|s_i, a_{i,1}) = 1 - 1/i$, $p_n(s_j|s_i, a_{i,1}) = 1/(2i)$ for $j \neq i$ and $p_n(s_i|s_i, a_{i,2}) = 1 - 1/(i+1)$, $p_n(s_j|s_i, a_{i,2}) = 1/(2(i+1))$ for $j \neq i$ for $n = 1, 2, \ldots, N$.

   (a) Provide a graphical representation of the model as in Figure 1.7.

   (b) Represent a one- and two-period model as a decision tree when $P(X_1 = s_i) = \gamma(s_i) = \frac{1}{3}$ for $i = 1, 2, 3$. Compute the total reward of each path through the tree.

   (c) In a one-period model, find the distribution of X_2 for each possible initial state s_1, s_2, s_3 when the deterministic decision rule $d(s_i) = a_{i,1}$ is used at decision epoch 1.

   (d) Use (1.39) to find the expected total reward $v^{\pi}(s)$ for each $s \in S$ in a one-period model for the policy π that uses the above decision rule d at decision epoch 1.

   (e) In a one-period model, find the distribution of X_2 for each possible initial state s_1, s_2, s_3 when the randomized decision rule d' with distribution $P(d'(s_i) = a_{i,1}) = 1 - P(d'(s_i) = a_{i,2}) = e^{-0.5i}$ for $i = 1, 2, 3$ is used at decision epoch 1.

   (f) Use (1.40) to find the expected total reward $v^{\pi'}(s)$ for each $s \in S$ in a one-period model for the policy π' that uses the above decision rule d' at decision epoch 1.

   (g) Find the optimal policy in a one-period model using value functions and state-action value functions.

2. Using 5000 replications of Algorithm 1.1, simulate the total reward for the policies π and π' from Exercise 1 when $N = 2$.

   (a) Estimate the expected total reward of each policy and compare your results to those in Exercise 1.

(b) Provide histograms of your estimates and comment on their shape and any differences you observe between the histograms of π and π'.

(c) Compute the standard deviation and 95th percentile of the total returns for each policy. Interpret these quantities verbally and note why we might be interested in such quantities.

(d) Suppose we measure the value of a reward stream (R_1, R_2) by multiplicative utility $e^{\gamma R_1} e^{\gamma R_2}$ and compare policies on the basis of their expected utility. Repeat parts (a) to (c) above using this utility function.

3. Construct a deterministic and randomized history-dependent policy for a two-period version of the model in Example 1.

4. Show that when $\pi \in \Pi^{\text{MR}}$, the sequence of state and action pairs is a discrete time Markov Chain. Devise an example that shows that when $\pi \in \Pi^{\text{HR}}$, the sequence may not be Markov Chain.

5. Construct an example where an epoch-dependent state space S_n is appropriate (i.e., where using $S = \cup_n S_n$ would unnecessarily enlarge the state space at each decision epoch). Hint: consider a shortest path problem.

6. Write out (1.60) and (1.61) for the one-period version of the two-state problem in which $r_2(s_1) = c_1$ and $r_2(s_2) = c_2$. Plot the optimal policy as a function of the terminal rewards c_1 and c_2.

7. *Consider the following deterministic model. Let $S = \{s_1, s_2\}, A_{s_1} = \{a_{1,1}, a_{1,2}\}, A_{s_2} = \{a_{2,1}, a_{2,2}\}, r(s_1, a_{1,1}) = r(s_1, a_{1,2}) = 2, r(s_2, a_{2,1}) = r(s_2, a_{2,2}) = -2,$ and $p(s_1|s_1, a_{1,1}) = p(s_2|s_1, a_{1,2}) = p(s_1|s_2, a_{2,1}) = p(s_2|s_2, a_{2,2}) = 1$.

(a) Provide a graphical representation of the model as in Figure 1.7.

(b) Show that for each stationary policy π and each state $s \in S$ that the following limit exists

$$\lim_{N\to\infty} \frac{1}{N} E^{\pi} \left[\sum_{n=1}^{N} r(X_n, Y_n) \,|\, X_1 = s \right]. \tag{1.70}$$

(c) Consider a history-dependent policy π that when the initial state is s_1 chooses action $a_{1,1}$ for one period, then chooses $a_{1,2}$ so that the system proceeds to s_2 and then chooses action $a_{2,2}$ so the system remains in s_2 for three periods,at which point it chooses action $a_{2,1}$ so that the system returns to s_1 and then chooses action $a_{1,1}$ so that it stays in state s_1 for $3^2 = 9$ periods and then chooses actions so that it proceeds to s_2 and remains there for $3^3 = 27$ periods and so forth.

Show that for π the limit in (1.70) does not exist by showing that

$$\liminf_{N\to\infty} \frac{1}{N} E^{\pi}\left[\sum_{n=1}^{N} r(X_n, Y_n) \,|\, X_1 = s\right] \neq \limsup_{N\to\infty} \frac{1}{N} E^{\pi}\left[\sum_{n=1}^{N} r(X_n, Y_n) \,|\, X_1 = s\right].$$

with $S = \{s_1, s_2, s_3\}$, $A_{s_1} = \{a_{1,1}\}$, $A_{s_2} = \{a_{2,1}\}$ $\mathrm{A}_{s_3} = \{a_{3,1\}}$

Chapter 2

Simulation with Function Approximation

Everything we know is only some kind of approximation, because we know that we do not know all the laws yet. Therefore, things must be learned only to be unlearned again or, more likely, to be corrected.

Richard Feynman, American theoretical physicist, 1918-1988.

Modern applications of Markov decision processes require both function approximation (Chapter **??**) and simulation[1] (Chapter **??**). Such methods are usually referred to as *reinforcement learning (RL)* but as noted at the beginning of Part III, RL refers to any setting in which the decision maker (or agent) learns by trial and error.

Methods in this chapter motivate, describe and apply methods for:

- value-function approximation,
- state-action value function approximation, and
- policy approximation.

Whereas value function and state-action value function approximation generalize methods in Chapters **??** and **??**, this chapter also introduces the concept of policy approximation. Policy approximation refers to approximating the action-choice probability distribution of a randomized policy.

The general approach is to represent a value function, a state-action value function, or an action-choice probability in terms of a parametric function of features and then to estimate the parameters on-line or in batch mode using output from simulations. Often parameters are referred to as *weights*.

Figure 2.1 summarizes the methods described in this chapter. Note that this chapter considers only discounted and episodic models and requires familiarity with gradient descent and least-squares regression.

[1] As noted earlier we use the expression *simulation* to refer to either computer-based simulators or real-time experimentation.

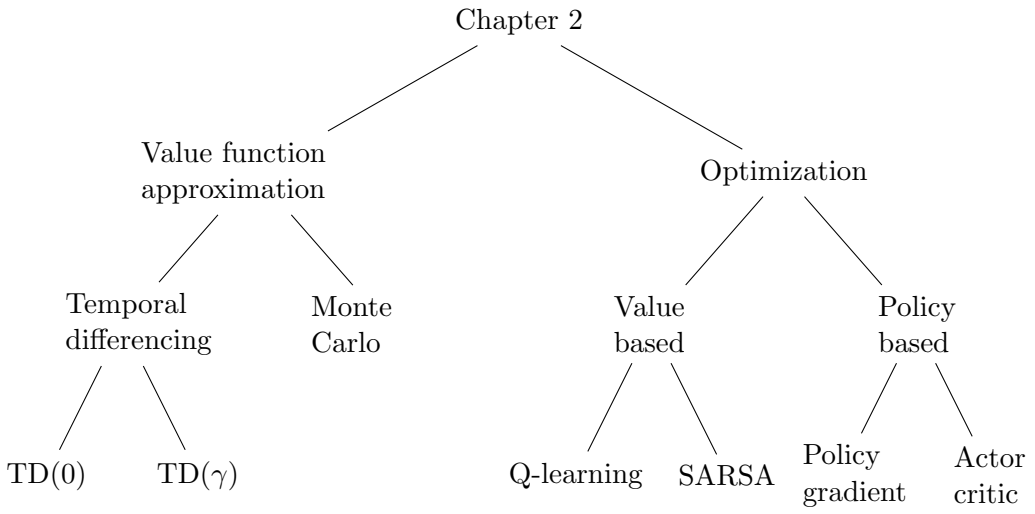


Figure 2.1: Schematic representation of methods described in Chapter 2

2.1 The challenge of large models

In contrast to the tabular models analyzed in Chapter **??**, models with large state (and action) spaces require methods to represent value functions and decision rules in forms suitable for computation and application. The following sections discuss methods for doing so.

2.1.1 Features

The underlying approach is to summarize the set of states or state-action pairs by lower-dimensional sets of real-valued functions defined on the set of states or state-action pairs. We refer to these functions as *features* or *basis functions*.

Specifying features

Choosing suitable features presents challenges. In physical models, such as navigating a drone through three-dimensional space, features may represent the position, velocity and angular orientation. In discrete models, with numerical-valued states and actions such as controlling a multi-dimensional queuing system, features may be (scaled) powers and products of powers of the number of entities in each queue.

In grid-world type models, specifying features may be more challenging. For example, in a robot guidance problem on an $M \times N$ grid, a possible choice of features is the row index, the column index and some higher order and cross-product terms of these quantities. Another possibility is to aggregate cells by choosing features to be indicator functions of overlapping or disjoint subsets of cells.

Note that choosing features to be indicator functions of all individual states or state-action pairs is equivalent to using a *tabular model*.

Combining features

Once features are specified, the issue of how to combine features to obtain estimates of quantities of interest arises. In general, value functions, state-action value functions or action choice probabilities can be approximated by:

1. linear functions of features
2. non-linear functions of features
3. neural networks[2]

Regarding notation, vectors of features evaluated at s or (s, a) are denoted by $\mathbf{b}(s)$ or $\mathbf{b}(s, a)$ and vectors of weights in both cases by $\boldsymbol{\beta}$. Components of vectors are written as $b_k(s)$ or $b_{k,j}(s, a)$.

2.1.2 Value function approximation

We begin with a discussion of approximating the value function approximation[3] of a fixed policy. For ease of exposition and to encompass many practical applications, we assume a linear value function approximation of the form:

$$v(s; \beta_0, \ldots, \beta_K) := \beta_0 b_0(s) + \beta_1 b_1(s) + \ldots + \beta_K b_K(s), \tag{2.1}$$

where $\{\beta_0, \ldots, \beta_k\}$ denote parameters and $\{b_0(s), \ldots, b_K(s)\}$ denote the value of the features evaluated at $s \in S$. We begin the indexing at 0 too conform to the convention in linear regression models in which $b_0(s_0) = 1$ for all $s \in S$ so that β_0 corresponds to a constant.

In vector notation (which we prefer) we write this as:

$$v(s; \boldsymbol{\beta}) = \mathbf{b}(s)^T \boldsymbol{\beta} \tag{2.2}$$

where $\mathbf{b}(s)$ denotes a (column) vector of pre-specified features evaluated at $s \in S$ and $\boldsymbol{\beta}$ denotes a weight vector. Since the quantity in (2.2) is a scalar, it equivalently can be written as $\boldsymbol{\beta}^T \mathbf{b}(s)$.

The beauty of such a representation is that in an expression such as (2.1) the coefficients can be interpreted as *marginal* rewards or costs. For example if s denotes the number of entities in a single-server queuing system, $v(s; \boldsymbol{\beta}) = \beta_0 + \beta_1 s$ represents an approximation to the expected infinite horizon discounted cost, then β_1 represents the increase in expected cost of adding an additional customer to the system.

In greater generality, $v(s, \boldsymbol{\beta})$ may be a nonlinear function such as a neural network. As a convention estimates of β will be denoted by $\hat{\boldsymbol{\beta}}$ and estimates of value functions by either $\hat{v}(s)$ or $v(s; \hat{\boldsymbol{\beta}})$.

[2]Of course, neural networks are non-linear functions but we distinguish them because they have a vast and specialized array of methods for parameter estimation.

[3]As noted earlier, the problem of estimating a value function for a specified policy is referred to as *value function prediction* in the computer science literature.

2.1.3 State-action value function approximation

Approximations of state-action value functions require features expressed in terms of states and actions. Determination of suitable functional forms for the components of the vector $\mathbf{b}(s,a)$ can present some challenges.

One can set

$$b_{k,j}(s,a) = f_k(s)h_j(a) \tag{2.3}$$

where $f_k(s)$ for $k = 0, 1, \ldots, K$ is a function defined on states and $j = 1, \ldots, J$ and $h_j(a)$ is a function defined on actions[4]. In the linear case this becomes

$$q(s,a;\boldsymbol{\beta}) \approx \sum_{k=0}^{K}\sum_{j=1}^{J} \beta_{k,j} f_k(s)h_j(a) \tag{2.4}$$

where $\boldsymbol{\beta} = \{b_{0,1}, \ldots, \beta_{K,J}\}$. In greater generality, $q(s,a,\boldsymbol{\beta})$ may be a pre-specified non-linear function of features.

When $A_s = \{a_1, \ldots, a_J\}$ contains a small number of elements and doesn't vary with the state, such as in the queuing service rate control model, one may choose $h_j(a)$ to be the indicator function of the action a_j, that is :

$$h_j(a) = I_{\{a_j\}}(a) = \begin{cases} 1 & a = a_j \\ 0 & a \neq a_j. \end{cases} \tag{2.5}$$

Consequently

$$b_{k,j}(s,a) = \begin{cases} f_k(s) & a = a_j \\ 0 & a \neq a_j \end{cases} \tag{2.6}$$

for $a \in A_s$ and $s \in S$. This is equivalent to representing $q(s,a)$ as a function of s that uses different weights for each $a \in A_s$.

To make this concrete consider a queuing-control model (such as in Section **??** with service rates a_1 and a_2 and suppose $f_0(s) = 1$ and $f_1(s) = s$ and $h_j(a) = I_{\{a_j\}}(a)$ as in (2.5). Then using (2.4), the approximation of $q(s,a)$ can be written as

$$\begin{aligned} q(s,a;\boldsymbol{\beta}) = &\ \beta_{0,1}f_0(s)h_1(a) + \beta_{0,2}f_0(s)h_2(a) + \beta_{1,1}f_1(s)h_1(a) + \beta_{1,2}f_1(s)h_2(a) \\ = &\ \beta_{0,1}b_0(s,a_1) + \beta_{0,2}b_0(s,a_2) + \beta_{1,1}b_1(s,a_1) + \beta_{1,2}b_1(s,a_2) \end{aligned}$$

which is equivalent to using the following possibly different linear functions for each action as follows

$$\begin{aligned} q(s,a_1;\boldsymbol{\beta}) &= \beta_{0,1} + \beta_{1,1}s \\ q(s,a_2,\boldsymbol{\beta}) &= \beta_{0,2} + \beta_{1,2}s. \end{aligned}$$

[4]Note that we start indexing $f_k(s)$ at 0 to allow for a constant function in the state approximation.

This means that using this approximation corresponds to approximating $q(s,a)$ by lines with different slopes and intercepts for each action.

We find it convenient to write (2.4) in vector form

$$q(s,a;\boldsymbol{\beta}) = \boldsymbol{\beta}^T \mathbf{b}(s,a) \tag{2.7}$$

where $\boldsymbol{\beta}$ is a column vector of weights of length $(K+1)J$ and $\mathbf{b}(s,a)$ is a column vector of features evaluated at (s,a) of length $(K+1)J$ where the ordering of weights and features in the vectors is consistent.

In the above example

$$\boldsymbol{\beta} = \begin{bmatrix} \beta_{0,1} \\ \beta_{0,2} \\ \beta_{1,1} \\ \beta_{1,2} \end{bmatrix}, \quad \mathbf{b}(s,a_1) = \begin{bmatrix} 1 \\ 0 \\ s \\ 0 \end{bmatrix} \quad \text{and} \quad \mathbf{b}(s,a_2) = \begin{bmatrix} 0 \\ 1 \\ 0 \\ s \end{bmatrix}. \tag{2.8}$$

Another alternative, which is appropriate when the states and actions are vectors $\mathbf{s}$ and $\mathbf{a}$ respectively, is to approximate $q(\mathbf{s},\mathbf{a};\boldsymbol{\beta})$ by a neural network with inputs $\mathbf{s}$ and $\mathbf{a}$.

State-action functions and value functions

Recall that given a state-action value function, the value function of the stationary policy d^∞ can be represented by:

$$v_d(s) = \begin{cases} q(s,d(s)) & \text{if } d \text{ is deterministic} \\ \sum_{a\in A_s} w_d(a|s) q(s,a) & \text{if } d \text{ is randomized.} \end{cases}$$

Combining this observation with (2.7) shows that we can approximation value functions by

$$\hat{v}_d(s) = \begin{cases} \boldsymbol{\beta}^T \mathbf{b}(s,d(s)) & \text{if } d \text{ is deterministic} \\ \sum_{a\in A_s} w_d(a|s) \boldsymbol{\beta}^T \mathbf{b}(s,a) & \text{if } d \text{ is randomized.} \end{cases}$$

State-action value function approximations and decision rules

When we analyzed tabular models, we were were able to specify decision rules **explicitly**. In the case of deterministic decision rules we could specify the action to choose in each state in the form of a look-up table and for a randomized policy we could specify a distribution over the set of actions. However in models with large state spaces this will not be possible, or even necessary.

When using function approximation, the quantity $\boldsymbol{\beta}$ assumes the role of a decision rule. That is instead of explicitly encoding the policy, one can use $\boldsymbol{\beta}$ to generate actions in practice or in an algorithm as follows:

Algorithm 2.1. Implicit action choice

1. Specify $\boldsymbol{\beta}$.
2. Select a state $s \in S$.
3. Compute $q(s, a; \boldsymbol{\beta}) = \boldsymbol{\beta}^T \mathbf{b}(s, a)$ for all $a \in A_s$.
4. (a) **Deterministic action selection:** Set $a_s = \arg\max_{a \in A_s} q(s, a; \boldsymbol{\beta})$.
   (b) **Randomized action selection:** Sample a_s using ϵ-greedy or using softmax action selection based on $q(s, a; \boldsymbol{\beta})$.
5. Return a_s.

To think of this another way, instead of carrying around a look-up table, our agent will have a (much lighter) weight function $\boldsymbol{\beta}$ in hand. When in state s, the agent can then apply Algorithm 2.1 and use the resulting action a_s.

From this perspective it would be difficult to specify $\boldsymbol{\beta}$ so as to represent a specific policy implicitly. However in the special case when the model is known and rewards are independent of the subsequent state, Choosing $\boldsymbol{\beta} = \mathbf{0}$ would result in setting $q(s, a; \boldsymbol{\beta}) = r(s, a)$ so that greedy action selection would generate a myopic policy.

2.1.4 Policy approximation

A third approach focuses on parameterizing the policy directly. It represents $w(a|s) = P[Y_n = a|X_n = s]$ as a parametric function of features defined in terms of states and actions. Of course the functional form must ensure that $w(a|s) \geq 0$ and $\sum_{a \in A_s} w(a|s) = 1$. Features are represented by the vector $\mathbf{b}(s, a)$ and weights by the vector $\boldsymbol{\beta}$. A convenient representation, referred to as a softmax[5] or logistic transformation, is given by

$$w(a|s; \boldsymbol{\beta}) = \frac{e^{\boldsymbol{\beta}^T \mathbf{b}(s,a)}}{\sum_{a' \in A_s} e^{\boldsymbol{\beta}^T \mathbf{b}(s,a')}} \tag{2.9}$$

where $\boldsymbol{\beta}_a$ is the parameter associated with choosing action a in state s.

An alternative is to represent $w(a|s; \boldsymbol{\beta})$ as a *neural network* where $\boldsymbol{\beta}$ is the vector of weights.

2.2 Value function approximation

This section focuses on simulation methods for approximating value function. It describes Monte Carlo and temporal differencing methods.

[5]Previously, the softmax function was used for exploration, here it is used to represent a policy directly.

2.2.1 Monte Carlo value function approximation

The idea is quite simple. Simulate (or observe) the process under a fixed policy for a set of starting states, store the observed values and then estimate the value function using least squares. For ease of exposition we only provide a starting state version of the algorithm.

We take the view that the objective of this section is to show how one might approximate value functions in applications in which the value function approximation is generated without analyst input as would be the case when seeking an optimal policy. Therefore the set of starting states and features are not chosen to conform to a specific policy.

Monte Carlo estimation of value function approximations in an episodic model

Recall that S_Δ denotes the set of stopped states[6] and $g(s)$ denotes the value on termination in state s. The algorithm assumes a randomized stationary policy.

Algorithm 2.2. Starting state Monte Carlo value function approximation in an episodic model

1. **Initialize:**
   (a) Choose $d \in D^{MR}$ and the number of replicates M.
   (b) Specify a subset of starting states $\bar{S}$ of S.
   (c) Create empty list VALUES(s) for each $s \in \bar{S}$.
2. **Generate values:**
   (a) For each $s \in \bar{S}$:
   (b) $m \leftarrow 1$,
   (c) While $m < M$,
      i. **Start episode:** $v(s) \leftarrow 0$.
      ii. Sample a from $w_d(\cdot|s)$.
      iii. Generate (s', r) or sample s' from $p(\cdot|s, a)$ and set $r = r(s, a, s')$.
      iv.
$$v(s) \leftarrow \begin{cases} r + v(s) & s' \notin S_\Delta \\ g(s') + v(s), & s' \in S_\Delta. \end{cases}$$

[6] As noted earlier we can assume without loss of generality that S_Δ consists of a single state by adding a zero-reward transition from each $s \in S_\Delta$ into a single absorbing state Δ.

v. **End episode:** If $s' \in S_\Delta$, append $v(s)$ to VALUES(s) and $m \leftarrow m+1$. Else, $s \leftarrow s'$ and return to 2c(ii).

3. **Estimate parameters:** Use data $\{(s,v) : s \in \bar{S}, v \in \text{VALUES}(s)\}$ to obtain (weighted) least squares estimates of parameters $\hat{\boldsymbol{\beta}} = (\hat{\beta}_0, \ldots, \hat{\beta}_K)$.

Some comments follow:

1. Instead of specifying a set of states at which to evaluate $v(s)$, states can be chosen randomly. Specifying states judiciously may result in more accurate parameter estimates.

2. The above algorithm generates $|\bar{S}|M$ data points of the form $(s, v(s))$. When $v(s)$ is assumed to be linear in parameters, least squares parameter estimates are available in closed form. When a nonlinear approximation is used, iterative estimation methods are required. Since these methods will be coded, both approaches can easily be included in optimization algorithms.

3. The above algorithm records values for the starting state only. As noted in Chapter **??** it was easy and more efficient to modify the algorithm to store estimates for all visited states.

4. When the variability of the Monte Carlo estimates vary with state, weighted least squares provides an alternative approach to parameter estimation.

An example

To illustrate these concepts, we apply the above algorithm to compute the value function in a model of optimal stopping in a random walk.

