REINFORCEMENT LEARNING AND STOCHASTIC OPTIMIZATION

A unified framework for sequential decisions

Warren B. Powell

August 22, 2021

WILEY-INTERSCIENCE

A JOHN WILEY & SONS, INC., PUBLICATION

Published by John Wiley & Sons, Inc., Hoboken, New Jersey.
Published simultaneously in Canada.

Library of Congress Cataloging-in-Publication Data:

Optimization Under Uncertainty: A unified framework
Printed in the United States of America.

10 9 8 7 6 5 4 3 2 1

CHAPTER 3

ONLINE LEARNING

There is a massive community that has evolved under names such as statistics, statistical learning, machine learning, and data sciences. The vast majority of this work, known as *supervised learning*, involves taking a dataset $(x^n, y^n),\ n = 1, \ldots, N$ of input data x^n and corresponding observations (sometimes called "labels") y^n and using this to design a statistical model $f(x|\theta)$ that produces the best fit between $f(x^n|\theta)$ and the associated observation (or label) y^n. This is the world of big data.

This book is on the topic of making decisions (that we call x). So why do we need a chapter on learning? The simple explanation is that machine learning arises throughout the process of helping computers make decisions. Classical machine learning is focused on learning something about an exogenous process: forecasting weather, predicting demand, estimating the performance of a drug or material. In this book, we need exogenous learning for the same reason everyone else does, but most of the time we will focus on *endogenous learning*, where we are learning about value functions, policies, and response surfaces, which are learning problems that arise in the context of methods for making decisions.

We open this chapter with an overview of the role of machine learning in the context of sequential decision making. The remainder of the chapter is an introduction to machine learning, with an emphasis on learning over time, a topic known as online learning, since this will dominate the applications of machine learning for sequential decisions.

As elsewhere, the sections marked with an * can easily be skipped on an initial pass through this chapter. Readers should understand the information that is available in this chapter, but otherwise should view it as a reference that is turned to on an as needed basis (and there will be many references to this chapter in the rest of the book).

3.1 MACHINE LEARNING FOR SEQUENTIAL DECISIONS

It is useful to begin our discussion of statistical learning by describing the learning issues that arise in the context of sequential decisions. This section provides an overview of the following dimensions of learning problems:

- Observations and data in sequential decisions - While classical statistical learning problems consist of datasets comprised of input (or independent) variables x and output (or dependent) variables y, in sequential decision making the input variables x^n are decisions that we control (at least partially).
- Indexing data - When we do batch learning, we use a dataset $(x^n, y^n),\ n = 1, \ldots, N$ where y^n is the response associated with the input data x^n. In the context of sequential decisions, we pick x^n and then observe y^{n+1}.
- Functions we are learning - There are a half dozen different classes of functions that we may need to approximate in different stochastic optimization contexts.
- Sequential learning - Most of our applications involve starting with little or no data, and then successively acquiring more data. This often means we have to transition from low-dimensional models (which can be fitted with little data) to higher-dimensional models.
- Approximation strategies - Here we summarize the three major classes of approximation strategies from the statistical learning literature. The rest of this chapter summarizes these strategies.
- Objectives - Sometimes we are trying to fit a function to data which minimizes errors, and sometimes we are finding a function to maximize contributions or minimize costs. Either way, learning functions is always its own optimization problem, sometimes buried within a larger stochastic optimization problem.
- Batch vs. recursive learning - Most of the statistical learning literature focuses on using a given dataset (and of late, these are very large datasets) to fit complex statistical models. In the context of sequential decision problems, we primarily depend on adaptive (or online) learning, so this chapter describes recursive learning algorithms.

3.1.1 Observations and data in stochastic optimization

Before we present our overview of statistical techniques, we need to say a word about the data we are using to estimate functions. In statistical learning, it is typically assumed that we are given input data x, after which we observe a response y. Some examples include

- We may observe the characteristics x of a patient to predict the likelihood y of responding to a treatment regime.
- We wish to predict the weather y based on meteorological conditions x that we observe now.
- We observe the pricing behavior of nearby hotels along with the price of rooms in our hotel, which we represent by x, to predict the response y of whether a customer books a room, y.

In these settings, we obtain a dataset where we associate the response y^n with the observations x^n, which gives us a dataset $(x^n, y^n)_{n=1}^N$.

In the context of sequential decision problems, x may be a decision, such as a choice of drug treatment, the price of a product, the inventory of vaccines, or the choice of movies to display on a user's internet account. In many settings, x may consist of a mixture of controllable elements (such as a drug dosage), and uncontrollable elements (the characteristics of the patient). We can always view machine learning as taking information that is known, x, to predict or estimate something that is unknown, which we call y.

3.1.2 Indexing input x^n and response y^{n+1}

Most work in machine learning uses a batch dataset that we can describe by $(x^n, y^n),\ n = 1, \ldots, N$, where x^n is the input, or independent, variables, and y^n is the associated response (sometimes called a label).

In the context of sequential decisions, we are going to find it more convenient to pick a decision $x^n = X^\pi(S^n)$ based on what we know, given by S^n, and some rule or policy $X^\pi(S^n)$. The decision x^n is based on our history of observations $y^1, \ldots, y^n$ that are used to create our state variable S^n. We then observe y^{n+1}, which gives us an updated state S^{n+1}. Note that we start with $n = 0$, where x^0 is the first decision, which we have to make before seeing any observations.

This style of indexing is consistent with how we index time, where $x_t = S^\pi(S_t)$, after which we observe W_{t+1} which is the information that arrives between t and $t + 1$. It can, however, create unnatural labeling. Imagine a medical setting where we have treated n patients. We use what we know from the first n patients, captured in S^n, to decide the treatment for the $n+1^{st}$ patient, after which we observe the response by the $n+1^{st}$ patient as y^{n+1} (or W^{n+1} if we use our "W" notation). This can seem unnatural. It is important, however, to keep to the principle that if a variable is indexed by n, it depends only on information from the first n observations.

3.1.3 Functions we are learning

The need to approximate functions arises in a number of settings in stochastic optimization. Some of the most important include:

1) Approximating the expectation of a function $\mathbb{E}F(x, W)$ to be maximized, where we assume that we have access to unbiased observations $\hat{F} = F(x, W)$ for a given decision x, which draws on a major branch of statistical learning known as supervised learning.

2) Creating an approximate policy $X^\pi(S|\theta)$. We may fit these functions using one of two ways. We may assume that we have an exogenous source of decisions x that we can use to fit our policy $X^\pi(S|\theta)$ (this would be supervised learning). More frequently, we are tuning the policy to maximize a contribution (or minimize a cost), which is sometimes referred to as a kind of reinforcement learning.

3) Approximating the value of being in a state S, given by $V_t(S_t)$. We wish to find an approximation $\overline{V}_t(S_t)$ that will give us an estimate even when one or more of the elements of S_t is continuous, and/or when S_t is multidimensional. The difference between approximating $\mathbb{E}F(x, W)$ vs. $V_t(S_t)$ is that we can get unbiased observations of $\mathbb{E}F(x, W)$, whereas observations of $V_t(S_t)$ depend on simulations

using suboptimal policies to make decisions over $t+1, t+2, \ldots$, which introduces the bias.

4) Learning any of the underlying models in a dynamic system. These include:

4a) The *transition function* that describes how the system evolves over time, which we will write as $S^M(S_t, x_t, W_{t+1})$ which is used to compute the next state S_{t+1}. This arises in complex environments where the dynamics are not known, such as modeling how much water is retained in a reservoir, which depends in a complex way on rainfall and temperature. We might approximate losses using a parametric model that has to be estimated.

4b) The cost or contribution functions (also known as rewards, gains, losses). This might be unknown if a human is making a decision to maximize an unknown utility, which we might represent as a linear model with parameters to be determined from observed behaviors.

4c) The evolution of exogenous quantities such as wind or prices, where we might model an observation W_{t+1} as a function of the history $W_t, W_{t-1}, W_{t-2}, \ldots$, where we have to fit our model from past observations.

There are three strategies we can use to approach the learning problems in this category:

Exogenous learning - An example of a transition function is a time series model of wind speeds w_t which we might write as

$$w_{t+1} = \bar{\theta}_{t0} w_t + \bar{\theta}_{t1} w_{t-1} + \bar{\theta}_{t2} w_{t-2} + \varepsilon_{t+1},$$

where the input $x_t = (w_t, w_{t-1}, w_{t-2})$ and the response $y_{t+1} = w_{t+1}$ allows us to update our estimate of the parameter vector $\bar{\theta}_t$. The response y_{t+1} comes from outside the system.

Endogenous learning - We may have an estimate of a value function

$$\overline{V}_t^n(S_t|\bar{\theta}_t) = \sum_{f \in \mathcal{F}} \bar{\theta}_{tf}^n \phi_f(S_t).$$

We can then generate a sampled observation $\hat{v}_t^n$ using

$$\hat{v}_t^n = \max_{a_t} \big(C(S_t^n, a_t) + \mathbb{E}_{W_{t+1}}\{\overline{V}_{t+1}(S_{t+1}^n|\bar{\theta}^{n-1})|S_t^n\}\big),$$

to update our parameters $\bar{\theta}_t^n$. The sampled estimate $\hat{v}_t^n$ is created endogenously.

Inverse optimization - Imagine that we are watching a human make decisions (playing a game, managing a robot, dispatching a truck, deciding on a medical treatment) where we do not have a well defined contribution function $C(S_t, x_t)$. Assume that we can come up with a parameterized contribution function $C(S_t, x_t|\theta^{cont})$. We do not have exogenous observations of contributions, and we also do not have endogenous calculations such as $\hat{v}_t$ that provide noisy estimates of the contribution. However, we are given a history of actual decisions x_t. Assume that we are using a policy $X^\pi(S_t|\theta^{cont})$ that depends

on $C(S_t, x_t|\theta^{cont})$ (and therefore depends on θ^{cont}). In this case, the policy $X^\pi(S_t|\theta^{cont})$ plays a role exactly analogous to a statistical model, where we choose θ^{cont} to get the best fit between our policy $X^\pi(S_t|\theta^{cont})$ and the observed decisions. Of course this is a form of exogenous learning, but the decisions only hint at what the contribution function should be.

5) Later we will introduce a class of policies that we call *parametric cost function approximations* where we have to learn two types of functions:

   5a) Parametric modifications of cost functions (for example, a penalty for not serving a demand now but instead holding it for the future). This is not the same as estimating the reward function (see bullet 4) from observed decisions.

   5b) Parametric modifications of constraints (for example, inserting schedule slack into an airline schedule to handle uncertainty in travel times).

   Each of these parametric modifications have to be tuned (which is a form of function estimation) to produce the best results over time.

3.1.4 Sequential learning: from very little data to . . . more data

A common theme in learning problems in the context of sequential decision problems is that the learning has to be done adaptively. This typically means that instead of fitting just one model, we have to transition from models with relatively few parameters (we might call these low-dimensional architectures) to higher-dimensional architectures.

There has been considerable attention to the online updating of parameter estimates. This is particularly easy in the case of linear models, although more challenging with nonlinear models like neural networks. However, there has been much less attention given to the updating of the structure of the model itself in an online setting.

3.1.5 Approximation strategies

Our tour of statistical learning makes a progression through the following classes of approximation strategies:

Lookup tables - Here we estimate a function $f(x)$ where x falls in a discrete region $\mathcal{X}$ given by a set of points $x_1, x_2, \ldots, x_M$. A point x_m could be the characteristics of a person, a type of material, or a movie. Or it could be a point in a discretized, continuous region. As long as x is some discrete element, $f(x)$ is a function where we pick x, and then "look up" its value $f(x)$. Some authors call these "tabular" representations.

In most applications, lookup tables work well in one or two dimensions, then become difficult (but feasible) in three or four dimensions, and then quickly become impractical starting at four or five dimensions. This is the classical "curse of dimensionality." Our presentation focuses on using aggregation, and especially hierarchical aggregation, both to handle the curse of dimensionality, as well as to manage the transition in recursive estimation from initial estimates with very little data, to produce better estimates as more data becomes available.

Parametric models - There are many problems where we can approximate a function using an analytical model in terms of some unknown parameters. These come in two broad categories:

Linear models - The simplest parametric model is linear in the parameters, which we might write

$$f(x|\theta) = \theta_0 + \theta_1\phi_1(x) + \theta_2\phi_2(x) + \dots, \tag{3.1}$$

where $(\phi_f(x))_{f\in\mathcal{F}}$ are *features* that extract possibly useful information from x which could be a vector, or the data describing a movie or ad. Equation (3.1) is called a linear model because it is linear in θ (it may be highly nonlinear in x). Alternatively, we may have a nonlinear model such as

$$f(x|\theta) = e^{\sum_{f\in\mathcal{F}}\theta_f\phi_f(x)}.$$

Parametric models may be low-dimensional (1-100 parameters), or high-dimensional (e.g. several hundred to thousands of parameters).

Nonlinear models - Nonlinear parametric models are usually chosen with a particular form motivated by the problem. Some examples are step functions (useful in asset buying and selling or inventory problems)

$$f(x|\theta) = \begin{cases} \text{-1} & x \le \theta^{low}, \\ 0 & \theta^{low} < x < \theta^{high}, \\ \text{+1} & x \ge \theta^{high}, \end{cases} \tag{3.2}$$

or logistic regression (useful for pricing and recommendation problems)

$$f(x|\theta) = \frac{1}{1 + e^{\theta_0+\theta_1x_1+\dots}}. \tag{3.3}$$

There are models such as neural networks whose primary advantage is that they do not impose any structure, which means they can approximate almost anything (especially the very large instances known as deep neural networks). These models can feature tens of thousands to as many as hundreds of millions of parameters. Not surprisingly, they require very large datasets to determine these parameters.

Nonparametric models - Nonparametric models create estimates by building a structure directly from the data. A simple example is where we estimate $f(x)$ from a weighted combination of nearby observations drawn from a set $(f^n, x^n),\ n = 1, \dots, N$. We can also construct approximations through locally linear approximations.

The three categories of statistical models - lookup tables, parametric and nonparametric - are best thought of as overlapping sets, as illustrated in figure 3.1. For example, neural networks, which we describe below, can be classified as either parametric models (for simpler neural networks) or nonparametric models (for deep neural networks). Other methods are effectively hybrids, such as those based on tree regression which might create a linear approximation (parametric) around specific regions of the input data (the definitions of the regions are lookup table).

Notably missing from this chapter is approximation methods for convex functions. There are many applications where $F(x, W)$ is convex in x. This function is so special that we defer handling this problem class until chapter 5 (and especially chapter 18) when we address stochastic convex (or concave) stochastic optimization problems such as linear programs with random data.

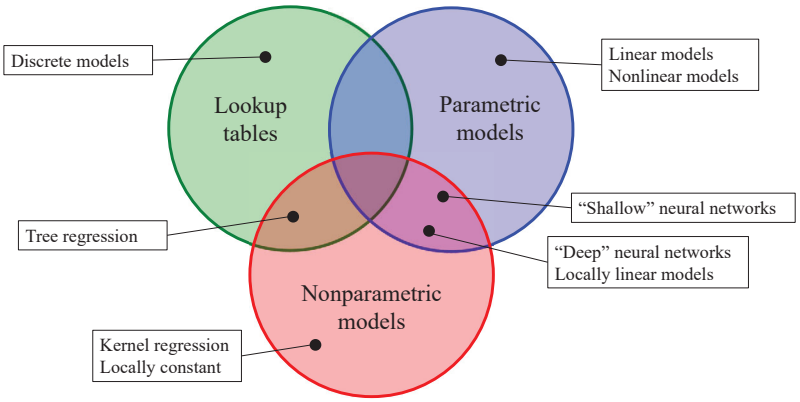


Figure 3.1 Illustration of the overlap between lookup table, parametric and nonparametric statistical models.

We begin our presentation with lookup tables, which are the simplest way to represent a function without assuming any structure. We begin by presenting lookup tables from both frequentist and Bayesian perspectives. In sequential decision problems, we need both belief models. As a general rule, Bayesian models are best when we have access to some prior, and where function evaluations are expensive.

3.1.6 From data analytics to decision analytics

Learning in the context of sequential decision problems can be approached from the perspective of two broad objectives:

- Learning a function - We might want to learn an approximation of a function such as an objective function $\mathbb{E}F(x,W)$, or a value function $V(s)$ or perhaps even the transition function $S^M(s,x,W)$. In these settings, we assume we have a source of observations of our function that may be noisy, and even biased. For example, we might have access to y^{n+1} which is a noisy observation of $\mathbb{E}F(x^n,W^{n+1})$ that we are going to approximate with some function $f(x|\theta)$. If we collect a dataset $(x^0,y^1,x^1,y^2,\ldots,x^{n-1},y^n)$, we would look to find θ that minimizes the error between the observations y and $f(x|\theta)$ using

$$\min_{\theta} \frac{1}{N} \sum_{n=0}^{N-1} \left(y^{n+1} - f(x^n|\theta)\right)^2. \tag{3.4}$$

- Maximizing rewards (or minimizing costs) - We can search for a policy $X^\pi(S|\theta)$ that maximizes a contribution function $C(S,x)$ using

$$\max_{\theta} \mathbb{E}C(S,X^\pi(S)) \approx \frac{1}{N} \sum_{n=0}^{N-1} C(S^n,X^\pi(S^n|\theta)), \tag{3.5}$$

where the states evolve according to a known transition function $S^{n+1} = S^M(S^n,x^n,W^{n+1})$.

The objective function in (3.4) is characteristic of classical machine learning, which we put under the umbrella of "data analytics." There are different ways to express objectives (for example, we might want to use $|y^{n+1} - f(x^n|\theta)|$, but they always involve predictions from a model, $f(x|\theta)$, and observations y.

The objective function in (3.5) is characteristic of optimization problems, which we put under the umbrella of "decision analytics." It assumes some form of pre-defined performance metric (cost, contribution, reward, utility), and notably does not require an exogenous dataset $(y^n)_{n=1}^N$.

3.1.7 Batch vs. online learning

Equation (3.4) (or (3.5)) are the standard problems that arise in batch learning problems, where we use a fixed dataset (possibly a very large one in the modern era of "big data") to fit a model (increasingly high-dimensional models such as neural networks which we introduce below).

While batch learning can arise in stochastic optimization, the most common learning problems are adaptive, which means updating estimates as new data arrives as happens in online applications. Imagine that after n iterations (or samples), we have the sequence

$$(x^0, W^1, y^1, x^1, W^2, y^2, x^2, \ldots, W^n, y^n).$$

Assume that we use this data to obtain an estimate of our function that we call $\bar{F}^n(x)$. Now assume we use this estimate to make a decision x^n, after which we experience exogenous information W^{n+1} and then the response y^{n+1}. We need to use our prior estimate $\bar{F}^n(x)$ along with the new information (W^{n+1}, y^{n+1}) to produce a new estimate $\bar{F}^{n+1}(x)$.

We could, of course, just solve a new batch problem with one more observation. This can be computationally expensive, and it also puts equal weight on the entire history. There are some settings where the more recent observations are more important.

3.2 ADAPTIVE LEARNING USING EXPONENTIAL SMOOTHING

The most common method we will use for adaptive learning is known by various names, but is popularly referred to as *exponential smoothing*. Assume we have a sequence of observations of some quantity, which might be the number of people booking a room, the response of a patient to a particular drug, or the travel time on a path. Let μ be the unknown truth, which could be the average number of people booking a room at a particular price, or the probability a patient responds to a drug, or the average travel time of our path. We want to estimate the average from a sequence of observations.

Let W^n be the n^{th} observation of the quantity we are trying to estimate, and let $\bar{\mu}^n$ be our estimate of the true mean μ after n observations. The most widely used method for computing $\bar{\mu}^{n+1}$ given $\bar{\mu}^n$ and a new observation W^{n+1} is given by

$$\bar{\mu}^{n+1} = (1 - \alpha_n)\bar{\mu}^n + \alpha_n W^{n+1}. \tag{3.6}$$

In chapter 5 we are going to motivate (3.6) using an algorithmic strategy known as stochastic gradient algorithms for solving a specific optimization problem. For now, it is enough to say that this basic equation will arise frequently in a variety of online learning problems.

Not surprisingly, the biggest challenge with this method is choosing α_n. The variable α_n is known variously as a learning rate, smoothing factor or (in this book) a stepsize (we

will see the motivation for the term stepsize in chapter 5). This topic is so rich that we dedicate an entire chapter (chapter 6) to this topic. For now, we can hint at some simple strategies:

- Constant stepsizes - Easily the simplest strategy is one that is actually widely used, which is to simply set $\alpha_n = \bar{\alpha}$ where $\bar{\alpha}$ is a constant chosen in advance.
- Harmonic stepsize - This is an arithmetically declining sequence

$$\alpha_n = \frac{\theta^{step}}{\theta^{step} + n - 1}.$$

  If $\theta^{step} = 1$, this gives us $\alpha_n = 1/n$ (we show in chapter 6 that this produces a simple average). Often this stepsize declines too quickly. Increasing θ^{step} slows the decline in the stepsize which can accelerate learning. It is also possible to have a declining sequence that approaches a limit point.
- In chapter 6 we also introduce a family of adaptive stepsizes that respond to the data.

3.3 LOOKUP TABLES WITH FREQUENTIST UPDATING

The frequentist view is arguably the approach that is most familiar to people with an introductory course in statistics. Assume we are trying to estimate the mean μ of a random variable W which might be the performance of a device or policy. Let W^n be the nth sample observation, such as the sales of a product or the blood sugar reduction achieved by a particular medication. Also let $\bar{\mu}^n$ be our estimate of μ, and $\hat{\sigma}^{2,n}$ be our estimate of the variance of W. We know from elementary statistics that we can write $\bar{\mu}^n$ and $\hat{\sigma}^{2,n}$ using

$$\bar{\mu}^n = \frac{1}{n}\sum_{m=1}^{n} W^m, \tag{3.7}$$

$$\hat{\sigma}^{2,n} = \frac{1}{n-1}\sum_{m=1}^{n}(W^m - \bar{\mu}^n)^2. \tag{3.8}$$

The estimate $\bar{\mu}^n$ is a random variable (in the frequentist view) because it is computed from other random variables, namely $W^1, W^2, \ldots, W^n$. Imagine if we had 100 people each choose a sample of n observations of W. We would obtain 100 different estimates of $\bar{\mu}^n$, reflecting the variation in our observations of W. The best estimate of the variance of the estimator $\bar{\mu}^n$ is given by

$$\bar{\sigma}^{2,n} = \frac{1}{n}\hat{\sigma}^{2,n}.$$

Note that as $n \to \infty$, $\bar{\sigma}^{2,n} \to 0$, but $\hat{\sigma}^{2,n} \to \sigma^2$ where σ^2 is the true variance of W. If σ^2 is known, there would be no need to compute $\hat{\sigma}^{2,n}$ and $\bar{\sigma}^{2,n}$ would be given as above with $\hat{\sigma}^{2,n} = \sigma^2$.