Example 2.1. Monte Carlo value function estimation on a random walk with stopping.

Consider the random walk model described in Example **??** but with $S = \{1, \ldots, 500\}$, continuation cost $c = 8$, stopping reward $g(s) = .3s + .7s^2$ and $p = .51$ where p denotes the probability of a transition from s to $s+1$ except in state N where it denotes the probability of remaining in state N.

The decision maker trades off the cost of continuing versus the cost of reaching the high reward states. There are two regions where the decision maker might consider stopping, in very low states when the cost of reaching the high reward states might exceed the benefit of waiting or in very high states.

This example investigates the use of linear polynomial approximations of the form

$$v(s; \beta_0, \beta_1, \ldots, \beta_K) = \beta_0 + \beta_1 s + \ldots + \beta_K s^K$$

for small values of K and considers three policies:

π_1: Stop in all states, that is $S_\Delta = S$.

π_2: Stop only when $N = 500$, that is $S_\Delta = \{500\}$.

π_3: Stop in states $\{1, \ldots, 50\}$ and $\{475, \ldots, 500\}$, that is $S_\Delta = \{1, \ldots, 50\} \cup \{475, \ldots, 500\}$.

Clearly under π_1, no approximation is necessary and $v^{\pi_1}(s) = g(s)$. To apply Algorithm 2.2 to π_2 and π_3, set $\bar{S} = \{1, 25, 50, 100, 150, \ldots, 400, 450, 475, 500\}$ and $M = 10$ replications for each $s \in \bar{S}$[a]

Figure 2.2a) shows linear (solid line) and cubic ($K = 3$) approximations to the Monte Carlo values for policy π_2. The linear approximation was

$$v^{\pi_2}(s; \hat{\beta}_0, \hat{\beta}_1) = -26549.1 + 404.7s.$$

and the cubic approximation was

$$v^{\pi_2}(s; \hat{\beta}_0, \hat{\beta}_1, \hat{\beta}_2, \hat{\beta}_3) = -11459.94 + 17.90s + 1.72s^2 - .0021s^3.$$

The cubic fit was slightly more accurate with a smaller standard error. The fourth order fits were almost identical to that of the cubic model.

Figure 2.2b shows linear and fourth-order approximations for policy π_3. Note that a fourth-order polynomial was necessary to well represent the values in the stopped region. The estimated approximations were:

$$v^{\pi_3}(s; \hat{\beta}_0, \hat{\beta}_1) = -14594.7 + 359.6s$$

and

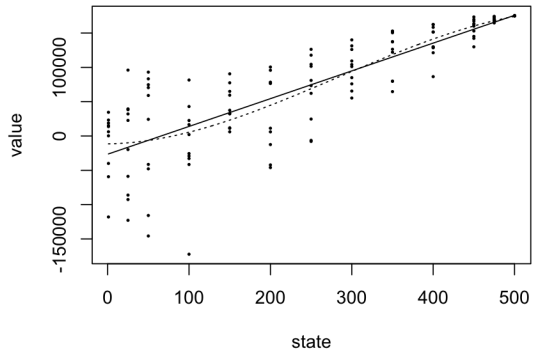
$$v^{\pi_3}(s, \hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_4) = 647.3 - 123.43s + 3.41s^2 - 0.009s^3 + 0.000009s^4.$$

We leave it as an exercise to develop suitable function approximations for other parameter values.

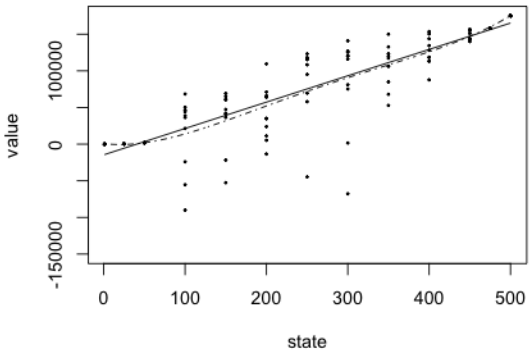
[a]The example chooses extra values close to the endpoints to achieve greater accuracy. Note also that replications in the stopping regions were unnecessary since $v(s) = g(s)$ therein.

Discounted models using truncation and Monte Carlo estimation

Our approach to approximating the value function in a discounted model is similar to that in an episodic model however we can either truncate trajectories after a fixed number of iterations or at a geometric stopping time. The truncation version runs each trajectory for N decision epochs. The geometric stopping time version is identical to that of an episodic model in which at each decision epoch the system can enter an



(a) Linear (solid line) and cubic (dashed line) value function approximations for policy π_2.



(b) Linear (solid line) and fourth-order (dashed line) value function approximations for policy π_3.

absorbing state with probability $(1 - \lambda)$. We formally state a truncation version of the algorithm here and leave algorithmic description and application of the geometric stopping version as an exercise. The algorithm assumes a randomized stationary policy.

Algorithm 2.3. Starting state Monte Carlo policy approximation in a discounted model with truncation.

1. **Initialize:**
   (a) Choose $d \in D^{MR}$.
   (b) Specify the number of replicates M and the truncation level N.
   (c) Specify a subset of starting states $\bar{S}$.
   (d) Create empty list VALUES(s) for all $s \in \bar{S}$.
2. **Generate values:**
   (a) For each $s \in \bar{S}$:
   (b) $m \leftarrow 1$.
   (c) While $m \leq M$,
      i. **Start episode:** $v(s) \leftarrow 0$ and $n \leftarrow 0$.
      ii. While $n < N$:
         A. Sample a from $w_d(\cdot|s)$.
         B. Simulate to obtain (s', r) or sample s' from $p(\cdot|s, a)$ and set $r = r(s, a, s')$.

C. $v(s) \leftarrow \lambda^n r + v(s)$.

D. $s \leftarrow s'$.

E. $n \leftarrow n + 1$.

iii. Add $v(s)$ to VALUES(s).

(d) $m \leftarrow m + 1$.

3. **Estimate parameters:** Use data $\{(s, v) : s \in \bar{S}, v \in \text{VALUES}(s)\}$ to obtain (weighted) least squares estimates of parameters $\hat{\boldsymbol{\beta}} = (\hat{\beta}_0, \ldots, \hat{\beta}_K)$.

Note that an every-state version of this algorithm would be problematic because the truncation lengths would vary from state to state.

An example

Example 2.2. Monte Carlo estimates of value function in queuing service rate control model.

This example applies Algorithm 2.3 to the queuing control model on $S = \{0, \ldots, 50\}$. It explores the quality of cubic polynomial approximations to the deterministic stationary policies derived from decision rules

$$d_1(s) = \begin{cases} a_1 & \text{for } s \leq 25 \\ a_3 & \text{for } s > 25 \end{cases}$$

and

$$d_2(s) = a_2 \quad \text{for } 0 \leq s \leq 50,$$

with $\lambda = 0.9, 0.95, 0.98$ over 40 replicates with common random number seeds. In each replicate, episodes are truncated at $N = 600$, $\bar{S} = \{0, 5, 10, \ldots, 45, 50\}$ and $M = 10$.

Outcomes were compared on the basis of the RMSE of the fit:

$$\text{RMSE} = \left(\frac{1}{|S|} \sum_{s \in S} (v(s, \hat{\boldsymbol{\beta}}) - v_\lambda^{d^\infty}(s))^2 \right)^{\frac{1}{2}}. \tag{2.10}$$

Note that $v_\lambda^{d^\infty}(s)$ can be easily computed exactly using policy evaluation methods in Chapter **??**. which can be computed because the values of these policies can be evaluated exactly.

Table 2.1 summarizes results. The column "True" establishes a baseline by providing the RMSE of a cubic polynomial fit to the exact value function. Observe that:

1. The accuracy of a cubic fit to the true model decreases with λ as does the accuracy of each of the Monte Carlo estimates.

2. The cubic model fits d_2 better than d_1. This is expected because of the step change in d_1 at $s = 26$ (see Figure 2.3a).

3. Least square fits provide more accurate estimates of the value function than weighted least square estimates using the estimated variance of values at each $s \in \bar{S}$.

4. Results (not shown) indicated that accuracy varied considerably with the truncation level N. When $N = 300$ the RMSE was more than three times greater than that shown when $\lambda = 0.98$.

Figure 2.3 compares, for a single replicate, the true value function, its cubic approximation and a cubic polynomial fitted to Monte Carlo estimates. Figure 2.3a shows that for decision rule d_1 and $\lambda = 0.95$, the cubic polynomial deviates from the true value function and the cubic fit based on the Monte Carlo simulation lies below the true value function. Figure 2.3b shows that for decision rule d_2 and $\lambda = 0.98$, the cubic polynomial approximation well approximates the true value function, the cubic fit based on the Monte Carlo simulation deviates from the true value function at high levels.

We investigate the impact of these discrepancies on optimization below.

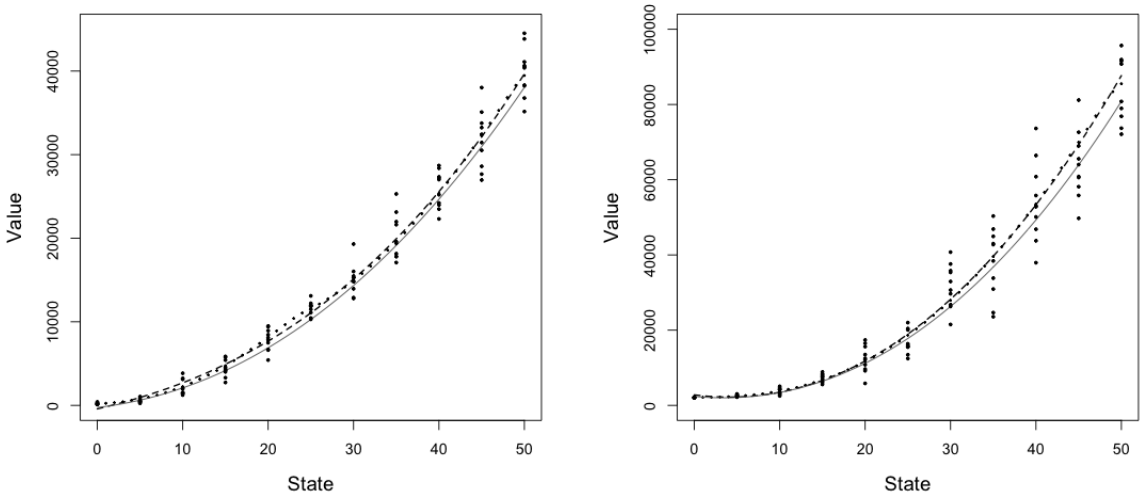
	d_1			d_2		
λ	True	MC-LS	MC-WLS	True	MC-LS	MC-WLS
0.90	67.99	551.14	569.17	59.15	567.90	579.10
0.95	344.73	1029.94	1135.56	153.05	1098.57	1151.50
0.98	1261.33	2161.07	2801.03	432.18	2480.18	2770.44

Table 2.1: Mean of RMSE over 40 replicates for least squares (MC-LS) and weighted least squares (MC-WLS) fit to Monte Carlo estimates and the RMSE for a cubic regression fit to the exact value (True) as a function of discount rate and decision rule.

2.2.2 TD(0) with value function approximation

As an alternative to Monte Carlo estimation in which estimates are derived after simulating (or observing) many trajectories originating from each of a subset of states, this section describes an online approach that generalizes TD(0) by updating parameter estimates after each state transition[7].

[7]As noted above, when the features are indicators of states or states and actions, this is equivalent to the online methods in Chapter **??**.



(a) Decision rule d_1 with $\lambda = 0.95$

(b) Decision rule d_2 with $\lambda = 0.98$.

Figure 2.3: Plots of estimated and true value functions based on starting state Monte Carlo based on a single replicate. Points indicate Monte Carlo values, the dotted line equals the true value function, the dashed line indicates the fitted cubic polynomial approximation to the true value function and the grey solid line denotes the least square estimate based on the Monte Carlo values. Note that the y-axis scales for the two figures differ.

Motivation

This section applies stochastic approximation methods in Chapter **??** Appendix B to obtain a recursive method for estimating a vector of weights in an approximate value function.

Let d denote a Markovian randomized decision rule, d^∞ the corresponding stationary policy and s an arbitrary state in S. The objective is find a vector of weights $\boldsymbol{\beta} \in \Re^K$ so as to "solve" the Bellman equation

$$v(s;\boldsymbol{\beta}) \approx E\big[r(X,Y,X') + \lambda v(X';\boldsymbol{\beta})\big|X = s\big]$$

for all $s \in S$, where the expectation is with respect to the conditional distribution of the action Y and next state X' under decision rule d given state $s \in S$.

To make this more concrete; the goal is to find a $\boldsymbol{\beta}$ that minimizes the expected squared error loss

$$E^{d^\infty}\Big[\big(r(X,Y,X') + \lambda v(X';\boldsymbol{\beta}) - v(X;\boldsymbol{\beta})\big)^2\Big|X = s\Big] \tag{2.11}$$

for all $s \in S$ where the expectation is respect to the conditional distribution of action Y and the next state X' under the probability distribution corresponding to decision

rule d in state s. Since a different $\boldsymbol{\beta}$ may achieve the minimum above for each $s \in S$, an alternative is the objective

$$G_d(\boldsymbol{\beta}) := E^{d^\infty}\Big[(r(X,Y,X') + \lambda v(X';\boldsymbol{\beta}) - v(X;\boldsymbol{\beta}))^2\Big] \tag{2.12}$$

where the expectation is now with respect to the state X as well. Possible choice for a distribution for X are a uniform distribution over states or the stationary distribution of the Markov chain corresponding to d^∞.

If one could evaluate the above expectation directly then the gradient descent recursion

$$\boldsymbol{\beta}' = \boldsymbol{\beta} - \tau \nabla_{\boldsymbol{\beta}} G_d(\boldsymbol{\beta})$$

would provide an estimate of $\boldsymbol{\beta}$.

We now describe an alternative approach suitable for use on simulated data. This scheme assumes that given a specified value of $\boldsymbol{\beta}'$ in state s, one observes a pair (r, s'), evaluates $u := r + \lambda v(s';\boldsymbol{\beta})$ and seeks a vector $\boldsymbol{\beta}$ to minimize

$$g(\boldsymbol{\beta}) := (u - v(s;\boldsymbol{\beta}))^2$$

regarding the scalar u as a fixed value independent of $\boldsymbol{\beta}$. To do so, TD(0) applies gradient descent to $g(\boldsymbol{\beta})$ by taking the gradient of $g(\boldsymbol{\beta})$,

$$\nabla_{\boldsymbol{\beta}} g(\boldsymbol{\beta}) = -2(u - v(s;\boldsymbol{\beta}))\nabla_{\boldsymbol{\beta}} v(s;\boldsymbol{\beta})$$

and using the recursion

$$\boldsymbol{\beta}' = \boldsymbol{\beta} - \tau \nabla_{\beta} v(s;\boldsymbol{\beta})\big(u - v(s;\boldsymbol{\beta})\big) \tag{2.13}$$

where the factor 2 is absorbed into τ.

When $v(s,\boldsymbol{\beta})$ is a linear function of $\boldsymbol{\beta}$, $v(s;\boldsymbol{\beta}) = \boldsymbol{\beta}^T \mathbf{b}(s)$, t $\nabla_{\boldsymbol{\beta}} v(s;\boldsymbol{\beta}) = \mathbf{b}(s)$, so that the gradient descent recursion becomes:

$$\boldsymbol{\beta}' = \boldsymbol{\beta} - \tau \mathbf{b}(s)\big(u - \boldsymbol{\beta}^T \mathbf{b}(s)\big). \tag{2.14}$$

The scalar quantity $\delta := (r + \lambda v(s';\boldsymbol{\beta}) - v(s;\boldsymbol{\beta}))$ is often referred to as a *temporal difference* or *Bellman error* and the following recursion is referred to as TD(0):

$$\boldsymbol{\beta}' = \boldsymbol{\beta} - \tau\, \delta\, \mathbf{b}(s). \tag{2.15}$$

Note that in this expression τ and δ are scalars and $\boldsymbol{\beta}$ and $\mathbf{b}(s)$ are column vectors.

TD(0) with value function approximation in algorithmic form; linear approximation

We now provide an algorithm for implementing TD(0) in a discounted model. It evaluates a Markovian random decision rule d and allows for flexibility in selecting the sequence of states.

Algorithm 2.4. TD(0) with linear value function approximation.

1. **Initialize:**
   (a) Specify the number of iterations N and the learning rate sequence $\{\tau_n : n = 1, \ldots\}$.
   (b) Specify a decision rule d to be evaluated, an initial vector $\boldsymbol{\beta}$ and a state $s \in S$.
   (c) $n \leftarrow 1$.
2. **Iterate:** For $n < N$:
   (a) Sample a' from $w_d(a|s)$.
   (b) Generate (s', r) or sample $s' \in S$ from $p(s'|s, a)$ and set $r = r(s, a', s')$.
   (c) **Compute the temporal difference:** Set $v(s; \boldsymbol{\beta}) = \mathbf{b}(s)^T\boldsymbol{\beta}$, $v(s'; \boldsymbol{\beta}) = \mathbf{b}(s')^T\boldsymbol{\beta}$ and
   $$\delta = r + \lambda v(s'; \boldsymbol{\beta}) - v(s; \boldsymbol{\beta}). \tag{2.16}$$
   (d) **Update $\boldsymbol{\beta}$:**
   $$\boldsymbol{\beta}' \leftarrow \boldsymbol{\beta} - \tau_n \delta \mathbf{b}(s) \tag{2.17}$$
   (e) $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta}'$ and $n \leftarrow n + 1$.
   (f) **Next state generation:**
   (Long-trajectory) $s \leftarrow s'$;
   (Random) Generate s from a uniform distribution on S.
   (Hybrid) Specify δ small and a state s_0. With probability δ, $s \leftarrow s_0$ and with probability $1 - \delta$, $s \leftarrow s'$.
3. Return $\hat{\boldsymbol{\beta}}$.

Some comments on implementing this algorithm follow:

1. **Specifying d:** The algorithm is expressed in a form that assumes $d(s)$ can be explicitly specified for all $s \in S$. Unfortunately when S is large, this may not be possible. Optimization algorithms, described below, get around this requirement in different ways.
   - **Q-learning**, which is based on state-action value functions $q(s, a)$, chooses a' (deterministically) in step 2(a) by
   $$a' \in \arg\max_{a \in A_s} q(s, a; \boldsymbol{\beta}).$$

- **Actor-critic** samples a' from $w(\cdot|s;\boldsymbol{\beta}^C)$ where the estimate of the weight $\boldsymbol{\beta}$ is updated using gradient ascent.

2. **Convergence:** When $v(s;\boldsymbol{\beta})$ is a linear function of $\boldsymbol{\beta}$ and the features are linearly independent, the iterates of $\boldsymbol{\beta}$ converge[8] with probability 1 provided the samples follow the trajectory of the Markov chain corresponding to δ and $(\tau_n : n = 1, 2, \ldots)$ satisfies the Robbins-Monro conditions. Of course this also means that the corresponding value functions converge. Generalization to non-linear approximations is more problematic.

3. **Initialization:** As in most nonlinear optimization problems, convergence is sensitive to initial specification of $\boldsymbol{\beta}$. We believe a limited Monte Carlo analysis based on one or more replicates at a selected subset of states followed by least squares parameter estimate provides reliable staring values.

4. **Stopping Rules:** We omit a precise stopping rule in the algorithm statement above. One could be based on either changes in parameter values

$$\left[\frac{1}{K+1}\sum_{k=0}^{K}(\beta'_k - \beta_k)^2\right]^{\frac{1}{2}} \tag{2.18}$$

or in changes in fitted values:

$$\left[\frac{1}{|S|}\sum_{s\in S}(v(s,\boldsymbol{\beta}') - v(s,\boldsymbol{\beta}))^2\right]^{\frac{1}{2}}. \tag{2.19}$$

or a weighted variant thereof. Use of these criteria have limitations: (2.18) may be dominated by the impact of a few large coefficients while (2.19) may be computationally prohibitive due to the magnitude of S. Note that (2.19) may be the more appropriate because it is consistent with least squares objective function used to derive TD(0).

5. **State updating:** Step 2(f) provides three approaches for generating "subsequent" states for evaluation: following the trajectory, randomly restarting or a combined approach. When this algorithm is used in a simulation environment such as in Example 2.3 below, the random restart and hybrid methods will provide better coverage of S, especially if s_0 is chosen judiciously. This is because under a specified decision rule, the process may occupy only a small portion of the state space. In real-world implementations random restarts may be difficult to implement so that the long-trajectory approach may be necessary. The hybrid approach provides a restart method that might be easier to implement.

[8] See Tsitsiklis and van Roy [1997].

6. **Episodic models:** Since episodic models terminate at a state in S_Δ after a finite (but random) number of iterations, it is necessary to provide a mechanism to generate enough data to accurately estimate parameters. One possibility is to jump to a random state after termination. Also, in an episodic model, λ may be set equal to 1.

Value function approximation in the queuing service rate control model

Because we intend to generalize this algorithm to state-action value function, we carried out a detailed study of the impact of learning rate and restart method for 3 decision rules.

Example 2.3. This example applies Algorithm 2.4 to the model in Example 2.2 with $\lambda = 0.95$. It uses a cubic polynomial approximation to the value function of three deterministic stationary policies based on the decision rules:

$$d_1(s) = \begin{cases} a_1 & 0 \le s \le 25 \\ a_3 & 25 < s \le 50 \end{cases}$$

$$d_2(s) = \quad a_2 \quad 0 \le s \le 50$$

$$d_3(s) = \begin{cases} a_2 & 0 \le s \le 9 \\ a_3 & 10 \le s \le 29 \\ a_1 & 30 \le s \le 50 \end{cases}$$

Step 2(f) used long-trajectory, random-restart options and a hybrid variant that with probability 0.008 jumped to $s_0 = 50$ and with probability 0.992 followed the existing trajectory. Note that under d_3 choosing $s_0 = 0$ would be more appropriate.

As a result of initial experimentation four learning rates are compared: $\tau_n = 0.1n^{-0.5}$, $\frac{0.1log(n+1)}{n}$, $\frac{0.1}{1+10^{-5}n^2}$, $\frac{150}{750+n}$ referred to respectively as polynomial, logarithmic, STC and ratio. Experiments compared all 36 combinations of these configurations over 40 replicates of simulations of length K =20,000 using common random number seeds for all instances in a replicate.

Preliminary calculations showed that that "convergence" of estimates was highly sensitive to starting values and feature scaling. To obtain reliable initial weights, implementations:

> Were initialized with one replicate of starting-state Monte Carlo on a subset $\bar{S}$ of states;
>
> Scaled the states by subtracting the mean and dividing by the standard deviation, and
>
> Used least squares estimation (linear regression) to obtain preliminary estimates of the parameters of a cubic polynomial.