We can write these expressions recursively using

$$\bar{\mu}^n = \left(1 - \frac{1}{n}\right)\bar{\mu}^{n-1} + \frac{1}{n}W^n, \tag{3.9}$$

$$\hat{\sigma}^{2,n} = \frac{n-2}{n-1}\hat{\sigma}^{2,n-1} + \frac{1}{n}(W^n - \bar{\mu}^{n-1})^2, \quad n \geq 2. \tag{3.10}$$

We will often speak of our belief state which captures what we know about the parameters we are trying to estimate. Given our observations, we would write our belief state as

$$B^n = \left(\bar{\mu}^n, \hat{\sigma}^{2,n}\right).$$

Equations (3.9) and (3.10) describe how our belief state evolves over time.

3.4 LOOKUP TABLES WITH BAYESIAN UPDATING

The Bayesian perspective casts a different interpretation on the statistics we compute which is particularly useful in the context of learning when observations are expensive (imagine having to run expensive simulations or field experiments). In the frequentist perspective, we do not start with any knowledge about the system before we have collected any data. It is easy to verify from equations (3.9) and (3.10) that we never use $\bar{\mu}^0$ or $\hat{\sigma}^{2,0}$.

By contrast, in the Bayesian perspective we assume that we begin with a prior distribution of belief about the unknown parameter μ. In other words, any number whose value we do not know is interpreted as a random variable, and the distribution of this random variable represents our belief about how likely μ is to take on certain values. So if μ is the true but unknown mean of W, we might say that while we do not know what this mean is, we think it is normally distributed around θ^0 with standard deviation σ^0.

Thus, the true mean μ is treated as a random variable with a known mean and variance, but we are willing to adjust our estimates of the mean and variance as we collect additional information. If we add a distributional assumption such as the normal distribution, we would say that this is our initial distribution of belief, known generally as the Bayesian prior.

The Bayesian perspective is well suited to problems where we are collecting information about a process where observations are expensive. This might arise when trying to price a book on the internet, or plan an expensive laboratory experiment. In both cases, we can be expected to have some prior information about the right price for a book, or the behavior of an experiment using our knowledge of physics and chemistry.

We note a subtle change in notation from the frequentist perspective, where $\bar{\mu}^n$ was our statistic giving our estimate of μ. In the Bayesian view, we let $\bar{\mu}^n$ be our estimate of the mean of the random variable μ after we have made n observations. It is important to remember that μ is a random variable whose distribution reflects our prior belief about μ. The parameter $\bar{\mu}^0$ is not a random variable. This is our initial estimate of the mean of our prior distribution. After n observations, $\bar{\mu}^n$ is our updated estimate of the mean of the random variable μ (the true mean).

Below we first use some simple expressions from probability to illustrate the effect of collecting information. We then give the Bayesian version of (3.9) and (3.10) for the case of independent beliefs, where observations of one choice do not influence our beliefs about other choices. We follow this discussion by giving the updating equations for correlated beliefs, where an observation of μ_x for alternative x tells us something about $\mu_{x'}$. We round out our presentation by touching on other important types of distributions.

3.4.1 The updating equations for independent beliefs

We begin by assuming (as we do through most of our presentation) that our random variable W is normally distributed. Let σ_W^2 be the variance of W, which captures the noise in our

ability to observe the true value. To simplify the algebra, we define the *precision* of W as

$$\beta^W = \frac{1}{\sigma_W^2}.$$

Precision has an intuitive meaning: smaller variance means that the observations will be closer to the unknown mean, that is, they will be more precise.

Now let $\bar{\mu}^n$ be our estimate of the true mean μ after n observations, and let β^n be the precision of this estimate. If we observe W^{n+1}, $\bar{\mu}^n$ and β^n are updated according to

$$\bar{\mu}^{n+1} = \frac{\beta^n \bar{\mu}^n + \beta^W W^{n+1}}{\beta^n + \beta^W}, \tag{3.11}$$

$$\beta^{n+1} = \beta^n + \beta^W. \tag{3.12}$$

Equations (7.26) and (7.27) are the Bayesian counterparts of (3.9) and (3.10), although we have simplified the problem a bit by assuming that the variance of W is known. The belief state in the Bayesian view (with normally distributed beliefs) is given by the belief state

$$B^n = (\bar{\mu}^n, \beta^n).$$

If our prior distribution of belief about μ is normal, and if the observation W is normal, then the posterior distribution is also normal. It turns out that after a few observations (perhaps five to 10), the distribution of belief about μ will be approximately normal due to the law of large numbers for almost any distribution of W. For the same reason, the posterior distribution is also approximately normal regardless of the distribution of W! So, our our updating equations (7.26) and (7.27) produce the mean and precision of a normal distribution for almost all problems!

3.4.2 Updating for correlated beliefs

We are now going to make the transition that instead of one number μ, we now have a vector $\mu_{x_1}, \mu_{x_2}, \ldots, \mu_{x_M}$ where $\mathcal{X} = \{x_1, \ldots, x_M\}$ is our set we are choosing among. We can think of an element of μ as μ_x, which might be our estimate of a function $\mathbb{E}F(x, W)$ at x. Often, μ_x and $\mu_{x'}$ are correlated, as might happen when x is continuous, and x and x' are close to each other. There are a number of examples that exhibit what we call *correlated beliefs*:

EXAMPLE 3.1

We are interested in finding the price of a product that maximizes total revenue. We believe that the function $R(p)$ that relates revenue to price is continuous. Assume that we set a price p^n and observe revenue R^{n+1} that is higher than we had expected. If we raise our estimate of the function $R(p)$ at the price p^n, our beliefs about the revenue at nearby prices should be higher.

EXAMPLE 3.2

We choose five people for the starting lineup of our basketball team and observe total scoring for one period. We are trying to decide if this group of five people is better

than another lineup that includes three from the same group with two different people. If the scoring of these five people is higher than we had expected, we would probably raise our belief about the other group, since there are three people in common.

EXAMPLE 3.3

A physician is trying to treat diabetes using a treatment of three drugs, where she observes the drop in blood sugar from a course of a particular treatment. If one treatment produces a better-than-expected response, this would also increase our belief of the response from other treatments that have one or two drugs in common.

EXAMPLE 3.4

We are trying to find the highest concentration of a virus in the population. If the concentration of one group of people is higher than expected, our belief about other groups that are close (either geographically, or due to other relationships) would also be higher.

Correlated beliefs are a particularly powerful device in learning functions, allowing us to generalize the results of a single observation to other alternatives that we have not directly measured.

Let $\bar{\mu}_x^n$ be our belief about alternative x after n measurements. Now let

$Cov^n(\mu_x, \mu_y)$ = the covariance in our belief about μ_x and μ_y given the first n observations.

We let Σ^n be the covariance matrix, with element $\Sigma^n_{xy} = Cov^n(\mu_x, \mu_y)$. Just as we defined the precision β_x^n to be the reciprocal of the variance, we are going to define the precision matrix M^n to be

$$M^n = (\Sigma^n)^{-1}.$$

Let e_x be a column vector of zeroes with a 1 for element x, and as before we let W^{n+1} be the (scalar) observation when we decide to measure alternative x. We could label W^{n+1} as W_x^{n+1} to make the dependence on the alternative more explicit. For this discussion, we are going to use the notation that we choose to measure x^n and the resulting observation is W^{n+1}.

If we choose to measure x^n, we can also interpret the observation as a column vector given by $W^{n+1}e_{x^n}$. Keeping in mind that $\bar{\mu}^n$ is a column vector of our beliefs about the expectation of μ, the Bayesian equation for updating this vector in the presence of correlated beliefs is given by

$$\bar{\mu}^{n+1} = (M^{n+1})^{-1}\left(M^n\bar{\mu}^n + \beta^W W^{n+1}e_{x^n}\right), \tag{3.13}$$

where M^{n+1} is given by

$$M^{n+1} = (M^n + \beta^W e_{x^n}(e_{x^n})^T). \tag{3.14}$$

Note that $e_x(e_x)^T$ is a matrix of zeroes with a one in row x, column x, whereas β^W is a scalar giving the precision of our measurement W.

It is possible to perform these updates without having to deal with the inverse of the covariance matrix. This is done using a result known as the Sherman-Morrison formula.

If A is an invertible matrix (such as Σ^n) and u is a column vector (such as e_x), the Sherman-Morrison formula is

$$[A + uu^T]^{-1} = A^{-1} - \frac{A^{-1}uu^TA^{-1}}{1 + u^TA^{-1}u}. \tag{3.15}$$

See section 3.14.2 for the derivation of this formula.

Using the Sherman-Morrison formula, and letting $x = x^n$, we can rewrite the updating equations as

$$\bar{\mu}^{n+1}(x) = \bar{\mu}^n + \frac{W^{n+1} - \bar{\mu}_x^n}{\sigma_W^2 + \Sigma_{xx}^n}\Sigma^n e_x, \tag{3.16}$$

$$\Sigma^{n+1}(x) = \Sigma^n - \frac{\Sigma^n e_x (e_x)^T \Sigma^n}{\sigma_W^2 + \Sigma_{xx}^n}, \tag{3.17}$$

where we express the dependence of $\bar{\mu}^{n+1}(x)$ and $\Sigma^{n+1}(x)$ on the alternative x which we have chosen to measure.

To illustrate, assume that we have three alternatives with mean vector

$$\bar{\mu}^n = \begin{bmatrix} 20 \\ 16 \\ 22 \end{bmatrix}.$$

Assume that $\sigma_W^2 = 9$ and that our covariance matrix Σ^n is given by

$$\Sigma^n = \begin{bmatrix} 12 & 6 & 3 \\ 6 & 7 & 4 \\ 3 & 4 & 15 \end{bmatrix}.$$

Assume that we choose to measure $x = 3$ and observe $W^{n+1} = W_3^{n+1} = 19$. Applying equation (3.16), we update the means of our beliefs using

$$\begin{aligned} \bar{\mu}^{n+1}(3) &= \begin{bmatrix} 20 \\ 16 \\ 22 \end{bmatrix} + \frac{19 - 22}{9 + 15} \begin{bmatrix} 12 & 6 & 3 \\ 6 & 7 & 4 \\ 3 & 4 & 15 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \\ &= \begin{bmatrix} 20 \\ 16 \\ 22 \end{bmatrix} + \frac{-3}{24} \begin{bmatrix} 3 \\ 4 \\ 15 \end{bmatrix} \\ &= \begin{bmatrix} 19.625 \\ 15.500 \\ 20.125 \end{bmatrix}. \end{aligned}$$

The update of the covariance matrix is computed using

$$\begin{aligned}
\Sigma^{n+1}(3) &= \begin{bmatrix} 12 & 6 & 3 \\ 6 & 7 & 4 \\ 3 & 4 & 15 \end{bmatrix} - \frac{\begin{bmatrix} 12 & 6 & 3 \\ 6 & 7 & 4 \\ 3 & 4 & 15 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} [0\ 0\ 1] \begin{bmatrix} 12 & 6 & 3 \\ 6 & 7 & 4 \\ 3 & 4 & 15 \end{bmatrix}}{9+15} \\
&= \begin{bmatrix} 12 & 6 & 3 \\ 6 & 7 & 4 \\ 3 & 4 & 15 \end{bmatrix} - \frac{1}{24} \begin{bmatrix} 3 \\ 4 \\ 15 \end{bmatrix} [3\ 4\ 15] \\
&= \begin{bmatrix} 12 & 6 & 3 \\ 6 & 7 & 4 \\ 3 & 4 & 15 \end{bmatrix} - \frac{1}{24} \begin{bmatrix} 9 & 12 & 45 \\ 12 & 16 & 60 \\ 45 & 60 & 225 \end{bmatrix} \\
&= \begin{bmatrix} 12 & 6 & 3 \\ 6 & 7 & 4 \\ 3 & 4 & 15 \end{bmatrix} - \begin{bmatrix} 0.375 & 0.500 & 1.875 \\ 0.500 & 0.667 & 2.500 \\ 1.875 & 2.500 & 9.375 \end{bmatrix} \\
&= \begin{bmatrix} 11.625 & 5.500 & 1.125 \\ 5.500 & 6.333 & 1.500 \\ 1.125 & 1.500 & 5.625 \end{bmatrix}.
\end{aligned}$$

These calculations are fairly easy, which means we can execute them even if we have thousands of alternatives. However, the method starts to become impractical if the number of alternatives is in the range of 10^5 or more, which arises when we consider problems where an alternative x is itself a multidimensional vector.

3.4.3 Gaussian process regression

A common strategy for approximating continuous functions is to discretize them, and then capture continuity by noting that the value of nearby points will be correlated, simply because of continuity. This is known as *Gaussian process regression*.

Assume that we have an unknown function $f(x)$ that is continuous in x which for the moment we will assume is a scalar that is discretized into the values $(x_1, x_2, \ldots, x_M)$. Let $\bar{\mu}^n(x)$ be our estimate of $f(x)$ over our discrete set. Let $\mu(x)$ be the true value of $f(x)$ which, with our Bayesian hat on, we will interpret as a normally distributed random variable with mean $\bar{\mu}_x^0$ and variance $(\sigma_x^0)^2$ (this is our prior). We will further assume that μ_x and $\mu_{x'}$ are correlated with covariance

$$Cov(\mu_x, \mu_{x'}) = (\sigma^0)^2 e^{\alpha \|x - x'\|}, \tag{3.18}$$

where $\|x - x'\|$ is some distance metric such as $|x - x'|$ or $(x - x')^2$ (if x is a scalar) or $\sqrt{\sum_{i=1}^{I}(x_i - x_i')^2}$ if x is a vector. If $x = x'$ then we just pick up the variance in our belief about μ_x. The parameter α captures the degree to which x and x' are related as they get further apart.

Figure 3.2 illustrates a series of curves randomly generated from a belief model using the covariance function given in equation (3.18) for different values of α. Smaller values of α produce smoother curves with fewer undulations, because a smaller α translates to a higher covariance between more distant values of x and x'. As α increases, the covariance drops off and two different points on the curve become more independent.

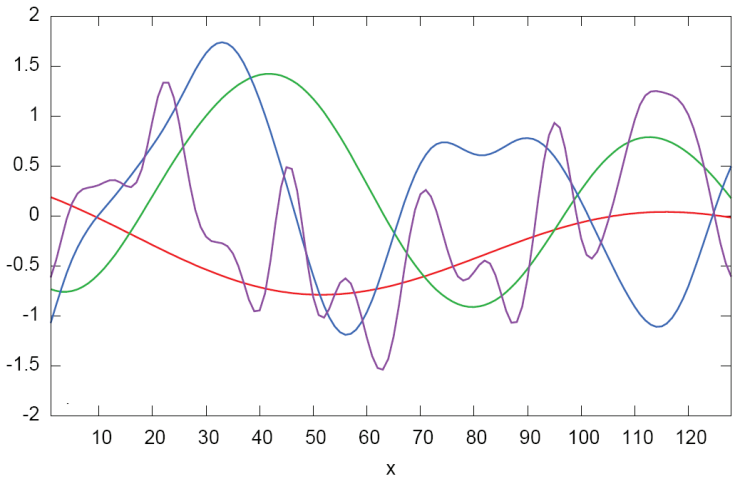


Figure 3.2 Illustration of a series of functions generated using Gaussian process regression (correlated beliefs) for different values of α.

Gaussian process regression (often shortened to just "GPR") is a powerful approach for approximating smooth functions that are continuous but otherwise have no specific structure. We present GPR here as a generalization of lookup table belief models, but it can also be characterized as a form of nonparametric statistics which we discuss below. In chapter 7 we will show how using GPR as a belief model can dramatically accelerate optimizing functions of continuous parameters such as drug dosages for medical applications, or the choice of temperature, pressure and concentration in a laboratory science application.

3.5 COMPUTING BIAS AND VARIANCE*

A powerful strategy for estimating functions of multidimensional vectors using lookup tables is hierarchical aggregation, where we estimate the function at different levels of aggregation. To lay the foundation for this approach, we are going to need some basic results on bias and variance in statistical estimation.

Assume we are trying to estimate a true but unknown parameter μ which we can observe, but we have to deal with both bias β and noise ε, which we write as

$$\hat{\mu}^n = \mu + \beta + \varepsilon^n. \tag{3.19}$$

Both μ and β are unknown, but we are going to assume that we have some way to make a noisy estimate of the bias that we are going to call $\hat{\beta}^n$. Later we are going to provide examples of how to get estimates of β.

Now let $\bar{\mu}^n$ be our estimate of μ after n observations. We will use the following recursive formula for $\bar{\mu}^n$

$$\bar{\mu}^n = (1 - \alpha_{n-1})\bar{\mu}^{n-1} + \alpha_{n-1}\hat{\mu}^n.$$

We are interested in estimating the variance of $\bar{\mu}^n$ and its bias $\bar{\beta}^n$. We start by computing the variance of $\bar{\mu}^n$. We assume that our observations of μ can be represented using equation

(3.19), where $\mathbb{E}\varepsilon^n = 0$ and $Var[\varepsilon^n] = \sigma^2$. With this model, we can compute the variance of $\bar{\mu}^n$ using

$$Var[\bar{\mu}^n] = \lambda^n \sigma^2, \tag{3.20}$$

where λ^n (this is λ at iteration n, not raised to the n^{th} power) can be computed from the simple recursion

$$\lambda^n = \begin{cases} \alpha_{n-1}^2, & n = 1, \\ (1-\alpha_{n-1})^2 \lambda^{n-1} + \alpha_{n-1}^2, & n > 1. \end{cases} \tag{3.21}$$

To see this, we start with $n = 1$. For a given (deterministic) initial estimate $\bar{\mu}^0$, we first observe that the variance of $\bar{\mu}^1$ is given by

$$\begin{aligned} Var[\bar{\mu}^1] &= Var[(1-\alpha_0)\bar{\mu}^0 + \alpha_0 \hat{\mu}^1] \\ &= \alpha_0^2 Var[\hat{\mu}^1] \\ &= \alpha_0^2 \sigma^2. \end{aligned}$$

For $\bar{\mu}^n$ for $n > 1$, we use a proof by induction. Assume that $Var[\bar{\mu}^{n-1}] = \lambda^{n-1}\sigma^2$. Then, since $\bar{\mu}^{n-1}$ and $\hat{\mu}^n$ are independent, we find

$$\begin{aligned} Var[\bar{\mu}^n] &= Var\left[(1-\alpha_{n-1})\bar{\mu}^{n-1} + \alpha_{n-1}\hat{\mu}^n\right] \\ &= (1-\alpha_{n-1})^2 Var\left[\bar{\mu}^{n-1}\right] + \alpha_{n-1}^2 Var[\hat{\mu}^n] \\ &= (1-\alpha_{n-1})^2 \lambda^{n-1}\sigma^2 + \alpha_{n-1}^2 \sigma^2 \qquad (3.22) \\ &= \lambda^n \sigma^2. \qquad (3.23) \end{aligned}$$

Equation (3.22) is true by assumption (in our induction proof), while equation (3.23) establishes the recursion in equation (3.21). This gives us the variance, assuming of course that σ^2 is known.

Using our assumption that we have access to a noisy estimate of the bias given by β^n, we can compute the mean-squared error using

$$\mathbb{E}\left[\left(\bar{\mu}^{n-1} - \bar{\mu}^n\right)^2\right] = \lambda^{n-1}\sigma^2 + \beta^{2,n}. \tag{3.24}$$

See exercise 3.11 to prove this. This formula gives the variance around the known mean, $\bar{\mu}^n$. For our purposes, it is also useful to have the variance around the observations $\hat{\mu}^n$. Let

$$\nu^n = \mathbb{E}\left[\left(\bar{\mu}^{n-1} - \hat{\mu}^n\right)^2\right]$$

be the mean squared error (including noise and bias) between the current estimate $\bar{\mu}^{n-1}$ and the observation $\hat{\mu}^n$. It is possible to show that (see exercise 3.12)

$$\nu^n = (1+\lambda^{n-1})\sigma^2 + \beta^{2,n}, \tag{3.25}$$

where λ^n is computed using (3.21).

In practice, we do not know σ^2, and we certainly do not know the bias β. As a result, we have to estimate both parameters from our data. We begin by providing an estimate of the bias using

$$\bar{\beta}^n = (1-\eta_{n-1})\bar{\beta}^{n-1} + \eta_{n-1}\beta^n,$$

where η_{n-1} is a (typically simple) stepsize rule used for estimating the bias and variance. As a general rule, we should pick a stepsize for η_{n-1} which produces larger stepsizes than α_{n-1} because we are more interested in tracking the true signal than producing an estimate with a low variance. We have found that a constant stepsize such as .10 works quite well on a wide range of problems, but if precise convergence is needed, it is necessary to use a rule where the stepsize goes to zero such as the harmonic stepsize rule (equation (6.15)).

To estimate the variance, we begin by finding an estimate of the total variation ν^n. Let $\bar{\nu}^n$ be the estimate of the total variance which we might compute using

$$\bar{\nu}^n = (1-\eta_{n-1})\bar{\nu}^{n-1} + \eta_{n-1}(\bar{\mu}^{n-1} - \hat{\mu}^n)^2.$$

Using $\bar{\nu}^n$ as our estimate of the total variance, we can compute an estimate of σ^2 using

$$\bar{\sigma}^{2,n} = \frac{\bar{\nu}^n - \bar{\beta}^{2,n}}{1+\lambda^{n-1}}.$$

We can use $(\bar{\sigma}^n)^2$ in equation (3.20) to obtain an estimate of the variance of $\bar{\mu}^n$.

If we are doing true averaging (as would occur if we use a stepsize of $1/n$), we can get a more precise estimate of the variance for small samples by using the recursive form of the small sample formula for the variance

$$\hat{\sigma}^{2,n} = \frac{n-2}{n-1}\hat{\sigma}^{2,n-1} + \frac{1}{n}(\bar{\mu}^{n-1} - \hat{\mu}^n)^2. \tag{3.26}$$

The quantity $\hat{\sigma}^{2,n}$ is an estimate of the variance of $\hat{\mu}^n$. The variance of our estimate $\bar{\mu}^n$ is computed using

$$\bar{\sigma}^{2,n} = \frac{1}{n}\hat{\sigma}^{2,n}.$$

We are going to draw on these results in two settings, which are both distinguished by how estimates of the bias β^n are computed:

- Hierarchical aggregation - We are going to estimate a function at different levels of aggregation. We can assume that the estimate of the function at the most disaggregate level is noisy but unbiased, and then let the difference between the function at some level of aggregation and the function at the most disaggregate level as an estimate of the bias.
- Transient functions - Later, we are going to use these results to approximate value functions. It is the nature of algorithms for estimating value functions that the underlying process varies over time (we see this most clearly in chapter 14). In this setting, we are making observations from a truth that is changing over time, which introduces a bias.

3.6 LOOKUP TABLES AND AGGREGATION*

Lookup table representations are the simplest and most general way to represent a function. If we are trying to model a function $f(x) = \mathbb{E}F(x,W)$, or perhaps a value function $V_t(S_t)$, assume that our function is defined over a discrete set of values $x_1, \ldots, x_M$ (or discrete states $\mathcal{S} = \{1, 2, \ldots, |\mathcal{S}|\}$). We wish to use observations of our function, whether they be

$f^n = F(x^n, W^{n+1})$ (or $\hat{v}_t^n$, derived from simulations of the value of being in a state S_t), to create an estimate $\bar{F}^{n+1}$ (or $\overline{V}_t^{n+1}(S_t)$).