Figure 2.4 compares the effect of the the factors on the quality of the fit for each configuration. From it one can conclude that:

1. Fits for d_1 and d_2 were more accurate than for d_3.

2. The random restart method gave the best result over the three decision rules, however the hybrid method gave similar results to random restart for d_1 and d_2. The reason it didn't work well for d_3 is that under this policy the system remained in high occupancy states. If the hybrid method was modified to jump to a low occupancy state, we suspect its performance would be equivalent to the random restart method.

3. For d_1 and d_2, long trajectory methods gave the least accurate and most variable estimates in comparison to the other restart methods. This was because under these policies, the system remained in low occupancy states in steady state. Consequently, by allowing a small probability of restarting at state 50, accuracy and variability improved considerably.

4. For random restart, the effect of learning rate was small. With random restart, the STC and exponential learning rates gave the best results. Surprisingly the exponential learning rate worked well across all configurations, especially when using long trajectories.

We now describe 5 randomly selected replicates in more detail. Each used decision rule d_1, STC learning rate and compared random and hybrid state generation. Table 2.2 summarizes the RMSE of the fit. As a baseline the RMSE from fitting a cubic regression function to the exact value function is 344.73 so we would not expect any approximation algorithm to generate a better fit. To obtain a better fit one needs to change the features, For example spline functions provide an attractive alternative. The command *bs()* in R [2021] provides several options.

From Table 2.2 observe that:

1. In replicate 3 both TD(0) methods gave a worse fit than the Monte Carlo initialization that was based on one realization at 5 states. In all other replicates at least one of the TD(0) methods improved the fit from the initial value.

2. There was no clear pattern as to whether hybrid or random state updating gave more accurate estimates.

3. Neither method gave estimates with RMSE close to that of the regression fit to the true value function.

We conclude from this example that using Algorithm 2.4 does not provide very accurate estimates of the value function. The following section considers a TD(γ) variant that might provide better estimates.

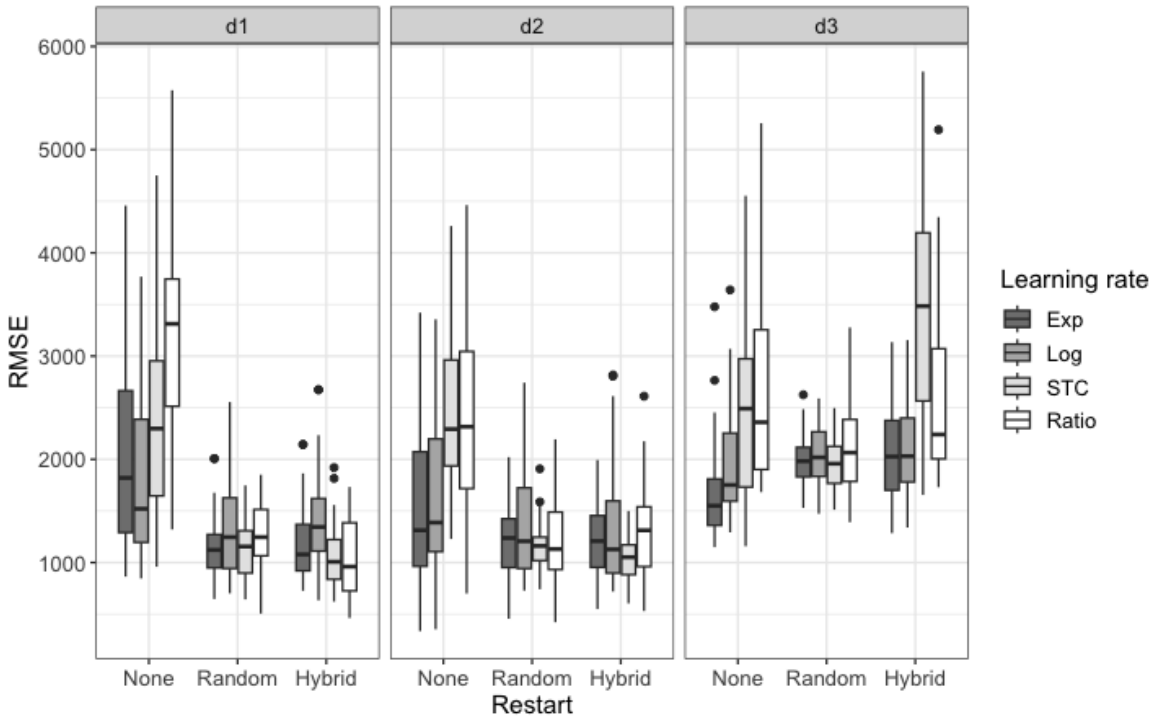


Figure 2.4: Comparison of the RMSE of value function estimates for the queuing control model in Example 2.3. As a frame of reference, Table 2.1 showed that a cubic polynomial approximation derived from Monte Carlo estimates had an average RMSE of 1029.04 for d_1 and 1098.57 for d_2. **(change Exp to Poly in plot)**

2.2.3 TD(γ) with linear value function approximation

This section generalizes TD(γ) to models with value function approximation. It provides it in algorithmic form, offers some comments and illustrates it with an example. It consider the case of linear approximations.

Algorithm 2.5. TD(γ) with linear value function approximation.

1. **Initialize:**
   (a) Specify a decision rule d, an initial vector $\boldsymbol{\beta}$ a state $s \in S$.
   (b) Specify the number of iterations N and the learning rate sequence $(\tau_n : n = 1, \ldots)$.

	Replicate				
Method	1	2	3	4	5
MC - initialization	1209	1259	794	973	2111
TD(0) - hybrid	967	1209	873	882	1620
TD(0) - random	1270	1070	1441	926	1131

Table 2.2: RMSE of fitted value functions for 5 replicates of Monte Carlo initialization and TD(0) with two state generation methods for decision rule d_1. Note that the cubic regression fit to the exact value function has RMSE equal to 344.73. Table 2.1 shows that the MC estimate with LS estimation had an average RMSE of 1029.94 over 40 replicates.

(c) Set the $(K+1)$-dimensional eligibility trace vector $\mathbf{z} = \mathbf{0}$ and specify $\gamma \in [0, 1]$.

(d) $n \leftarrow 1$.

2. **Iterate:** For $n < N$:

(a) Sample a from $w_d(\cdot|s)$.

(b) Generate (s', r) or sample $s' \in S$ from $p(\cdot|s, a)$ and set $r = r(s, a, s')$.

(c) **Compute the temporal difference:** Set $v(s; \boldsymbol{\beta}) = \mathbf{b}(s)^T\boldsymbol{\beta}$, $v(s'; \boldsymbol{\beta}) = \mathbf{b}(s')^T\boldsymbol{\beta}$ and

$$\delta = r + \lambda v(s'; \boldsymbol{\beta}) - v(s; \boldsymbol{\beta}). \tag{2.20}$$

(d) **Update eligibility trace:**

$$\mathbf{z}' = \gamma\lambda\mathbf{z} + \mathbf{b}(s) \tag{2.21}$$

(e) **Update $\boldsymbol{\beta}$:**

$$\boldsymbol{\beta}' \leftarrow \boldsymbol{\beta} - \tau_n\delta\mathbf{z}' \tag{2.22}$$

(f) $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta}'$, $\mathbf{z} \leftarrow \mathbf{z}'$, $n \leftarrow n + 1$, $s \leftarrow s'$.

3. Return $\boldsymbol{\beta}$.

Some comments follow:

1. We state the long-trajectory discounted variant of the algorithm only.

2. The eligibility trace is a weighted linear combination of past feature vectors. The smaller the value of γ, the more quickly the effect of past features dies out. Note that when using non-linear value function approximations, the gradient of

the value-function (with respect to its parameters) replaces the feature vector in (2.21).

3. Note that when $\gamma = 0$, the algorithm reduces to TD(0).

4. We leave it as an exercise to show that this reduces to Algorithm **??** when features are indicators of states.

Example 2.4. This example investigates the application of the long-trajectory variant of TD(γ) to the queuing model analyzed in Example 2.3. That example showed the the best estimates occurred with polynomial and logarithmic learning rates. Consequently this example compares 40 replicates with common random seeds with these two learning rates and $\gamma = 0, 0.25, 0.5, 0.75$ for each of three decision rules. In each instance, N =20,000. Estimates are compared on the basis of the RMSE of the fitted values. Figure 2.5 summarizes results graphically. Observe that the effect of γ differed across decision rules:

1. For d_1, estimates using $\gamma = 0.75$ gave the most accurate estimates with those using the polynomial learning rate the least variable.

2. For d_2, the TD(0) estimates were least variable and had the lowest average[a] RMSE.

3. For d_3, the TD(0) estimates were the most accurate and least variable.

Thus in this example the benefits of using TD(γ) over TD(0) were small.

[a]Result not shown, the boxplot below shows the median RMSE.

2.3 Optimization based on value function approximation: Value iteration type algorithms

This section develops simulation-based methods in which value functions or state-action value functions are approximated by functions of features. It considers value iteration type algorithms such as Q-learning and policy iteration algorithms.

We restrict attention to discounted models. Algorithms can be easily generalized to episodic models by adding a mechanism to start the next episode after termination of the previous episode. Recall that value iteration is based on either the value function recursion

$$v(s) \leftarrow \max_{a \in A_s} \Big\{ \sum_{j \in S} p(j|s,a)(r(s,a,j) + \lambda v(j) \Big\} \tag{2.23}$$

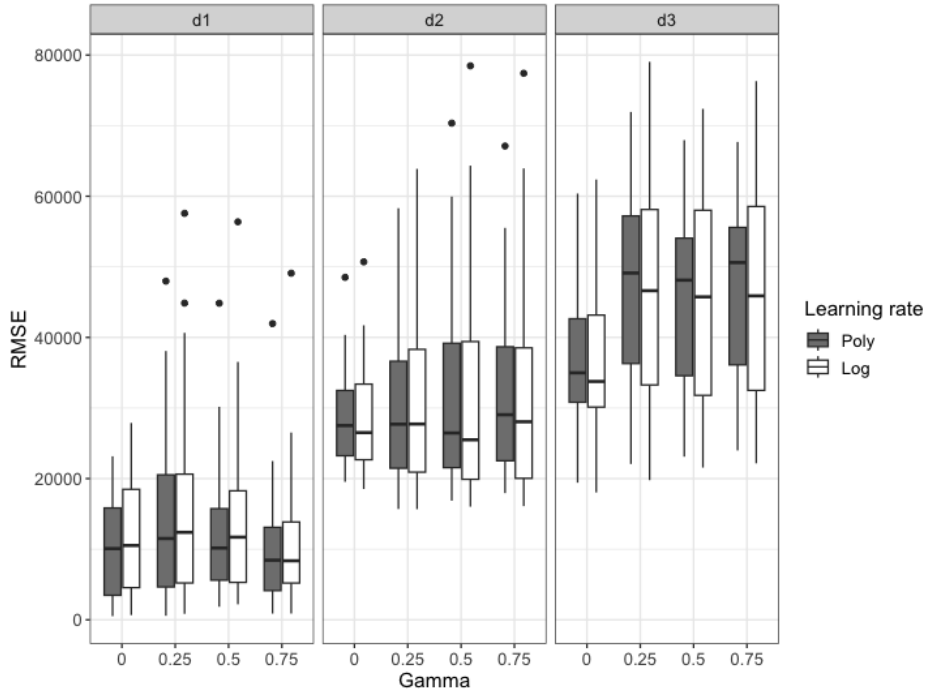


Figure 2.5: Boxplots of the RMSE of TD(γ) estimates based on 40 replicates of two learning rates and three decision rules.

or the state-action value function recursion

$$q(s,a) \leftarrow \sum_{j \in S} p(j|s,a)\big(r(s,a,j) + \lambda \max_{a \in A_s} q(j,a)\big). \tag{2.24}$$

Note that when a model is available, that is when we know $r(s,a,j)$, $p(j|s,a)$ and the optimal value-function $v^*(s)$, we can set

$$q(s,a) = \sum_{j \in S} p(j|s,a)\big(r(s,a,j) + \lambda v^*(j)\big). \tag{2.25}$$

however in model-free environments we must work directly with the state-action value function.

2.3.1 Q-learning with function approximation

(I'm not completely happy with this intro. I don't think the justification is sufficiently rigorous.) As in the case of a tabular model, Q-learning seeks to find a $\boldsymbol{\beta}$ that minimizes

$$E\left[\Big(r(X,Y,X') + \lambda \max_{a' \in A_X} q(X',a';\boldsymbol{\beta}) - q(X,Y;\boldsymbol{\beta})\Big)^2 \Big| X = s, Y = a\right] \tag{2.26}$$

over $\boldsymbol{\beta} \in \Re^K$ where the expectation is with respect to the distribution $p(\cdot|s,a)$.

The gradient[9] of this expression may be written as

$$-2E\left[\Big(r(s,a,X') + \lambda \max_{a' \in A_{X'}} q(X',a';\boldsymbol{\beta}) - q(s,a;\boldsymbol{\beta})\Big)\nabla_{\boldsymbol{\beta}} q(s,a,\boldsymbol{\beta})\right]$$

which when $q(s,a)$ is approximated by a linear function of features $\mathbf{b}(s,a)$, becomes

$$-2E\left[\Big(r(s,a,X') + \lambda \max_{a' \in A_{X'}} \boldsymbol{\beta}^T\mathbf{b}(X',a') - \boldsymbol{\beta}^T\mathbf{b}(s,a)\Big)\nabla_{\boldsymbol{\beta}} q(s,a,\boldsymbol{\beta})\right].$$

Noting this expression for the gradient leads to the following recursion for estimating weights $\boldsymbol{\beta}$:

$$\boldsymbol{\beta}' \leftarrow \boldsymbol{\beta} + \tau\Big(r(s,a,s') + \lambda \max_{a \in A'_s} \boldsymbol{\beta}^T\mathbf{b}(s',a') - \boldsymbol{\beta}^T\mathbf{b}(s,a)\Big)\mathbf{b}(s,a). \tag{2.27}$$

where s' is the result of some form of sampling. Note that in the above expression the $\boldsymbol{\beta}$ and $\mathbf{b}(s,a)$ are column vectors and the expression in ()'s is a scalar.

An algorithm

The following Q-learning algorithm seeks to find the coefficients of a linear approximation to optimal state-action value function $q^*(s,a)$. Because we are implicitly assuming that the set of states is large, the output of the algorithm is a low dimension vector $\hat{\boldsymbol{\beta}}$ of weights which can be used in subsequent applications to determine action choice in state s by setting

$$\hat{d}(s) \in \arg\max_{a \in A_s} q(s,a;\hat{\boldsymbol{\beta}}). \tag{2.28}$$

Moreover the corresponding value function approximation can be obtained from

$$\hat{v}(s) = \max_{a \in A_s} q(s,a;\boldsymbol{\beta}). \tag{2.29}$$

over a range of states.

Algorithm 2.6. Q-Learning with linear state-action value function approximation in a discounted model

1. **Initialization:**
   (a) Initialize $\boldsymbol{\beta}$.
   (b) Specify the learning rate $\{\tau_n : n = 1, 2, \ldots\}$ and a sequence $\{\epsilon_n\, n = 1, 2, \ldots\}$ for ϵ-greedy action selection.

[9]Note that when computing this gradient, the state-action pair is regarded as fixed so that the gradient is evaluated at (s,a).

(c) Choose $s \in S$.

(d) Specify the number of iterations N and $n \leftarrow 1$.

2. **Re-evaluation** While $n < N$

(a) Choose $a \in A_s$ ϵ_n-greedily.

(b) Generate (s', r) or sample $s' \in S$ from $p(s'|s, a)$ and set $r = r(s, a, s')$.

(c) Set

$$q(s, a; \boldsymbol{\beta}) = \boldsymbol{\beta}^T \mathbf{b}(s, a) \tag{2.30}$$

(d) Set

$$\delta = \Big(r(s, a, s') + \lambda \max_{a' \in A_s} q(s', a'; \boldsymbol{\beta}) - q(s, a; \boldsymbol{\beta}) \Big) \tag{2.31}$$

(e) Update $\boldsymbol{\beta}$ according to

$$\boldsymbol{\beta}' \leftarrow \boldsymbol{\beta} + \tau_n \mathbf{b}(s, a)\delta \tag{2.32}$$

(f) $s \leftarrow s'$, $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta}'$ and $n \leftarrow n + 1$

3. **Termination:** Return $\boldsymbol{\beta}$.

We comment briefly on this algorithm.

1. The algorithm requires ϵ-greedy or softmax action selection to ensure exploration. Without it, the algorithm would end up evaluating a sub-optimal policy.

2. When computing the learning rate, the index is chosen to be the iteration number to avoid storing the number of visits to each state-action pair.

3. As stated, the algorithm generates sequences of states and actions according to the underlying transition probabilities. Note that in step 2(f), $s \leftarrow s'$ can be replaced by "sample s from S" or the hybrid variant described previously.

4. The above algorithm never explicitly computes a decision rule. Action selection in step 2(b) implicitly corresponds to a policy but as $\boldsymbol{\beta}$ changes, the elusive policy shifts. We avoid this issue in the policy iteration algorithms below by fixing $\boldsymbol{\beta}$ for several iterations.

5. Note this is an off-policy algorithm since the update in (2.36) may be based on a different action than that followed by the trajectory. A SARSA variant, as in Chapter **??**, would provide an on-policy alternative.

6. Unlike its tabular counterpart, that corresponds to using state-action indicators as basis functions, there are no general convergence guarantees for Q-learning with function approximation.

7. The algorithm can be easily modified to allow for non-linear state-action value function approximations. This would require replacing 2(d) to allow for a non-linear functional form and replacing $\mathbf{b}(s, a)$ in 2(f) by the gradient of the non-linear function.

2.3.2 Q-learning in a discounted model

We now apply Q-learning to the infinite horizon discounted queuing service rate control model.

Example 2.5. This example again considers the queuing service rate control model on $S = \{0, \ldots, 50\}$. Basis functions are of the form (2.3) where the terms in $f_k(s)$ equals the scaled powers of a cubic polynomial and the terms in $h_j(a)$ represent indicator functions of each possible action. This is a equivalent to representing $q(s, a)$ by a cubic polynomial in which the weights vary with the action.

As seen in Example 2.3, good starting values were required to ensure convergence. By using the above specification for the approximation, preliminary weight estimates for the cubic polynomial approximation can be obtained using Monte Carlo for each action separately or alternative randomizing over all actions with equal probability. Note that these differ from the q-functions sought by the algorithm because after choosing a state and action, they follow the value function of the specified policy, not the optimal value function.

The algorithm required considerable tuning to obtain convergence to reasonable policies. Results are described in the case N =250,000, learning rate

$$\tau_n = \frac{5000}{50000 + n}$$

and greedy parameter

$$\epsilon_n = \frac{100}{400 + n}.$$

The indices of the learning rate and ϵ-greedy parameter were the iteration number.

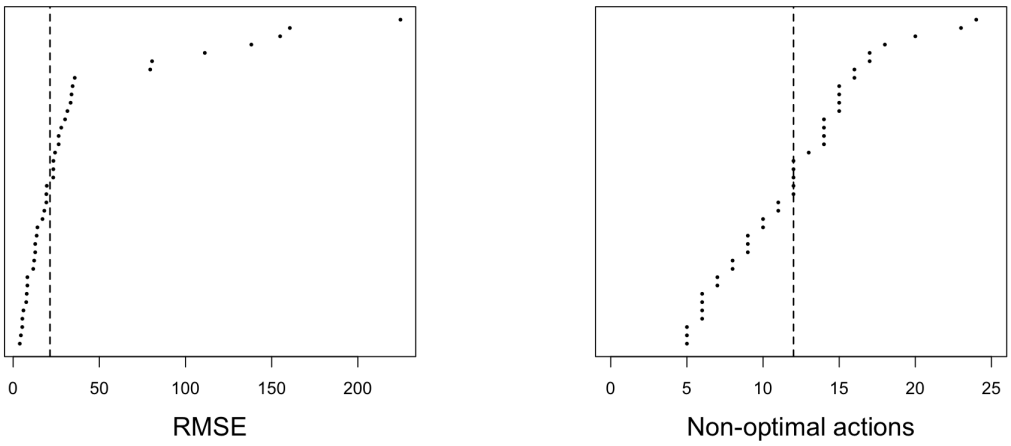
Figure 2.6 summarizes output from 40 replicates on the basis of the RMSE of the value function of the greedy policy relative to the optimal value function, and the number of states at which the greedy policy differed from the optimal policy.

Figure 2.6a show that in 33 out of the 40 replicates the RMSE was less than 35.7 with a median of 21.4. To put this in context the the value function range was (221, 39,441) indicating a high degree of accuracy.

In spite of this, the greedy policy differed from the optimal policy in all replicates (Figure 2.6b). In 35 out of 40 replicates the greedy policy had the same

structure as the optimal policy however the points at which actions changed differed. The median number of non-optimal actions was 12.

Consequently from the perspective of RMSE, the results were satisfactory however this approach failed to identify the optimal policy in all cases. Changing the starting values for the algorithm and basis functions did not improve the accuracy of the greedy-policy. We leave it to the reader to try to improve on these results.



(a) Plot of sorted RMSEs. Vertical dashed line denotes median.

(b) Plot of sorted number of non-optimal actions. Vertical dashed line denotes median.

Figure 2.6: Summary order statistics of the RMSE and number of non-optimal actions in 40 replicates obtained in Example 2.5.

2.3.3 Q-learning in an episodic model

This simple example applies Q-learning to a simple shortest path problem on a rectangular grid with M rows and N columns. In each cell, the robot can *try* to move up, down, right or left. If a move is impeded by a boundary, the robot remains in its current location. The goal is to reach a pre-specified cell (m^*, n^*). When that cell is reached the robot receives a reward of R. The cost of each attempted step is c (corresponding to a reward of $-c$). The objective is to maximize the expected total reward.

Markov decision process formulation

An MDP formulation follows:

States: $S = \{(i, j) : i = 1, \ldots, M, j = 1, \ldots, N\}$

Actions: For all $s = (i, j) \in S$, $A_s = A$

$$A = \begin{cases} \{\text{up, down, left, right}\} & (i,j) \neq (m^*, n^*) \\ \{\delta\} & (i,j) = (m^*, n^*) \end{cases}$$

Rewards:

$$r\big((i,j), a, (i',i')\big) = \begin{cases} c & (i',j') \neq (m^*, n^*),\ a \in A_{(i,j)} \\ R & (i,j) \neq (m^*, n^*),\ (i',j') = (m^*, n^*),\ a \in A_{(i,j)} \\ 0 & (i,j) = (m^*, n^*), a = \delta \end{cases}$$

Transition probabilities:

$$p((i',j')|(i,j),a) = \begin{cases} 1 & \text{if move from } (i,j) \text{ to } (i',j') \text{ is possible under action } a \\ 0 & \text{otherwise.} \end{cases}$$

Note that the transition probabilities are awkward to write down explicitly because there are many cases to enumerate[10], but easy to code in a simulation.

This model can be analyzed as an undiscounted total return model in which the action choice probabilities introduce randomness in transitions. There are many improper policies, each having reward $-\infty$, however there exists (many) deterministic optimal policies that can be easily found by inspection. Although exact methods (from Chapter **??** converge, we found that it was necessary to introduce a discount factor to ensure reliable convergence.

Feature choice

This example compared three implementations of Q-learning differing by the choice of the form of the approximate state-action value function. All were based on representation (2.4) in which $h_j(a)$ was an indicator function of the action. That is

$$h_1(a) := I_{\{\text{up}\}}(a), \quad h_2(a) := I_{\{\text{down}\}}(a), \quad h_3(a) := I_{\{\text{left}\}}(a), \quad h_4(a) := I_{\{\text{right}\}}(a)$$

for $a \in A$. They differ on the form of features $f_k(i, j)$ as follows:

1. **Indicator functions of each cell (the tabular model):**

   $$f_k(i', j') = I_{\{i,j\}}(i', j')$$

   for each $(i, j) \in S$. For each action there are $M \times N$ features. In codes, it might be more convenient to index $f_k(i', j')$ by the row and column index.

[10] For example if the agent is at the left boundary, that is in state $(i, 1)$, and tries to move left, it remains in state $(i, 1)$ but if it tries, for example, to move right, a transition to $(i, 2)$ occurs.

2. **Indicator functions of each row and column:**

$$f_k(i',j') = \begin{cases} I_{\{i\}}(i',j') & \text{for } i = 1,\ldots,M \text{ and } k = 1,\ldots,M, \\ I_{\{j\}}(i',j') & \text{for } j = 1,\ldots,N \text{ and } k = N+1,\ldots,N+M. \end{cases}$$

   Note that in this case there are $M + N$ features for each action.

3. **A parametric function of the row and column number:**

   Motivated by calculations in the tabular case (described below) we hypothesized that a linear representation with an interaction term would well approximate the true value function.

$$f_0(i,j) := 1, \quad f_1(i,j) := j, \quad f_2(i,j) := j, \quad f_3(i,j) := ij.$$

   This means that for each action, $q(s,a;\boldsymbol{\beta})$ is approximated by a linear function of $(1,r,c,rc)$ with coefficients varying across actions.

   It is possible to also add higher order terms or even some judiciously chosen indicator functions. Note that when M and/or N are large, it may be judicious to scale the row and column index.

Results: Tabular model

Calculations for the tabular model apply Algorithm 2.6 to two instances with $(m^*, n^*) = (M, N)$, $R = 10$ and $c = 0.1$. In this application the robot seeks to learn a path to cell (M, N) from each the starting cell $(1, 1)$.

It uses ϵ-greedy action selection with $\epsilon_n = 10/(100 + n)$ and a learning rate of $\tau_n = 50/(1000+n)$ where n denotes the episode number. After completion of an episode the next episode again started from cell $(1, 1)$[11]. The experiments used a discount rate of $\lambda = 1$, 10,000 episodes and set all values $q((i, j), a) = 0$ for initialization.

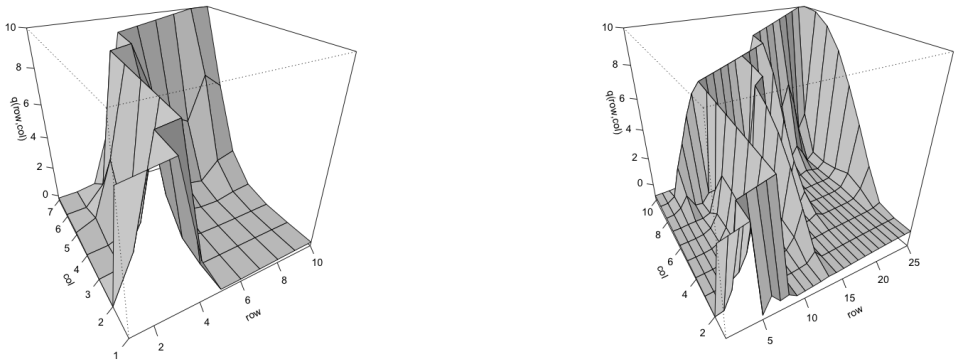
For small values of M and N, it was convenient to use the tabular representation of Q-learning to gain some insight into the form of the q-functions. For all replications of the experiment, using Q-learning in the tabular model found an optimal path through the grid.

Figure 2.7 shows the optimal q-functions from a single replicate for $(M, N) = (10, 7)$ and $(M, N) = (25, 10)$. It shows that in both cases, the algorithm identified a policy that moved close to the main diagonal of the grid. This suggested that the parametric model above with a cross-product term might identify optimal policies.

Results: Indicator functions of rows and columns

We replicated the above calculations using indicator functions of grid row and grid column using the same choice for ϵ_n and learning rate τ_n and the two grid sizes. To reliably find an optimal policy required discounting; ($\lambda = 0.95$) gave reliable results.

[11] If one sought a good policy for every starting state, episodes could be restarted at a random cell not equal to the target.



(a) 10×7 grid (b) 25×10 grid

Figure 2.7: Optimal q-values in grid-world model identified by Q-learning using a tabular representation.

Unlike the policies generated using the tabular representation, which tended to move down along the main diagonal, the greedy policies chose actions that moved the robot along the boundaries (row 1 and column N or column 1 and row M) of the grid. Moreover the algorithm found an optimal policy from all starting states even when the target was an interior point of the grid.

Results: Parametric representation

Developing a convergent implementation required considerable experimentation. Issues that arose were divergence of q-values or termination with non-optimal greedy policies. The challenge was that because rewards are only received when reaching the goal state, it required many iterations for the impact of the reward in the goal state to impact q-values in distant states. In light of these observations, successful implementations[12] used:

1. $\boldsymbol{\beta}$ initialized to be $\mathbf{0}$,
2. high initial exploration rates ($\epsilon_n = 20/(20+n)$),
3. an upper bound (1000) on the number of steps in an episode,
4. small learning rates ($\tau_n = 50/(1000+n)$),
5. a discount factor of 0.9, and
6. random starting states.

[12]The literature also suggests using experience replay to enhance convergence.

With these specifications, greedy-policies based on Q-learning always achieved the objective and were optimal when starting in state $(1, 1)$. Moreover they were similar to those found using the tabular representation, that is, they tended to step down through the interior of the region. They exhibited fast learning as shown in Figure 2.8.

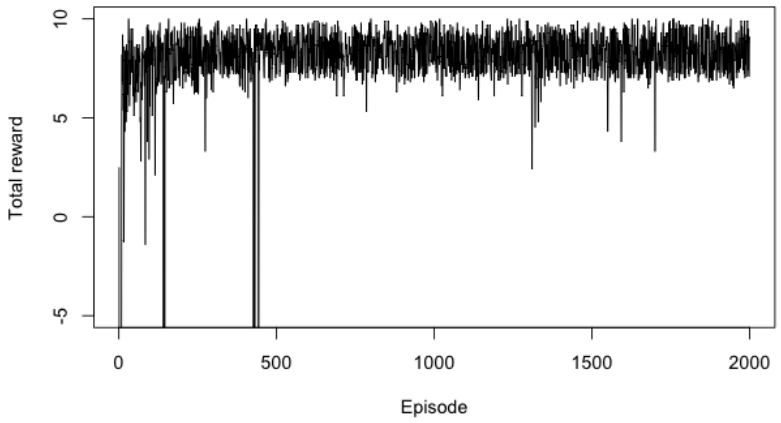


Figure 2.8: Reward per episode in a typical realization of Q-learning in a 10×25 grid. The eventual variability of the total reward per episode is due to the random starting state.

The beauty of using this parameterization was that it was not required to know the dimension of the grid before implementing Q-learning.

Summary

The above approximations used MN, $M + N$ and 4 features respectively for each action. While the first two approximations, which were based on indicator functions, reliably found optimal policies, those based on a low-dimensional parametric function required considerable experimentation to identify optimal policies.

We leave it to the reader to explore parametric approximations based on other functional forms or neural networks.

2.4 Value-based policy iteration

Policy iteration algorithms require methods for both evaluating and improving stationary policies. This presents challenges in large models which in which policies are represented implicitly through vectors of weights.

It is more expedient to focus on state-action value functions than value functions

to avoid determining an action in state s on the basis of

$$a' \in \arg\max_{a \in A_s} \left\{ \sum_{j \in S} p(j|s,a) \Big(r(s,a,j) + \lambda v(j; \hat{\boldsymbol{\beta}}) \Big) \right\}.$$

Evaluating such an expression would be prohibitive in a large model, even if both $p(j|s,a)$ and $r(s,a,j)$ were known.

We now describe a policy iteration type algorithm to achieve this.

2.4.1 Q-policy iteration

The following algorithm, which we refer to as *Q-policy iteration*[13] generalizes Q-learning with function approximation by running a policy that is close to a fixed implicitly-defined policy for several iterations before updating the weight vector. Essentially, it evaluates an ϵ-greedy variant [14] of the policy implicitly defined by the "current" weight vector to enable exploration. It may also be viewed as an extension of its tabular counterpart in Section **??** to an environment with state-action value function approximation.

Given a weight vector $\boldsymbol{\beta}$ and an $\epsilon \in (0,1)$, define the Markovian randomized decision rule d_0^ϵ by

$$w_{d_0^\epsilon}(a|s) = \begin{cases} 1-\epsilon & a = a_s \text{ where } a_s \in \arg\max_{a \in A_s} q(s,a;\boldsymbol{\beta}_0) \\ \frac{\epsilon}{|S|-1} & a \neq a_s. \end{cases} \tag{2.33}$$

We refer to the stationary randomized policy $(d^\epsilon)^\infty$ as the ϵ-greedy policy corresponding to $\boldsymbol{\beta}$. This policy can be implemented in a simulation by selecting action a_s with probability $1-\epsilon$ and sampling among other actions with probability ϵ.

The algorithm may be viewed as a simulation based version of modified[15] policy iteration (Algorithm **??**). We state the algorithm from the perspective of implicit policy specification in terms of a weight function and discuss why this should be regarded as a policy iteration type algorithm.

Algorithm 2.7. Q-policy iteration with linear state-action value function approximation in a discounted model

1. **Initialization:**

   (a) Initialize $\boldsymbol{\beta}_0$.

[13] This algorithm hasn't been widely discussed but seems to be a logical extension of policy iteration, see p.338 in Bertsekas and Tsitsiklis [1996] for a related discussion.

[14] Tabular models required the concept of ϵ-altered policies to account for the possibility that a decision rule being evaluated is not chosen greedily.

[15] Note that some authors, most notably, Bertseksas [2012] refer to this as *optimistic* policy iteration.

(b) Specify a learning rate sequence $\{\tau_n\}$ and a sequence $\{\epsilon_n\}$ for ϵ-altered action selection.

(c) Specify the number of evaluation iterations M_E and the number of evaluation loops N.

(d) Choose $s \in S$ and $n \leftarrow 1$.

2. While $n < N$.

3. **Policy Evaluation for fixed β_0:** While $m < M_E$;

(a) $m \leftarrow 1$ and $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta}_0$

(b) Generate $a \in A_s$ ϵ_n-greedily.

(c) Generate (s', r) or sample $s' \in S$ from $p(\cdot|s, a)$ and set $r = r(s, a, s')$.

(d) Set

$$q(s, a; \boldsymbol{\beta}_0) = \boldsymbol{\beta}_0^T \mathbf{b}(s, a) \tag{2.34}$$

(e) Generate $a' \in A_{s'}$ ϵ_n-greedily.

(f) Set

$$q(s', a'; \boldsymbol{\beta}_0) = \boldsymbol{\beta}_0^T \mathbf{b}(s', a') \tag{2.35}$$

(g) Update $\boldsymbol{\beta}$ according to

$$\boldsymbol{\beta}' \leftarrow \boldsymbol{\beta} + \tau_n \mathbf{b}(s, a)\delta \tag{2.36}$$

(h) $s \leftarrow s'$, $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta}'$ and $m \leftarrow m + 1$

4. **Weight updating:**

(a) $\boldsymbol{\beta}_0 \leftarrow \boldsymbol{\beta}$.

(b) $n \leftarrow n + 1$.

5. **Termination:** Output $\hat{\boldsymbol{\beta}} = \boldsymbol{\beta}$.

On the surface this does not look like a policy iteration algorithm since there is no clear improvement step. The algorithm as stated embeds improvement in the implicit generation of a new decision rule in steps 2(b) and 2(c) using fixed weights $\boldsymbol{\beta}_0$ updated after M_E evaluation iterations.

Suppose instead the algorithm starts with a decision rule d rather than a vector of weights $\boldsymbol{\beta}_0$. Then the policy evaluation step approximates an ϵ-greedy variant of that decision rule , d_ϵ, yielding a vector of weights $\boldsymbol{\beta}_{d_\epsilon}$. Then a full improvement step would choose

$$d'(s) \in \arg\max_{a \in A_s} \boldsymbol{\beta}_{d_\epsilon}^T \mathbf{b}(s, a)$$

for **all** $s \in S$ and then return to evaluating the ϵ-altered variant of this new decision

rule.

Some further comments follow:

1. When $M_E = 1$, this algorithm is equivalent to a SARSA variant of Q-learning. When $M_E > 1$ the evaluation step may be viewed as a TD(0) approximation to the state-action value function of the ϵ-greedy policy corresponding to $\boldsymbol{\beta}_0$.

2. No explicit stopping criterion is specified. One might terminate the algorithm when successive weight vectors differ by a small amount.

3. The algorithm is stated assuming long trajectory updates. Of course step 3(k) can update s in a different way to enhance coverage of the state space.

4. Softmax action selection can replace ϵ-greedy action selection in steps 3(b) and 3(f).

5. The reason ϵ-greedy policies are used in 3(c) and 3(f) are to obtain off-policy realizations of $q(s, a; \boldsymbol{\beta}_0)$.

6. The policy evaluation step can be replaced by Monte Carlo estimation of the ϵ-greedy policy.

7. Similarly to Q-learning, there is no guarantee of convergence of this algorithm.

The newsvendor model with approximate value functions

This section applies

1. Monte Carlo estimation

2. Q-learning

3. Q-policy iteration

to the newsvendor inventory model introduced in Section **??**. The reason why we consider this extremely simple model is that by removing the effect of state transitions we can distinguish the subtle differences between Q-learning and Q-policy iteration.

Since this model has a single state $s = 0$, the state-action value function $q(s, a)$ is a function **only** of the action a. We use a cubic polynomial approximation expressed in terms of scaled actions as follows

$$b_0(a) = 1, b_1(a) = \tilde{a},\ b_2(a) = \tilde{a}^2 \text{ and } b_3(a) = \tilde{a}^3$$

where $\tilde{a}$ represents a scaled by subtracting its mean and standard deviation[16] so that

$$q(a; \boldsymbol{\beta}) = \boldsymbol{\beta}^T \mathbf{b}(a).$$

[16]This means $\tilde{a} = (a - mean(a))/sd(a)$ where the mean and standard deviation are over $\{0, 1, \ldots, a_{max}\}$.

To implement Monte Carlo, for each element in a subset of $a \in A_0$, sample the demand N times to obtain $z^0, \ldots, z^N$ and set $\hat{q}(a)$ equal to the mean of $r(a, z^1), \ldots, r(a, z^N)$[17]. Then use least-squares to approximate $\hat{q}(a)$ by a linear function of its features to obtain $\hat{\boldsymbol{\beta}}_{MC}$.

The following algorithms apply Q-learning and Q-policy iteration apply directly to the newsvendor model.

Q-learning in the newsvendor model:

1. Initialize $\boldsymbol{\beta}$ and $n \leftarrow 1$.
2. Specify sequences $\{\epsilon_n : n = 1, 2, \ldots\}$ and $\{\tau_n : n = 1, 2, , \ldots)\}$.
3. Repeat N times:
   (a) Sample a' using ϵ-greedy (or softmax) sampling with respect to
   $$q(a; \boldsymbol{\beta}) = \boldsymbol{\beta}^T \mathbf{b}(a)$$
   (b) Generate demand z^n.
   (c) Update
   $$\boldsymbol{\beta}' \leftarrow \boldsymbol{\beta} + \tau_n(r(a', z^n) - \boldsymbol{\beta}^T \mathbf{b}(a))\mathbf{b}(a') \tag{2.37}$$
   (d) $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta}'$ and $n \leftarrow n + 1$.
4. Return $\hat{\boldsymbol{\beta}}_{QL} = \boldsymbol{\beta}$.

[17]Note the same realization of δ can be used for all a.

Q-policy iteration in the newsvendor model:

1. Initialize $\boldsymbol{\beta}_0$ and $n \leftarrow 1$.
2. Specify sequences $(\epsilon_n : n = 1, 2, \ldots)$ and $(\tau_n : n = 1, 2, , \ldots)$.
3. Repeat N times:
   (a) $m \leftarrow 1$
   (b) Repeat M times:
      i. Sample a' using ϵ-greedy (or softmax) sampling with respect to
      $$q(a) = \boldsymbol{\beta}_0^T \mathbf{b}(a)$$
      ii. Generate z^m.
      iii. Update
      $$\boldsymbol{\beta}' \leftarrow \boldsymbol{\beta} + \tau_m (r(a', z^n) - \boldsymbol{\beta}^T \mathbf{b}(a')) \mathbf{b}(a')$$
      iv. $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta}'$ and $m \leftarrow m + 1$.
   (c) $\boldsymbol{\beta}_0 \leftarrow \boldsymbol{\beta}$ and $n \leftarrow n + 1$.
4. Return $\hat{\boldsymbol{\beta}}_{QPI}$.

While these algorithms appear very similar, there are subtle but important differences.

1. In each of these algorithms the "max" in the Bellman update disappears because the Bellman equation reduces to $\max_{a \in A_0} E[r(Z, a)]$ so that the continuation cost is zero.
2. In Q-learning, action selection in step 2(a) is respect to a different $\boldsymbol{\beta}$ at each iteration, moreover this $\boldsymbol{\beta}$ is also updated in (2.37). In Q-policy iteration a fixed $\boldsymbol{\beta}_0$ is used for action generation in step 2(b)i for M evaluation steps and a different $\boldsymbol{\beta}$ (corresponding to the policy that is being evaluated) is used in the update equation. This is because the inner loop seeks to evaluate a fixed policy based on the ϵ-altered decision rule corresponding $\boldsymbol{\beta}_0$.
3. When $M = 1$ these algorithms are equivalent.

Example 2.6. This example considers the newsvendor problem with two discrete demand distributions (rounded and truncated normal with mean 50 and standard deviation 15 and a rounded gamma distribution with shape parameter 20 and scale parameter 4). It sets the item cost to 20, the salvage value to 5 and the price to be each of $21, 30$ and 120. The corresponding ratios of $\frac{G}{G+L}$ were $0.0625, 0.400, 0.870$ which represent a wide range of quantiles of the demand

distributions. The order quantities were $A_0 = \{0, 1, \ldots, 120\}$.

To obtain a baseline we generate 10,000 Monte Carlo replicates for each $a \in A_0$ and estimate the expected reward to be the average reward over the samples. Table 2.3 gives the optimal order quantity using the closed form representation (**??**), the maximum of the Monte Carlo estimates and the maximum obtained when fitting the Monte Carlo estimates with a cubic polynomial and a cubic spline with knots at 40 and 80.

Observe that order quantity that maximizes the Monte Carlo estimates well approximates the optimal order quantity. Observe also the maximum obtained from the spline approximation better approximates the Monte Carlo maximum than that based on a cubic approximation. We would expect this to be the case because the spline is a more flexible function with more parameters than a cubic polynomial.

We then applied Q-learning and Q-policy iteration using a cubic approximation based on scaled values of a. In each case the algorithm was initiated with $\boldsymbol{\beta} = \mathbf{0}$, used the STC learning rate $\tau_n = .05(1 + 10^{-5}n^2)^{-1}$ and a step-wise exploration rate ϵ_n that equaled 0.2 for $n <$ 10,000 , 0.15 for $n \geq$ 10,000. We used $N =$ 20,000 for Q-learning and $N = 10$ and $M =$ 2,000 for Q-policy iteration so that the total number of iterates in each case was the same. We ran 40 replicates of each algorithm for each combination of demand distribution and price using common random seeds.

Figure 2.9 shows the variability of estimates of the optimal order quantity over replicates obtained by choosing

$$a^* \in \arg\max_{a \in A_0} \hat{\boldsymbol{\beta}}_i^T \mathbf{b}(a)$$

for $i = QL, QPI$. Comparison with Table 2.3 shows that the Q-policy iteration better approximated the optima obtained using a cubic approximation to the Monte Carlo estimates then the Q-learning estimates in **all** cases.
This example may be viewed as a learning experiment since no information was used regarding the underlying demand distribution or reward structure.

Q-policy iteration in the queuing service rate control model

We now apply Q-policy iteration to the queuing service rate control model.

Example 2.7. This example applies Algorithm 2.7 to the queuing control model and compares results to those obtained applying Q-learning directly. Again it assumes $q(s, a)$ is approximated by distinct cubic functions of the state for each action.

Distribution	Price	Optimal	Monte Carlo	Cubic	Spline
Normal	21	27	26	23	29
	30	46	45	41	44
	120	67	65	76	69
Gamma	21	55	53	47	54
	30	74	76	76	75
	120	100	99	101	99

Table 2.3: Optimal order quantity in newsvendor model compared to three estimates obtained from 10,000 Monte Carlo replicates for each $a \in A_0$. The Monte Carlo estimate is is the maximum over the mean reward for each action while the Cubic and Spline estimates are the maximums of the least squares estimates using these approximations.

To obtain reasonable policy estimates required considerable experimentation with the number of evaluation iterations M_E and the learning rate and ϵ-greedy parameters. Table 2.4 summarizes results for $M_E = 50$, $N = 5{,}000$, $\tau_n = 5000/(50000 + n)$ and $\epsilon_n = 100/(400 + n)$ where the index for the parameters was the cumulative iteration number. Note that the total number of iterations was the same as in the experiment with Q-learning. States were sampled randomly at each iteration.

Table 2.4 summarizes results of the estimates and compares them to those obtained using Q-learning. Observe that the results using Q-policy iteration were quite similar to those obtained using Q-learning. Moreover the plots of order statistics (not shown) were similar to those in Figure 2.6.

Method	Quantity	min	25th pctl	median	75th pctl	max
Q-Policy iteration	Non-optimal actions	1	8	10.5	12.25	40
	RMSE	0.09	9.28	43.4	49.42	243.43
Q-learning	Non-optimal actions	5	8	12	15	24
	RMSE	3.86	12.07	21.39	33.48	224.70

Table 2.4: Summary statistics of 40 replicates of Q-policy iteration applied to the queuing control model.