The problem with lookup table representations is that if our variable x (or state S) is a vector, then the number of possible values grows exponentially with the number of dimensions. This is the classic curse of dimensionality. One strategy for overcoming the curse of dimensionality is to use aggregation, but picking a single level of aggregation is generally never satisfactory. In particular, we typically have to start with no data, and steadily build up an estimate of a function.

We can accomplish this transition from little to no data, to increasing numbers of observations, by using hierarchical aggregation. Instead of picking a single level of aggregation, we work with a family of aggregations which are hierarchically structured.

3.6.1 Hierarchical aggregation

Lookup table representations of functions often represent the first strategy we consider because it does not require that we assume any structural form. The problem is that lookup tables suffer from the curse of dimensionality. A powerful strategy that makes it possible to extend lookup tables is the use of hierarchical aggregation. Rather than simply aggregating a state space into a smaller space, we pose a family of aggregations, and then combine these based on the statistics of our estimates at each level of aggregation. This is not a panacea (nothing is), and should not be viewed as a method that "solves" the curse of dimensionality, but it does represent a powerful addition to our toolbox of approximation strategies. As we will see, this is particularly useful when being applied in the context of sequential decision problems.

We can illustrate hierarchical aggregation using our nomadic trucker example that we first introduced in section 2.3.4. In this setting, we are managing a truck driver who is picking up and dropping off loads (imagine taxicabs for freight), where the driver has to choose loads based on both how much money he will make moving the load, and the value of landing at the destination of the load. Complicating the problem is that the driver is described by a multidimensional attribute vector $a = (a_1, a_2, \ldots, a_d)$ which includes attributes such as the location of a truck (which means location in a region), his equipment type, and his home location (again, a region).

If our nomadic trucker is described by the state vector $S_t = a_t$ which we act on with an action x_t (moving one of the available loads), the transition function $S_{t+1} = S^M(S_t, x_t, W_{t+1})$ may represent the state vector at a high level of detail (some values may be continuous). But the decision problem

$$\max_{x_t \in \mathcal{X}} \left(C(S_t, x_t) + \mathbb{E}\{\overline{V}_{t+1}(G(S_{t+1}))|S_t\} \right) \tag{3.27}$$

uses a value function $\overline{V}_{t+1}(G(S_{t+1}))$, where $G(\cdot)$ is an aggregation function that maps the original (and very detailed) state S into something much simpler. The aggregation function G may ignore a dimension, discretize it, or use any of a variety of ways to reduce the number of possible values of a state vector. This also reduces the number of parameters we have to estimate. In what follows, we drop the explicit reference of the aggregation function G and simply use $\overline{V}_{t+1}(S_{t+1})$. The aggregation is implicit in the value function approximation.

Some major characteristics that can be used for aggregation are:

- Spatial - A transportation company is interested in estimating the value of truck drivers at a particular location. Locations may be calculated at the level of a five-

digit zip code (there are about 55,000 in the United States), three-digit zip code (about 1,000), or the state level (48 contiguous states).

- Temporal - A bank may be interested in estimating the value of holding an asset at a point in time. Time may be measured by the day, week, month, or quarter.

- Continuous parameters - The state of an aircraft may be its fuel level; the state of a traveling salesman may be how long he has been away from home; the state of a water reservoir may be the depth of the water; the state of the cash reserve of a mutual fund is the amount of cash on hand at the end of the day. These are examples of systems with at least one dimension of the state that is at least approximately continuous. The variables may all be discretized into intervals of varying lengths.

- Hierarchical classification - A portfolio problem may need to estimate the value of investing money in the stock of a particular company. It may be useful to aggregate companies by industry segment (for example, a particular company might be in the chemical industry, and it might be further aggregated based on whether it is viewed as a domestic or multinational company). Similarly, problems of managing large inventories of parts (for cars, for example) may benefit by organizing parts into part families (transmission parts, engine parts, dashboard parts).

The examples below provide additional illustrations.

EXAMPLE 3.5

The state of a jet aircraft may be characterized by multiple attributes which include spatial and temporal dimensions (location and flying time since the last maintenance check), as well other attributes. A continuous parameter could be the fuel level, an attribute that lends itself to hierarchical aggregation might be the specific type of aircraft. We can reduce the number of states (attributes) of this resource by aggregating each dimension into a smaller number of potential outcomes.

EXAMPLE 3.6

The state of a portfolio might consist of the number of bonds which are characterized by the source of the bond (a company, a municipality or the federal government), the maturity (six months, 12 months, 24 months), when it was purchased, and its rating by bond agencies. Companies can be aggregated up by industry segment. Bonds can be further aggregated by their bond rating.

EXAMPLE 3.7

Blood stored in blood banks can be characterized by type, the source (which might indicate risks for diseases), age (it can be stored for up to 42 days), and the current location where it is being stored. A national blood management agency might want to aggregate the state space by ignoring the source (ignoring a dimension is a form of aggregation), discretizing the age from days into weeks, and aggregating locations into more aggregate regions.

Aggregation level	Location	Fleet type	Domicile	Size of state space
0	Sub-region	Fleet	Region	$400 \times 5 \times 100 = 200,000$
1	Region	Fleet	Region	$100 \times 5 \times 100 = 50,000$
2	Region	Fleet	Zone	$100 \times 5 \times 10 = 5,000$
3	Region	Fleet	-	$100 \times 5 \times 1 = 500$
4	Zone	-	-	$10 \times 1 \times 1 = 10$

Table 3.1 Examples of aggregations of the state space for the nomadic trucker problem. '-' indicates that the particular dimension is ignored.

EXAMPLE 3.8

The value of an asset is determined by its current price, which is continuous. We can estimate the asset using a price discretized to the nearest dollar.

There are many applications where aggregation is naturally hierarchical. For example, in our nomadic trucker problem we might want to estimate the value of a truck based on three attributes: location, home domicile, and fleet type. The first two represent geographical locations, which can be represented (for this example) at three levels of aggregation: 400 sub-regions, 100 regions, and 10 zones. Table 3.1 illustrates five levels of aggregation that might be used. In this example, each higher level can be represented as an aggregation of the previous level.

Aggregation is also useful for continuous variables. Assume that our state variable is the amount of cash we have on hand, a number that might be as large as $10 million dollars. We might discretize our state space in units of $1 million, $100 thousand, $10 thousand, $1,000, $100, and $10. This discretization produces a natural hierarchy since 10 segments at one level of aggregation naturally group into one segment at the next level of aggregation.

Hierarchical aggregation is a natural way to generate a family of estimates, but in most cases there is no reason to assume that the structure is hierarchical. In fact, we may even use overlapping aggregations (sometimes known as "soft" aggregation), where the same state s aggregates into multiple elements in $\mathcal{S}^g$. For example, assume that s represents an (x, y) coordinate in a continuous space which has been discretized into the set of points $(x_i, y_i)_{i \in \mathcal{I}}$. Further assume that we have a distance metric $\rho((x, y), (x_i, y_i))$ that measures the distance from any point (x, y) to every aggregated point (x_i, y_i) , $i \in \mathcal{I}$. We might use an observation at the point (x, y) to update estimates at each (x_i, y_i) with a weight that declines with $\rho((x, y), (x_i, y_i))$.

3.6.2 Estimates of different levels of aggregation

Assume we are trying to approximate a function $f(x)$, $x \in \mathcal{X}$. We begin by defining a family of aggregation functions

$$G^g : \mathcal{X} \rightarrow \mathcal{X}^{(g)}.$$

$\mathcal{X}^{(g)}$ represents the g^{th} level of aggregation of the domain $\mathcal{X}$. Let

$\mathcal{G}$ = the set of indices corresponding to the levels of aggregation.

In this section, we assume we have a single aggregation function G that maps the disaggregate state $x \in \mathcal{X} = \mathcal{X}^{(0)}$ into an aggregated space $\mathcal{X}^{(g)}$. In section 3.6.3, we let $g \in \mathcal{G} = \{0, 1, 2, \ldots\}$ and we work with all levels of aggregation at the same time.

To begin our study of aggregation, we first need to characterize how we sample values x at the disaggregate level. For this discussion, we assume we have two exogenous processes: At iteration n, the first process chooses a value to sample (which we denote by x^n), and the second produces an observation of the value of being in state

$$\hat{f}^n(x^n) = f(x^n) + \varepsilon^n.$$

Later, we are going to assume that x^n is determined by some policy, but for now, we can treat this as purely exogenous.

We need to characterize the errors that arise in our estimate of the function. Let

$f_x^{(g)}$ = the true estimate of the g^{th} aggregation of the original function $f(x)$.

We assume that $f^{(0)}(x) = f(x)$, which means that the zeroth level of aggregation is the true function.

Let

$\bar{f}_x^{(g,n)}$ = the estimate of the value of $f(x)$ at the g^{th} level of aggregation after n observations.

Throughout our discussion, a bar over a variable means it was computed from sample observations. A hat means the variable was an exogenous observation.

When we are working at the most disaggregate level ($g = 0$), the state s that we measure is the observed state $s = \hat{s}^n$. For $g > 0$, the subscript x in $\bar{f}_x^{(g,n)}$ refers to $G^g(x^n)$, or the g^{th} level of aggregation of $f(x)$ at $x = x^n$. Given an observation $(x^n, \hat{f}^n(x^n))$, we would update our estimate of the $f^{(g)}(x)$ using

$$\bar{f}_x^{(g,n)} = (1 - \alpha_{x,n-1}^{(g)})\bar{f}_x^{(g,n-1)} + \alpha_{x,n-1}^{(g)}\hat{f}^n(x).$$

Here, we have written the stepsize $\alpha_{x,n-1}^{(g)}$ to explicitly represent the dependence on the decision x and level of aggregation. Implicit is that this is also a function of the number of times that we have updated $\bar{f}_x^{(g,n)}$ by iteration n, rather than a function of n itself.

To illustrate, imagine that our nomadic trucker is described by the vector $x = (\text{Loc}, \text{Equip}, \text{Home}, \text{DOThrs}, \text{Days})$, where "Loc" is location, "Equip" denotes the type of trailer (long, short, refrigerated), "Home" is the location of where he lives, "DOThrs" is a vector giving the number of hours the driver has worked on each of the last eight days, and "Days" is the number of days the driver has been away from home. We are going to estimate the value $f(x)$ for different levels of aggregation of x, where we aggregate purely by ignoring certain dimensions of s. We start with our original disaggregate observation $\hat{f}(x)$, which we are going to write as

$$\hat{f}\begin{pmatrix} \text{Loc} \\ \text{Equip} \\ \text{Home} \\ \text{DOThrs} \\ \text{Days} \end{pmatrix} = f(x) + \varepsilon.$$

We now wish to use this estimate of the value of a driver with attribute x to produce value functions at different levels of aggregation. We can do this by simply smoothing this disaggregate estimate in with estimates at different levels of aggregation, as in

$$\bar{f}^{(1,n)}\begin{pmatrix}\text{Loc}\\ \text{Equip}\\ \text{Home}\end{pmatrix} = (1-\alpha^{(1)}_{x,n-1})\bar{f}^{(1,n-1)}\begin{pmatrix}\text{Loc}\\ \text{Equip}\\ \text{Home}\end{pmatrix} + \alpha^{(1)}_{x,n-1}\hat{f}\begin{pmatrix}\text{Loc}\\ \text{Equip}\\ \text{Home}\\ \text{DOThrs}\\ \text{Days}\end{pmatrix},$$

$$\bar{f}^{(2,n)}\begin{pmatrix}\text{Loc}\\ \text{Equip}\end{pmatrix} = (1-\alpha^{(2)}_{x,n-1})\bar{f}^{(2,n-1)}\begin{pmatrix}\text{Loc}\\ \text{Equip}\end{pmatrix} + \alpha^{(2)}_{x,n-1}\hat{f}\begin{pmatrix}\text{Loc}\\ \text{Equip}\\ \text{Home}\\ \text{DOThrs}\\ \text{Days}\end{pmatrix},$$

$$\bar{f}^{(3,n)}\begin{pmatrix}\text{Loc}\end{pmatrix} = (1-\alpha^{(3)}_{x,n-1})\bar{f}^{(3,n-1)}\begin{pmatrix}\text{Loc}\end{pmatrix} + \alpha^{(3)}_{x,n-1}\hat{v}\begin{pmatrix}\text{Loc}\\ \text{Equip}\\ \text{Home}\\ \text{DOThrs}\\ \text{Days}\end{pmatrix}.$$

In the first equation, we are smoothing the value of a driver based on a five-dimensional state vector, given by x, in with an approximation indexed by a three-dimensional state vector. The second equation does the same using value function approximation indexed by a two-dimensional state vector, while the third equation does the same with a one-dimensional state vector. It is very important to keep in mind that the stepsize must reflect the number of times a state has been updated.

We need to estimate the variance of $\bar{f}^{(g,n)}_x$. Let

$(s^2_x)^{(g,n)}$ = The estimate of the variance of observations made of the function at x, using data from aggregation level g, after n observations.

$(s^2_x)^{(g,n)}$ is the estimate of the variance of the observations $\hat{f}$ when we observe the function at $x = x^n$ which aggregates to x (that is, $G^g(x^n) = x$). We are really interested in the variance of our estimate of the mean, $\bar{f}^{(g,n)}_x$. In section 3.5, we showed that

$$\begin{aligned}(\bar{\sigma}^2_x)^{(g,n)} &= \mathit{Var}[\bar{f}^{(g,n)}_x]\\ &= \lambda^{(g,n)}_x (s^2_x)^{(g,n)},\end{aligned} \tag{3.28}$$

where $(s^2_x)^{(g,n)}$ is an estimate of the variance of the observations $\hat{f}^n$ at the g^{th} level of aggregation (computed below), and $\lambda^{(g,n)}_s$ can be computed from the recursion

$$\lambda^{(g,n)}_x = \begin{cases}(\alpha^{(g)}_{x,n-1})^2, & n=1,\\ (1-\alpha^{(g)}_{x,n-1})^2\lambda^{(g,n-1)}_x + (\alpha^{(g)}_{x,n-1})^2, & n>1.\end{cases}$$

Note that if the stepsize $\alpha^{(g)}_{x,n-1}$ goes to zero, then $\lambda^{(g,n)}_x$ will also go to zero, as will $(\bar{\sigma}^2_x)^{(g,n)}$. We now need to compute $(s^2_x)^{(g,n)}$ which is the estimate of the variance of observations $\hat{f}^n$ at points x^n for which $G^g(x^n) = x$ (the observations of states that aggregate up to x). Let $\bar{\nu}^{(g,n)}_x$ be the total variation, given by

$$\bar{\nu}^{(g,n)}_x = (1-\eta_{n-1})\bar{\nu}^{(g,n-1)}_x + \eta_{n-1}(\bar{f}^{(g,n-1)}_x - \hat{f}^n_x)^2,$$

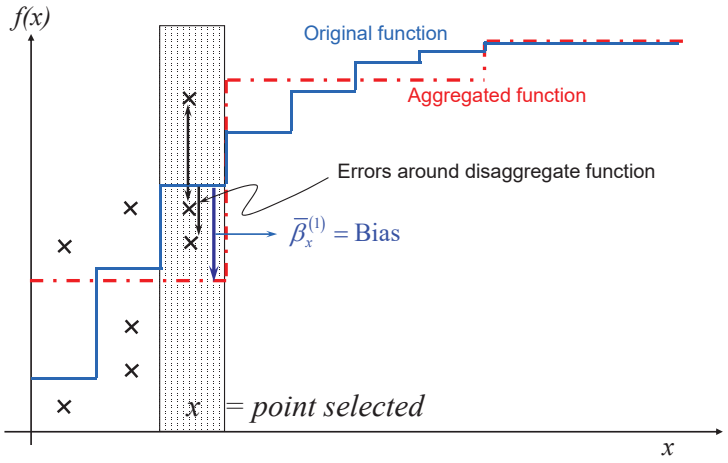


Figure 3.3 Illustration of a disaggregate function, an aggregated approximation and a set of samples. For a particular state s, we show the estimate and the bias.

where η_{n-1} follows some stepsize rule (which may be just a constant). We refer to $\bar{\nu}_x^{(g,n)}$ as the total variation because it captures deviations that arise both due to measurement noise (the randomness when we compute $\hat{f}^n(x)$) and bias (since $\bar{f}_x^{(g,n-1)}$ is a biased estimate of the mean of $\hat{f}^n(x)$).

We finally need an estimate of the bias from aggregation which we find by computing

$$\bar{\beta}_x^{(g,n)} = \bar{f}_x^{(g,n)} - \bar{f}_x^{(0,n)}. \tag{3.29}$$

We can separate out the effect of bias to obtain an estimate of the variance of the error using

$$(s_x^2)^{(g,n)} = \frac{\bar{\nu}_x^{(g,n)} - (\bar{\beta}_x^{(g,n)})^2}{1+\lambda^{n-1}}. \tag{3.30}$$

In the next section, we put the estimate of aggregation bias, $\bar{\beta}_x^{(g,n)}$, to work.

The relationships are illustrated in figure 3.3, which shows a simple function defined over a single, continuous state (for example, the price of an asset). If we select a particular state s, we find we have only two observations for that state, versus seven for that section of the function. If we use an aggregate approximation, we would produce a single number over that range of the function, creating a bias between the true function and the aggregated estimate. As the illustration shows, the size of the bias depends on the shape of the function in that region.

One method for choosing the best level of aggregation is to choose the level that minimizes $(\bar{\sigma}_s^2)^{(g,n)} + (\bar{\beta}_s^{(g,n)})^2$, which captures both bias and variance. In the next section, we use the bias and variance to develop a method that uses estimates at all levels of aggregation at the same time.

3.6.3 Combining multiple levels of aggregation

Rather than try to pick the best level of aggregation, it is intuitively appealing to use a weighted sum of estimates at different levels of aggregation. The simplest strategy is to use

$$\bar{f}_x^n = \sum_{g \in \mathcal{G}} w^{(g)} \bar{f}_x^{(g)}, \tag{3.31}$$

where $w^{(g)}$ is the weight applied to the g^{th} level of aggregation. We would expect the weights to be positive and sum to one, but we can also view these simply as coefficients in a regression function. In such a setting, we would normally write the regression as

$$\bar{F}(x|\theta) = \theta_0 + \sum_{g \in \mathcal{G}} \theta_g \bar{f}_x^{(g)},$$

(see section 3.7 for a presentation of linear models). The problem with this strategy is that the weight does not depend on the value of x. Intuitively, it makes sense to put a higher weight on points x which have more observations, or where the estimated variance is lower. This behavior is lost if the weight does not depend on x.

In practice, we will generally observe some states much more frequently than others, suggesting that the weights should depend on x. To accomplish this, we need to use

$$\bar{f}_x^n = \sum_{g \in \mathcal{G}} w_x^{(g)} \bar{f}_x^{(g,n)}.$$

Now the weight depends on the point being estimated, allowing us to put a higher weight on the disaggregate estimates when we have a lot of observations. This is clearly the most natural, but when the domain $\mathcal{X}$ is large, we face the challenge of computing thousands (perhaps hundreds of thousands) of weights. If we are going to go this route, we need a fairly simple method to compute the weights.

We can view the estimates $(\bar{f}^{(g,n)})_{g \in \mathcal{G}}$ as different ways of estimating the same quantity. There is an extensive statistics literature on this problem. For example, it is well known that the weights that minimize the variance of $\bar{f}_x^n$ in equation (3.31) are given by

$$w_x^{(g)} \propto \left((\bar{\sigma}_x^2)^{(g,n)} \right)^{-1}.$$

Since the weights should sum to one, we obtain

$$w_x^{(g)} = \left(\frac{1}{(\bar{\sigma}_x^2)^{(g,n)}} \right) \left(\sum_{g \in \mathcal{G}} \frac{1}{(\bar{\sigma}_x^2)^{(g,n)}} \right)^{-1}. \tag{3.32}$$

These weights work if the estimates are unbiased, which is clearly not the case. This is easily fixed by using the total variation (variance plus the square of the bias), producing the weights

$$w_x^{(g,n)} = \frac{1}{\left((\bar{\sigma}_x^2)^{(g,n)} + \left(\bar{\beta}_x^{(g,n)} \right)^2 \right)} \left(\sum_{g' \in \mathcal{G}} \frac{1}{\left((\bar{\sigma}_x^2)^{(g',n)} + \left(\bar{\beta}_x^{(g',n)} \right)^2 \right)} \right)^{-1}. \tag{3.33}$$

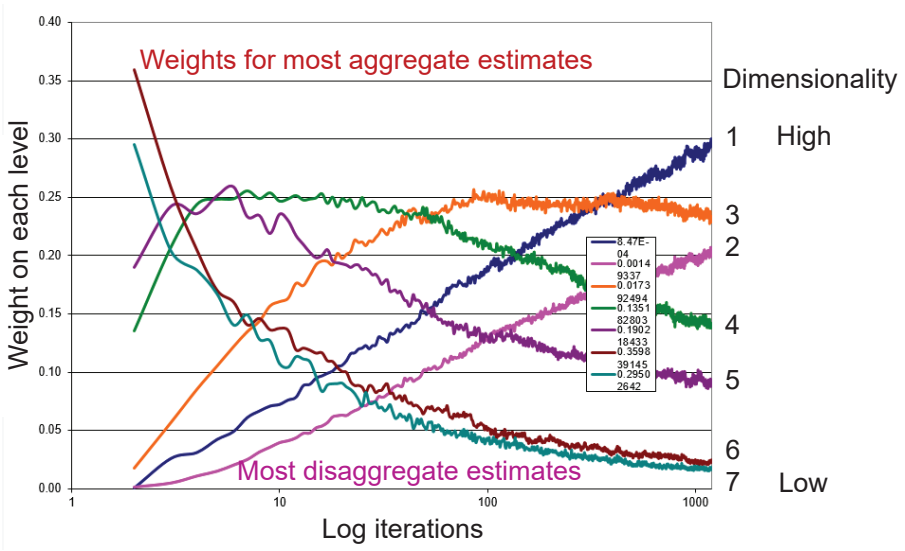


Figure 3.4 Average weight (across all states) for each level of aggregation using equation (3.33).

These are computed for each level of aggregation $g \in \mathcal{G}$. Furthermore, we compute a different set of weights for each point x. $(\bar{\sigma}_x^2)^{(g,n)}$ and $\bar{\beta}_x^{(g,n)}$ are easily computed recursively using equations (3.28) and (3.29), which makes the approach well suited to large scale applications. Note that if the stepsize used to smooth $\hat{f}^n$ goes to zero, then the variance $(\bar{\sigma}_x^2)^{(g,n)}$ will also go to zero as $n \to \infty$. However, the bias $\bar{\beta}_x^{(g,n)}$ will in general not go to zero.

Figure 3.4 shows the average weight put on each level of aggregation (when averaged over all the states s) for a particular application. The behavior illustrates the intuitive property that the weights on the aggregate level are highest when there are only a few observations, with a shift to the more disaggregate level as the algorithm progresses. This is a very important behavior when approximating functions recursively. It is simply not possible to produce good function approximations with only a few data points, so it is important to use simple functions (with only a few parameters).