Concluding remarks on examples

These numerical studies show that in the queuing control model, both Q-learning and Q-policy iteration produced similar results with Q-policy iteration slightly better at identifying a policy in agreement with the optimal policy. In the newsvendor model,

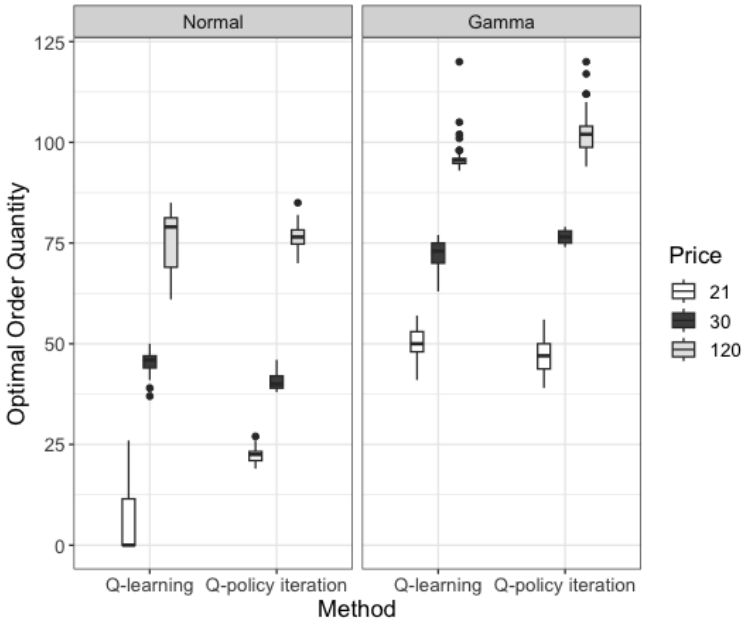


Figure 2.9: Boxplot comparing actions chosen by Q-learning and Q-policy iteration in the newsvendor model over 40 replicates for each demand distribution and price.

Q-policy iteration was better than Q-learning in identify a policy that agreed with that found using a cubic approximation to Monte Carlo estimates.

While these results were inconclusive, we believe that both Q-learning and Q-policy iteration provide the analyst with tools for analyzing models that use function approximation.

2.5 Policy space methods

This section describes a conceptually different approach to function approximation based on parameterizing the probability that a randomized stationary policy selects an action in a given state. Instead of directly improving a value function or state-action value function, this approach seeks to find a choice of parameter values that improves the policy directly.

It is advantage is that unlike state-action value focused methods (such as Q-learning) which may jump between deterministic policies, it allows for gradual changes in action selection probability leading to greater stability in values. This is especially attractive when controlling physical systems in real-time.

Two concepts underlie these methods:

1. *Expressing the probability that a stationary randomized policy d^∞ chooses action a in state s as a parametric function of both the state and action.* This is somewhat similar to the approach used to parameterize a state-action value function $q(s, a)$

where s and a are inputs and the value is an output, but when approximating policies directly, the state s is the input and the probability of choosing action a is an output.

2. *Updating parameter values using (stochastic) gradient ascent.* This approach seeks to iteratively find a zero of the gradient of a value function[18] by stochastic approximation where estimates are obtained by sampling from a distribution with a specified expectation.

2.5.1 Motivation: A policy gradient algorithm for a simple one-period model

We first analyze a one period, single-state multi-action model such as the newsvendor problem (Example 2.6). Since this model doesn't include state-transitions, it is similar to a classical stochastic optimization model. We retain the same notation for approximation as used when approximating state-action value functions although as noted above, these are two conceptually different approaches. That is, we represent features by vectors $\mathbf{b}(s,a)$ and parameters by compatible vectors[19] $\boldsymbol{\beta}$. In the single-state model, we ignore the dependence on s and write $\mathbf{b}(a)$ for the vector of features corresponding to action a.

We represent policies in terms of the softmax[20]function. For single-state models:

$$w(a|\boldsymbol{\beta}) = \frac{e^{\boldsymbol{\beta}^T\mathbf{b}(a)}}{\sum_{a'\in A_s} e^{\boldsymbol{\beta}^T\mathbf{b}(a')}} \tag{2.38}$$

where $\boldsymbol{\beta}$ is a K-dimensional column vector of parameters and $\mathbf{b}(a)$ is a K-dimensional column vector of features associated with action a. In multi-state models (Markov decision processes),

$$w(a|s;\boldsymbol{\beta}) = \frac{e^{\boldsymbol{\beta}^T\mathbf{b}(s,a)}}{\sum_{a'\in A_s} e^{\boldsymbol{\beta}^T\mathbf{b}(s,a')}}. \tag{2.39}$$

The objective in the single state ($S = \{0\}$) model can be represented by

$$v^* := \max_{\boldsymbol{\beta}\in\Re^K} v(\boldsymbol{\beta}) = \max_{\boldsymbol{\beta}\in\Re^K} \sum_{a\in A_0} w(a|\boldsymbol{\beta})r(a) = \max_{\boldsymbol{\beta}\in\Re^K} \sum_{a\in A_0} \frac{e^{\boldsymbol{\beta}^T\mathbf{b}(a)}}{\sum_{a'\in A_s} e^{\boldsymbol{\beta}^T\mathbf{b}(a')}} r(a) \tag{2.40}$$

where $r(a)$ denotes the (expected) reward for choosing action a, $w(a|\boldsymbol{\beta}) := w(a|0;\boldsymbol{\beta})$ and

$$v(\boldsymbol{\beta}) := \sum_{a\in A_0} w(a|\boldsymbol{\beta})r(a) := E_{\boldsymbol{\beta}}[r(Y)]$$

[18]Of course, the value function varies over states. To obtain a single value, one can use a weighted average over states.

[19]Many authors represent randomized decision rules by $\pi_{\boldsymbol{\phi}}(a|s)$ where $\boldsymbol{\phi}$ denotes a vector of parameters.

[20]This is also referred to as a *logistic* representation.

where Y denotes the random action selected by $w(\cdot|\boldsymbol{\beta})$.

We seek to maximize this expression by solving:

$$\nabla_{\boldsymbol{\beta}} v(\boldsymbol{\beta}) = \mathbf{0}. \tag{2.41}$$

In the newsvendor model, $r(a) = E[r(a,\delta)] = \sum_{k=0}^{\Delta} r(k)f(k)$ where $f(k) = P[\delta = k]$. In this simple model we can compute $v(\boldsymbol{\beta})$ analytically, numerically or through simulation. To simulate this model requires sampling from (both) $w(\cdot|\boldsymbol{\beta})$ and the demand distribution $f(\cdot)$.

The gradient of $v(\boldsymbol{\beta})$ in the newsvendor model.

We now develop the machinery to apply the approach in Figure **??**. To do so, we derive a representation for the expected gradient of $v(\boldsymbol{\beta})$.

The gradient of $v(\boldsymbol{\beta})$ can be written as

$$\begin{aligned}\nabla_{\boldsymbol{\beta}} v(\boldsymbol{\beta}) &= \sum_{a\in A_0} \nabla_{\boldsymbol{\beta}} w(a|\boldsymbol{\beta}) r(a) \\ &= \sum_{a\in A_0} w(a|\boldsymbol{\beta}) \nabla_{\boldsymbol{\beta}} \ln(w(a|\boldsymbol{\beta})) r(a) = E_{\boldsymbol{\beta}}[\nabla_{\boldsymbol{\beta}} \ln[w(Y|\boldsymbol{\beta})] r(Y)].\end{aligned} \tag{2.42}$$

where the second equality follows from the generalizing the easily demonstrated calculus identity

$$\frac{dg(x)}{dx} = g(x)\frac{d\ln(g(x))}{dx}. \tag{2.43}$$

The benefits of establishing this equivalence are that:

1. when using the softmax to represent $w(a|\boldsymbol{\beta})$, the gradient of $\ln(w(a|\boldsymbol{\beta})$ has a nice closed form representation, and
2. in multi-period models, expectations are based on products of probabilities so that logarithms transform a product to an easier to work with sum [21].

We leave it as an exercise that when $w(a|\boldsymbol{\beta})$ is defined by (2.38),

$$\nabla_{\boldsymbol{\beta}} \ln(w(a|\boldsymbol{\beta})) = \mathbf{b}(a) - \sum_{a'\in A_0} w(a'|\boldsymbol{\beta})\mathbf{b}(a') \tag{2.44}$$

where as above, $\mathbf{b}(a)$ is a vector of feature values. Thus we have the following closed-form expression for the gradient of $v(\boldsymbol{\beta})$ in the newsvendor model;

$$\nabla_{\boldsymbol{\beta}} v(\boldsymbol{\beta}) = \sum_{a\in A_0} w(a|\boldsymbol{\beta}) \left[\mathbf{b}(a) - \sum_{a'\in A_0} w(a'|\boldsymbol{\beta})\mathbf{b}(a')\right] r(a) = E_{\boldsymbol{\beta}}[(\mathbf{b}(Y) - C)r(Y)] \tag{2.45}$$

where $C = \sum_{a'\in A_0} w(a'|\boldsymbol{\beta})\mathbf{b}(a')$.

[21] Note that in statistical maximum likelihood estimation one bases results on maximizing log-likelihoods instead of likelihoods.

Applying stochastic approximation

This section applies stochastic gradient ascent[22] to solve $\nabla_{\boldsymbol{\beta}} v(\boldsymbol{\beta}) = \mathbf{0}$ by sampling from a distribution with expected value $E_{\boldsymbol{\beta}}[(\mathbf{b}(Y) - C)r(Y)]$. Action a is sampled from $w(\cdot|\boldsymbol{\beta})$, a sample from the distribution $f(\cdot)$ is used to estimate $r(a)$ and the gradient is estimated by $\nabla_{\boldsymbol{\beta}} ln[w(a|\boldsymbol{\beta})]r(a)$.

The stochastic approximation recursion becomes:

$$\boldsymbol{\beta}' \leftarrow \boldsymbol{\beta} + \tau \nabla_{\boldsymbol{\beta}} \ln[w(a|\boldsymbol{\beta})]r(a) \tag{2.46}$$

where the gradient is computed using (2.45) (or by numerical differentiation). This observation leads to the following policy gradient algorithm for a one-period model (assuming that simulation of both actions and rewards):

Algorithm 2.8. Policy gradient algorithm for a single-state, one-period model:

1. Specify $\boldsymbol{\beta}$, $\{\tau_n : n = 1, 2, \ldots\}$ and N.
2. $n \leftarrow 1$
3. While $n < N$:
   (a) Sample action a from $w(a|\boldsymbol{\beta})$.
   (b) Sample reward r from $f(\cdot)$.
   (c) **Estimate gradient:**
   $$\nabla_{\boldsymbol{\beta}} v(\boldsymbol{\beta}) = \Big(\mathbf{b}(a) - \sum_{a' \in A_0} w(a'|\boldsymbol{\beta})\mathbf{b}(a')\Big)r \tag{2.47}$$
   (d) **Update:**
   $$\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} + \tau_n \nabla_{\boldsymbol{\beta}} v(\boldsymbol{\beta}) \tag{2.48}$$
   (e) $n \leftarrow n + 1$
4. Return $w(a|\boldsymbol{\beta})$.

Some comments follow:

1. As stated the algorithm samples both the action and the reward given the action. In some examples the reward may be a deterministic function of the action so it can be computed directly once the action is known.

[22] We use the expression "ascent" since we are maximizing.

2. In a model free environment, instead of sampling the reward, it can be observed. However one would still sample the action from $w(\cdot|\boldsymbol{\beta})$.

3. The policy gradient algorithm converges to a global maximum under mild conditions on $r(a)$.

4. The algorithm terminates with a stationary randomized policy based on $w(a|\boldsymbol{\beta})$.

5. Alternatively to stopping after a specified number of iterations the algorithm can terminate when the change in $\boldsymbol{\beta}$ achieves a pre-specified tolerance.

An example

We now apply this algorithm to the newsvendor model. We take as features, indicator variables of actions:

$$b_i(a) = \begin{cases} 1 & a = a_i \\ 0 & a \neq a_i \end{cases}$$

for $i = \{0, 1, \ldots, a_{max}\}$ resulting in $a_{max} + 1$ features. Note this choice of features is equivalent to using a tabular representation. (We leave it to the reader to investigate other feature choices.)

The advantage of this choice of features is that it provides very clear insight in to the workings of gradient ascent as the following calculations show. For concreteness, assume that $a \in \{0, 1, 2, 3\}$ and $\boldsymbol{\beta} = [0, 0, 0, 0]^T$ so that $w(a|\boldsymbol{\beta}) = 0.25$ for $a \in \{0, 1, 2, 3\}$.

Suppose sampling generates action $a = 1$ and $r(a) = 17$ so that applying (2.47) shows that

$$\nabla_{\boldsymbol{\beta}} v(\boldsymbol{\beta}) = 17 \left(\begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} - \begin{bmatrix} 0.25 \\ 0.25 \\ 0.25 \\ 0.25 \end{bmatrix} \right) = 17 \begin{bmatrix} -0.25 \\ 0.75 \\ -0.25 \\ -0.25 \end{bmatrix}.$$

Observe that the component of the gradient corresponding to $a = 1$ is positive and all other components are negative. Thus as a result of applying gradient ascent in step 2(b) of Algorithm 2.8 the component of $\boldsymbol{\beta}$ corresponding to $a = 1$, $\phi_1' > \phi_1$ and all other components of $\boldsymbol{\beta}'$ will be smaller than the corresponding component of $\boldsymbol{\beta}$. Hence the policy $w(a|\boldsymbol{\beta}')$ will select $a = 1$ with greater probability and all other actions with smaller probability than $w(a|\boldsymbol{\beta})$. If on the other hand, $r(a) < 0$, the reverse would occur, namely the probability of selecting $a = 1$ would decrease and all other probabilities would increase.

We now apply this algorithm to some numerical instances.

Example 2.8. Policy gradient in the newsvendor model.

We set $a_{max} = 20$, cost equal to 20, salvage value equal to 5 and vary the price between $21, 30, 120$. We consider three demand distributions: truncated

discrete normal with mean 10 and standard deviation 3, binomial with $N = 10$ and $p = 0.6$ and discrete uniform on $\{0, \ldots, a_{max}\}$. We apply Algorithm 2.8 with N =10,000 and $\tau_n = 0.0001$ to each of 40 replicates for all nine combinations of price and demand distribution.

We observed that in **all** replicates $w(a|\boldsymbol{\beta})$ evaluated at the estimated $\boldsymbol{\beta}$ was unimodal with a single probability extremely close to 1 and all others near 0. This means that the algorithm converged to a deterministic policy.

Table 2.5 summarizes results. The column "Mean" denotes the average of $\arg\max_{a \in A_0} w(a|\boldsymbol{\beta})$ and the column "S.D." denotes the standard deviation of this quantity over 40 replicates. Observe that in all cases, the average over replicates well approximated the optimal value computed using (**??**). Estimates were most variable when price equalled 120 corresponding to the 87th percentile of the demand distribution.

Distribution	Price	Optimal	Mean	S. D.
Normal	21	5.40	5.55	0.64
	30	9.23	9.50	0.72
	120	13.37	14.43	2.05
Binomial	21	4	3.73	0.45
	30	6	5.58	0.50
	120	8	7.93	1.16
Uniform	21	1.63	1.15	0.36
	30	10.40	10.75	1.69
	120	22.61	21.93	2.10

Table 2.5: Mean and standard deviation over 40 replicates of order quantity corresponding to maximum estimated probability obtained using Algorithm 2.8 . Note the optimal order quantity for the normal and uniform distributions is based on continuous representations of these distributions.

2.5.2 A policy gradient algorithm for a Markov decision process

We now use the intuition and results in the previous section to develop a policy gradient algorithm for an episodic Markov decision process.

Recall that the goal in an episodic model is to maximize the expected total reward prior to reaching a set of zero-reward absorbing states Δ. In such models one seeks to find a policy π that maximizes

$$v^{\pi}(s) = E^{\pi}\left\{ \sum_{n=1}^{N_{\Delta}} r(X_n, Y_n, X_{n+1}) \middle| X_1 = s \right\}$$

over the set of all history dependent randomized policies where N_Δ denotes the random time the system enters Δ. Recall (see Section 1.2.3) that the expectation is respect to the probability distribution of sequences of states and actions generated by the stochastic process corresponding to π. This probability distribution combines both Markov decision process transition probabilities and action choice probabilities corresponding to randomized decision rules.

To compute a gradient requires a single objective function as opposed to one for each $s \in S$. This is easily accomplished by adding an initial state distribution $\rho(s)$ so that the value of policy π becomes scalar-valued and represented by

$$v^\pi = \sum_{s \in S} \rho(s) v^\pi(s). \tag{2.49}$$

As a result of Theorem **??** under mild conditions there is a stationary deterministic policy that maximizes (2.49). But instead of focusing on deterministic policies the methods herein operate within the larger family of randomized stationary policies. Recall that that a randomized stationary policy d^∞ chooses actions according to distribution $w_d(a|s)$.

As done previously in this chapter, let $\mathbf{b}(s,a)$ denote a K-dimensional state and action-dependent feature vector with corresponding K-dimensional weight vector $\boldsymbol{\beta}$ and let $w(a|s,\boldsymbol{\beta})$ denote the corresponding parameterized action-choice probability distribution. and let d_β represents the decision rule that chooses actions according $w(a|s,\boldsymbol{\beta})$. That is $w_{d_{\boldsymbol{\beta}}}(a|s) := w(a|s,\boldsymbol{\beta})$.

Hence the (scalar) objective becomes that of finding a weight vector $\boldsymbol{\beta} \in \Re^K$ that maximizes

$$v(\boldsymbol{\beta}) := \sum_{s \in S} \rho(s) v^{(d_{\boldsymbol{\beta}})^\infty}(s) \tag{2.50}$$

or expressed differently, we seek

$$\hat{\boldsymbol{\beta}} \in \arg\max_{\boldsymbol{\beta} \in \Re^K} v(\boldsymbol{\beta}). \tag{2.51}$$

2.5.3 The gradient of the value function for an undiscounted infinite horizon Markov decision process

Chapter **??** provides details regarding the evaluation of an expression similar to $v(\boldsymbol{\beta})$ in terms of transition probabilities, action choice probabilities and rewards. It shows that

$$\begin{aligned} v(\boldsymbol{\beta}) = \sum_{s^1 \in S} \sum_{a^1 \in A_{s^1}} \sum_{s^2 \in S} \rho(s^1) w(a^1|s^1,\boldsymbol{\beta}) p(s^2|s^1,a^1) \Big(r(s^1,a^1,s^2) \\ + \sum_{a^2 \in A_{s^2}} \sum_{s^3 \in S} w(a^2|s^2,\boldsymbol{\beta}) p(s^3|s^2,a^2) \big(r(s^2,a^2,s^3) + \dots \big) \Big) \end{aligned} \tag{2.52}$$

where eventually the rewards are zero after reaching an absorbing state.

Now consider the expected reward in the first period only

$$v_1(\boldsymbol{\beta}) := \sum_{s^1 \in S} \sum_{a^1 \in A_{s^1}} \sum_{s^2 \in S} \rho(s^1) w(a^1|s^1, \boldsymbol{\beta}) p(s^2|s^1, a^1) r(s^1, a^1, s^2).$$

Then

$$\nabla_\beta v_1(\boldsymbol{\beta}) = \sum_{s^1 \in S} \sum_{a^1 \in A_{s^1}} \sum_{s^2 \in S} \nabla_\beta \rho(s^1) w(a^1|s^1, \boldsymbol{\beta}) p(s^2|s^1, a^1) r(s^1, a^1, s^2).$$

Observe that this gradient involves both the initial distribution and the transition probabilities and is not amenable (especially higher order terms) to direct computation or simulation. Instead using the "trick"

$$\frac{du(x)}{dx} = u(x) \frac{d \ln(u)}{dx} \tag{2.53}$$

yields

$$\begin{aligned} \nabla_\beta v_1(\boldsymbol{\beta}) = \sum_{s^1 \in S} \sum_{a^1 \in A_{s^1}} \sum_{s^2 \in S} & \rho(s^1) w(a^1|s^1, \boldsymbol{\beta}) p(s^2|s^1, a^1) r(s^1, a^1, s^2) \\ & \times \nabla_\beta \ln \big(\rho(s^1) w(a^1|s^1, \boldsymbol{\beta}) p(s^2|s^1, a^1) r(s^1, a^1, s^2) \big) \big). \end{aligned} \tag{2.54}$$

Since $\rho(s)$ and $p(s^2|s^1, a^1)$ don't involve $\boldsymbol{\beta}$,

$$\begin{aligned} & \nabla_\beta \ln \big(\rho(s^1) w(a^1|s^1, \boldsymbol{\beta}) p(s^2|s^1, a^1) r(s^1, a^1, s^2) \big) \\ & \qquad = \nabla_\beta \ln \big(\rho(s) \big) + \nabla_\beta \ln \big(w(a^1|s^1, \boldsymbol{\beta}) \big) + \nabla_\beta \ln \big(p(s^2|s^1, a^1) \big) \\ & \qquad = \nabla_\beta \ln \big(w(a^1|s^1, \boldsymbol{\beta}) \big). \end{aligned}$$

Therefore it follows from (2.54) that

$$\nabla_\beta v_1(\boldsymbol{\beta}) = \sum_{s^1 \in S} \sum_{a^1 \in A_{s^1}} \sum_{s^2 \in S} \rho(s^1) w(a^1|s^1, \boldsymbol{\beta}) p(s^2|s^1, a^1) \nabla_\beta \Big(\ln \big(w(a^1|s^1, \boldsymbol{\beta}) \big) r(s^1, a^1, s^2) \Big). \tag{2.55}$$

The significance of this expression is that it shows that $\nabla_\beta v_1(\boldsymbol{\beta})$ can be evaluated by averaging $\ln \big(w(a^1|s^1, \boldsymbol{\beta}) \big) r(s^1, a^1, s^2)$ over replicates of (s^1, a^1, s^2). **(Add exercise that does this.)**

The following result, which is proved in the appendix to this chapter, generalizes the above argument. It provides the basis for estimating the gradient of the value function using a sampled trajectory.

Theorem 2.1. Suppose N_Δ is finite with probability 1 and that $w(a|s; \boldsymbol{\beta})$ is differentiable with respect to each component of $\boldsymbol{\beta}$. Then

$$\nabla_{\boldsymbol{\beta}} v(\boldsymbol{\beta}) = E^{(d_{\boldsymbol{\beta}})^\infty} \left[\sum_{j=1}^{N_\Delta} \nabla_{\boldsymbol{\beta}} \ln \big(w(Y_j|X_j; \boldsymbol{\beta}) \big) \left(\sum_{i=j}^{N_\Delta} r(X_i, Y_i, X_{i+1}) \right) \right]. \tag{2.56}$$

Observe that the multiplier of the the gradient $\nabla_{\boldsymbol{\beta}} \ln\big(w(Y_j|X_j;\boldsymbol{\beta})$ only involves rewards after decision epoch j.