3.7 LINEAR PARAMETRIC MODELS

Up to now, we have focused on lookup-table representations of functions, where if we are at a point x (or state s), we compute an approximation $\bar{F}(x)$ (or $\overline{V}(s)$) that is an estimate of the function at x (or state s). Using aggregation (even mixtures of estimates at different levels of aggregation) is still a form of look-up table (we are just using a simpler lookup-table). Lookup tables offer tremendous flexibility, but generally do not scale to higher dimensional variables (x or s), and do not allow you to take advantage of structural relationships.

There has been considerable interest in estimating functions using regression methods. A classical presentation of linear regression poses the problem of estimating a parameter vector θ to fit a model that predicts a variable y using a set of observations (known as covariates in the machine learning community) $(x_i)_{i \in \mathcal{I}}$, where we assume a model of the

form

$$y = \theta_0 + \sum_{i=1}^{I} \theta_i x_i + \varepsilon. \tag{3.34}$$

The variables x_i might be called independent variables, explanatory variables, or covariates, depending on the community. In dynamic programming where we want to estimate a value function $V^\pi(S_t)$, we might write

$$\overline{V}(S|\theta) = \sum_{f\in\mathcal{F}} \theta_f \phi_f(S),$$

where $(\phi_f(S))_{f\in\mathcal{F}}$ are known variously as *basis functions* or *features*, but are also referred to by names such as *covariates* or simply "independent variables." We might use this vocabulary regardless of whether we are approximating a value function or the policy itself. In fact, if we write our policy using

$$X^\pi(S_t|\theta) = \sum_{f\in\mathcal{F}} \theta_f \phi_f(S_t),$$

we would refer to $X^\pi(S_t|\theta)$ as a *linear decision rule* or, alternatively, as an *affine policy* ("affine" is just a fancy name for linear, by which we mean linear in θ).

Linear models are arguably the most popular approximation strategy for complex problems because they handle high-dimensionality by imposing a linear structure (which also means separable and additive). Using this language, instead of an independent variable x_i, we would have a basis function $\phi_f(S)$, where $f \in \mathcal{F}$ is a *feature*. $\phi_f(S)$ might be an indicator variable (e.g., 1 if we have an 'X' in the center square of our tic-tac-toe board), a discrete number (the number of X's in the corners of our tic-tac-toe board), or a continuous quantity (the price of an asset, the amount of oil in our inventories, the amount of $AB-$ blood on hand at the hospital). Some problems might have fewer than 10 features; others may have dozens; and some may have hundreds of thousands. In general, however, we would write our value function in the form

$$\overline{V}(S|\theta) = \sum_{f\in\mathcal{F}} \theta_f \phi_f(S).$$

In a time dependent model, the parameter vector θ would typically also be indexed by time, which can dramatically increase the number of parameters we have to estimate.

In the remainder of this section, we provide a brief review of linear regression, followed by some examples of regression models. We close with a more advanced presentation that provides insights into the geometry of basis functions (including a better understanding of why they are called "basis functions"). Given the tremendous amount of attention this class of approximations has received in the literature, we defer to chapter 16 a full description of how to fit linear regression models recursively for an ADP setting.

3.7.1 Linear regression review

Let y^n be the n^{th} observation of our dependent variable (what we are trying to predict) based on the observation $(x_1^n, x_2^n, \ldots, x_I^n)$ of our independent (or explanatory) variables

(the x_i are equivalent to the basis functions we used earlier). Our goal is to estimate a parameter vector θ that solves

$$\min_{\theta} \sum_{m=1}^{n} \left(y^m - (\theta_0 + \sum_{i=1}^{I} \theta_i x_i^m) \right)^2. \tag{3.35}$$

This is the standard linear regression problem.

Throughout this section, we assume that the underlying process from which the observations y^n are drawn is stationary (an assumption that is often not the case in the context of sequential decision problems).

If we define $x_0 = 1$, we let

$$x^n = \begin{pmatrix} x_0^n \\ x_1^n \\ \vdots \\ x_I^n \end{pmatrix}$$

be an $I+1$-dimensional column vector of observations. Throughout this section, and unlike the rest of the book, we use traditional vector operations, where $x^T x$ is an inner product (producing a scalar) while xx^T is an outer product, producing a matrix of cross terms.

Letting θ be the column vector of parameters, we can write our model as

$$y = \theta^T x + \varepsilon.$$

We assume that the errors $(\varepsilon^1, \ldots, \varepsilon^n)$ are independent and identically distributed. We do not know the parameter vector θ, so we replace it with an estimate $\bar{\theta}$ which gives us the predictive formula

$$\bar{y}^n = (\bar{\theta})^T x^n,$$

where $\bar{y}^n$ is our predictor of y^{n+1}. Our prediction error is

$$\hat{\varepsilon}^n = y^n - (\bar{\theta})^T x^n.$$

Our goal is to choose θ to minimize the mean squared error

$$\min_{\theta} \sum_{m=1}^{n} (y^m - \theta^T x^m)^2. \tag{3.36}$$

It is well known that this can be solved very simply. Let X^n be the n by $I+1$ matrix

$$X^n = \begin{pmatrix} x_0^1 & x_1^1 & & x_I^1 \\ x_0^2 & x_1^2 & & x_I^2 \\ \vdots & \vdots & \cdots & \vdots \\ x_0^n & x_1^n & & x_I^n \end{pmatrix}.$$

Next, denote the vector of observations of the dependent variable as

$$Y^n = \begin{pmatrix} y^1 \\ y^2 \\ \vdots \\ y^n \end{pmatrix}.$$

The optimal parameter vector $\bar{\theta}$ (after n observations) is given by

$$\bar{\theta} = [(X^n)^T X^n]^{-1}(X^n)^T Y^n. \tag{3.37}$$

These are known as the *normal equations*.

Solving a static optimization problem such as (3.36), which produces the elegant equations for the optimal parameter vector in (3.37), is the most common approach taken by the statistics community. It has little direct application in the context of our sequential decision problems since our applications tend to be recursive in nature, reflecting the fact that at each iteration we obtain new observations, which require updates to the parameter vector. In addition, our observations tend to be notoriously nonstationary. Later, we show how to overcome this problem using the methods of recursive statistics.

3.7.2 Sparse additive models and Lasso

It is not hard to create models where there are a large number of explanatory variables. Some examples include:

EXAMPLE 3.9

A physician is trying to choose the best medical treatment for a patient, which may be described by thousands of different characteristics. It is unlikely that all of these characteristics have strong explanatory power.

EXAMPLE 3.10

A scientist is trying to design probes to identify the structure of RNA molecules. There are hundreds of locations where a probe can be attached. The challenge is to design probes to learn a statistical model that has hundreds of parameters (corresponding to each location).

EXAMPLE 3.11

An internet provider is trying to maximize ad-clicks, where each ad is characterized by an entire dataset consisting of all the text and graphics. A model can be created by generating hundreds (perhaps thousands) of features based on word patterns within the ad. The problem is to learn which features are most important by carefully selecting ads.

In these settings, we are trying to approximate a function $f(S)$ where S is our "state variable" consisting of all the data (describing patients, the RNA molecule, or the features within an ad). $f(S)$ might be the response (medical successes or costs, or clicks on ads), which we approximate using

$$\bar{F}(S|\theta) = \sum_{f \in \mathcal{F}} \theta_f \phi_f(S). \tag{3.38}$$

Now imagine that there are hundreds of features in the set $\mathcal{F}$, but we anticipate that $\theta_f = 0$ for many of these. In this case, we would view equation (3.38) as a *sparse additive* model,

where the challenge is to identify a model with the highest explanatory power which means excluding the parameters which do not contribute very much.

Imagine we have a dataset consisting of $(f^n, S^n)_{n=1}^N$ where f^n is the observed response corresponding to the information in S^n. If we use this data to fit (3.38), virtually every fitted value of θ_f will be nonzero, producing a huge model with little explanatory power. To overcome this, we introduce what is known as a *regularization* term where we penalize nonzero values of θ. We would write the optimization problem as

$$\min_{\theta} \left(\sum_{n=1}^{N} (f^n - \bar{F}(S^n|\theta))^2 + \lambda \sum_{f \in \mathcal{F}} \|\theta_f\|_1 \right), \tag{3.39}$$

where $\|\theta_f\|_1$ represents what is known as "L_1" regularization, which is the same as taking the absolute value $|\theta_f|$. L_2 regularization would use θ_f^2, which means that there is almost no penalty for values of θ_f that are close to zero. This means we are assessing a penalty when $\theta_f \neq 0$, and the marginal penalty is the same for any value of θ_f other than zero.

We refer to $\lambda \sum_f \|\theta_f\|_1$ as a *regularization* term. As we increase λ, we put a higher penalty for allowing θ_f to be in the model. It is necessary to increase λ, take the resulting model, and then test it on an out-of-sample dataset. Typically, this is done repeatedly (five times is typical) where the out-of-sample observations are drawn from a different 20 percent of the data (this process is known as *cross-validation*). We can plot the error from this testing for each value of λ, and find the best value of λ.

This procedure is known as Lasso, for "Least absolute shrinkage and selection operator." The procedure is inherently batch, although there is a recursive form that has been developed. The method works best when we assume there is access to an initial testing dataset that can be used to help identify the best set of features.

A challenge with regularization is that it requires determining the best value of λ. It should not be surprising that you will get the best fit if you set $\lambda = 0$, creating a model with a large number of parameters. The problem is that these models do not offer the best predictive power, because many of the fitted parameters $\theta_f > 0$ reflect spurious noise rather than the identification of truly important features.

The way to overcome this is to use cross-validation, which works as follows. Imagine fitting the model on an 80 percent sample of the data, and then evaluating the model on the remaining 20 percent. Now, repeat this five times by rotating through the dataset, using different portions of the data for testing. Finally, repeat this entire process for different values of λ to find the value of λ that produces the lowest error.

Regularization is sometimes referred to as modern statistical learning. While not an issue for very low dimensional models where all the variables are clearly important, regularization is arguably one of the most powerful tools for modern models which feature large numbers of variables. Regularization can be introduced into virtually any statistical model, including nonlinear models and neural networks.

3.8 RECURSIVE LEAST SQUARES FOR LINEAR MODELS

Perhaps one of the most appealing features of linear regression is the ease with which models can be updated recursively. Recursive methods are well known in the statistics and machine learning communities, but these communities often focus on batch methods. Recursive statistics is especially valuable in stochastic optimization because they are well suited to any adaptive algorithm.

We start with a basic linear model

$$y = \theta^T x + \varepsilon,$$

where $\theta = (\theta_1, \ldots, \theta_I)^T$ is a vector of regression coefficients. We let X^n be the $n \times I$ matrix of observations (where n is the number of observations). Using batch statistics, we can estimate θ from the normal equation

$$\theta = [(X^n)^T X^n]^{-1} (X^n)^T Y^n. \tag{3.40}$$

We note in passing that equation (3.40) represents an optimal solution of a statistical model using a sampled dataset, one of the major solution strategies that we are going to describe in chapter 4 (stay tuned!).

We now make the conversion to the vocabulary where instead of a feature x_i, we are going to let x be our data and let $\phi_f(x)$ be a feature (also known as basis functions), where $f \in \mathcal{F}$ is our set of features. We let $\phi(x)$ be a column vector of the features, where $\phi^n = \phi(x^n)$ replaces x^n. We also write our function approximation using

$$\bar{F}(x|\theta) = \sum_{f \in \mathcal{F}} \theta_f \phi_f(x) = \phi(x)^T \theta.$$

Throughout our presentation, we assume that we have access to an observation $\hat{f}^n$ of our function $F(x, W)$.

3.8.1 Recursive least squares for stationary data

In the setting of adaptive algorithms in stochastic optimization, estimating the coefficient vector θ using batch methods such as equation (3.40) would be very expensive. Fortunately, it is possible to compute these formulas recursively. The updating equation for θ is

$$\theta^n = \theta^{n-1} - H^n \phi^n \hat{\varepsilon}^n, \tag{3.41}$$

where H^n is a matrix computed using

$$H^n = \frac{1}{\gamma^n} M^{n-1}. \tag{3.42}$$

The error $\hat{\varepsilon}^n$ is computed using

$$\hat{\varepsilon}^n = \bar{F}(x|\theta^{n-1}) - \hat{y}^n. \tag{3.43}$$

Note that it is common in statistics to compute the error in a regression using "actual minus predicted" while we are using "predicted minus actual" (see equation (3.43) above). Our sign convention is motivated by the derivation from first principles of optimization, which we cover in more depth in chapter 5.

Now let M^n be the $|\mathcal{F}| \times |\mathcal{F}|$ matrix given by

$$M^n = [(X^n)^T X^n]^{-1}.$$

Rather than do the matrix inversion, we can compute M^n recursively using

$$M^n = M^{n-1} - \frac{1}{\gamma^n} (M^{n-1} \phi^n (\phi^n)^T M^{n-1}), \tag{3.44}$$

where γ^n is a scalar computed using

$$\gamma^n = 1 + (\phi^n)^T M^{n-1} \phi^n. \tag{3.45}$$

The derivation of equations (3.41)-(3.45) is given in section 3.14.1.

It is possible in any regression problem that the matrix $(X^n)^T X^n$ (in equation (3.40)) is non-invertible. If this is the case, then our recursive formulas are not going to overcome this problem. When this happens, we will observe $\gamma^n = 0$. Alternatively, the matrix may be invertible, but unstable, which occurs when γ^n is very small (say, $\gamma^n < \epsilon$ for some small ϵ). When this occurs, the problem can be circumvented by using

$$\bar{\gamma}^n = \gamma^n + \delta,$$

where δ is a suitably chosen small perturbation that is large enough to avoid instabilities. Some experimentation is likely to be necessary, since the right value depends on the scale of the parameters being estimated.

The only missing step in our algorithm is initializing M^0. One strategy is to collect a sample of m observations where m is large enough to compute M^m using full inversion. Once we have M^m, we use it to initialize M^0 and then we can then proceed to update it using the formula above. A second strategy is to use $M^0 = \epsilon I$, where I is the identity matrix and ϵ is a "small constant." This strategy is not guaranteed to give the exact values, but should work well if the number of observations is relatively large.

In our stochastic optimization applications, the observations $\hat{f}^n$ will represent observations of the value of a function, or estimates of the value of being in a state, or even decisions we should make given a state. Our data can be a decision x (or possibly the decision x and initial state S_0), or a state S. The updating equations assume implicitly that the estimates come from a stationary series.

There are many problems where the number of basis functions can be extremely large. In these cases, even the efficient recursive expressions in this section cannot avoid the fact that we are still updating a matrix where the number of rows and columns may be large. If we are only estimating a few dozen or a few hundred parameters, this can be fine. If the number of parameters extends into the thousands, even this strategy would probably bog down. It is very important to work out the approximate dimensionality of the matrices before using these methods.

3.8.2 Recursive least squares for nonstationary data*

It is generally the case in approximate dynamic programming that our observations $\hat{f}^n$ (typically, updates to an estimate of a value function) come from a nonstationary process. This is true even when we are estimating the value of a fixed policy if we use TD learning, but it is always true when we introduce the dimension of optimizing over policies. Recursive least squares puts equal weight on all prior observations, whereas we would prefer to put more weight on more recent observations.

Instead of minimizing total errors (as we do in equation (3.35)) it makes sense to minimize a geometrically weighted sum of errors

$$\min_{\theta} \sum_{m=1}^{n} \lambda^{n-m} \left(f^m - (\theta_0 + \sum_{i=1}^{I} \theta_i \phi_i^m) \right)^2, \tag{3.46}$$

where λ is a discount factor that we use to discount older observations. If we repeat the derivation in section 3.8.1, the only changes we have to make are in the updating formula

for M^n, which is now given by

$$M^n = \frac{1}{\lambda}\left(M^{n-1} - \frac{1}{\gamma^n}(M^{n-1}\phi^n(\phi^n)^T M^{n-1})\right), \tag{3.47}$$

and the expression for γ^n, which is now given by

$$\gamma^n = \lambda + (\phi^n)^T M^{n-1}\phi^n. \tag{3.48}$$

λ works in a way similar to a stepsize, although in the opposite direction. Setting $\lambda = 1$ means we are putting an equal weight on all observations, while smaller values of λ puts more weight on more recent observations. In this way, λ plays a role similar to our use of λ in TD(λ).

We could use this logic and view λ as a tunable parameter. Of course, a constant goal in the design of algorithms is to avoid the need to tune yet another parameter. For the special case where our regression model is just a constant (in which case $\phi^n = 1$), we can develop a simple relationship between α_n and the discount factor (which we now compute at each iteration, so we write it as λ_n). Let $G^n = (H^n)^{-1}$, which means that our updating equation is now given by

$$\theta^n = \theta^{n-1} - (G^n)^{-1}\phi^n\hat{\varepsilon}^n.$$

Recall that we compute the error ε^n as predicted minus actual as given in equation (3.43). This is required if we are going to derive our optimization algorithm based on first principles, which means that we are minimizing a stochastic function. The matrix G^n is updated recursively using

$$G^n = \lambda_n G^{n-1} + \phi^n(\phi^n)^T, \tag{3.49}$$

with $G^0 = 0$. For the case where $\phi^n = 1$ (in which case G^n is also a scalar), $(G^n)^{-1}\phi^n = (G^n)^{-1}$ plays the role of our stepsize, so we would like to write $\alpha_n = G^n$. Assume that $\alpha_{n-1} = \left(G^{n-1}\right)^{-1}$. Equation (3.49) implies that

$$\begin{aligned}\alpha_n &= (\lambda_n G^{n-1} + 1)^{-1} \\ &= \left(\frac{\lambda_n}{\alpha_{n-1}} + 1\right)^{-1}.\end{aligned}$$

Solving for λ_n gives

$$\lambda_n = \alpha_{n-1}\left(\frac{1-\alpha_n}{\alpha_n}\right). \tag{3.50}$$

Note that if $\lambda_n = 1$, then we want to put equal weight on all the observations (which would be optimal if we have stationary data). We know that in this setting, the best stepsize is $\alpha_n = 1/n$. Substituting this stepsize into equation (3.50) verifies this identity.

The value of equation (3.50) is that it allows us to relate the discounting produced by λ_n to the choice of stepsize rule, which has to be chosen to reflect the nonstationarity of the observations. In chapter 6, we introduce a much broader range of stepsize rules, some of which have tunable parameters. Using (3.50) allows us to avoid introducing yet another tunable parameter.

3.8.3 Recursive estimation using multiple observations*

The previous methods assume that we get one observation and use it to update the parameters. Another strategy is to sample several paths and solve a classical least-squares problem for estimating the parameters. In the simplest implementation, we would choose a set of realizations $\hat{\Omega}^n$ (rather than a single sample ω^n) and follow all of them, producing a set of estimates $(f(\omega))_{\omega\in\hat{\Omega}^n}$ that we can use to update our estimate of the function $\bar{F}(s|\theta)$.

If we have a set of observations, we then face the classical problem of finding a vector of parameters $\hat{\theta}^n$ that best match all of these function estimates. Thus, we want to solve

$$\hat{\theta}^n = \arg\min_{\theta} \frac{1}{|\hat{\Omega}^n|} \sum_{\omega\in\hat{\Omega}^n} (\bar{F}(s|\theta) - f(\omega))^2.$$

This is the standard parameter estimation problem faced in the statistical estimation community. If $\bar{F}(s|\theta)$ is linear in θ, then we can use the usual formulas for linear regression. If the function is more general, we would typically resort to nonlinear programming algorithms to solve the problem. In either case, $\hat{\theta}^n$ is still an update that needs to be smoothed in with the previous estimate θ^{n-1}, which we would do using

$$\theta^n = (1-\alpha_{n-1})\theta^{n-1} + \alpha_{n-1}\hat{\theta}^n. \tag{3.51}$$

One advantage of this strategy is that in contrast with the updates that depend on the gradient of the value function, updates of the form given in equation (3.51) do not encounter a scaling problem, and therefore we return to our more familiar territory where $0 < \alpha_n \leq 1$. Of course, as the sample size $\hat{\Omega}$ increases, the stepsize should also be increased because there is more information in $\hat{\theta}^n$. Using stepsizes based on the Kalman filter (sections 6.3.2 and 6.3.3) will automatically adjust to the amount of noise in the estimate.

The usefulness of this particular strategy will be very problem-dependent. In many applications, the computational burden of producing multiple estimates $\hat{v}^n(\omega), \omega \in \hat{\Omega}^n$ before producing a parameter update will simply be too costly.

3.9 NONLINEAR PARAMETRIC MODELS

While linear models are exceptionally powerful (recall that "linear" means linear in the parameters), it is inevitable that some problems will require models that are nonlinear in the parameters. We might want to model the nonlinear response of price, dosage, or temperature. Nonlinear models introduce challenges in model estimation as well as learning in stochastic optimization problems.

We begin with a presentation on maximum likelihood estimation, one of the most widely used estimation methods for nonlinear models. We then introduce the idea of a sampled nonlinear model, which is a simple way of overcoming the complexity of a nonlinear model. We close with an introduction to neural networks, a powerful approximation architecture that has proven to be useful in machine learning as well as dynamic programs arising in engineering control problems.

3.9.1 Maximum likelihood estimation

The most general method for estimating nonlinear models is known as maximum likelihood estimation. Let $f(x|\theta)$ the function given θ, and assume that we observe

$$y = f(x|\theta) + \epsilon$$

where $\epsilon \sim N(0, \sigma^2)$ is the error with density

$$f^{\epsilon}(w) = \frac{1}{\sqrt{2\pi}\sigma} \exp \frac{w^2}{2\sigma^2}.$$

Now imagine that we have a set of observations $(y^n, x^n)_{n=1}^N$. The likelihood of observing $(y^n)_{n=1}^N$ is given by

$$L(y|x, \theta) = \Pi_{n=1}^N \exp \frac{(y^n - f(x^n|\theta))^2}{2\sigma^2}.$$

It is common to use the log likelihood $\mathcal{L}(y|x, \theta) = \log L(y|x, \theta)$, which gives us

$$\mathcal{L}(y|x, \theta) = \sum_{n=1}^{N} \frac{1}{\sqrt{2\pi}\sigma}(y^n - f(x^n|\theta))^2, \tag{3.52}$$

where we can, of course, drop the leading constant $\frac{1}{\sqrt{2\pi}\sigma}$ when maximizing $\mathcal{L}(y|x, \theta)$.

Equation (3.52) can be used by nonlinear programming algorithms to estimate the parameter vector θ. This assumes that we have a batch dataset $(y^n, x^n)_{n=1}^N$, which is not our typical setting. In addition, the log likelihood $\mathcal{L}(y|x, \theta)$ can be nonconvex when $f(x|\theta)$ is nonlinear in θ, which further complicates the optimization challenge.

The next section describes a method for handling nonlinear models in a recursive setting.

3.9.2 Sampled belief models

A powerful strategy for estimating models that are nonlinear in the parameters assumes that the unknown parameter θ can only take on one of a finite set $\theta_1, \theta_2, \ldots, \theta_K$. Let $\boldsymbol{\theta}$ be a random variable representing the true value of θ, where $\boldsymbol{\theta}$ takes on one of the values $\Theta = (\theta_k)_{k=1}^K$.

Assume we start with a prior set of probabilities $p_k^0 = \mathbb{P}[\boldsymbol{\theta} = \theta_k]$, and let $p^n = (p_k^n), k = 1, \ldots K$ be the probabilities after n experiments. This is framework we use when we adopt a Bayesian perspective: we view the true value of θ as a random variable $\boldsymbol{\theta}$, with a prior distribution of belief p^0 (which might be uniform).

We refer to $B^n = (p^n, \Theta)$ as a *sampled belief model*. Sampled belief models are powerful ways for representing the uncertainty in a nonlinear belief model. The process of generating the set Θ (which actually can change with iterations) makes it possible for a user to ensure that each member of the sample is reasonable (for example, we can ensure that some coefficients are positive). Updating the probability vector p^n can be done fairly simply using Bayes theorem, as we show below.

What we are now going to do is to use observations of the random variable Y to update our probability distribution. To illustrate this, assume that we are observing successes and failures, so $Y \in \{0, 1\}$, as might happen with medical outcomes. In this setting, the vector x would consist of information about a patient as well as medical decisions. Assume that the probability that $Y = 1$ is given by a logistic regression, given by

$$f(y|x, \theta) = \mathbb{P}[Y = 1|x, \theta] \tag{3.53}$$

$$= \frac{\exp^{U(x|\theta)}}{1 + \exp^{U(x|\theta)}}, \tag{3.54}$$

where $U(x|\theta)$ is a linear model given by

$$U(x|\theta) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_M.$$

We assume that θ is one of the elements $(\theta_k)_{k=1}^K$, where θ_k is a vector of elements $(\theta_{km})_{m=1}^M$. Let $H^n = (y^1, \ldots, y^n)$ be our history of observations of the random outcome Y. Now assume that $p_k^n = \mathbb{P}[\boldsymbol{\theta} = \theta_k | H^n]$, and that we next choose x^n and observe $Y = y^{n+1}$ (later, we are going to talk about how to choose x^n). We can update our probabilities using Bayes theorem

$$p_k^{n+1} = \frac{\mathbb{P}[Y = y^{n+1}|x^n, \theta_k, H^n]\mathbb{P}[\boldsymbol{\theta} = \theta_k|H^n]}{\mathbb{P}[Y = y^{n+1}|x^n, H^n]}. \tag{3.55}$$

We start by observing that $p_k^n = \mathbb{P}[\boldsymbol{\theta} = \theta_k|H^n]$. The conditional probability $\mathbb{P}[Y = y^{n+1}|x^n, \theta_k, H^n]$ comes from our logistic regression in (7.58):

$$\mathbb{P}[Y = y^{n+1}|x^n, \theta_k, H^n] = \begin{cases} f(x^n|\theta^n) & \text{if } y^{n+1} = 1, \\ 1 - f(x^n|\theta^n) & \text{if } y^{n+1} = 0. \end{cases}$$

Finally, we compute the denominator using

$$\mathbb{P}[Y = y^{n+1}|x^n, H^n] = \sum_{k=1}^K \mathbb{P}[Y = y^{n+1}|x^n, \theta_k, H^n]p_k^n.$$

This idea can be extended to a wide range of distributions for Y. Its only limitation (which may be significant) is the assumption that θ can be only one of a finite set of discrete values. A strategy for overcoming this limitation is to periodically generate new possible values of θ, use the past history of observations to obtain updated probabilities, and then drop the values with the lowest probability.

3.9.3 Neural networks - parametric*

Neural networks represent an unusually powerful and general class of approximation strategies that has been widely used in optimal control and statistical learning. There are a number of excellent textbooks on the topic along with widely available software packages, so our presentation is designed only to introduce the basic idea and encourage readers to experiment with this technology if simpler models are not effective.

In this section, we restrict our attention to low-dimensional neural networks, although these "low-dimensional" neural networks may still have thousands of parameters. Neural networks in this class have been very popular for many years in the engineering controls community, where they are used for approximating both policies and value functions for deterministic control problems.

We return to neural networks in section 3.10.4 where we discuss the transition to "deep" neural networks, which are extremely high-dimensional functions allowing them to approximate almost anything, earning them the classification as a nonparametric model.

In this section we describe the core algorithmic steps for performing estimation with neural networks. We defer until chapter 5 the description of how we optimize the parameters, since we are going to use the methods of derivative-based stochastic optimization which are covered in that chapter.

Up to now, we have considered approximation functions of the form

$$\bar{F}(x|\theta) = \sum_{f \in \mathcal{F}} \theta_f \phi_f(x),$$

where $\mathcal{F}$ is our set of features, and $(\phi_f(x))_{f\in\mathcal{F}}$ are the basis functions which extract what are felt to be the important characteristics of the state variable which explain the value of being in a state. We have seen that when we use an approximation that is linear in the parameters, we can estimate the parameters θ recursively using standard methods from linear regression. For example, if x^n is the n^{th} input with element x_i^n, our approximation might look like

$$\bar{F}(x^n|\theta) = \sum_{i\in\mathcal{I}} \left(\theta_{1i}x_i^n + \theta_{2i}(x_i^n)^2\right).$$

Now assume that we feel that the best function might not be quadratic in R_i, but we are not sure of the precise form. We might want to estimate a function of the form

$$\bar{F}(x^n|\theta) = \sum_{i\in\mathcal{I}} \left(\theta_{1i}x_i^n + \theta_{2i}(x_i^n)^{\theta_3}\right).$$

Now we have a function that is nonlinear in the parameter vector $(\theta_1, \theta_2, \theta_3)$, where θ_1 and θ_2 are vectors and θ_3 is a scalar. If we have a training dataset of state-value observations, $(\hat{f}^n, R^n)_{n=1}^N$, we can find θ by solving

$$\min_\theta F(\theta) = \sum_{n=1}^N \left(\hat{f}^n - \bar{F}(x^n|\theta)\right)^2, \tag{3.56}$$

which generally requires the use of nonlinear programming algorithms. One challenge is that nonlinear optimization problems do not lend themselves to the simple recursive updating equations that we obtained for linear (in the parameters) functions. More problematic is that we have to experiment with various functional forms to find the one that fits best.

Neural networks are, ultimately, a form of nonlinear model which can be used to approximate the function $\mathbb{E}f(x, W)$ (or a policy $X^\pi(S)$, or a value function $V(S)$). We will have an input x (or S), and we are using a neural network to predict an output $\hat{f}$ (or a decision x^n, or a value v^n). Using the traditional notation of statistics, let x^n be a vector of inputs which could be features $\phi_f(S^n)$ for $f \in \mathcal{F}$. If we were using a linear model, we would write

$$f(x^n|\theta) = \theta_0 + \sum_{i=1}^I \theta_i x_i^n.$$

In the language of neural networks, we have I inputs (we have $I + 1$ parameters since we also include a constant term), which we wish to use to estimate a single output f^{n+1} (a random observations of our function). The relationships are illustrated in figure 3.5 where we show the I inputs which are then "flowed" along the links to produce $f(x^n|\theta)$. After this, we then learn the sample realization $\hat{f}^{n+1}$ that we were trying to predict, which allows us to compute the error $\epsilon^{n+1} = \hat{f}^{n+1} - f(x^n|\theta)$.

Define the random variable X to describe a set of inputs (where x^n is the value of X at the n^{th} iteration), and let $\hat{f}$ be the random variable giving the response from input X. We would like to find a vector θ that solves

$$\min_\theta \mathbb{E}\frac{1}{2}(f(X|\theta) - \hat{f})^2.$$

Let $F(\theta) = \mathbb{E}\big(0.5(f(X|\theta) - \hat{f})^2\big)$, and let $F(\theta, \hat{f}) = 0.5(f(X|\theta) - \hat{f})^2$ where $\hat{f}$ is a sample realization of our function. As before, we can solve this iteratively using the algorithm we

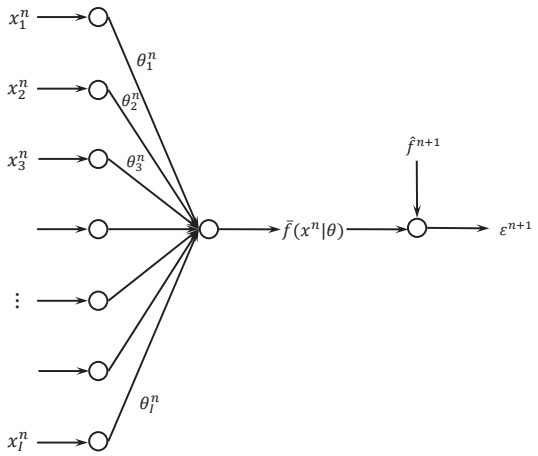


Figure 3.5 Neural networks with a single layer.

first introduced in section 3.2 which gives us the updating equation

$$\theta^{n+1} = \theta^n - \alpha_n \nabla_\theta F(\theta^n, \hat{f}^{n+1}), \tag{3.57}$$

where $\nabla_\theta F(\theta^n, \hat{f}^{n+1}) = \epsilon^{n+1} = (f(x^n|\theta) - \hat{f}^{n+1})$ for a given input $X = x^n$ and observed response $\hat{f}^{n+1}$.

We illustrated our linear model by assuming that the inputs were the individual dimensions of the control variable which we denoted x_i^n. We may not feel that this is the best way to represent the state of the system (imagine representing the states of a Connect-4 game board). We may feel it is more effective (and certainly more compact) if we have access to a set of basis functions $\phi_f(X),\ f \in \mathcal{F}$, where $\phi_f(X)$ captures a relevant feature of our system given the inputs X. In this case, we would be using our standard basis function representation, where each basis function provides one of the inputs to our neural network.

This was a simple illustration, but it shows that if we have a linear model, we get the same basic class of algorithms that we have already used. A richer model, given in figure 3.6, illustrates a more classical neural network. Here, the "input signal" x^n (this can be the state variable or the set of basis functions) is communicated through several layers. Let $x^{(1,n)} = x^n$ be the input to the first layer (recall that x_i^n might be the i^{th} dimension of the state variable itself, or a basis function). Let $\mathcal{I}^{(1)}$ be the set of inputs to the first layer (for example, the set of basis functions).

Here, the first linear layer produces $|\mathcal{I}^{(2)}|$ outputs given by

$$y_j^{(2,n)} = \sum_{i \in \mathcal{I}^{(1)}} \theta_{ij}^{(1)} x_i^{(1,n)}, \quad j \in \mathcal{I}^{(2)}.$$

$x_j^{(2,n)}$ becomes the input to a nonlinear *perceptron* node which is characterized by a nonlinear function that may dampen or magnify the input. A typical functional form for a perceptron node is the logistics function given by

$$\sigma(y) = \frac{1}{1 + e^{-\beta y}}, \tag{3.58}$$

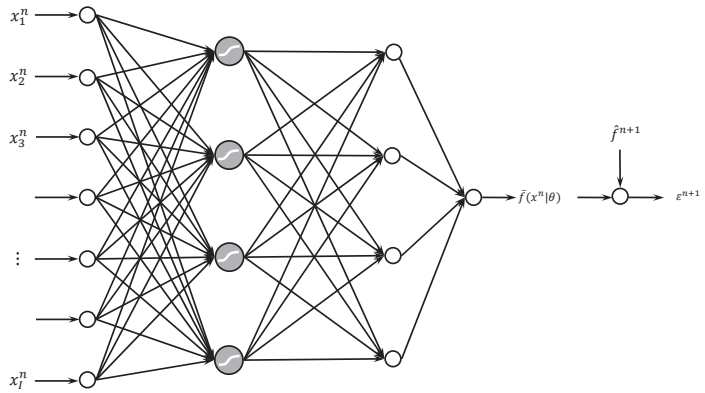


Figure 3.6 A three-layer neural network.

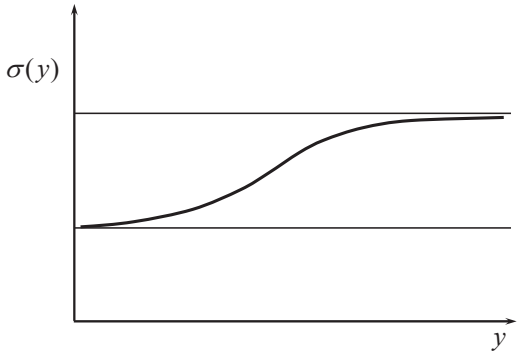


Figure 3.7 Illustrative logistics function for introducing nonlinear behavior into neural networks.

where β is a scaling coefficient. The function $\sigma(y)$ is illustrated in figure 3.7. The sigmoid function $\sigma(x)$ introduces nonlinear behavior into the communication of the "signal" x^n. In addition let

$$\sigma'(y) = \frac{\partial \sigma(y)}{\partial y}.$$

We next calculate

$$x_i^{(2,n)} = \sigma(y_i^{(2,n)}), \;\; i \in \mathcal{I}^{(2)}$$

and use $x_i^{(2,n)}$ as the input to the second linear layer. We then compute

$$y_j^{(3,n)} = \sum_{i \in \mathcal{I}^{(2)}} \theta_{ij}^{(2)} x_i^{(2,n)}, \quad j \in \mathcal{I}^{(3)}.$$

and then calculate the input to layer 3

$$x_i^{(3,n)} = \sigma(y_i^{(3,n)}), \;\; i \in \mathcal{I}^{(3)}.$$

Finally, we compute the single output using

$$\bar{f}^n(x^n|\theta) = \sum_{i \in \mathcal{I}^{(3)}} \theta_i^{(3)} x_i^{(3,n)}.$$

As before, f^n is our estimate of the response from input x^n. This is our function approximation $\bar{F}^n(s|\theta)$ which we update using the observation $\hat{f}^{n+1}$. Now that we know how to produce estimates using a neural network given the vector θ, the next step is optimize θ.

We update the parameter vector $\theta = (\theta^{(1)}, \theta^{(2)}, \theta^{(3)})$ using the stochastic gradient algorithm given in equation (3.57). The only difference is that the derivatives have to capture the fact that changing $\theta^{(1)}$, for example, impacts the "flows" through the rest of the network. There are standard packages for fitting neural networks to data using gradient algorithms, but for readers interested in the algorithmic side, we defer until section 5.5 the presentation of this algorithm since it builds on the methods of derivative-based stochastic search.

This presentation should be viewed as a simple illustration of an extremely rich field. The advantage of neural networks is that they offer a much richer class of nonlinear functions ("nonlinear architectures" in the language of machine learning) which can be trained in an iterative way. Calculations involving neural networks exploit the layered structure, and naturally come in two forms: feed forward propagation, where we step forward through the layers "simulating" the evolution of the input variables to the outputs, and backpropagation, which is used to compute derivatives so we can calculate the marginal impact of changes in the parameters (shown in section 5.5).

3.9.4 Limitations of neural networks

Neural networks offer an extremely flexible architecture, which reduces the need for designing and testing different nonlinear (parametric) models. They have been particularly successful in the context of deterministic problems such as optimal control of engineering systems, and the familiar voice and image recognition tools that have been so visible. There is a price, however, to this flexibility:

- To fit models with large numbers of parameters, you need large datasets. This is problematic in the context of sequential decision problems since we are often starting with little or no data, and then generating a series of inputs that allow us to create increasingly more accurate estimates of whatever function we are approximating.

- The flexibility of a neural network also means that, when applied to problems with noise, the network may just be fitting the noise (this is classic overfitting, well known to the statistical learning community). When the underlying problem exhibits noise (and many of the sequential decision problems in this book exhibit high levels of noise), the data requirements grow dramatically. Sadly, this is often overlooked in the neural network community, where it is not unusual to fit neural networks to datasets where there are more parameters in the neural network than data points.

- Neural networks struggle to replicate structure. There are many problems in business, engineering, economics and the sciences that exhibit structure: monotonicity (the higher the price, the lower the demand); concavity (very common in resource allocation problems); unimodularity (there is an optimum response to dosage, which declines when the dosage is too high or too low).

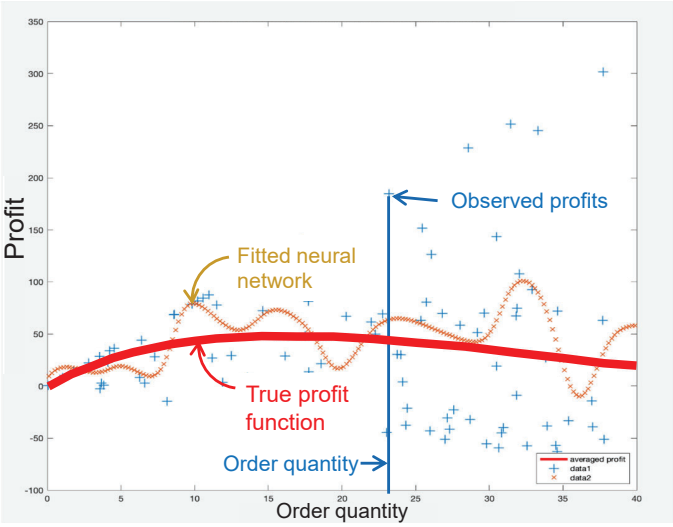


Figure 3.8 A neural network fitted to sampled data from a newsvendor problem, demonstrating the tendency of neural networks to overfit noisy data without capturing problem structure (such as concavity).

The problem of handling noise, and not being able to capture structure, is illustrated in figure 3.8, where we sampled data from the newsvendor problem

$$F(x, W) = 10 \min\{x, W\} - 8x,$$

where W is distributed according to the density

$$f^W(w) = .1e^{-.1w}.$$

We sampled 1000 observations of the demand W and the profit $F(x, W)$, for values of x that were drawn uniformly between 0 and 40. This data was then fitted with a neural network.

The expected profit $F(x) = \mathbb{E}_W F(x, W)$ is shown as the concave red line. The fitted neural network does not come close to capturing this structure. 1000 observations is a lot of data for approximating a one-dimensional function. We urge readers to use caution when using neural networks in the context of noisy but structured applications since these arise frequently in the application domains discussed in the opening of chapter 1.

3.10 NONPARAMETRIC MODELS*

The power of parametric models is matched by their fundamental weakness: they are only effective if you can find the right structure, and this remains a frustrating art. For this reason, nonparametric statistics have attracted recent attention. They avoid the art of specifying a parametric model, but introduce other complications. Nonparametric methods work primarily by building local approximations to functions using observations rather than depending on functional approximations.

Nonparametric models are characterized by the property that as the number of observations $N \rightarrow \infty$, we can approximate any function with arbitrary accuracy. This means that the working definition of a nonparametric model is that with enough data, they can approximate any function. However, the price of such flexibility is that you need very large datasets.

There is an extensive literature on the use of approximation methods for continuous functions. These problems, which arise in many applications in engineering and economics, require the use of approximation methods that can adapt to a wide range of functions. Interpolation techniques, orthogonal polynomials, Fourier approximations and splines are just some of the most popular techniques. Often, these methods are used to closely approximate the expectation using a variety of numerical approximation techniques.

We note that lookup tables are, technically, a form of nonparametric approximation methods, although these can also be expressed as parametric models by using indicator variables (this is the reason why the three classes of statistical models are illustrated as overlapping functions in figure 3.1). For example, assume that $\mathcal{X} = \{x_1, x_2, \ldots, x_M\}$ is a set of discrete inputs, and let

$$\mathbb{1}_{\{X=x\}} = \begin{cases} 1 & \text{if } X = x \in \mathcal{X}, \\ 0 & \text{otherwise.} \end{cases}$$

be an indicator variable that tells us when X takes on a particular value. We can write our function as

$$f(X|\theta) = \sum_{x \in \mathcal{X}} \theta_x \mathbb{1}_{\{X=x\}}$$

This means that we need to estimate a parameter θ_x for each $x \in \mathcal{X}$. In principle, this is a parametric representation, but the parameter vector θ has the same dimensionality as the input vector x. However, the working definition of a nonparametric model is one that, given an infinite dataset, will produce a perfect representation of the true function, a property that our lookup table model clearly satisfies. It is precisely for this reason that we treat lookup tables as a special case since parametric models are always used for settings where the parameter vector θ is much lower dimensional than the size of $\mathcal{X}$.

In this section, we review some of the nonparametric methods that have received the most attention within the approximate dynamic programming community. This is an active area of research which offers potential as an approximation strategy, but significant hurdles remain before this approach can be widely adopted. We start with the simplest methods, closing with a powerful class of nonparametric methods known as support vector machines.

3.10.1 K-nearest neighbor

Perhaps the simplest form of nonparametric regression forms estimates of functions by using a weighted average of the k-nearest neighbors. As above, we assume we have a response y^n corresponding to a measurement $x^n = (x_1^n, x_2^n, \ldots, x_I^n)$. Let $\rho(x, x^n)$ be a distance metric between a query point x (in dynamic programming, this would be a state), and an observation x^n. Then let $\mathcal{N}^n(x)$ be the set of the k-nearest points to the query point x, where clearly we require $k \leq n$. Finally let $\bar{Y}^n(x)$ be the response function, which is our best estimate of the true function $Y(x)$ given the observations $x^1, \ldots, x^n$. When we use a k-nearest neighbor model, this is given by

$$\bar{Y}^n(x) = \frac{1}{k} \sum_{n \in \mathcal{N}^n(x)} y^n. \tag{3.59}$$

Thus, our best estimate of the function $Y(x)$ is made by averaging the k points nearest to the query point x.

Using a k-nearest neighbor model requires, of course, choosing k. Not surprisingly, we obtain a perfect fit of the data by using $k = 1$ if we base our error on the training dataset.

A weakness of this logic is that the estimate $\bar{Y}^n(x)$ can change abruptly as x changes continuously, as the set of nearest neighbors changes. An effective way of avoiding this behavior is using kernel regression, which uses a weighted sum of all data points.