The quantity $\nabla_{\boldsymbol{\beta}} \ln(w(Y_j|X_j;\boldsymbol{\beta}))$ is often referred to as the *score function* and is a fundamental quantity in maximum likelihood estimation. In fact, if $q(X_j, Y_j)$ was replaced by the constant one, the above expression would correspond to the gradient used when applying gradient ascent to obtain the maximum likelihood estimator of the parameters of the decision rule.

2.5.4 A policy gradient algorithm

The following algorithm, referred to in the literature as REINFORCE[23], describes an early implementation of gradient ascent over the space of parameterized randomized policies. We frequently refer to this algorithm as "the policy gradient algorithm".

Algorithm 2.9. REINFORCE with baseline.

1. Initialize $\boldsymbol{\beta}$ and specify a learning rate sequence $\{\tau_n : n = 1, 2, \ldots\}$, a baseline $b(s)$ and the number of replicates N'.
2. Repeat N' times:
   (a) Generate an episode and save a sequence of states, actions and rewards $(s^1, a^1, s^2, r^1, \ldots, s^N, a^N, s^{N+1}, r^N)$ in which N denotes the termination time of the episode.
   (b) $N \leftarrow 1$
   (c) While $n \leq N$
      i. Update
      $$\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} + \tau_n \nabla_{\boldsymbol{\beta}} \ln\big(w(a^n|s^n;\boldsymbol{\beta})\big) \left(\sum_{k=n}^{N} r^k - b(s^n)\right) \tag{2.57}$$
      ii. $n \leftarrow n + 1$.
3. Return $w(a|s, \boldsymbol{\beta})$.

Observe that:

1. Note that many implementations of REINFORCE are stated with $b(s) = 0$. Including it does not invalidate the gradient derivation in Theorem 2.1 since the baseline does not change the expectation in (2.56). Moreover choosing the baseline appropriately enhances convergence by reducing the variance of the estimates.

[23]REINFORCE is an acronym for the awkward expression "REward INcrement = Non-negative Factor × Offset Reinforcement × Characteristic Eligibility".

2. Representation (2.57) provides insight into the qualitative behavior of the policy gradient algorithm. When $\left(\sum_{k=n}^{N} r^k - b(s^n)\right)$ is positive, components of $\boldsymbol{\beta}$ that make action a^n more likely in state s^n will be increased.

   As a concrete example, consider the case in which the features are indicator functions of state-action pair and $w(a|s,\boldsymbol{\beta})$ is the softmax function. Then

   $$\frac{\partial}{\partial \beta_{s,a}} \ln\left(w(a^n|s^n,\boldsymbol{\beta})\right) = \begin{cases} 1 - w(a|s;\boldsymbol{\beta}) & \text{for } a = a^n, s = s^n \\ -w(a|s;\boldsymbol{\beta}) & \text{otherwise.} \end{cases}$$

   so that the only positive component of $\nabla_\beta \ln\left(w(a|s;\boldsymbol{\beta})\right)$ in state s^n is a^n. Hence when $\left(\sum_{k=n}^{N} r^k - b(s^n)\right)$ is positive, β_{s^n,a^n} will increase and all other components will decrease resulting in an increased probability of choosing action a^n in state s^n at future iterations.

3. The algorithm requires an entire episode before updating the parameter estimates. Once the data is generated, REINFORCE updates the parameters stepwise.

4. The update in step 2(b) can be also be implemented in a single step by accumulating the terms of the gradient as in (2.56) prior to an update or even averaging the gradient over several trajectories.

5. The expression $\nabla_{\boldsymbol{\beta}} \ln\left(w(a_n|s_n;\boldsymbol{\beta})\right)$ can be evaluated numerically, in closed form when $w(a|s,\boldsymbol{\beta})$ is a softmax function of features, or by back-propagation when $w(a|s;\boldsymbol{\beta})$ is a neural net.

6. Step 2c. describes a direct application of gradient ascent. Enhanced versions are available[24].

7. The above algorithm is on policy, that is the evaluates the same policy that was used to generate the sequence of states, actions and rewards. An off-policy variant uses *importance sampling* to a adjust iterates appropriately.

2.5.5 An example

We consider the problem of optimal stopping in a reflecting random walk (Example **??**). In it, $S = \{-M, -(M-1), \ldots, M-1, M\}$, $A_s = \{a_0, a_1\}$ with a_0 corresponding to stopping and a_1 corresponding to continuing. Stopping in state s yield a reward of $g(s)$ while continuing costs c and results in a transition to state s' according to

$$p(s'|s,a_1) = \begin{cases} p & \text{if } s' = s+1, s < N \quad \text{or } s' = s = M \\ 1-p & \text{if } s' = s-1, s > 1 \quad \text{or } s' = s = -M \\ 0 & \text{otherwise} \end{cases}$$

[24]The smoothed gradient descent algorithm ADAM is described Kingma and Ba [2017].

for $0 < p < 1$. **(check this is consistent with optimal stopping formulation earlier)**. Let Δ denote the stopped state.

This is a stochastic shortest[25] path model with the reward of the improper policy that continues in every state equal to $-\infty$. Under all other policies the system terminates in finite expected time so we can analyze it as an episodic model.

Illustrative calculations

Before providing computational results, we illustrate the update in step 3 of Algorithm 2.9 on a single hypothetical episode. Suppose an episode generates two continuation actions followed by stopping at the third step. For concreteness, set the observed sequence to $5 \to 6 \to 7 \to \Delta$ and suppose $c = -2$ and $g(s) = s^2$. Then the output of the episode would be the realization[26] when a transition occurs to the stopped state.

$$(5, a_1, -2, 6, a_1, -2, 7, a_0, 49, \Delta, 0).$$

The following steps illustrate the calculation in (2.57) with $b = 0$. Ignoring "fake" transition in Δ, the quantity $\sigma_n := \left(\sum_{k=n}^{N} r^k - b\right)$ takes on the values

$$\sigma_3 = 49, \quad \sigma_2 = 47, \quad \sigma_1 = 45.$$

Now assume the features are linear in the state for each action so that

$$b(s) = \beta_{0,0} I_{\{a=a_0\}} + \beta_{0,1} I_{\{a=a_1\}} + \beta_{1,0} s I_{\{a=a_0\}} + \beta_{1,1} s I_{\{a=a_1\}}$$

and the softmax function $w(a|s, \boldsymbol{\beta})$ becomes

$$w(a_0|s, \boldsymbol{\beta}) = \frac{e^{\beta_{0,0}+\beta_{1,0}s}}{e^{\beta_{0,0}+\beta_{1,0}s} + e^{\beta_{0,1}+\beta_{1,1}s}}$$

with $w(a_1|s, \boldsymbol{\beta}) = 1 - w(a_0|s, \boldsymbol{\beta})$. Some simple calculus establishes that the gradients are given by

$$\nabla_{\boldsymbol{\beta}} ln\big(w(a_0|s, \boldsymbol{\beta})\big) = \begin{bmatrix} 1 - w(a_0|s, \boldsymbol{\beta}) \\ -w(a_1|s, \boldsymbol{\beta}) \\ s(1 - w(a_0|s, \boldsymbol{\beta})) \\ -sw(a_1|s, \boldsymbol{\beta}) \end{bmatrix} = w(a_1|s, \boldsymbol{\beta}) \begin{bmatrix} 1 \\ -1 \\ s \\ -s \end{bmatrix}$$

and

$$\nabla_{\boldsymbol{\beta}} \ln\big(w(a_1|s, \boldsymbol{\beta})\big) = w(a_0|s, \boldsymbol{\beta}) \begin{bmatrix} -1 \\ 1 \\ -s \\ s \end{bmatrix}$$

[25] Since we are maximizing rewards, this is actually a *longest* path model.

[26] Assuming the reward is received immediately after the stopped action is chosen.

where $\boldsymbol{\beta} = (\beta_{0,0}, \beta_{0,1}, \beta_{1,0}, \beta_{1,1})^T$. Note that in examples such as this it is more convenient to represent the components of $\boldsymbol{\beta}$ in matrix form with columns corresponding to actions so that gradient can be represented in terms of a Jacobian matrix.

Thus when $a = a_0$, (2.57) becomes

$$\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} + \tau_n \left(\sum_{k=n}^{N} r^k \right) w(a_1|s, \boldsymbol{\beta}) \begin{bmatrix} 1 \\ -1 \\ s \\ -s \end{bmatrix}. \tag{2.58}$$

Observe that the only stochastic element in this expression is the term $\left(\sum_{k=n}^{N} r^k\right)$ so that the variability of this quantity effects the stability of the estimates of $\boldsymbol{\beta}$. Standardizing rewards can reduce this variability.

Suppose that $\boldsymbol{\beta} = (0, 0, 0, 0)^T$ and $\tau_n = 0.1$, then the iterates become

$$\boldsymbol{\beta} = 0.1 \cdot 49 \cdot 0.5 \begin{bmatrix} 1 \\ -1 \\ 7 \\ -7 \end{bmatrix} = \begin{bmatrix} 2.45 \\ -2.45 \\ 17.15 \\ -17.15 \end{bmatrix}, \ \boldsymbol{\beta} = \begin{bmatrix} -2.35 \\ 2.35 \\ -14.10 \\ 14.10 \end{bmatrix}, \text{and } \boldsymbol{\beta} = \begin{bmatrix} -2.35 \\ 2.35 \\ -14.10 \\ 14.10 \end{bmatrix}.$$

Note that corresponding to the last $\boldsymbol{\beta}$ above, the action choice probabilities in state 5 are $w(a_0|5, \boldsymbol{\beta}) = 1$ and $w(a_1|5, \boldsymbol{\beta}) = 0$.

Alternatively setting $b = 47$, which equals the mean of the σ_i, we obtain the sequence

$$\boldsymbol{\beta} = \begin{bmatrix} 0.30 \\ -0.30 \\ 1.70 \\ -1.60 \end{bmatrix}, \ \boldsymbol{\beta} = \begin{bmatrix} 0.30 \\ -0.30 \\ 1.70 \\ -1.60 \end{bmatrix}, \text{and } \boldsymbol{\beta} = \begin{bmatrix} 0.50 \\ -0.50 \\ 2.70 \\ -2.70 \end{bmatrix}.$$

Although the $\boldsymbol{\beta}$ estimates have changed and have become smaller in magnitude, the corresponding action selection probabilities in state 5 remain $w(a_0|5, \boldsymbol{\beta}) = 1$ and $w(a_1|5, \boldsymbol{\beta}) = 0$.

A numerical experiment

Now we provide a more detailed numerical study that implements Algorithm 2.9 in the context of an optimal stopping problem on $S = \{-50, \ldots, 50\}$ and $g(s) = -s^2$. The objective is maximize the expected total (undiscounted) reward averaged over the initial state distribution. With this choice of $g(s)$ the goal is stop as close to zero as possible.

Figure 2.10 symbolically represents the *optimal* value functions and stopping region for three choices of the continuation cost. The optimal value function was found to a high degree accuracy using value iteration with $v^0(s) = -50^2$. This starting value, which was a lower bound on the value function, was chosen to ensure monotone convergence of value iteration (see Chapter **??**).

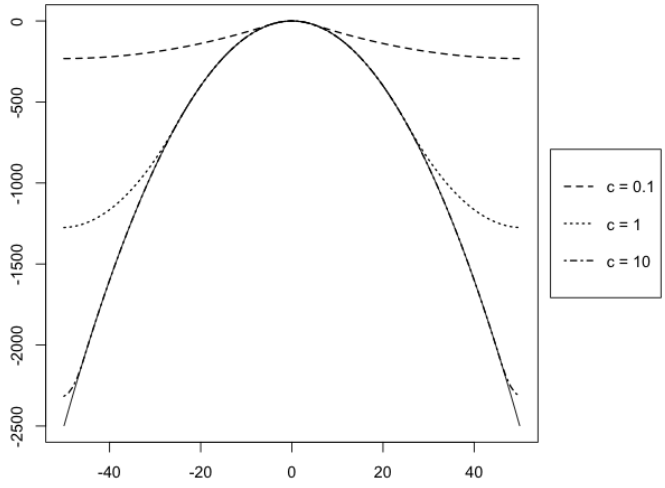


Figure 2.10: Optimal value functions (dashed lines) for three choices of c for the problem of optimally stopping in a random walk with $p = 0.5$ The solid line shows the reward of stopping in each state. The optimal policy is to stop when the optimal value function and the reward from stopping agree. Observe that the stopping region increases with respect to the continuation cost.

To apply the policy gradient algorithm above, first assume a tabular representation so that features have the form

$$b(s, a) = I_{\{s',a'\}}(s, a), \text{ for } s' \in S \text{ and } a' = a_0, a_1$$

so that

$$w(a'|s', \boldsymbol{\beta}) = \frac{e^{\eta\beta_{s',a'}}}{\sum_{s=-50}^{50}\sum_{j=0}^{1} e^{\eta\beta_{s,a_j}}} \tag{2.59}$$

where $\beta_{s',a'}$ is the coefficient corresponding to $b(s', a')$.

For this model, (assuming[27] that $\eta = 1$) the gradient is available in closed form as:

$$\frac{\partial}{\partial\beta_{s,a}} \ln w(a'|s', \boldsymbol{\beta}) = \begin{cases} \big(1 - w(a|s, \boldsymbol{\beta})\big) & s = s', a = a' \\ -w(a|s, \boldsymbol{\beta}) & \text{otherwise.} \end{cases} \tag{2.60}$$

Since $0 \leq w(a|s, \boldsymbol{\beta}) \leq 1$, this means that the only positive term in the gradient corresponds to the state-action pair that is being evaluated.

Algorithm 2.9 was implemented with the following specifications:

- Initial distribution uniform on S;

[27]When $\eta \neq 1$ it will appear in each of the the derivative terms however it can be ignored as it can be absorbed in the learning rate.

- $c = 0.1, 10, 50$;
- $p = 0.5$;
- $\boldsymbol{\beta}$ initialized to $\mathbf{0}$;
- N' =500,000;
- episodes were truncated after 100 steps;
- softmax exponent $\eta = 1/100$;
- learning rate, $\tau_n = \frac{1000}{10000+n}$ where n represented the episode number, and
- baseline $b(s) = -50^2$.

The baseline was chosen so as to avoid overflow in probability estimates. A common random seed was used across all values of c.

Figure 2.11 below shows estimates of $\boldsymbol{\beta}$ for the three choices of c for a single (typical) replicate and as well for a discounted model with $c = 0$ and the discount rate $\lambda = 0.95$. The lines correspond to 4th order polynomial fits to components of $\boldsymbol{\beta}$ corresponding to each action. The figures show that individual component values (dots) are quite noisy. Smoothing them with a fourth order polynomial in the state values for each action reveals some interesting and anticipated behavior.

To interpret these figures note that when solid line lies above the dashed line, it is more likely for the policy to stop[28]. Thus these figures show that the stopping region which is centered on $s = 0$ and increases with respect to c. When c is large, it's optimal to stop in all states and when $c = 0.1$ and in the discounted model the stopping region is narrower. Note that when $c = 0.1$, the policy found using the above algorithm differs considerably form the optimum policy indicate in Figure 2.10.

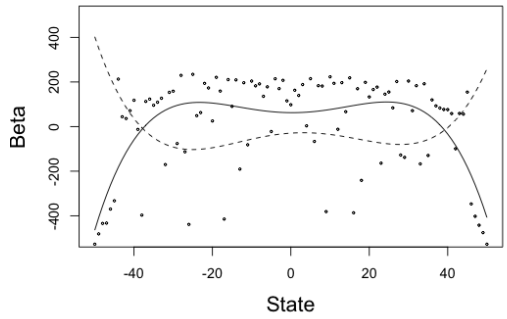
To conclude the analysis, this example fits a fourth-order polynomial using centered and standardized states as features for each action. That is

$$w(a_j|s, \boldsymbol{\beta}) = \frac{e^{\eta\left(\sum_{i=0}^{4} \beta_{i,j}\tilde{s}^i\right)}}{\sum_{k=0}^{1} e^{\eta\left(\sum_{i=0}^{4} \beta_{i,k}\tilde{s}^i\right)}}$$

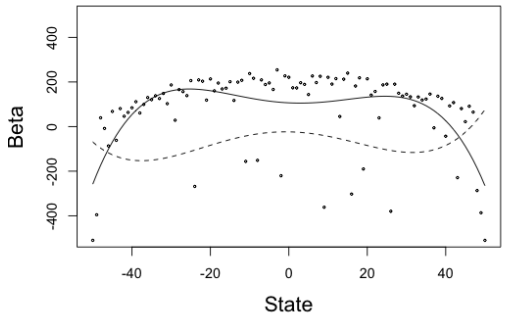
for $j = 0, 1$ where $\tilde{s}$ represents the standardized state. This model has 10 parameters as opposed to the tabular model above which has 202 parameters. It is convenient to represent $\boldsymbol{\beta}$ in matrix form as

$$\boldsymbol{\beta} = \begin{bmatrix} \beta_{0,0} & \beta_{0,1} \\ \beta_{1,0} & \beta_{1,1} \\ \beta_{2,0} & \beta_{2,1} \\ \beta_{3,0} & \beta_{3,1} \\ \beta_{4,0} & \beta_{4,1} \end{bmatrix} := \begin{bmatrix} \boldsymbol{\beta}_0 & \boldsymbol{\beta}_1 \end{bmatrix}$$

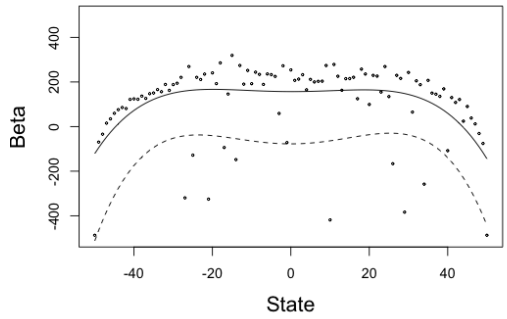
[28] Alternatively the estimates can be plotted on the probability scale



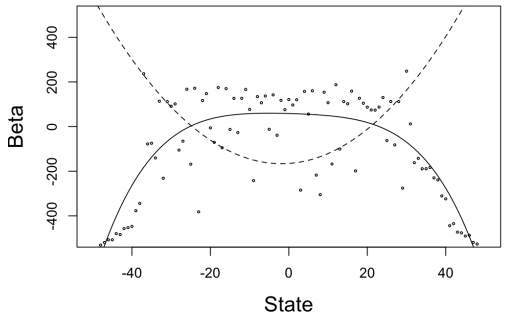
(a) $c = 0.1$ and $\lambda = 1$



(b) $c = 10$ and $\lambda = 1$



(c) $c = 50$ and $\lambda = 1$



(d) $c = 0$ and $\lambda = 0.95$

Figure 2.11: Parameter estimates obtained applying Algorithm 2.9 to optimal stopping on a random walk in a single typical replicate. The dots correspond to the estimates of β_{s,a_0}, the solid line to the fit to these points using a fourth order polynomial and the dashed line indicates the fitted values to β_{s,a_1} using a fourth order polynomial.

Again (with $\eta = 1$) the gradient of $\ln w(a|s, \boldsymbol{\beta})$ is available in closed form with components

$$\frac{\partial}{\partial \beta_{i,j}} \ln w(a_0|s, \boldsymbol{\beta}) = \begin{cases} \big(1 - w(a_0|s, \boldsymbol{\beta})\big)\tilde{s}^i & j = 0 \\ -w(a_1|s, \boldsymbol{\beta})\tilde{s}^i & j = 1 \end{cases} \tag{2.61}$$

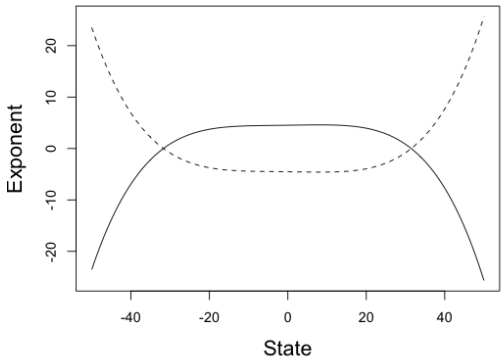
and

$$\frac{\partial}{\partial \beta_{i,j}} \ln w(a_1|s, \boldsymbol{\beta}) = \begin{cases} -w(a_0|s, \boldsymbol{\beta})\tilde{s}^i & j = 0 \\ \big(1 - w(a_1|s, \boldsymbol{\beta})\big)\tilde{s}^i & j = 1. \end{cases}$$

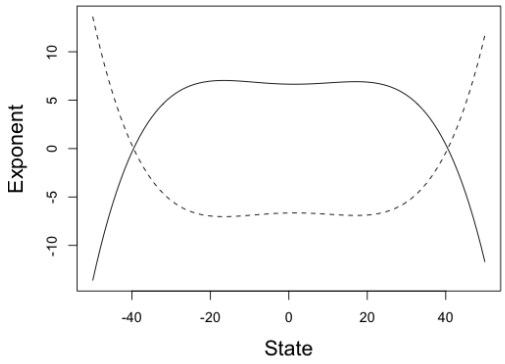
Note that in this model, $w(a_1) = 1 - w(a_0)$ so that the above expressions can be simplified further but we leave them in the above form in order to easily generalize to cases with more than two actions such as the queuing service rate control model analyzed earlier.

Figure 2.12 show the fitted exponents as a function of the state using a fourth

degree polynomial with $\eta = 1/100$ and $b(s) = -.3(50)^2$. Observe that the stopping regions differ from the optimal (in the case $c = 0.1$ and $\lambda = 1.0$) and also from those using a polynomial fit applied to the tabular model. Note also that with this baseline choice and parameter choice, the estimates when $c = 10$ diverged resulting in overflow in the exponents of the softmax.



(a) $c = 0.1$ and $\lambda = 1$



(b) $c = 0$ and $\lambda = 0.95$

Figure 2.12: Parameter estimates obtained applying Algorithm 2.9 to optimal stopping on a random walk in a single typical replicate using a fourth order polynomial. Solid line corresponds to stopping and the dashed line to continuing.

2.5.6 Policy gradient in a discounted model

As noted frequently, an infinite-horizon discounted model may be analyzed through simulation as either:

1. a random horizon expected total reward model in with the horizon length is sampled from a geometric distribution with parameter λ, or
2. a finite horizon model in which the total discounted reward is truncated at a fixed decision epoch.

Algorithm 2.9 applies directly to the first representation where the stopping time is either be sampled before each episode or realized by sampling from a Bernoulli distribution at each decision epoch within an episode. To apply truncation, (2.57) must be modified to include the discount factor as follows:

$$\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} + \tau_n \nabla_{\boldsymbol{\beta}} \ln\left(w(a^n|s^n;\boldsymbol{\beta})\right)\left(\sum_{k=n}^{N} \lambda^{k-n} r^k - b(s)\right) \tag{2.62}$$

Otherwise the algorithm is applied as is.