3.10.2 Kernel regression

Kernel regression has attracted considerable attention in the statistical learning literature. As with k-nearest neighbor, kernel regression forms an estimate $\bar{Y}(x)$ by using a weighted sum of prior observations which we can write generally as

$$\bar{Y}^n(x) = \frac{\sum_{m=1}^{n} K_h(x, x^m) y^m}{\sum_{m=1}^{n} K_h(x, x^m)} \tag{3.60}$$

where $K_h(x, x^m)$ is a weighting function that declines with the distance between the query point x and the measurement x^m. h is referred to as the *bandwidth* which plays an important scaling role. There are many possible choices for the weighting function $K_h(x, x^m)$. One of the most popular is the Gaussian kernel, given by

$$K_h(x, x^m) = e^{-\left(\frac{\|x - x^m\|}{h}\right)^2}.$$

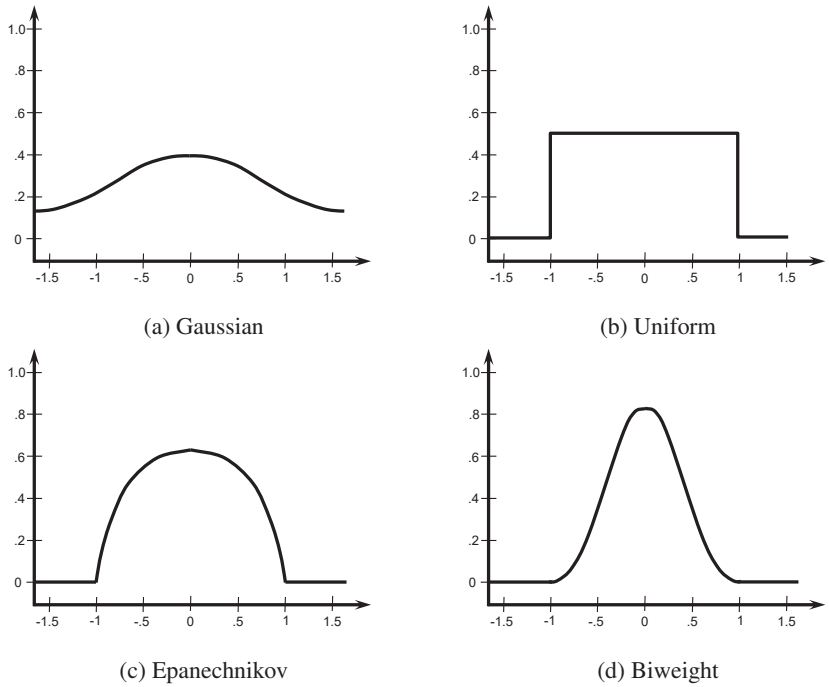
where $\|\cdot\|$ is the Euclidean norm. Here, h plays the role of the standard deviation. Note that the bandwidth h is a tunable parameter that captures the range of influence of a measurement x^m. The Gaussian kernel, often referred to as *radial basis functions*, provides a smooth, continuous estimate $\bar{Y}^n(x)$. Another popular choice of kernel function is the symmetric Beta family, given by

$$K_h(x, x^m) = \max(0, (1 - \|x - x^m\|)^2)^h.$$

Here, h is a nonnegative integer. $h = 1$ gives the uniform kernel; $h = 2$ gives the Epanechnikov kernel; and $h = 3$ gives the biweight kernel. Figure 3.9 illustrates each of these four kernel functions.

We pause to briefly discuss some issues surrounding k-nearest neighbors and kernel regression. First, it is fairly common in the ADP literature to see k-nearest neighbors and kernel regression being treated as a form of aggregation. The process of giving a set of states that are aggregated together has a certain commonality with k-nearest neighbor and kernel regression, where points near each other will produce estimates of $Y(x)$ that are similar. But this is where the resemblance ends. Simple aggregation is actually a form of parametric regression using dummy variables, and it offers neither the continuous approximations, nor the asymptotic unbiasedness of kernel regression.

Kernel regression is a method of approximation that is fundamentally different from linear regression and other parametric models. Parametric models use an explicit estimation step, where each observation results in an update to a vector of parameters. At any point in time, our approximation consists of the pre-specified parametric model, along with the current estimates of the regression parameters. With kernel regression, all we do is store data until we need an estimate of the function at a query point. Only then do we trigger the



(a) Gaussian (b) Uniform

(c) Epanechnikov (d) Biweight

Figure 3.9 Illustration of Gaussian, uniform, Epanechnikov and biweight kernel weighting functions.

approximation method, which requires looping over all previous observation, a step that clearly can become expensive as the number of observations grow.

Kernel regression enjoys an important property from an old result known as Mercer's theorem. The result states that there exists a set of basis functions $\phi_f(S),\ f \in \mathcal{F}$, possibly of very high dimensionality, where

$$K_h(S, S') = \phi(S)^T \phi(S'),$$

as long as the kernel function $K_h(S, S')$ satisfies some basic properties (satisfied by the kernels listed above). In effect this means that using appropriately designed kernels is equivalent to finding potentially very high dimensional basis functions, without having to actually create them.

Unfortunately, the news is not all good. First, there is the annoying dimension of bandwidth selection, although this can be mediated partially by scaling the explanatory variables. More seriously, kernel regression (and this includes k-nearest neighbors), cannot be immediately applied to problems with more than about five dimensions (and even this can be a stretch). The problem is that these methods are basically trying to aggregate points in a multidimensional space. As the number of dimensions grows, the density of points in the d-dimensional space becomes quite sparse, making it very difficult to use "nearby" points to form an estimate of the function. A strategy for high-dimensional applications is to use separable approximations. These methods have received considerable attention in the broader machine learning community, but have not been widely tested in an ADP setting.

3.10.3 Local polynomial regression

Classical kernel regression uses a weighted sum of responses y^n to form an estimate of $Y(x)$. An obvious generalization is to estimate locally linear regression models around each point x^n by solving a least squares problem which minimizes a weighted sum of least squares. Let $\bar{Y}^n(x|x^i)$ be a linear model around the point x^k, formed by minimizing the weighted sum of squares given by

$$\min_{\theta} \left(\sum_{m=1}^{n} K_h(x^k, x^m) \left(y^m - \sum_{i=1}^{I} \theta_i x_i^m \right)^2 \right). \tag{3.61}$$

Thus, we are solving a classical linear regression problem, but we do this for each point x^k, and we fit the regression using all the other points $(y^m, x^m),\ m = 1, \ldots, n$. However, we weight deviations between the fitted model and each observation y^m by the kernel weighting factor $K_h(x^k, x^m)$ which is centered on the point x^k.

Local polynomial regression offers significant advantages in modeling accuracy, but with a significant increase in complexity.

3.10.4 Deep neural networks

Low-dimensional (basically finite) neural networks are a form of parametric regression. Once you have specified the number of layers and the nodes per layer, all that is left are the weights in the network, which represent the parameters. However, there is a class of high-dimensional neural networks known as deep learners, which typically have four or more layers (see figure 3.10). These behave as if they have an unlimited number of layers and nodes per layer.

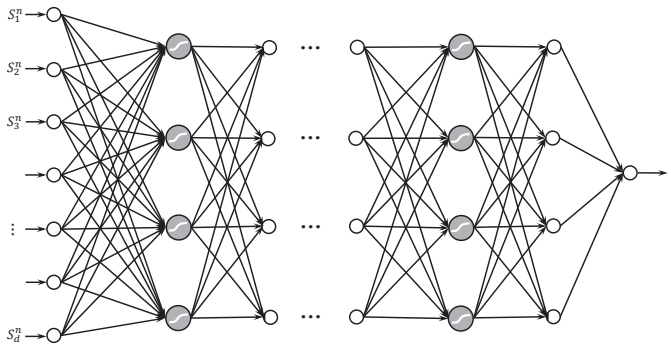


Figure 3.10 Illustration of a deep neural network.

Deep learners have shown tremendous power in terms of their ability to capture complex patterns in language and images. It is well known that they require notoriously large datasets for training, but there are settings where massive amounts of data are available such as the results of internet searches, images of people, and text searches. In the context of algorithms for sequential decision problems, there are settings (such as the algorithms used for playing video games) where it is possible to run the algorithm for millions of iterations.

As of this writing, it is not yet clear if deep learners will prove useful in stochastic optimization, partly because our data comes from the iterations of an algorithm, and partly

because the high-dimensional capabilities of neural networks raise the risk of overfitting in the context of stochastic optimization problems. Deep neural networks are very high-dimensional architectures, which means that they tend to fit noise, as we illustrated in figure 3.8. In addition, they are not very good at imposing structure such as monotonicity (although this has been a topic of research).

3.10.5 Support vector machines

Support vector machines (for classification) and support vector regression (for continuous problems) have attracted considerable interest in the machine learning community. For the purpose of fitting value function approximations, we are primarily interested in support vector regression, but we can also use regression to fit policy function approximations, and if we have discrete actions, we may be interested in classification. For the moment, we focus on fitting continuous functions.

Support vector regression, in its most basic form, is linear regression with a different objective than simply minimizing the sum of the squares of the errors. With support vector regression, we consider two goals. First, we wish to minimize the absolute sum of deviations that are larger than a set amount ξ. Second, we wish to minimize the regression parameters themselves, to push as many as possible close to zero.

As before, we let our predictive model be given by

$$y = \theta x + \epsilon.$$

Let $\epsilon^i = y^i - \theta x^i$ be the error. We then choose θ by solving the following optimization problem

$$\min_{\theta} \left(\frac{\eta}{2} \|\theta\|^2 + \sum_{i=1}^{n} \max\{0, |\epsilon^i| - \xi\} \right). \tag{3.62}$$

The first term penalizes positive values of θ, encouraging the model to minimize values of θ unless they contribute in a significant way to producing a better model. The second term penalizes errors that are greater than ξ. The parameters η and ξ are both tunable parameters. The error ϵ^i and error margin ξ are illustrated in figure 3.11.

It can be shown by solving the dual that the optimal value of θ and the best fit $\bar{Y}(x)$ have the form

$$\begin{aligned} \theta &= \sum_{i=1}^{n} (\bar{\beta}^i - \bar{\alpha}^i) x^i, \\ \bar{Y}(x) &= \sum_{i=1}^{n} (\bar{\beta}^i - \bar{\alpha}^i)(x^i)^T x^i. \end{aligned}$$

Here, $\bar{\beta}^i$ and $\bar{\alpha}^i$ are scalars found by solving

$$\min_{\bar{\beta}^i, \bar{\alpha}^i} \xi \sum_{i=1}^{n} (\bar{\beta}^i + \bar{\alpha}^i) - \sum_{i=1}^{n} y^i (\bar{\beta}^i + \bar{\alpha}^i) + \frac{1}{2} \sum_{i=1}^{n} \sum_{i'=1}^{n} (\bar{\beta}^i + \bar{\alpha}^i)(\bar{\beta}^{i'} + \bar{\alpha}^{i'})(x^i)^T x^{i'},$$

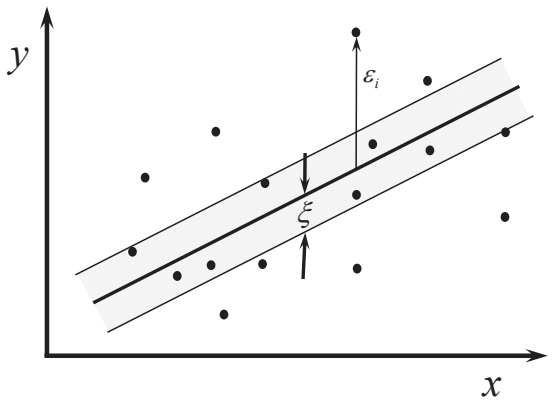


Figure 3.11 Illustration of penalty structure for support vector regression. Deviations within the gray area are assessed a value of zero. Deviations outside the gray area are measured based on their distance to the gray area.

subject to the constraints

$$\begin{aligned} 0 \leq \bar{\alpha}^i, \bar{\beta}^i &\leq 1/\eta, \\ \sum_{i=1}^{n} (\bar{\beta}^i - \bar{\alpha}^i) &= 0, \\ \bar{\alpha}^i \bar{\beta}^i &= 0. \end{aligned}$$

3.10.6 Indexed functions, tree structures and clustering

There are many problems where we feel comfortable specifying a simple set of basis functions for some of the parameters, but we do not have a good feel for the nature of the contribution of other parameters. For example, we may wish to plan how much energy to hold in storage over the course of the day. Let R_t be the amount of energy stored at time t, and let H_t be the hour of the day. Our state variable might be $S_t = (R_t, H_t)$. We feel that the value of energy in storage is a concave function in R_t, but this value depends in a complex way on the hour of day. It would not make sense, for example, to specify a value function approximation using

$$\overline{V}(S_t) = \theta_0 + \theta_1 R_t + \theta_2 R_t^2 + \theta_3 H_t + \theta_4 H_t^2.$$

There is no reason to believe that the hour of day will be related to the value of energy storage in any convenient way. Instead, we can estimate a function $\overline{V}(S_t|H_t)$ given by

$$\overline{V}(S_t|h) = \theta_0(h) + \theta_1(h) R_t + \theta_2(h) R_t^2.$$

What we are doing here is estimating a linear regression model for each value of $h = H_t$. This is simply a form of lookup table using regression given a particular value of the complex variables. Imagine that we can divide our state variable S_t into two sets: the first set, f_t, contains variables where we feel comfortable capturing the relationship using linear regression. The second set, g_t, includes more complex variables whose contribution is not

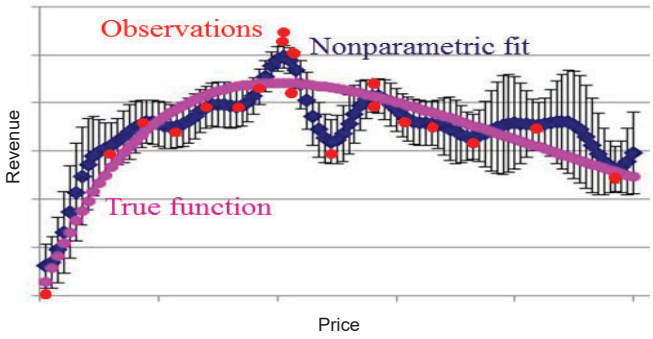


Figure 3.12 Fitting noisy data of revenue as a function of price using kernel regression versus a quadratic function.

as easily approximated. If g_t is a discrete scalar (such as hour of day), we can consider estimating a regression model for each value of g_t. However, if g_t is a vector (possibly with continuous dimensions), then there will be too possible values.

When the vector g_t cannot be enumerated, we can resort to various clustering strategies. These fall under names such as regression trees and local polynomial regression (a form of kernel regression). These methods cluster g_t (or possibly the entire state S_t) and then fit simple regression models over subsets of data. In this case, we would create a set of clusters $\mathcal{C}^n$ based on n observations of states and values. We then fit a regression function $\overline{V}(S_t|c)$ for each cluster $c \in \mathcal{C}^n$. In traditional batch statistics, this process proceeds in two stages: clustering and then fitting. In approximate dynamic programming, we have to deal with the fact that we may change our clusters as we collect additional data.

A much more sophisticated strategy is based on a concept known as Dirichlet process mixtures. This is a fairly sophisticated technique, but the essential idea is that you form clusters that produce good fits around local polynomial regressions. However, unlike traditional cluster-then-fit methods, the idea with Dirichlet process mixtures is that membership in a cluster is probabilistic, where the probabilities depend on the query point (e.g., the state whose value we are trying to estimate).

3.10.7 Comments on nonparametric models

Nonparametric models are extremely flexible, but two characteristics make them hard to work with:

- They need a *lot* of data.
- Due to their ability to closely fit data, nonparametric models are susceptible to overfitting when used to fit functions where observations are subject to noise (which describes almost everything in this book). Figure 3.12 illustrates observations of revenue as we vary price. We are expecting a smooth, concave function. A kernel regression model closely fits the data, producing a behavior that does not seem realistic. By contrast, we might fit a quadratic model that captures the structure that we are expecting.

- Nonparametric models can be very clumsy to store. Kernel regression models effectively need the entire dataset. Deep neural networks may involve hundreds of thousands or even millions of parameters.

Neural networks have attracted considerable attention in recent years as they have demonstrated their ability to recognize faces and voices. These are problems that do not have a well-known structure matching bitmapped images to the identity of a person. We also note that the right answer is deterministic, which helps with training.

We anticipate that parametric models will remain popular for problems which have known structure. A difficulty with parametric models is that they are generally accurate only over some region. This is not a problem if we are searching for a unique point on a function, such as the best price, the best dosage of a drug or the right temperature for running an experiment. However, there are problems such as estimating the value $V_t(S_t)$ of being in a state S_t at time t, which is a random variable that depends on the history up to time t. If we want to develop an approximate $\overline{V}_t(S_t) \approx V_t(S_t)$, then it has to be accurate over the range of states that we are likely to visit (and of course we may not know this).

3.11 NONSTATIONARY LEARNING*

There are a number of settings where the true mean varies over time. We begin with the simplest setting where the mean may evolve up or down, but on average stays the same. We then consider the situation where the signal is steadily improving up to some unknown limit.

In chapter 7 we are going to use this in the context of optimizing functions of nonstationary random variables, or time-dependent functions of (typically) stationary random variables.

3.11.1 Nonstationary learning I - Martingale truth

In the stationary case, we might write observations as

$$W_{t+1} = \mu + \varepsilon_{t+1},$$

where $\varepsilon \sim N(0, \sigma_\varepsilon^2)$. This means that $\mathbb{E}W_{t+1} = \mu$, which is an unchanging truth that we are trying to learn. We refer to this as the stationary case because the distribution of W_t does not depend on time.

Now assume that the true mean μ is also changing over time. We write the dynamics of the mean using

$$\mu_{t+1} = \mu_t + \varepsilon_{t+1}^\mu,$$

where ε^μ is a random variable with distribution $N(0, \sigma_\mu^2)$. This means that $\mathbb{E}\{\mu_{t+1}|\mu_t\} = \mu_t$, which is the definition of a martingale process. This means that on average, the true mean μ_{t+1} at time $t+1$ will be the same as at time t, although the actual may be different. Our observations are then made from

$$W_{t+1} = \mu_{t+1} + \varepsilon_{t+1}.$$

Typically, the variability of the mean process $\mu_0, \mu_1, \ldots, \mu_t, \ldots$ is much lower than the variance of the noise of an observation W of μ.

Now assume that μ_t is a vector with element μ_{tx}, where x will allow us to capture the performance of different drugs, paths through a network, people doing a job, or the price of a product. Let $\bar{\mu}_{tx}$ be the estimate of μ_{tx} at time t. Let Σ_t be the covariance matrix at time t, with element $\Sigma_{txx'} = Cov^n(\mu_{tx}, \mu_{tx'})$. This means we can write the distribution of μ_t as

$$\mu_t \sim N(\bar{\mu}_t, \Sigma_t).$$

This is the posterior distribution of μ_t, which is to say the distribution of μ_t given our prior observations $W_1, \ldots, W_t$, and our prior $N(\bar{\mu}_0, \sigma_0)$. Let Σ^μ be the covariance matrix for the random variable ε^μ describing the evolution of μ. The *predictive distribution* is the distribution of μ_{t+1} given μ_t, which we write as

$$\mu_{t+1}|\mu_t \sim N(\bar{\mu}_t, \tilde{\Sigma}^\mu_t),$$

where

$$\tilde{\Sigma}^\mu_t = \Sigma_t + \Sigma^\mu.$$

Let e_{t+1} be the error in a vector of observations W_{t+1} given by

$$e_{t+1} = W_{t+1} - \bar{\mu}_t.$$

Let Σ^ε be the covariance matrix for e_{t+1}. The updated mean and covariance is computed using

$$\begin{aligned}
\bar{\mu}_{t+1} &= \bar{\mu}_t + \tilde{\Sigma}^\mu_t\big(\Sigma^\varepsilon + \tilde{\Sigma}^\mu_t\big)^{-1} e_{t+1}, \\
\Sigma_{t+1} &= \tilde{\Sigma}^\mu_t - \tilde{\Sigma}^\mu_t\big(\Sigma^\varepsilon + \tilde{\Sigma}^\mu_t\big)\tilde{\Sigma}^\mu_t.
\end{aligned}$$

3.11.2 Nonstationary learning II - Transient truth

A more general, but slightly more complex model, allows for predictable changes in θ_t. For example, we may know that θ_t is growing over time (perhaps θ_t is related to age or the population size), or we may be modeling variations in solar energy and have to capture the rising and setting of the sun.

We assume that μ_t is a vector with element x. Now assume we have a diagonal matrix M_t with factors that govern the predictable change in μ_t, allowing us to write the evolution of μ_t as

$$\mu_{t+1} = M_t\mu_t + \delta_{t+1}.$$

The evolution of the covariance matrix Σ_t becomes

$$\tilde{\Sigma}_t = M_t\Sigma_t M_t + \Sigma^\delta.$$

Now the evolution of the estimates of the mean and covariance matrix $\bar{\mu}_t$ and Σ_t are given by

$$\begin{aligned}
\bar{\mu}_{t+1} &= M_t\bar{\mu}_t + \tilde{\Sigma}_t\big(\Sigma^\varepsilon + \tilde{\Sigma}_t\big)^{-1} e_{t+1}, \\
\Sigma_{t+1} &= \tilde{\Sigma}_t - \tilde{\Sigma}_t\big(\Sigma^\varepsilon + \tilde{\Sigma}_t\big)\tilde{\Sigma}_t.
\end{aligned}$$

Note there is no change in the formula for Σ_{t+1} since M_t is built into $\tilde{\Sigma}_t$.

3.11.3 Learning processes

There are many settings where we know that a process is improving over time up to an unknown limit. We refer to these as *learning processes* since we are modeling a process that learns as it progresses. Examples of learning processes are:

■ **EXAMPLE 3.12**

We have to choose a new basketball player x and then watch him improve as he gains playing time.

■ **EXAMPLE 3.13**

We observe the reduction in blood sugar due to diabetes medication x for a patient who has to adapt to the drug.

■ **EXAMPLE 3.14**

We are testing an algorithm where x are the parameters of the algorithm. The algorithm may be quite slow, so we have to project how good the final solution will be.

We model our process by assuming that observations come from

$$W_x^n = \mu_x^n + \varepsilon^n. \tag{3.63}$$

where the true mean μ_x^n rises according to.

$$\mu_x^n(\theta) = \theta_x^s + [\theta_x^\ell - \theta_x^s][1 - e^{-n\theta_x^r}]. \tag{3.64}$$

Here, θ_x^s is the expected starting point at $n = 0$, while θ_x^ℓ is the limiting value as $n \to \infty$. The parameter θ_x^r controls the rate at which the mean approaches θ_x^ℓ. Let $\theta = (\theta^s, \theta^\ell, \theta^r)$ be the vector of unknown parameters.

If we fix θ^r, then $\mu_x^n(\theta)$ is linear in θ^s and θ^ℓ, allowing us to use our equations for recursive least squares for linear models that we presented in section 3.8. This will produce estimates $\bar{\theta}^{s,n}(\theta^r)$ and $\bar{\theta}^{\ell,n}(\theta^r)$ for each possible value of θ^r.

To handle the one nonlinear parameter θ^r, assume that we discretize this parameter into the values $\theta_1^r, \ldots, \theta_K^r$. Let $p_k^{r,n}$ be the probability that $\theta^r = \theta_k^r$, which can be shown to be given by

$$p_k^{r,n} = \frac{L_k^n}{\sum_{k'=1}^K L_{k'}^n}$$

where L_k^n is the likelihood that $\theta^r = \theta_k^r$ which is given by

$$L_k^n \propto e^{-\left(\frac{W^{n+1} - \bar{\mu}_x^n}{\sigma_\varepsilon}\right)^2},$$

where σ_ε^2 is the variance of ε. This now allows us to write

$$\bar{\mu}_x^n(\theta) = \sum_{k=1}^K p_k^{r,n} \bar{\mu}_x^n(\theta|\theta^r).$$

This approach provides us with conditional point estimates and variances of $\bar{\theta}^{s,n}(\theta^r)$, $\bar{\theta}^{\ell,n}(\theta^r)$ for each θ^r, along with the distribution $p^{r,n}$ for θ^r.

3.12 THE CURSE OF DIMENSIONALITY

There are many applications where state variables have multiple, possibly continuous dimensions. In some applications, the number of dimensions can number in the millions or larger (see section 2.3.4). Some examples are

---

EXAMPLE 3.15

An unmanned aerial vehicle may be described by location (three dimensions), velocity (three dimensions), in addition to fuel level. All dimensions are continuous.

EXAMPLE 3.16

A utility is trying to plan the amount of energy that should be put in storage as a function of the wind history (six hourly measurements), the history of electricity spot prices (six measurements), and the demand history (six measurements).

EXAMPLE 3.17

A trader is designing a policy for selling an asset that is priced against a basket of 20 securities, creating a 20-dimensional state variable.

EXAMPLE 3.18

A medical patient can be described by several thousand characteristics, beginning with basic information such as age, weight, gender, but extending to lifestyle variables (diet, smoking, exercise) to an extensive array of variables describing someone's medical history.

Each of these problems has a multi-dimensional state vector, and in all but the last example the dimensions are continuous. If we have 10 dimensions, and discretize each dimension into 100 elements, our input vector x (which might be a state) is $100^{10} = 10^{20}$ which is clearly a very large number. A reasonable strategy might be to aggregate. Instead of discretizing each dimension into 100 elements, what if we discretize into 5 elements? Now our state space is $5^{10} = 9.76 \times 10^6$, or almost 10 million states. Much smaller, but still quite large. Figure 3.13 illustrates the growth in the state space with the number of dimensions.

Each of our examples explode with the number of dimensions because we are using a lookup table representation for our function. It is important to realize that the curse of dimensionality is tied to the use of lookup tables. The other approximation architectures avoid the curse, but they do so by assuming structure such as a parametric form (linear or nonlinear).

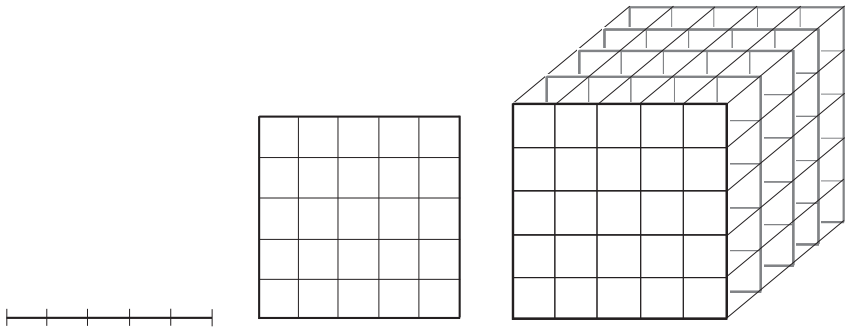


Figure 3.13 Illustration of the effect of higher dimensions on the number of grids in an aggregated state space.

Approximating high-dimensional functions is fundamentally intractable without exploiting structure. Beware of anyone claiming to "solve the curse of dimensionality." Pure lookup tables (which make no structural assumptions) are typically limited to four or five dimensions (depending on the number of values each dimension can take). However, we can handle thousands, even millions, of dimensions if we are willing to live with a linear model with separable, additive basis functions.

We can improve the accuracy of a linear model by adding features (basis functions) to our model. For example, if we use a second order parametric representation, we might approximate a two-dimensional function using

$$F(x) \approx \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_{11} x_1^2 + \theta_{22} x_2^2 + \theta_{12} x_1 x_2.$$

If we have N dimensions, the approximation would look like

$$F(x) \approx \theta_0 + \sum_{i=1}^{N} \theta_i x_i + \sum_{i=1}^{N} \sum_{j=1}^{N} \theta_{ij} x_i x_j,$$

which means we have to estimate $1 + N + N^2$ parameters. As N grows, this grows very quickly, and this is only a second order approximation. If we allow N^{th} order interactions, the approximation would look like

$$F(x) \approx \theta_0 + \sum_{i=1}^{N} \theta_i x_i + \sum_{i_1=1}^{N} \sum_{i_2=1}^{N} \theta_{i_1 i_2} x_{i_1} x_{i_2} + \sum_{i_1=1}^{N} \sum_{i_2=1}^{N} \cdots \sum_{i_N=1}^{N} \theta_{i_1, i_2, \ldots, i_N} x_{i_1} x_{i_2} \cdots x_{i_N}.$$

The number of parameters we now have to estimate is given by $1+N+N^2+N^3+\ldots+N^N$. Not surprisingly, this becomes intractable even for relatively small values of N.

The problem follows us if we were to use kernel regression, where an estimate of a function at a point s can be estimated from a series of observations $(\hat{f}^i, x^i)_{i=1}^N$ using

$$F(x) \approx \frac{\sum_{i=1}^{N} \hat{f}^i k(x, x^i)}{\sum_{i=1}^{N} k(x, x^i)}$$

where $k(x, x^i)$ might be the Gaussian kernel

$$k(x, x^i) = e^{-\frac{\|x - x^i\|^2}{b}}$$

where b is a bandwidth. Kernel regression is effectively a soft form of the aggregation depicted in figure 3.13(c). The problem is that we would have to choose a bandwidth that covers most of the data to get a statistically reliable estimate of a single point.

To see this, imagine that our observations are uniformly distributed in an N-dimensional cube that measures 1.0 on each side, which means it has a volume of 1.0. If we carve out an N-dimensional cube that measures .5 on a side, then this would capture 12.5 percent of the observations in a 3-dimensional cube, and 0.1 percent of the observations in a 10-dimensional cube. If we would like to choose a cube that captures $\eta = .1$ of our cube, we would need a cube that measures $r = \eta^{1/N} = .1^{1/10} = .794$, which means that our cube is covering almost 80 percent of the range of each input dimension.

The problem is that we have a multidimensional function, and we are trying to capture the joint behavior of all N dimensions. If we are willing to live with separable approximations, then we can scale to very large number of dimensions. For example, the approximation

$$F(x) \approx \theta_0 + \sum_{i=1}^{N} \theta_{1i} x_i + \sum_{i=1}^{N} \theta_{2i} x_i^2,$$

captures quadratic behavior but without any cross terms. The number of parameters is $1+2N$, which means we may be able to handle very high-dimensional problems. However, we lose the ability to handle interactions between different dimensions.

Kernel regression, along with essentially all nonparametric methods, is basically a fancy form of lookup table. Since these methods do not assume any underlying structure, they depend on capturing the local behavior of a function. The concept of "local," however, breaks down in high dimensions, where by "high" we typically mean four or five or more.

3.13 DESIGNING APPROXIMATION ARCHITECTURES IN ADAPTIVE LEARNING

Most solution methods in stochastic optimization are adaptive, which means that the data is arriving over time as a sequence of inputs x^n and observations $\hat{f}^{n+1}$. With each observation, we have to update our estimate of whatever function we are approximating, which might be the objective function $\mathbb{E}F(x, W)$, a value function $V(s)$, a policy $X^\pi(s)$, or a transition function $S^M(s, x, W)$. This entire chapter has focused on adaptive learning, but in the context where we used a fixed model and just adapt the parameters to produce the best fit.

Adaptive learning means that we have to start with small datasets (sometimes no data at all), and then adapt as new decisions and observations arrive. This raises a challenge we have not addressed above: we need to do more than just update a parameter vector θ^n with new data to produce θ^{n+1}. Instead, we need to update the architecture of the function we are trying to estimate. Said differently, the dimensionality of θ^n (or at least the set of nonzero elements of θ^n) will need to change as we acquire more data.

A key challenge with any statistical learning problem is designing a function that strikes the right tradeoff between the dimensionality of the function and the amount of data available for approximating the function. For a batch problem, we can use powerful tools such as regularization (see equation (3.39)) for identifying models that have the right number of variables given the available data. But this only works for batch estimation, where the size of the dataset is fixed.

As of this writing, additional research is needed to create the tools that can help to identify not just the best parameter vector θ^n, but the structure of the function itself. One

technique that does this is hierarchical aggregation which we presented in the context of lookup tables in section 3.6. This is a powerful methodology that adaptively adjusts from a low dimensional representation (that is, estimates of the function at a high level of aggregation) to higher dimensional representations, which is accomplished by putting higher weights on the more disaggregate estimates. However, lookup table belief models are limited to relatively low dimensional problems.

3.14 WHY DOES IT WORK?**

3.14.1 Derivation of the recursive estimation equations

Here we derive the recursive estimation equations given by equations (3.41)-(3.45). To begin, we note that the matrix $(X^n)^T X^n$ is an $I+1$ by $I+1$ matrix where the element for row i, column j is given by

$$[(X^n)^T X^n]_{i,j} = \sum_{m=1}^{n} x_i^m x_j^m.$$

This term can be computed recursively using

$$[(X^n)^T X^n]_{i,j} = \sum_{m=1}^{n-1} (x_i^m x_j^m) + x_i^n x_j^n.$$

In matrix form, this can be written

$$[(X^n)^T X^n] = [(X^{n-1})^T X^{n-1}] + x^n (x^n)^T.$$

Keeping in mind that x^n is a column vector, $x^n(x^n)^T$ is an $I+1$ by $I+1$ matrix formed by the cross products of the elements of x^n. We now use the Sherman-Morrison formula (see section 3.14.2 for a derivation) for updating the inverse of a matrix

$$[A + uu^T]^{-1} = A^{-1} - \frac{A^{-1}uu^T A^{-1}}{1 + u^T A^{-1} u},$$

where A is an invertible $n \times n$ matrix, and u is an n-dimensional column vector. Applying this formula to our problem, we obtain

$$\begin{aligned}
[(X^n)^T X^n]^{-1} &= [(X^{n-1})^T X^{n-1} + x^n (x^n)^T]^{-1} \\
&= [(X^{n-1})^T X^{n-1}]^{-1} \\
&\quad - \frac{[(X^{n-1})^T X^{n-1}]^{-1} x^n (x^n)^T [(X^{n-1})^T X^{n-1}]^{-1}}{1 + (x^n)^T [(X^{n-1})^T X^{n-1}]^{-1} x^n}. \qquad (3.65)
\end{aligned}$$

The term $(X^n)^T Y^n$ can also be updated recursively using

$$(X^n)^T Y^n = (X^{n-1})^T Y^{n-1} + x^n (y^n). \qquad (3.66)$$

To simplify the notation, let

$$\begin{aligned}
M^n &= [(X^n)^T X^n]^{-1}, \\
\gamma^n &= 1 + (x^n)^T [(X^{n-1})^T X^{n-1}]^{-1} x^n.
\end{aligned}$$

This simplifies our inverse updating equation (3.65) to

$$M^n = M^{n-1} - \frac{1}{\gamma^n}(M^{n-1}x^n(x^n)^T M^{n-1}).$$

Recall that

$$\bar{\theta}^n = [(X^n)^T X^n]^{-1}(X^n)^T Y^n. \tag{3.67}$$

Combining (3.67) with (3.65) and (3.66) gives

$$\begin{aligned}
\bar{\theta}^n &= [(X^n)^T X^n]^{-1}(X^n)^T Y^n \\
&= \left(M^{n-1} - \frac{1}{\gamma^n}(M^{n-1}x^n(x^n)^T M^{n-1})\right)\left((X^{n-1})^T Y^{n-1} + x^n y^n\right), \\
&= M^{n-1}(X^{n-1})^T Y^{n-1} \\
&\quad -\frac{1}{\gamma^n}M^{n-1}x^n(x^n)^T M^{n-1}\left[(X^{n-1})^T Y^{n-1} + x^n y^n\right] + M^{n-1}x^n y^n.
\end{aligned}$$

We can start to simplify by using $\bar{\theta}^{n-1} = M^{n-1}(X^{n-1})^T Y^{n-1}$. We are also going to bring the term $x^n M^{n-1}$ inside the square brackets. Finally, we are going to bring the last term $M^{n-1}x^n y^n$ inside the brackets by taking the coefficient $M^{n-1}x^n$ outside the brackets and multiplying the remaining y^n by the scalar $\gamma^n = 1 + (x^n)^T M^{n-1}x^n$, giving us

$$\begin{aligned}
\bar{\theta}^n &= \bar{\theta}^{n-1} - \frac{1}{\gamma^n}M^{n-1}x^n\left[(x^n)^T(M^{n-1}(X^{n-1})^T Y^{n-1})\right. \\
&\quad \left. + (x^n)^T M^{n-1}x^n y^n - (1 + (x^n)^T M^{n-1}x^n)y^n\right].
\end{aligned}$$

Again, we use $\bar{\theta}^{n-1} = M^{n-1}(X^{n-1})^T Y^{n-1}$ and observe that there are two terms $(x^n)^T M^{n-1}x^n y^n$ that cancel, leaving

$$\bar{\theta}^n = \bar{\theta}^{n-1} - \frac{1}{\gamma^n}M^{n-1}x^n\left((x^n)^T\bar{\theta}^{n-1} - y^n\right).$$

We note that $(\bar{\theta}^{n-1})^T x^n$ is our prediction of y^n using the parameter vector from iteration $n-1$ and the explanatory variables x^n. y^n is, of course, the actual observation, so our error is given by

$$\hat{\varepsilon}^n = y^n - (\bar{\theta}^{n-1})^T x^n.$$

Let

$$H^n = -\frac{1}{\gamma^n}M^{n-1}.$$

We can now write our updating equation using

$$\bar{\theta}^n = \bar{\theta}^{n-1} - H^n x^n \hat{\varepsilon}^n. \tag{3.68}$$

3.14.2 The Sherman-Morrison updating formula

The Sherman-Morrison matrix updating formula (also known as the Woodbury formula or the Sherman-Morrison-Woodbury formula) assumes that we have a matrix A and that we

are going to update it with the outer product of the column vector u to produce the matrix B, given by

$$B = A + uu^T. \tag{3.69}$$

Pre-multiply by B^{-1} and post-multiply by A^{-1}, giving

$$A^{-1} = B^{-1} + B^{-1}uu^TA^{-1}. \tag{3.70}$$

Post-multiply by u

$$\begin{aligned} A^{-1}u &= B^{-1}u + B^{-1}uu^TA^{-1}u \\ &= B^{-1}u\left(1 + u^TA^{-1}u\right). \end{aligned}$$

Note that $u^TA^{-1}u$ is a scalar. Divide through by $\left(1 + u^TA^{-1}u\right)$

$$\frac{A^{-1}u}{(1 + u^TA^{-1}u)} = B^{-1}u.$$

Now post-multiply by u^TA^{-1}

$$\frac{A^{-1}uu^TA^{-1}}{(1 + u^TA^{-1}u)} = B^{-1}uu^TA^{-1}. \tag{3.71}$$

Equation (3.70) gives us

$$B^{-1}uu^TA^{-1} = A^{-1} - B^{-1}. \tag{3.72}$$

Substituting (3.72) into (3.71) gives

$$\frac{A^{-1}uu^TA^{-1}}{(1 + u^TA^{-1}u)} = A^{-1} - B^{-1}. \tag{3.73}$$

Solving for B^{-1} gives us

$$\begin{aligned} B^{-1} &= [A + uu^T]^{-1} \\ &= A^{-1} - \frac{A^{-1}uu^TA^{-1}}{(1 + u^TA^{-1}u)}, \end{aligned}$$

which is the desired formula.

3.14.3 Correlations in hierarchical estimation

It is possible to derive the optimal weights for the case where the statistics $\bar{v}_s^{(g)}$ are not independent. In general, if we are using a hierarchical strategy and have $g' > g$ (which means that aggregation g' is more aggregate than g), then the statistic $\bar{v}_s^{(g',n)}$ is computed using observations $\hat{v}_s^n$ that are also used to compute $\bar{v}_s^{(g,n)}$.

We begin by defining

$\mathcal{N}_s^{(g,n)}$ = The set of iterations n where $G^g(\hat{s}^n) = G^g(s)$ (that is, $\hat{s}^n$ aggregates to the same state as s).

$N_s^{(g,n)} = |\mathcal{N}_s^{(g,n)}|$

$\bar{\varepsilon}_s^{(g,n)}$ = An estimate of the average error when observing state $s = G(\hat{s}^n)$.

$$= \frac{1}{N_s^{(g,n)}} \sum_{n \in \mathcal{N}_s^{(g,n)}} \hat{\varepsilon}_s^{(g,n)}.$$

The average error $\bar{\varepsilon}_s^{(g,n)}$ can be written

$$\begin{aligned}\bar{\varepsilon}_s^{(g,n)} &= \frac{1}{N_s^{(g,n)}}\left(\sum_{n\in\mathcal{N}_s^{(0,n)}}\varepsilon^n + \sum_{n\in\mathcal{N}_s^{(g,n)}\setminus\mathcal{N}_s^{(0,n)}}\varepsilon^n\right)\\ &= \frac{N_s^{(0,n)}}{N_s^{(g,n)}}\bar{\varepsilon}_s^{(0)} + \frac{1}{N_s^{(g,n)}}\sum_{n\in\mathcal{N}_s^{(g,n)}\setminus\mathcal{N}_s^{(0,n)}}\varepsilon^n. \qquad (3.74)\end{aligned}$$

This relationship shows us that we can write the error term at the higher level of aggregation g' as a sum of a term involving the errors at the lower level of aggregation g (for the same state s) and a term involving errors from other states s'' where $G^{g'}(s'') = G^{g'}(s)$, given by

$$\begin{aligned}\bar{\varepsilon}_s^{(g',n)} &= \frac{1}{N_s^{(g',n)}}\left(\sum_{n\in\mathcal{N}_s^{(g,n)}}\varepsilon^n + \sum_{n\in\mathcal{N}_s^{(g',n)}\setminus\mathcal{N}_s^{(g,n)}}\varepsilon^n\right)\\ &= \frac{1}{N_s^{(g',n)}}\left(N_s^{(g,n)}\frac{\sum_{n\in\mathcal{N}_s^{(g,n)}}\varepsilon^n}{N_s^{(g,n)}} + \sum_{n\in\mathcal{N}_s^{(g',n)}\setminus\mathcal{N}_s^{(g,n)}}\varepsilon^n\right)\\ &= \frac{N_s^{(g,n)}}{N_s^{(g',n)}}\bar{\varepsilon}_s^{(g,n)} + \frac{1}{N_s^{(g',n)}}\sum_{n\in\mathcal{N}_s^{(g',n)}\setminus\mathcal{N}_s^{(g,n)}}\varepsilon^n. \qquad (3.75)\end{aligned}$$

We can overcome this problem by rederiving the expression for the optimal weights. For a given (disaggregate) state s, the problem of finding the optimal weights $(w_s^{(g,n)})_{g\in\mathcal{G}}$ is stated by

$$\min_{w_s^{(g,n)},g\in\mathcal{G}} \mathbb{E}\left[\frac{1}{2}\left(\sum_{g\in\mathcal{G}} w_s^{(g,n)}\cdot\bar{v}_s^{(g,n)} - \nu_s^{(g,n)}\right)^2\right] \qquad (3.76)$$

subject to

$$\sum_{g\in\mathcal{G}} w_s^{(g,n)} = 1 \qquad (3.77)$$

$$w_s^{(g,n)} \geq 0, \quad g\in\mathcal{G}. \qquad (3.78)$$

Let

$$\begin{aligned}\bar{\delta}_s^{(g,n)} &= \text{The error in the estimate } \bar{v}_s^{(g,n)} \text{ from the true value associated with attribute vector } s.\\ &= \bar{v}_s^{(g,n)} - \nu_s.\end{aligned}$$

The optimal weights are computed using the following theorem:

Theorem 1. *For a given attribute vector, s, the optimal weights, $w_s^{(g,n)}$, $g\in\mathcal{G}$, where the individual estimates are correlated by way of a tree structure, are given by solving the*

following system of linear equations in (w, λ)*:*

$$\sum_{g\in\mathcal{G}} w_s^{(g,n)} \mathbb{E}\left[\bar{\delta}_s^{(g,n)}\bar{\delta}_s^{(g',n)}\right] - \lambda = 0 \quad \forall\ g' \in \mathcal{G} \tag{3.79}$$

$$\sum_{g\in\mathcal{G}} w_s^{(g,n)} = 1 \tag{3.80}$$

$$w_s^{(g,n)} \geq 0 \quad \forall\ g \in \mathcal{G}. \tag{3.81}$$

Proof: The proof is not too difficult and it illustrates how we obtain the optimal weights. We start by formulating the Lagrangian for the problem formulated in (3.76)-(3.78), which gives us

$$\begin{aligned} L(w,\lambda) &= \mathbb{E}\left[\frac{1}{2}\left(\sum_{g\in\mathcal{G}} w_s^{(g,n)}\cdot\bar{v}_s^{(g,n)} - \nu_s^{(g,n)}\right)^2\right] + \lambda\left(1 - \sum_{g\in\mathcal{G}} w_s^{(g,n)}\right) \\ &= \mathbb{E}\left[\frac{1}{2}\left(\sum_{g\in\mathcal{G}} w_s^{(g,n)}\left(\bar{v}_s^{(g,n)} - \nu_s^{(g,n)}\right)\right)^2\right] + \lambda\left(1 - \sum_{g\in\mathcal{G}} w_s^{(g,n)}\right). \end{aligned}$$

The first order optimality conditions are

$$\mathbb{E}\left[\sum_{g\in\mathcal{G}} w_s^{(g,n)}\left(\bar{v}_s^{(g,n)} - \nu_s^{(g,n)}\right)\left(\bar{v}_s^{(g',n)} - \nu_s^{(g,n)}\right)\right] - \lambda = 0 \quad \forall\ g' \in \mathcal{G} \tag{3.82}$$

$$\sum_{g\in\mathcal{G}} w_s^{(g,n)} - 1 = 0. \tag{3.83}$$

To simplify equation (3.82), we note that,

$$\begin{aligned} \mathbb{E}\left[\sum_{g\in\mathcal{G}} w_s^{(g,n)}\left(\bar{v}_s^{(g,n)} - \nu_s^{(g,n)}\right)\left(\bar{v}_s^{(g',n)} - \nu_s^{(g,n)}\right)\right] &= \mathbb{E}\left[\sum_{g\in\mathcal{G}} w_s^{(g,n)}\bar{\delta}_s^{(g,n)}\bar{\delta}_s^{(g',n)}\right] \\ &= \sum_{g\in\mathcal{G}} w_s^{(g,n)}\mathbb{E}\left[\bar{\delta}_s^{(g,n)}\bar{\delta}_s^{(g',n)}\right]. \end{aligned} \tag{3.84}$$

Combining equations (3.82) and (3.84) gives us equation (3.79) which completes the proof. □

Finding the optimal weights that handle the correlations between the statistics at different levels of aggregation requires finding $\mathbb{E}\left[\bar{\delta}_s^{(g,n)}\bar{\delta}_s^{(g',n)}\right]$. We are going to compute this expectation by conditioning on the set of attributes $\hat{s}^n$ that are sampled. This means that our expectation is defined over the outcome space Ω^ε. Let $N_s^{(g,n)}$ be the number of observations of state s at aggregation level g. The expectation is computed using:

Proposition 3.14.1. *The coefficients of the weights in equation (3.80) can be expressed as follows:*

$$\mathbb{E}\left[\bar{\delta}_s^{(g,n)}\bar{\delta}_s^{(g',n)}\right] = \mathbb{E}\left[\bar{\beta}_s^{(g,n)}\bar{\beta}_s^{(g',n)}\right] + \frac{N_s^{(g,n)}}{N_s^{(g',n)}}\mathbb{E}\left[\bar{\varepsilon}_s^{(g,n)^2}\right] \quad \forall g \leq g' \text{ and } g, g' \in \mathcal{G}. \tag{3.85}$$

The proof is given in section 3.14.4.