A numerical example

We illustrate the truncation approach in the context of the frequently analyzed two-state model (Example 1.5). Experiments used softmax action selections probabilities and tabular representations for the policy so that $\boldsymbol{\beta} \in \{\beta_{1,1}, \beta_{1,2}, \beta_{2,1}, \beta_{2,2}\}$ where the first subscript corresponds to the state and the second subscript corresponds to the action. The implementation used a discount rate $\lambda = 0.9$, $N' = 1000$ episodes, truncation at $N = 100$, learning rate $\tau_n = 50/(1000+n)$ where n denotes the episode number and weights initialized at $\boldsymbol{\beta} = \mathbf{0}$. Experiments explored the impact of the softmax exponent and the baseline over varying random number seeds.

By choosing the softmax exponent equal to 1/10, the impact of baseline was negligible and the algorithm terminated with action selection probabilities close to 1 for the optimal policy across a wide range of random seeds. For example, for a specific random number seed

$$w(a_{1,2}|s_2, \boldsymbol{\beta}) = 0.988 \quad \text{and} \quad w(a_{2,2}|s_2, \boldsymbol{\beta}) = 0.999.$$

Using a softmax exponent equal to 1 resulted in frequent convergence to a non-optimal policy.

2.5.7 Policy gradient: concluding remarks

In the discrete domain numerical examples in this section, we have found that the policy gradient algorithm is extremely sensitive algorithmic specification including: initialization, baseline specification, state and reward scaling, softmax scaling, and learning rate.

We observed that a configuration that identifies an optimal policy for one random number seed may converge to a non-optimal policy for another random number seed. These results are consistent with observations in the literature for instance [29] noted:

- Experiments are difficult to reproduce because numerical results are sensitive to hyper-parameters and can vary over random number seeds.
- A major share of claimed performance increments is less achieved by innovative algorithmic properties but more through clever implementation.
- Results obtained using algorithms based on neural networks can be achieved by simpler models based on linear function approximations.

Therefore we encourage the reader to experiment with algorithmic features when attempting to use the methods herein.

[29] Gronauer et al. [2021]

2.6 Actor-Critic Algorithms: Combining policy and value function approximation

We saw in the previous section that the expression:

$$\delta(s) := \left(\sum_{k=n}^{N} r^k - b(s^n) \right)$$

in (2.57) is fundamental to behavior of iterates of the policy gradient algorithm. In this expression N represented a realization of the episode length and $\sum_{k=n}^{N} r^k$ a realization of the total reward over the episode starting in in state s^n and choosing action a^n. In other words, the sum represents a realization of the state-action value function $q^{(d_\beta)^\infty}(s^n, a^n)$ under the randomized stationary policy $(d_\beta)^\infty$ corresponding to weights $\boldsymbol{\beta}$. To simplify notation let $q(s, a, \beta) := q^{(d_\beta)^\infty}(s, a)$.

This observation sheds light on choosing the baseline. Suppose we choose the baseline to be the expected value of the policy $(d_\beta)^\infty$ starting in s^n, $v^{(d_\beta)^\infty}(s^n)$. To simplify notation let $v(s, \boldsymbol{\beta}) := v^{(d_\beta)^\infty}(s)$.

Then reminiscent of policy iteration algorithms in Chapters **??** - **??**:

> If $q(s, a, \boldsymbol{\beta}) - v(s, \boldsymbol{\beta}) > 0$ it is desirable to increase the probability of selecting action a in state s.
>
> If $q(s, a, \boldsymbol{\beta}) - v(s, \boldsymbol{\beta}) < 0$ it is desirable to decrease the probability of choosing action a in state s.

Thus with this choice of baseline, *on average* the policy gradient algorithm above will generate a new weight vector that corresponds to a policy with an increased value. We include the expression "on average" in the previous statement to account for stochastic variation in the course of a simulation.

It is convenient to introduce the *advantage function*[30]

$$A(s, a, \boldsymbol{\beta}) := q(s, a, \boldsymbol{\beta}) - v(s, \boldsymbol{\beta})$$

defined for $s \in S$, $a \in A_s$ and $\boldsymbol{\beta} \in \Re^K$. Then the sign and magnitude of the advantage function impact the change in parameter values and the probability of selecting the designated action.

Note that one benefit of using the advantage function is that it may avoid explicit calculation of the two individual components. Since $q(s, a, \boldsymbol{\beta})$ and $v(s, \boldsymbol{\beta})$ are not known, the challenge is to incorporate these expressions in algorithms. Of course in a simple model, $v(s, \beta)$ can be computed exactly using appropriate policy evaluation

[30]Hopefully the use of the symbol A to represent a function will not be confused with the notation A_s for the set of actions in state A.

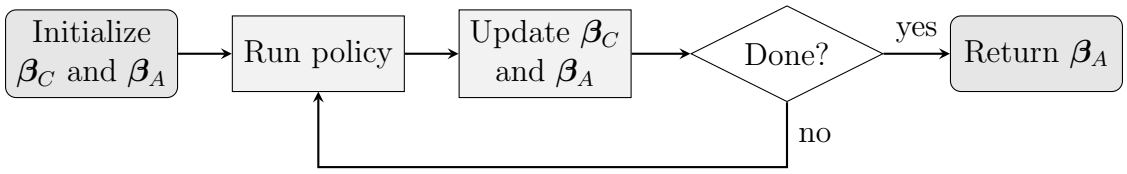


Figure 2.13: Steps in an actor-critic algorithm.

methods[31], however in doing so we found that convergence to an optimal policy was not guaranteed, especially in the random walk model of the previous section.

Note that in this notation the gradient of the value function can be represented by

$$\nabla_{\boldsymbol{\beta}} v(\boldsymbol{\beta}) = E^{(d_{\boldsymbol{\beta}})^\infty}\left[\sum_{j=1}^{N_\Delta} \nabla_{\boldsymbol{\beta}} \ln\big(w(Y_j|X_j;\boldsymbol{\beta})\big) A(X_i, Y_i, \boldsymbol{\beta})\right]. \tag{2.63}$$

We digress regarding nomenclature. The literature refers to algorithms which use the advantage function or state-action value function combined with policy gradient algorithm as *actor-critic algorithms*. The *actor* corresponds to a policy and the *critic* an evaluator. Feedback from the critic enables to actor to improve the policy. Policy gradient algorithms are frequently referred to as actor-only algorithms and q-learning algorithms as critic-only algorithms.

Actor-critic algorithm overview

Actor-critic algorithms are based on approximating both the policy and advantage function (or its components) by weighted combinations of features. Let $\mathbf{b}_A(s,a)$ denote the vector of actor features and $\boldsymbol{\beta}_A$ denote the corresponding vector of weights. Similarly let $\mathbf{b}_C(s,a)$ denote the vector of critic features and $\boldsymbol{\beta}_C$ denote the corresponding vector of critic weights. Note the features may be either the same or different.

Figure **??** provides a high level schematic of the actor-critic algorithm. The algorithm can be implemented online or in batch mode. The former is suitable for continuing tasks modelled by an infinite horizon total discounted (or average reward) model while the batch mode is designed for episodic task however it can be adopted for discounted models by including a geometric stopping time. "

In an online implementation, "Run policy" generates a single transition after which both weight vectors are updated. In batch mode "Run policy" generates a trajectory of states, actions and rewards using the current policy weights $\boldsymbol{\beta}_A$ and then both $\boldsymbol{\beta}_A$ and $\boldsymbol{\beta}_C$ are updated. In batch mode, variants of the algorithm interchange the order of updating the weights as well as the method. The critic weights $\boldsymbol{\beta}_C$ can be updated using TD(0), TD(γ) or iterative least squares and the actor weights $\boldsymbol{\beta}_A$ can be updated using a policy gradient with an estimate of the critic based on the current rewards or previous rewards in the advantage function.

[31] either value iteration of solving a evaluation equations

Note that an actor-critic algorithm avoids the need to use ϵ-altered policy in Q-policy iteration.

2.6.1 An online actor-critic algorithm

The online actor-critic algorithm is the easiest to describe and applies to discounted infinite horizon applications. We state it assuming a linear value function approximation

$$v(s;\boldsymbol{\beta}_C) = \boldsymbol{\beta}_C^T \mathbf{b}_C(s).$$

Algorithm 2.10. On-line actor critic algorithm[a]

1. **Initialization:**
   (a) Initialize $\boldsymbol{\beta}_C$ and $\boldsymbol{\beta}_A$.
   (b) Specify learning rate sequences $\{\tau_n^C; n = 1, 2, \ldots\}$ and $\{\tau_n^A; n = 1, 2, \ldots\}$.
   (c) Specify number of iterations N and set $n = 1$.
   (d) Choose $s \in S$.
2. **Loop:** While $n < N$:
   (a) Sample a from $w(\cdot|s, \boldsymbol{\beta}_A)$.
   (b) Generate (s', r).
   (c) **Update critic:**

$$\boldsymbol{\beta}_C \leftarrow \boldsymbol{\beta}_C - \tau_n^C \big(r + \lambda v(s', \boldsymbol{\beta}_C) - v(s, \boldsymbol{\beta}_C)\big)\mathbf{b}_C(s) \tag{2.64}$$

   (d) **Evaluate advantage:**

$$A(s, a, \boldsymbol{\beta}_C) := r + \lambda v(s', \boldsymbol{\beta}_C) - v(s, \boldsymbol{\beta}_C). \tag{2.65}$$

   (e) **Update actor:**

$$\boldsymbol{\beta}_A \leftarrow \boldsymbol{\beta}_A + \tau_n^A \Big(\nabla_{\boldsymbol{\beta}_A} w(a|s, \boldsymbol{\beta}_A)\Big) A(s, a, \boldsymbol{\beta}_C). \tag{2.66}$$

   (f) $n \leftarrow n + 1$, $s \leftarrow s'$.
3. Return $\boldsymbol{\beta}_A$.

[a] Some authors refer to this as *one-shot* actor critic.

Some comments about this algorithm follow:

1. As stated, the critic update uses TD(0). A TD(γ) update may be preferable however the critic update step will be less transparent.

2. Note that the advantage equals the temporal difference applied at the updated value function parameter. To avoid this extra calculation, the advantage can be set equal to the temporal difference in which case the updated critic parameter is not used until the next iterate.

3. If a nonlinear value function approximation is used, $\nabla_{\boldsymbol{\beta}_C} v(s, \boldsymbol{\beta}_C)$ replaces the expression $\mathbf{b}_C(s)$ in 2.64.

4. The advantage estimates $q(s, a)$ with its sampled version $r + \lambda v(s', \boldsymbol{\beta}_C)$. Alternatively, one can approximate $q(s, a, \boldsymbol{\beta}_C)$ and estimate its weights iteratively.

5. All the earlier comments about exploration and model tuning the apply.

6. Many variants in the literature consider updates of the critics based on parallel sampling states and rewards from parallel implementations.

7. **(Make more precise)**Since the algorithm is a variant of a policy gradient algorithm it will converge under standard conditions on the learning rate for the actor. By adaptively choosing the baseline using the advantage function, the convergence should be to an optimal policy.

An example

This example illustrates the online actor-critic algorithm in the context of the two-state example.

Example 2.9. This example applies Algorithm 2.10 to a discounted ($\lambda = 0.9$) version of the two-state example analyzed frequently throughout the book. It uses state-action indicators as features for the actor and indicators of the state as features for the critic. Consequently the updates for the critic correspond to TD(0) recursion

$$v(s) \leftarrow v(s) - \tau_n^C \Big(r + \lambda v(s') - v(s) \Big),$$

updates for the advantage are given by

$$A(s, a) = r + \lambda v(s') - v(s),$$

and updates for components of the actor weights are given by

$$\beta_{(s',a')} \leftarrow \begin{cases} \beta_{(s,a)} + \tau_n^A \Big(1 - w(a|s, \boldsymbol{\beta}_A) \Big) A(s, a) & \text{for } (s', a') = (s, a) \\ \beta_{(s',a')} + \tau_n^A \Big(- w(a|s, \boldsymbol{\beta}_A) \Big) A(s, a) & \text{otherwise.} \end{cases}$$

Note that at each iteration, the critic is updated only in the observed state s while the actor weights are updated for all state-action pairs.

The algorithm converged to the optimal value function and policy for a wide range of learning rates and numbers of iterations. Figure 2.14 shows the sequences of values for a typical replicate with N =20,000 iterations and $\tau_n^A = \tau_n^C = 100/(1000 + n)$. In this replicate the corresponding policy used action selection probabilities

$$w(a_{1,2}|s_1, \boldsymbol{\beta}_A) = 0.994 \quad \text{and} \quad w(a_{2,2}|s_2, \boldsymbol{\beta}_A) = 1.000$$

in agreement with the optimal policy.

We also observed that the algorithm converged to the optimal policy for random number seeds in which the policy gradient algorithm (Algorithm 2.9 converged to a sub-optimal policy. Thus by replacing an arbitrary baseline by a critic that tracked the "current policy", the algorithm avoided convergence to a sub-optimal policy.

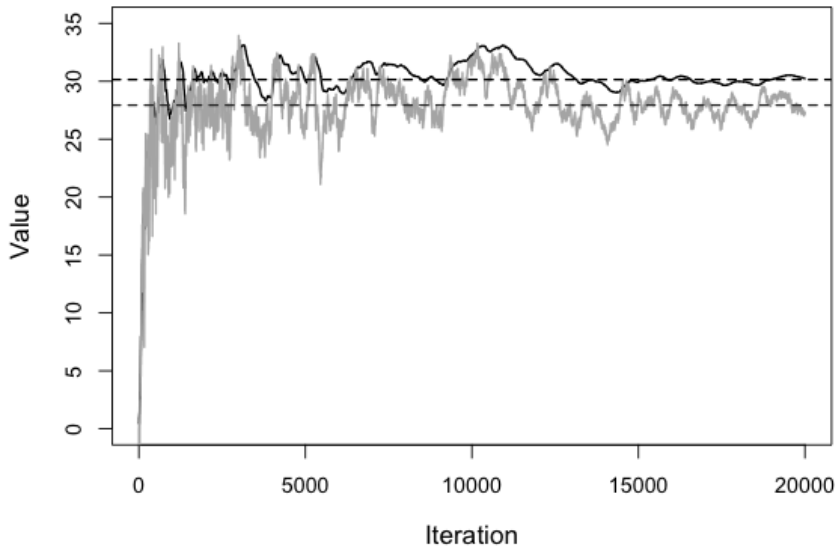


Figure 2.14: Value function estimates in two-state example using Algorithm 2.10 obtained in a typical replicate. Black lines correspond to $v^*(s_1)$ and grey lines to $v^*(s_2)$; solid lines correspond to estimates and dashed lines to true value.

2.6.2 A batch actor-critic algorithm

The following algorithm describes actor-critic implementations suitable for an undiscounted episodic models. It generalizes the (batch) policy gradient algorithm above (Algorithm 2.9) by using the critic to update the baseline.

Algorithm 2.11. Batch actor-critic

1. **Initialization:**
   (a) Initialize $\boldsymbol{\beta}_A$.
   (b) Specify learning rate sequence $\{\tau_n^A; n = 1, 2, \ldots\}$.
   (c) Specify number of episodes N'.
2. Repeat N' times:
   (a) Generate an episode and save a sequence of states, actions and rewards $(s^1, a^1, s^2, r^1, \ldots, s^N, a^N, s^{N+1}, r^N)$ in which N denotes the termination time of the episode.
   (b) **Update critic:** Choose

$$\boldsymbol{\beta}_C \in \arg\min_{\boldsymbol{\beta}\in\Re^K} \sum_{n=1}^{N} \Big(\sum_{k=n}^{N} r^k - v(s^n, \boldsymbol{\beta}) \Big)^2 \tag{2.67}$$

   (c) $n \leftarrow 1$
   (d) While $n \leq N$
      i. **Estimate advantage:**

$$A(s^n, a^n, \boldsymbol{\beta}_C) = \sum_{k=n}^{N} r^k - v(s^n; \boldsymbol{\beta}_C) \tag{2.68}$$

      ii. **Update actor:**

$$\boldsymbol{\beta}_A \leftarrow \boldsymbol{\beta}_A + \tau_n^A \nabla_{\boldsymbol{\beta}} \ln \big(w(a^n|s^n; \boldsymbol{\beta}_A) A(s^n, a^n, \boldsymbol{\beta}_C) \tag{2.69}$$

      iii. $n \leftarrow n + 1$.
3. Return $w(s|a, \boldsymbol{\beta}_A)$.

Some comments on the algorithm follow.

1. Critic weights $\boldsymbol{\beta}_C$ can be also be updated after updating actor weights $\boldsymbol{\beta}_A$. Our experience, as illustrated by the example below, is that the algorithm converged

more reliably when the critic was updated before updating the actor.

2. The above implementation uses regression to estimate the the critic weights based derived from an entire episode. An alternative is to minimize

$$h(\boldsymbol{\beta}) := \sum_{n=1}^{N} \Big(\sum_{k=n}^{N} r^k - v(s^n, \boldsymbol{\beta}) \Big)^2$$

by gradient descent. To do so, an algorithm would compute

$$\nabla_{\boldsymbol{\beta}} h(\boldsymbol{\beta}) = \nabla_{\beta} \sum_{n=1}^{N} \Big(\sum_{k=n}^{N} r^k - v(s^n, \boldsymbol{\beta}) \Big)^2$$

which in the linear case equals

$$\nabla_{\boldsymbol{\beta}} h(\boldsymbol{\beta}) = -2 \sum_{n=1}^{N} (\Big(\sum_{k=n}^{N} r^k - v(s^n, \boldsymbol{\beta}) \Big) \mathbf{b}^C(s^n)$$

where $\mathbf{b}^C(s)$ **(check notation for critic features)** denotes the critic weights evaluated at s. Absorbing the constant into the learning rate gives the recursion for $\boldsymbol{\beta}_C$:

$$\boldsymbol{\beta}_C \leftarrow \boldsymbol{\beta}_B - \tau_n^C \nabla_{\boldsymbol{\beta}} h(\boldsymbol{\beta}). \tag{2.70}$$

Note that this approach requires specifying an additional learning rate and monitoring the stability of gradient estimates.

3. The above algorithm uses the same advantage function as that used by the policy gradient algorithm. An alternative is to use the single step (bootstrapped) advantage function:

$$A(s^n, a^n, \boldsymbol{\beta}_C) := r^n + v(s^{n+1}, \boldsymbol{\beta}_C) - v(s^n, \boldsymbol{\beta}_C) \tag{2.71}$$

4. The actor weights can be updated iteratively as in the algorithm or accumulated and then updated using a single gradient ascent step.

5. Instead of using value functions, one could evaluate the advantage in terms of the state action function $q(s, a; \boldsymbol{\beta}_C)$ as follows:

$$A(s^n, a^n, \boldsymbol{\beta}_C) := q(s^{n+1}, a^{n+1}, \boldsymbol{\beta}_C) - q(s^n, a^n, \boldsymbol{\beta}_C). \tag{2.72}$$

(Check this makes sense)

An example

This simple example applies the batch actor critic algorithm to the shortest path model in Section 2.3.3

This example applies Algorithm 2.11 to an instance with $M = 10$, $N = 7$, $(m^*, n^*) = (10, 7)$ $R = 10$ and $c = 0.1$. In this application the robot seeks to learn a path to cell $(10, 7)$ from each starting cell. The implementation represents the critic by the model

$$v\big((i,j), \boldsymbol{\beta}^C\big) = \beta_0^C + \beta_{1,0}^C i + \beta_{0,1}^C j + \beta_{1,1}^C ij + \beta_{2,0}^C i^2 + \beta_{0,2}^C j^2$$

so that $\boldsymbol{\beta} = (\beta_0^C, \beta_{1,0}^C, \beta_{0,1}^C, \beta_{1,1}^C, \beta_{2,0}^C, \beta_{0,2}^C)$ where critic parameters are computed using ordinary least squares to obtain an estimate of $\boldsymbol{\beta}$ and compute the advantage. The actor was represented in tabular form with $\boldsymbol{\beta}^A = \{\beta_{(i,j,a)}^A : (i,j) \in S \text{ and } a \in A_{(i,j)}\}$.

Implementation details follow: The algorithm was initiated with components of $\boldsymbol{\beta}^A$ sampled from a standard normal distribution to so as to avoid too many long episodes. Episodes were truncated at 200 iterations and the algorithm was run for 50000 episodes, each starting from a random cell not equal to the target. The learning rate, $\tau_n^A = 40/(400 + n)$.

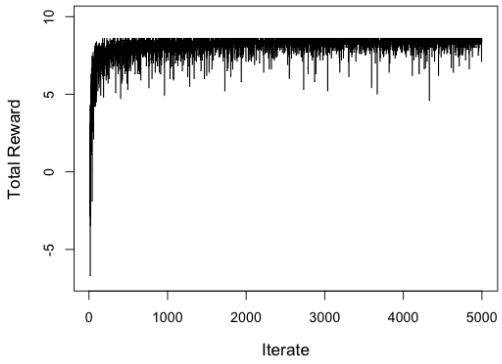
Figure 2.15 shows how the total reward per episode varies within a single replicate starting in cell $(1, 1)$ and at random. Observe that the reward increases and stabilizes with variability in total rewards when starting in a random cell more variable then when starting in cell $(1, 1)$. The advantage of the random start is that it identifies good policies for infrequently visited states.

Note that when starting in cell $(1, 1)$, the actor-critic algorithm often finds a sequence of actions[32] with probabilities close to 1 that correspond to a shortest path through the grid. However since most episodes will follow this path, the optimal probabilities off the path are not well estimated.

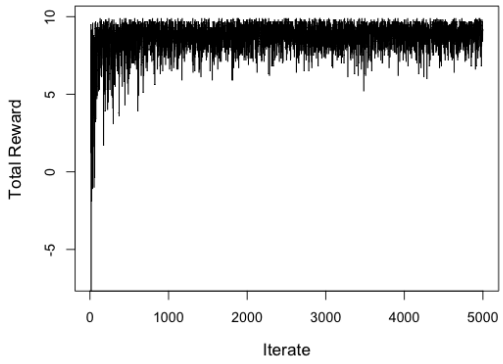
Note that when the critic was evaluated *after* the actor, the algorithm converged to sub-optimal policies that cycled between states and never reached the target cell.

We also applied the policy gradient algorithm (Algorithm 2.9) to this example. Our implementation was the same as above but instead of using the critic, it set the baseline (bl in Table 2.6 equal to $0, -8$ and the (running) mean of the previous total rewards accumulated from the starting state. Observe form Table 2.6 that the quality of estimates varied significantly with the baseline with the running mean giving the best results. In fact for this replicate the policy gradient with the mean of total rewards as the baseline gave a higher total reward than the actor-critic algorithm. However there were many instances where the policy gradient diverged and the actor-critic algorithm converged to a good policy.

[32]For example the sequence: "down" "down" "down" "down" "down" "right" "down" "down" "down" "right" "right" "down" "right" "right" "right".



(a) All episodes start in cell $(1, 1)$.