Now consider what happens when we make the assumption that the measurement error ε^n is independent of the attribute being sampled, $\hat{s}^n$. We do this by assuming that the variance of the measurement error is a constant given by ${\sigma_\varepsilon}^2$. This gives us the following result:

Corollary 3.14.1. *For the special case where the statistical noise in the measurement of the values is independent of the attribute vector sampled, equation (3.85) reduces to*

$$\mathbb{E}\left[\bar{\delta}_s^{(g,n)}\bar{\delta}_s^{(g',n)}\right] = \mathbb{E}\left[\bar{\beta}_s^{(g,n)}\bar{\beta}_s^{(g',n)}\right] + \frac{\sigma_\varepsilon^2}{N_s^{(g',n)}}. \tag{3.86}$$

For the case where $g = 0$ (the most disaggregate level), we assume that $\beta_s^{(0)} = 0$ which gives us

$$\mathbb{E}\left[\bar{\beta}_s^{(0,n)}\bar{\beta}_s^{(g',n)}\right] = 0.$$

This allows us to further simplify (3.86) to obtain

$$\mathbb{E}\left[\bar{\delta}_s^{(0,n)}\bar{\delta}_s^{(g',n)}\right] = \frac{\sigma_\varepsilon^2}{N_s^{(g',n)}}. \tag{3.87}$$

3.14.4 Proof of Proposition 3.14.1

We start by defining

$$\bar{\delta}_s^{(g,n)} = \bar{\beta}_s^{(g,n)} + \bar{\varepsilon}_s^{(g,n)}. \tag{3.88}$$

Equation (3.88) gives us

$$\begin{aligned}
\mathbb{E}\left[\bar{\delta}_s^{(g,n)}\bar{\delta}_s^{(g',n)}\right] &= \mathbb{E}\left[(\bar{\beta}_s^{(g,n)} + \bar{\varepsilon}_s^{(g,n)})(\bar{\beta}_s^{(g',n)} + \bar{\varepsilon}_s^{(g',n)})\right] \\
&= \mathbb{E}\left[\bar{\beta}_s^{(g,n)}\bar{\beta}_s^{(g',n)} + \bar{\beta}_s^{(g',n)}\bar{\varepsilon}_s^{(g,n)} + \bar{\beta}_s^{(g,n)}\bar{\varepsilon}_s^{(g',n)} + \bar{\varepsilon}_s^{(g,n)}\bar{\varepsilon}_s^{(g',n)}\right] \\
&= \mathbb{E}\left[\bar{\beta}_s^{(g,n)}\bar{\beta}_s^{(g',n)}\right] + \mathbb{E}\left[\bar{\beta}_s^{(g',n)}\bar{\varepsilon}_s^{(g,n)}\right] + \mathbb{E}\left[\bar{\beta}_s^{(g,n)}\bar{\varepsilon}_s^{(g',n)}\right] \\
&\quad + \mathbb{E}\left[\bar{\varepsilon}_s^{(g,n)}\bar{\varepsilon}_s^{(g',n)}\right].
\end{aligned} \tag{3.89}$$

We note that

$$\mathbb{E}\left[\bar{\beta}_s^{(g',n)}\bar{\varepsilon}_s^{(g,n)}\right] = \bar{\beta}_s^{(g',n)}\mathbb{E}\left[\bar{\varepsilon}_s^{(g,n)}\right] = 0.$$

Similarly

$$\mathbb{E}\left[\bar{\beta}_s^{(g,n)}\bar{\varepsilon}_s^{(g',n)}\right] = 0.$$

This allows us to write equation (3.89) as

$$\mathbb{E}\left[\bar{\delta}_s^{(g,n)}\bar{\delta}_s^{(g',n)}\right] = \mathbb{E}\left[\bar{\beta}_s^{(g,n)}\bar{\beta}_s^{(g',n)}\right] + \mathbb{E}\left[\bar{\varepsilon}_s^{(g,n)}\bar{\varepsilon}_s^{(g',n)}\right]. \tag{3.90}$$

We start with the second term on the right-hand side of equation (3.90). This term can be written as

$$\begin{aligned}\mathbb{E}\left[\bar{\varepsilon}_s^{(g,n)}\bar{\varepsilon}_s^{(g',n)}\right] &= \mathbb{E}\left[\bar{\varepsilon}_s^{(g,n)}\cdot\frac{N_s^{(g,n)}}{N_s^{(g')}}\bar{\varepsilon}_s^{(g,n)}\right]+\mathbb{E}\left[\bar{\varepsilon}_s^{(g,n)}\cdot\frac{1}{N_s^{(g')}}\sum_{n\in\mathcal{N}_s^{(g',n)}\setminus\mathcal{N}_s^{(g,n)}}\varepsilon^n\right]\\ &= \frac{N_s^{(g,n)}}{N_s^{(g')}}\mathbb{E}\left[\bar{\varepsilon}_s^{(g,n)}\bar{\varepsilon}_s^{(g,n)}\right]+\frac{1}{N_s^{(g')}}\underbrace{\mathbb{E}\left[\bar{\varepsilon}_s^{(g,n)}\cdot\sum_{n\in\mathcal{N}_s^{(g',n)}\setminus\mathcal{N}_s^{(g,n)}}\varepsilon^n\right]}_{I}.\end{aligned}$$

The term I can be rewritten using

$$\begin{aligned}\mathbb{E}\left[\bar{\varepsilon}_s^{(g,n)}\cdot\sum_{n\in\mathcal{N}_s^{(g',n)}\setminus\mathcal{N}_s^{(g,n)}}\varepsilon^n\right] &= \mathbb{E}\left[\bar{\varepsilon}_s^{(g,n)}\right]\mathbb{E}\left[\sum_{n\in\mathcal{N}_s^{(g',n)}\setminus\mathcal{N}_s^{(g,n)}}\varepsilon^n\right],\\ &= 0\end{aligned}$$

which means

$$\mathbb{E}\left[\bar{\varepsilon}_s^{(g,n)}\bar{\varepsilon}_s^{(g',n)}\right] = \frac{N_s^{(g,n)}}{N_s^{(g')}}\mathbb{E}\left[\bar{\varepsilon}_s^{(g)^2}\right]. \tag{3.91}$$

Combining (3.90) and (3.91) proves the proposition. □

The second term on the right-hand side of equation (3.91) can be further simplified using,

$$\begin{aligned}\mathbb{E}\left[\bar{\varepsilon}_s^{(g)^2}\right] &= \mathbb{E}\left[\left(\frac{1}{N_s^{(g,n)}}\sum_{n\in\mathcal{N}_s^{(g,n)}}\varepsilon^n\right)^2\right],\quad \forall\ g'\in\mathcal{G}\\ &= \frac{1}{\left(N_s^{(g,n)}\right)^2}\sum_{m\in\mathcal{N}_s^{(g,n)}}\sum_{n\in\mathcal{N}_s^{(g,n)}}\mathbb{E}\left[\varepsilon^m\varepsilon^n\right]\\ &= \frac{1}{\left(N_s^{(g,n)}\right)^2}\sum_{n\in\mathcal{N}_s^{(g,n)}}\mathbb{E}\left[(\varepsilon^n)^2\right]\\ &= \frac{1}{\left(N_s^{(g,n)}\right)^2}N_s^{(g,n)}{\sigma_\varepsilon}^2\\ &= \frac{\sigma_\varepsilon^2}{N_s^{(g,n)}}\end{aligned} \tag{3.92}$$

Combining equations (3.85), (3.91) and (3.92) gives us the result in equation (3.86). □

3.15 BIBLIOGRAPHIC NOTES

This chapter is primarily a tutorial into online (adaptive) learning. Readers looking to do serious algorithmic work should obtain a good statistical reference such as Bishop (2006)

and Hastie et al. (2009). The second reference can be downloaded from

http://www-stat.stanford.edu/~tibs/ElemStatLearn/.

Note that classical references in statistical learning tend to focus on batch learning, while we are primarily interested in online (or adaptive) learning.

Sections 3.6 - Aggregation has been a widely used technique in dynamic programming as a method to overcome the curse of dimensionality. Early work focused on picking a fixed level of aggregation (Whitt (1978), Bean et al. (1987)), or using adaptive techniques that change the level of aggregation as the sampling process progresses (Bertsekas & Castanon (1989), Mendelssohn (1982), Bertsekas & Tsitsiklis (1996)), but which still use a fixed level of aggregation at any given time. Much of the literature on aggregation has focused on deriving error bounds (Zipkin (1980)). For a good discussion of aggregation as a general technique in modeling, see Rogers et al. (1991). The material in section 3.6.3 is based on George et al. (2008) and Powell & George (2006). LeBlanc & Tibshirani (1996) and Yang (2001) provide excellent discussions of mixing estimates from different sources. For a discussion of soft state aggregation, see Singh et al. (1995). Section 3.5 on bias and variance is based on Powell & George (2006).

Section 3.7 - Basis functions have their roots in the modeling of physical processes. A good introduction to the field from this setting is Heuberger et al. (2005). Schweitzer & Seidmann (1985) describes generalized polynomial approximations for Markov decision processes for use in value iteration, policy iteration and the linear programming method. Menache et al. (2005) discusses basis function adaptations in the context of reinforcement learning. For a very nice discussion of the use of basis functions in approximate dynamic programming, see Tsitsiklis & Roy (1996) and Van Roy (2001). Tsitsiklis & Van Roy (1997) proves convergence of iterative stochastic algorithms for fitting the parameters of a regression model when the policy is held fixed. For section 17.6.1, the first use of approximate dynamic programming for evaluating an American call option is given in Longstaff & Schwartz (2001), but the topic has been studied for decades (see Taylor (1967)). Tsitsiklis & Van Roy (2001) also provide an alternative ADP algorithm for American call options. Clement et al. (2002) provides formal convergence results for regression models used to price American options. This presentation on the geometric view of basis functions is based on Tsitsiklis & Van Roy (1997).

Section 3.10 - An excellent introduction to continuous approximation techniques is given in Judd (1998) in the context of economic systems and computational dynamic programming. Ormoneit & Sen (2002) and Ormoneit & Glynn (2002) discuss the use of kernel-based regression methods in an approximate dynamic programming setting, providing convergence proofs for specific algorithmic strategies. For a thorough introduction to locally polynomial regression methods, see Fan & Gijbels (1996). An excellent discussion of a broad range of statistical learning methods can be found in Hastie et al. (2009). Bertsekas & Tsitsiklis (1996) provides an excellent discussion of neural networks in the context of approximate dynamic programming. Haykin (1999) presents a much more in-depth presentation of neural networks, including a chapter on approximate dynamic programming using neural networks. A very rich field of study has evolved around support vector machines and support

vector regression. For a thorough tutorial, see Smola & Schölkopf (2004). A shorter and more readable introduction is contained in chapter 12 of Hastie et al. (2009). Note that SVR does not lend itself readily to recursive updating, which we suspect will limit its usefulness in approximate dynamic programming.

Figure 3.8 was created by Larry Thul.

Section 3.12 - See Hastie et al. (2009), section 2.5, for a very nice discussion of the challenges of approximating high-dimensional functions.

Section 3.14.2 - The Sherman-Morrison updating formulas are given in a number of references, such as L. & Soderstrom (1983) and Golub & Loan (1996).

EXERCISES

Review questions

3.1 What are the five classes of approximations that may arise in sequential decision problems?

3.2 When using lookup table models with independent observations, what are the belief state variables for frequentist and Bayesian beliefs?

3.3 What is the belief state for lookup tables with correlated beliefs, when using a Bayesian belief model?

3.4 This chapter is organized around three major classes of approximation architectures: lookup table, parametric and nonparametric, but some have argued that there should only be two classes: parametric and nonparametric. Justify your answer by presenting an argument why a lookup table can be properly modeled as a parametric model, and then a counter argument why a lookup table is more similar to a nonparametric model. [Hint: What is the defining characteristic of a nonparametric model? - see section 3.10.]

3.5 What is the belief state if you are doing recursive updating of a linear model?

3.6 A deep neural network is just a bigger neural network. So why are deep neural networks considered nonparametric models? After all they are just a nonlinear model with a very large number of parameters. Where would you put neural networks (all neural networks) in figure 3.1.

Computational exercises

3.7 Use equations (3.16) and (3.17) to update the mean vector with prior

$$\bar{\mu}^0 = \begin{bmatrix} 10 \\ 18 \\ 12 \end{bmatrix}.$$

Assume that we test alternative 3 and observe $W = 19$ and that our prior covariance matrix Σ^0 is given by

$$\Sigma^0 = \begin{bmatrix} 12 & 4 & 2 \\ 4 & 8 & 3 \\ 2 & 3 & 10 \end{bmatrix}.$$

Assume that $\lambda^W = 4$. Give $\bar{\mu}^1$ and Σ^1.

3.8 In a spreadsheet, create a 4×4 grid where the cells are numbered 1, 2, ..., 16 starting with the upper left-hand corner and moving left to right, as shown below. We are

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

going to treat each number in the cell as the mean of the observations drawn from that cell. Now assume that if we observe a cell, we observe the mean plus a random variable that is uniformly distributed between -1 and $+1$. Next define a series of aggregations where aggregation 0 is the disaggregate level, aggregation 1 divides the grid into four 2×2 cells, and aggregation 2 aggregates everything into a single cell. After n iterations, let $\bar{f}_s^{(g,n)}$ be the estimate of cell "s" at the n^{th} level of aggregation, and let

$$\bar{f}_s^n = \sum_{g \in \mathcal{G}} w_s^{(g)} \bar{f}_s^{(g,n)}$$

be your best estimate of cell s using a weighted aggregation scheme. Compute an overall error measure using

$$(\bar{\sigma}^2)^n = \sum_{s \in \mathcal{S}} (\bar{f}_s^n - \nu_s)^2,$$

where ν_s is the true value (taken from your grid) of being in cell s. Also let $w^{(g,n)}$ be the average weight after n iterations given to the aggregation level g when averaged over all cells at that level of aggregation (for example, there is only one cell for $w^{(2,n)}$). Perform 1000 iterations where at each iteration you randomly sample a cell and measure it with noise. Update your estimates at each level of aggregation, and compute the variance of your estimate with and without the bias correction.

(a) Plot $w^{(g,n)}$ for each of the three levels of aggregation at each iteration. Do the weights behave as you would expect? Explain.

(b) For each level of aggregation, set the weight given to that level equal to one (in other words, we are using a single level of aggregation) and plot the overall error as a function of the number of iterations.

(c) Add to your plot the average error when you use a weighted average, where the weights are determined by equation (3.32) without the bias correction.

(d) Finally add to your plot the average error when you used a weighted average, but now determine the weights by equation (3.33), which uses the bias correction.

(e) Repeat the above assuming that the noise is uniformly distributed between -5 and $+5$.

3.9 In this exercise you will use the equations in section 3.8.1 to update a linear model. Assume you have an estimate of a linear model given by

$$\begin{aligned}\bar{F}(x|\theta^0) &= \theta_0 + \theta_1\phi_1(x) + \theta_2\phi_2(x) \\ &= -12 + 5.2\phi_1 + 2.8\phi_2.\end{aligned}$$

Assume that the matrix B^0 is a 3×3 identity matrix. Assume the vector $\phi = (\phi_0 \ \phi_1 \ \phi_2) = (5 \ 15 \ 22)$ and that you observe $\hat{f}^1 = 90$. Give the updated regression vector θ^1.

Theory questions

3.10 Show that

$$\sigma_s^2 = (\sigma_s^2)^{(g)} + (\beta_s^{(g)})^2 \tag{3.93}$$

which breaks down the total variation in an estimate at a level of aggregation is the sum of the variation of the observation error plus the bias squared.

3.11 Show that $\mathbb{E}\left[\left(\bar{\mu}^{n-1} - \mu(n)\right)^2\right] = \lambda^{n-1}\sigma^2 + (\beta^n)^2$ (equation (3.24)). [Hint: Add and subtract $\mathbb{E}\bar{\mu}^{n-1}$ inside the expectation and expand.]

3.12 Show that $\mathbb{E}\left[\left(\bar{\theta}^{n-1} - \hat{\theta}^n\right)^2\right] = (1 + \lambda^{n-1})\sigma^2 + (\beta^n)^2$ (which proves equation 3.25). [Hint: See previous exercise.]

3.13 Derive the small sample form of the recursive equation for the variance given in (3.26). Recall that if

$$\bar{\mu}^n = \frac{1}{n}\sum_{m=1}^{n}\hat{\mu}^m$$

then an estimate of the variance of $\hat{\theta}$ is

$$\mathit{Var}[\hat{\mu}] = \frac{1}{n-1}\sum_{m=1}^{n}(\hat{\mu}^m - \bar{\mu}^n)^2.$$

Problem solving questions

3.14 Consider the problem where you are observing the number of arrivals Y^{n+1} which you believe are coming from a Poisson distribution with mean λ which is given by

$$Prob[Y^{n+1} = y|\lambda] = \frac{\lambda^y e^{-\lambda}}{\lambda!}.$$

where we assume $y = 0, 1, 2, \ldots$. Your problem is that you do not know what λ is, but you think it is one of $\{\lambda_1, \lambda_2, \ldots, \lambda_K\}$. Assume that after n observations of the number of arrivals Y, we have estimated the probability

$$p_k^n = Prob[\lambda = \lambda_k|Y^1, \ldots, Y^n].$$

Using the methods of section 3.9.2 for sampled belief models, write the expression for p_k^{n+1} given the observation Y^{n+1}. Your expression has to be in terms of p_k^n and the Poisson distribution above.

3.15 Bayes' theorem comes from the identity $P(A|B)P(B) = P(B|A)P(A)$ where A and B are probabilistic events. From this, we can

$$P(B|A) = \frac{P(A|B)P(B)}{P(A)}$$

Use this identity to derive equation (3.55) used for updating beliefs for sampled belief models. Clearly identify events A and B. [Hint: an equivalent form of Bayes theorem involves conditionining everything on a third event C, as in

$$P(B|A,C) = \frac{P(A|B,C)P(B|C)}{P(A|C)}$$

What is the event C in equation (3.55)?]

Diary problem

The diary problem is a single problem you chose (see chapter 1 for guidelines). Answer the following for your diary problem.

3.16 Review the different classes of approximations described in section 3.1.3, and identify examples of as many of these that may arise in your approximation.

Bibliography

Bean, J. C., Birge, J. R. & Smith, R. L. (1987), 'Aggregation in Dynamic Programming', *Operations Research* **35**, 215–220.

Bertsekas, D. P. & Castanon, D. A. (1989), 'Adaptive Aggregation Methods for Infinite Horizon Dynamic Programming', *IEEE Transactions on Automatic Control* **34**, 589–598.

Bertsekas, D. P. & Tsitsiklis, J. N. (1996), *Neuro-Dynamic Programming*, Athena Scientific, Belmont, MA.

Bishop, C. M. (2006), *Pattern Recognition and Machine Learning*, Springer, New York.

Clement, E., Lamberton, D. & Protter, P. (2002), 'An analysis of a least squares regression method for American option pricing', *Finance and Stochastics* **17**, 448–471.

Fan, J. & Gijbels, I. (1996), *Local Polynomial Modelling and Its Applications*, Chapman and Hall, London.

George, A., Powell, W. B. & Kulkarni, S. (2008), 'Value Function Approximation using Multiple Aggregation for Multiattribute Resource Management', *J. Machine Learning Research* pp. 2079–2111.

Golub, G. H. & Loan, C. F. V. (1996), *Matrix Computations*, John Hopkins University Press, Baltimore, MD.

Hastie, T. J., Tibshirani, R. J. & Friedman, J. H. (2009), *The elements of statistical learning : data mining, inference, and prediction*, Springer, New York.

Haykin, S. (1999), *Neural Networks: A comprehensive foundation*, Prentice Hall, Englewood Cliffs, N.J.

Heuberger, P. S. C., den Hov, P. M. J. V. & Wahlberg, B., eds (2005), *Modeling and Identification with Rational Orthogonal Basis Functions*, Springer, New York.

Judd, K. L. (1998), *Numerical Methods in Economics*, MIT Press.

L., l. & Soderstrom, T. (1983), *Theory and Practice of Recursive Identification*, MIT Press, Cambridge, MA.

LeBlanc, M. & Tibshirani, R. (1996), 'Combining estimates in regression and classification', *Journal of the American Statistical Association* **91**, 1641–1650.

Longstaff, F. A. & Schwartz, E. S. (2001), 'Valuing American options by simulation: A simple least squares approach', *The Review of Financial Studies* **14**(1), 113–147.

Menache, I., Mannor, S. & Shimkin, N. (2005), 'Basis function adaptation in temporal difference reinforcement learning', *Annals of Operations Research* **134**(1), 215–238.

Mendelssohn, R. (1982), 'An Iterative Aggregation Procedure for Markov Decision Processes', *Operations Research* **30**, 62–73.

Ormoneit, D. & Glynn, P. W. (2002), 'Kernel-based reinforcement learning average-cost problems', *IEEE Trans. on Automatic Control* **vol**, 1624–1636.

Ormoneit, D. & Sen, Ś. (2002), 'Kernel-based reinforcement learning', *Machine Learning*.

Powell, W. B. & George, A. P. (2006), 'Adaptive stepsizes for recursive estimation with applications in approximate dynamic programming', *Journal of Machine Learning* **65**(1), 167–198.

Rogers, D., Plante, R., Wong, R. & Evans, J. (1991), 'Aggregation and Disaggregation Techniques and Methodology in Optimization', *Operations Research* **39**, 553–582.

Schweitzer, P. & Seidmann, A. (1985), 'Generalized polynomial approximations in Markovian decision processes', *Journal of Mathematical Analysis and Applications* **110**(6), 568–582.

Singh, S. P., Jaakkola, T. & Jordan, M. I. (1995), 'Reinforcement Learning with Soft State Aggregation', *In Advances in Neural Information Processing Systems, Vol. 7, MIT Press* **pp**, 361–368.

Smola, A. J. & Schölkopf, B. (2004), 'A tutorial on support vector regression', *Statistics and Computing* **14**(3), 199–222.

Taylor, H. (1967), 'Evaluating a call option and optimal timing strategy in the stock market', *Management Science* **12**, 111–120.

Tsitsiklis, J. N. & Roy, B. V. (1996), 'Feature-Based Methods for Large Scale Dynamic Programming', *Machine Learning, Vol.* **22**, 59–94.

Tsitsiklis, J. N. & Van Roy, B. (1997), 'An analysis of temporal-difference learning with function approximation', *IEEE Transactions on Automatic Control* **42**(5), 674–690.

Tsitsiklis, J. N. & Van Roy, B. (2001), 'Regression Methods for Pricing Complex American-Style Options', *IEEE Transactions on Neural Networks* **12**, 694–703.

Van Roy, B. (2001), Neuro-Dynamic Programming: Overview and Recent Trends, *in* E. Feinberg & A. Shwartz, eds, 'Handbook of Markov Decision Processes: Methods and Applications', Kluwer, Boston, pp. 431–460.

Whitt, W. (1978), 'Approximations of Dynamic Programs I', *Mathematics of Operations Research* **vol**, 231–243.

Yang, Y. (2001), 'Adaptive Regression by Mixing', *Journal of the American Statistical Association*.

Zipkin, P. (1980), 'Bounds on the Effect of Aggregating Variables in Linear Programming', *Operations Research* **28**, 155–177.