(b) Episodes start in non-terminal random cell.

Figure 2.15: A typical replicate of the total reward per episode in a model with a 10×7 grid with the target cell in lower right.

Method	Total Reward mean	Total Reward std dev
PG ($bl = 0$)	6.31	6.14
PG ($bl = -8$)	5.80	8.06
PG (bl =mean)	8.52	0.47
AC	8.24	0.78

Table 2.6: Mean and standard deviation of total rewards over episodes for a single replicate in which all four methods converged. The results used a common random number seeds obtained using the policy gradient algorithm with the specified baseline (bl) and the actor-critic algorithm as described above.

Delivering coffee

This section applies the actor-critic algorithm to find an optimal policy in the coffee delivering robot problem from Section **??** on a 5×3 grid. We divide rewards and costs by 10 to improve numerical stability so that the per step cost is -0.1, the reward for delivering the coffee is 5 and the cost for falling down the stairs is -10. An episode ends when the robot successfully delivers the coffee; if it falls down the stairs, it returns to cell 13 and starts over. This example assumes that the robot's moves are deterministic in the sense that if it plans to go right and such a move is possible, it moves right, however if there is a wall in the way, the robot stays in its current cell and incurs a cost associated with a single step. However, randomness is introduced through $w(a|s, \boldsymbol{\beta}^A)$.

Algorithmic details follow. The value function is approximated by

$$v\big((i, j), \boldsymbol{\beta}^C\big) = \boldsymbol{\beta}^C_{0,0} + \beta^C_{1,0} i + \beta^C_{0,1} j$$

where i denotes the row (indexed from the bottom) and j denotes the column (indexed from the left)[33] Its parameters are estimated in step 2(b) of Algorithm 2.11 using linear regression. The actor is again modelled by indicators of state-action pairs and estimated (by gradient ascent) using a learning rate of $\tau_n^A = 40/(400 + n)$ where n is the episode number. The model is run for 20000 episodes with initial values for $\boldsymbol{\beta}^A$ generated randomly from a standard normal distribution.

This implementation identified an optimal policy for roughly half of the random seeds. When it did not, it identified a policy that cycled without delivering the coffee. For example Figure 2.16a shows a typical replicate when the algorithm identified a good policy. For this replicate, the policy assigned probabilities exceeding 0.9 to each step on the "conservative" path $13 \to 14 \to 15 \to 12 \to 9 \to 6 \to 3 \to 2 \to 1$. Note that for a few earlier episodes, the robot incurred a cost of -10 when the robot fell down the stairs but eventually it learned to avoid such incidents. For this model the value function estimate was

$$v\big((i,j), \hat{\boldsymbol{\beta}}^C\big) = 4.164 + 0.127i + 0.055j.$$

So that the effect of moving one row closer to the destination increases the reward by more than twice as much as moving one column to the right.

We also applied the algorithm to a similar problem on a 10×5 grid with the stairs occupying 4 cells on the left hand side. Results from a single replicate are shown in Figure 2.16b. Observe that in several episodes the robot fell down the stairs more than once, with this event occurring less frequently in later episodes. The optimal path was similar to that in the smaller model, moving down to the edge of the stairwell, then moving right to column 3 and then downward toward the lower boundary.

In both models, estimating the critic weights by gradient descent failed to identify an optimal policy.

2.6.3 Actor-critic: remarks and enhancements

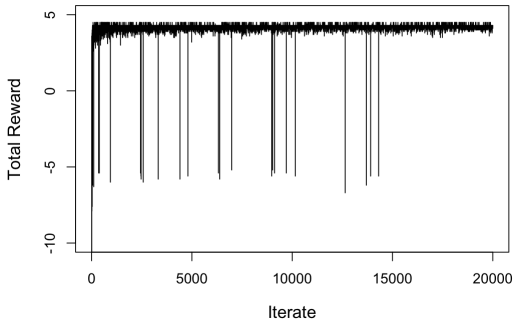
Policy gradient and actor-critic algorithms have been researched extensively in the literature. Various sources have suggested improvements to enhance convergence. They include:

Normalization: Centering and scaling states and rewards, especially when they are highly variable) can enhance convergence.

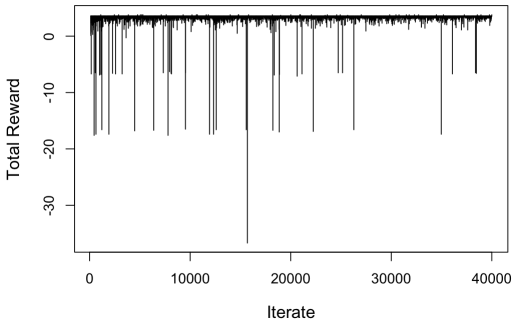
Parallelization: Running many episodes in parallel (on several robots) can provide better critic estimates.

Natural gradients: The gradients described above are sensitive to both the parameterization and the geometry (curvature) of f the underlying surface. The

[33]For example, cell 13 corresponds to row 1 and column 1.



(a) 5×3 grid; the median total reward equalled 4.3 with range between -20.9 and 4.5.



(b) 10×5 grid; the median total reward equalled 3.6 with range between -89.2 and 3.8.

Figure 2.16: Total reward by iterate for a single replicate of the coffee delivering robot model obtained using Algorithm 2.11 for two different grid sizes.

natural gradient accounts for this in the actor by multiplying the the gradient by the inverse of the Fisher information matrix given by

$$E\left[\Big(\nabla_\beta \ln w(a|s,\boldsymbol{\beta})\Big)\Big(\nabla_\beta \ln w(a|s,\boldsymbol{\beta})\Big)^T\right]$$

Generalized Advantage estimation: Generalized advantage estimation provides a TD(γ) -like approach for improved estimation of the advantage in actor-critic algorithms. It is based on taking weighted sums of expressions of the form:

$$r^n + \ldots + r^{n+m} + v(s^{n+m+1}, \boldsymbol{\beta}_C) - v(s^n, \boldsymbol{\beta}_C)$$

Improved gradient descent: Adaptive moment estimation (ADAM) provides enhanced estimates of gradients based on moving averages of the mean and variance of the gradient. It is a widely used tool in gradient descent methods.

Policy and value function networks: Instead of using linear parameterizations for $w(a|s,\boldsymbol{\beta}_A)$ and $v(a|s,\boldsymbol{\beta}_C)$ one can replaced them with neural networks using common or different features.

Replay buffer methods Trajectories are stored in a replay buffer and sampled as needed. Requires off policy methods.

Thus the analyst is faced with numerous options to improve the performance of policy gradient and actor-critic methods. Considerable experimentation is required to obtain good results. It is our intent that the introduction above provides an entry in to understanding and applying these concepts.

Appendix: Derivation of policy gradient representation

This appendix provides a proof of what is often referred to as "The policy gradient theorem". This fundamental equivalence underlies policy gradient and actor-critic methods. The proof below is given for finite horizon models. We subsequently discuss extensions to infinite horizon models.

Theorem A.1. Let N be finite and suppose $w(a|s;\boldsymbol{\beta})$ is differentiable with respect to each component of $\boldsymbol{\beta}$. Then

$$\nabla_{\boldsymbol{\beta}} v(\boldsymbol{\beta}) = E^{(d_{\boldsymbol{\beta}})^\infty}\left\{\sum_{j=1}^{N} \nabla_{\boldsymbol{\beta}} \ln(w(Y_j|X_j;\boldsymbol{\beta}))\left[\sum_{i=j}^{N} r(X_i, Y_i, X_{i+1})\right]\right\}. \tag{A.1}$$

The proof is an easy consequence of the following two results.

Lemma A.1. Let a_i for $i = 1, \ldots, N$ and b_j $j = 1, \ldots, N$ be real numbers. Then for N finite;

$$\sum_{i=1}^{N}\left[\sum_{j=1}^{i} b_j\right] a_i = \sum_{j=1}^{N} b_j \left[\sum_{i=j}^{N} a_i\right]. \tag{A.2}$$

This lemma and its proof for N infinite series as Theorem 8.3 in Rudin [1964]. Next we provide a representation for the expected value of an Markov decision process that only receives a reward $r(s, a, s')$ at the end of period n.

Lemma A.2. Consider an n-period Markov decision process that in which

$$r_i(s, a, j) = \begin{cases} 0 & i < n \\ 1 & i = n \end{cases}$$

and let $v_n(\phi)$ denotes the expected reward when using randomized decision rule $d_{\boldsymbol{\beta}}$ selects actions according to $w(a|s;\boldsymbol{\beta})$ averaged over initial states according to $\rho(\cdot)$.
Then if $w(a|s;\phi)$ is differentiable with respect to each component of ϕ:

$$\nabla_{\boldsymbol{\beta}} v_n(\boldsymbol{\beta}) = E^{(d_{\boldsymbol{\beta}})^\infty}\left\{\left(\sum_{j=1}^{n} \nabla_{\boldsymbol{\beta}} \ln(w(Y_j|X_j;\boldsymbol{\beta}))\right) r(X_n, Y_n, X_{n+1})\right\}. \tag{A.3}$$

Proof. From the definition of $v_n(\boldsymbol{\beta})$,

$$v_n(\phi) := \sum_{h \in H_n} P_\phi(h) r(s_n, a_n, s_{n+1}) = E^{(d_{\boldsymbol{\beta}})^\infty} \{r(X_n, Y_n, X_{n+1})\} \tag{A.4}$$

where H_n denotes the set of all histories of the form $(s_1, a_1, \ldots, s_n, a_n, s_{n+1})$ and $P_\phi(h)$ denotes the probability of each history where the probability distribution is parameterized by $\boldsymbol{\beta}$. By the Markov property, for a specific history,

$$P_\phi(h) = \rho(s_1) w(a_1|s_1; \boldsymbol{\beta}) p(s_2|s_1, a_1) \cdots w(a_n|s_n; \boldsymbol{\beta}) p(s_{n+1}|s_n, a_n). \tag{A.5}$$

Since $w(a|s; \boldsymbol{\beta})$ is differentiable with respect to each component of $\boldsymbol{\beta}$, we can show using (2.43) that

$$\nabla_{\boldsymbol{\beta}} P_{\boldsymbol{\beta}}(h) = P_{\boldsymbol{\beta}}(h) \nabla_{\boldsymbol{\beta}} \ln(P_{\boldsymbol{\beta}}(h)). \tag{A.6}$$

Therefore

$$\nabla_{\boldsymbol{\beta}} v_n(\phi) = \sum_{h \in H_n} P_{\boldsymbol{\beta}(h)} \nabla_{\boldsymbol{\beta}} \ln(P_{\boldsymbol{\beta}}(h)) r(s_n, a_n, s_{n+1}). \tag{A.7}$$

From (A.5)

$$\begin{aligned} \ln(P_{\boldsymbol{\beta}}(h)) = \ln(\rho(s_1)) + \ln(w(a_1|s_1, \boldsymbol{\beta})) + \ln(p(s_2|s_1, a_1)) \\ + \cdots + \ln(w(a_n|s_n, \boldsymbol{\beta})) + \ln(p(s_{n+1}|s_n, a_n)). \end{aligned}$$

Since $\rho(s)$ and $p(j|s, a)$ do not depend on $\boldsymbol{\beta}$,

$$\nabla_{\boldsymbol{\beta}} \ln(P_{\boldsymbol{\beta}}(h)) = \sum_{j=1}^{n} \nabla_{\boldsymbol{\beta}} \ln(w(a_j|s_j; \boldsymbol{\beta})). \tag{A.8}$$

Hence combining (A.7) and (A.8) gives

$$\nabla_{\boldsymbol{\beta}} v_n(\boldsymbol{\beta}) = \sum_{h \in H_n} P_{\boldsymbol{\beta}(h)} \left(\sum_{j=1}^{n} \nabla_{\boldsymbol{\beta}} \ln(w(a_j|s_j; \boldsymbol{\beta})) \right) r(s_n, a_n, s_{n+1}). \tag{A.9}$$

Rewriting this in expectation form yields the desired representation. □

Proof of Theorem A.1. It is easy to see that

$$\nabla_{\boldsymbol{\beta}} v(\boldsymbol{\beta}) = \sum_{n=1}^{N} \nabla_{\boldsymbol{\beta}} v_n(\boldsymbol{\beta})$$

so that substituting the representation in (A.3) into the above result and applying Lemma A.1 with $b_j = \ln w(Y_j|X_j; \boldsymbol{\beta})$ and $a_i = r(X_i, Y_i, X_{i+1})$. □

Extensions*

(I think this is correct)

Above we prove the policy gradient theorem for finite N. If we use the same proof for a random stopping time N which satisfies $P(N < \infty) = 1$, the result follows by applying the same proof to each sample path. Regarding an infinite horizon model as a model with random geometric stopping times allows extension to the geometric case.

To analyze infinite horizon discounted models directly requires some additional assumptions, including the variant of Lemma A.1 which allows $N = \infty$. For this to hold when N is infinite, the series must converge absolutely. Requiring $\lambda < 1$ and bounded rewards ensures this holds.

Bibliographic remarks

Out discussion just touches the surface of the vast literature of reinforcement learning with function approximation including value-based methods, policy based method and combinations thereof. Szepesvari [2010] and Bertsekas and Tsitsiklis [1996] provide comprehensive overviews of RL methods with references to theoretical results. Bertsekas and Tsitsiklis [1996] also contains a comprehensive historical perspective on the development of RL methods. Sutton and Barto [2018], and Kochendorfer et al. [2022] provide accessible and insightful presentations of this material.

Tsitsiklis and van Roy [1997] analyzes TD(γ) rigorously and motivated many subsequent results.

Sutton et al. [1999] establish convergence of Williams [1992] policy gradient algorithm REINFORCE for models with function approximation. Widrow et al. [1973] introduces the idea of using a critic and Barto et al. [1983] show how a system with *"two neuronlike elements can solve a difficult learning control problem."* Lehmann [2024] provides a state-of the art reference on policy gradient methods.

The vanilla policy-gradient algorithm is discussed in OpenAI [2024]. Course lecture notes available on the internet provide excellent guides for learning this material. We especially liked the notes of Levine [2024] who provides an insightful discussion of policy gradient and actor-critic algorithms and those of Katselis [2019] on Q-learning.

Lagoudakis and Parr [2003] describe a least squares policy iteration method that avoids the using of learning rates and is suitable for linear function approximations.

Exercises

You should try to replicate the computational results in this chapter and as well try out all the methods herein on examples you are interested in. Chapter **??** provides a wide range of possibilities. We think that advanced appointment scheduling **??** offers an excellent vehicle for testing these methods. Analysing it with neural networks offers considerable research potential.

The exercises below often some alternative suggestions.

1. Specify first-visit and every-visit Monte Carlo approximation algorithms for estimating the value of a policy in an episodic model. Apply them to the shortest path grid-world model in Section **??**.

2. Consider the problem of optimal stopping in a random walk in Example 2.1.

   (a) Obtain a linear value approximation (i.e., polynomial or piecewise-polynomial) for π_2 and π_3 when $p = 0.45, 0.48, 0.5, 0.52, 0.55$. Compare your analysis to that in the text.

   (b) Find a suitable value function approximation in cases in which a linear fit is inadequate.

   (c) Which fit would you use if you wished to incorporate this approximation in an optimization algorithm in which required value functions approximations for several policies.

   (d) Obtain approximations of state-action value functions.

3. (a) State a Monte Carlo value function approximation algorithm for a discounted MDP based on geometric stopping. Recall that in this case, you accumulate the undiscounted total reward up to a geometric stopping time instead of the discounted reward as was the case in the truncation version.

   (b) Use it to approximate in the value function in the two instances described in Example 2.2.

4. Show that when features are indicators of states, Algorithm 2.5 reduces to Algorithm **??**

5. Apply gradient descent and stochastic gradient decent to estimate the parameters in the regression model $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \varepsilon$ using methods similar to Example **??**.

6. **Hybrid SGD** Develop an SGD policy evaluation algorithm that randomly restarts in a random state after M transitions. Apply this approach to the model in Example **??**.

7. **SGD in an episodic model**. Develop an apply a version of SGD for policy evaluation in an episodic model. Apply it to obtain a linear approximation to the value function for π_2 in Example 2.1.

8. Provide a TD(γ) version of Algorithm **??** and apply it to the queuing control model in Example 2.3

9. Fit a cubic spline with two knots to the queuing control model of Example and use Algorithm **??** to estimate its parameters.

10. Show that when features are indicator functions of state-action pairs, the tabular Q-learning algorithm of Chapter **??** is equivalent to the Q-learning algorithm in this chapter.

11. Write out features when $f_k(s)$ and $h_j(a)$ are polynomials in s and a respectively.

12. Consider the episodic shortest path model in Section 2.3.3.

    (a) Verify the conclusions in Section 2.3.3 by developing your own codes.

    (b) Repeat the calculations using a neural network and other parametric state-action value function approximations.

    (c) Specify a SARSA variant of Q-learning algorithm 2.6 and apply it to this problem.

    (d) Solve the model using the Q-policy iteration algorithm, Algorithm 2.7.

13. One of the challenges of using Q-learning in an episodic model is that large rewards are not realized until a goal state is reached. Consider an alternative shortest path model in which the reward when reaching cell (i,j) is of the form $ci + dj$ so that rewards are generated along the path to the goal state.

    Apply Q-learning with function approximation to this variant of the shortest path model and summarize your conclusions.

14. **State aggregation in optimal stopping:** Apply policy gradient and actor-critic to the the optimal stopping model as follows. Let $N = 60$ and compare approximations that aggregate consecutive states into groups of $M = 1, 2, 4$. For example when $M = 4$ the state space is partitioned into K groups of states of the form $G_1 := \{1, 2, 3, 4\}, G_2 := \{5, 6, 7, 8\} \ldots, G_{15} = \{57, 58, 59, 60\}$. In this case features can be written as

$$b_{k,j}(s, a) = \begin{cases} 1 & s \in G_k,\ a = a_j \\ 0 & \text{otherwise.} \end{cases}$$

    for $k = 1, \ldots, 15$ and $j = 1, 2$. We can represent these features as 30 component vectors with the first 15 components corresponding to a_1 and the next 15 components corresponding to a_2.

15. Derive (2.44).

16. Apply policy gradient and actor-critic to the discounted queuing service rate control model.

17. Consider the shortest path through a grid analyzed in Section 2.6.2.

    (a) Replicate the analysis in the text using function approximations for both the actor and critic. See Section 2.3.3 for guidance on specificaiton of basis functions.

(b) Repeat you analysis assuming the cost per step $c(i, j)$ in cell (i, j) depends on the row i and column j. For example suppose $c(i, j) = -0.1 - 0.1i$. How do you think modifying the costs in this way effects policy choice. (Note we found actor-critic and policy gradient frequently diverged or converged to sub-optimal policies for this modification.)

18. Partial observable models and rl

Bibliography

Jr. A.F. Veinott. On discrete dynamic programming programming with sensitive discount optimality criteria. *Ann. Math. Stat.*, 40:1635–1660, 1969.

Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13(5):834–846, 1983.

R.E. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, 1957.

D. Bertsekas and J. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996.

D.P. Bertsekas. *Lessons from AlphaZero for Optimal, Model Predictive and Adaptive Control, 2nd Edition*. Athena Scientific, 2022.

D.P. Bertseksas. *Dynamic Programming and Optimal Control, Volume 2, 4th Edition*. Athena Scientific, 2012.

D. Blackwell. Discrete dynamic programming. *Ann. Math. Stat.*, 33:719–726, 1962.

C. Derman. *Finite State Markovian Decision Processes*. Academic Press, Newyork, NY., 1970.

J. Gessford and S. Karlin. Optimal policies for hydroelectric operations. In K. Arrow, S.Karlin, and H. Scarf, editors, *Studies in the Mathematical Theory of Inventory and Production*, pages 179–200. Stanford University Press, Stanford,CA, 1958.

S. Gronauer, M. Gottwald, and K. Diepold. The sucessful ingredients of policy gradient algorithms. In *Proceedings of the Thirtieth Joint Conference on Artificial Intelligence (IIJCAI-21)*, pages 2455–2461, 2021.

R. Howard. *Dynamic Programming and Markov Processes*. MIT Press, Cambridge, MA, 1960.

Dimitrios Katselis. Ece586: Mdps and reinforcement learning: Lecture 10, 2019. URL `http://katselis.web.engr.illinois.edu/ECE586/Lecture10.pdf`.

J. G. Kemeny and J. L. Snell. *Finite Markov Chains*. Van Nostrand-Reinhold, New York, 1960.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization, 2017.

M. J. Kochendorfer, T. A. Wheeler, and K. H. Wray. *Algorithms for Decision Making*. MIT Press, 2022.

M. Lagoudakis and R. Parr. Least-squares policy evaluation. *J. Mach. Learning Res.*, 4:1107–1149, 2003.

M. Lehmann. The definitive guide to policy gradients in deep reinforcement learning: Theory, algorithms and implementations, 2024.

Sergey Levine. Cs285: Deep reinforcement learning; lectures 4-9, 2024. URL `https://rail.eecs.berkeley.edu/deeprlcourse/`.

P. Massé. *Les reserves et la regulation se l'avenir dans la vie economique, Vol I and II* . Hermann, 1946.

OpenAI. Vanilla policy gradient, 2024. URL `https://spinningup.openai.com/en/latest/algorithms/vpg.html`. Accessed: 2024-06-11.

W. Powell. *Approximate Dynamic Programming*. J.Wiley and Sons, 2007.

M. L. Puterman. *Markov Decision Processes:Discrete Stochastic Dynamic Programming*. John Wiley & Sons., 1994.

R. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2021. URL `https://www.R-project.org/`.

W. Rudin. *Principles of Mathematical Analysis, 2nd Edition*. McGraw-Hill, Inc., 1964.

L. Sennott. *Stochastic Dynamic Programming and the Control of Queueing Systems* . John Wiley and Sons, 1999.

L.S. Shapley. Stochastic games. *Proc. Nat. Acad. Sci. USA*, 39:1095–1100, 1953.

R. Sutton, D. MacAllester, S. Singh, and Y.Manoour. Policy gradient methods for reinforcement learning with function approximation. In *NIPS Proceedings*, pages 1057–1063, 1999.

R. S. Sutton and A. G. Barto. *Reinforcement Learning: An introduction, second edition*. MIT Press, Cambridge, MA, 2018.

C. Szepesvari. *Algorithms for Reinforcement Learning* . Morgan and Claypool, 2010.

J.N. Tsitsiklis and B. van Roy. An analysis of temporal-difference learning with function approximations. *IEEE Trans. of Auto. Cont.*, 42:674–690, 1997.

A. Wald. *Sequential Analysis*. John Wiley & Sons, New York, 1947.

B. Widrow, N. K. Gupta, and S. Maitra. Punish/reward: learning with a critic in adaptive threshold systems. *IEEE Trans. Syst. Man Cybern.*, 3(5):455–465, 1973.

R. Williams. Some statistical gradient-following allgorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.

Index