Chapter 2

Supervised learning

A *supervised learning model* defines a mapping from one or more inputs to one or more outputs. For example, the input might be the age and mileage of a secondhand Toyota Prius, and the output might be the estimated value of the car in dollars.

The model is just a mathematical equation; when the inputs are passed through this equation, it computes the output, and this is termed *inference*. The model equation also contains *parameters*. Different parameter values change the outcome of the computation; the model equation describes a family of possible relationships between inputs and outputs, and the parameters specify the particular relationship.

When we *train* or *learn* a model, we find parameters that describe the true relationship between inputs and outputs. A learning algorithm takes a training set of input/output pairs and manipulates the parameters until the inputs predict their corresponding outputs as closely as possible. If the model works well for these training pairs, then we hope it will make good predictions for new inputs where the true output is unknown.

The goal of this chapter is to expand on these ideas. First, we describe this framework more formally and introduce some notation. Then we work through a simple example in which we use a straight line to describe the relationship between input and output. This linear model is both familiar and easy to visualize, but nevertheless illustrates all the main ideas of supervised learning.

2.1 Supervised learning overview

In supervised learning, we aim to build a model that takes an input $\mathbf{x}$ and outputs a prediction $\mathbf{y}$. For simplicity, we assume that both the input $\mathbf{x}$ and output $\mathbf{y}$ are vectors of a predetermined and fixed size and that the elements of each vector are always ordered in the same way; in the Prius example above, the input $\mathbf{x}$ would always contain the age of the car and then the mileage, in that order. This is termed *structured* or *tabular* data.

To make the prediction, we need a model $\mathbf{f}[\bullet]$ that takes input $\mathbf{x}$ and returns $\mathbf{y}$, so:

$$\mathbf{y} = \mathbf{f}[\mathbf{x}]. \tag{2.1}$$

When we compute the prediction $\mathbf{y}$ from the input $\mathbf{x}$, we call this *inference*.

The model is just a mathematical equation with a fixed form. It represents a family of different relations between the input and the output. The model also contains *parameters* $\boldsymbol{\phi}$. The choice of parameters determines the particular relation between input and output, so we should really write:

$$\mathbf{y} = \mathbf{f}[\mathbf{x}, \boldsymbol{\phi}]. \tag{2.2}$$

When we talk about *learning* or *training* a model, we mean that we attempt to find parameters $\boldsymbol{\phi}$ that make sensible output predictions from the input. We learn these parameters using a *training dataset* of I pairs of input and output examples $\{\mathbf{x}_i, \mathbf{y}_i\}$. We aim to select parameters that map each training input to its associated output as closely as possible. We quantify the degree of mismatch in this mapping with the *loss* L. This is a scalar value that summarizes how poorly the model predicts the training outputs from their corresponding inputs for parameters $\boldsymbol{\phi}$.

We can treat the loss as a function $L[\boldsymbol{\phi}]$ of these parameters. When we train the model, we are seeking parameters $\hat{\boldsymbol{\phi}}$ that minimize this *loss function*:[1]

Appendix A
Argmin function

$$\hat{\boldsymbol{\phi}} = \underset{\boldsymbol{\phi}}{\operatorname{argmin}}\Big[L\left[\boldsymbol{\phi}\right]\Big]. \tag{2.3}$$

If the loss is small after this minimization, we have found model parameters that accurately predict the training outputs $\mathbf{y}_i$ from the training inputs $\mathbf{x}_i$.

After training a model, we must now assess its performance; we run the model on separate *test data* to see how well it *generalizes* to examples that it didn't observe during training. If the performance is adequate, then we are ready to deploy the model.

2.2 Linear regression example

Let's now make these ideas concrete with a simple example. We consider a model $y = \mathrm{f}[x, \boldsymbol{\phi}]$ that predicts a single output y from a single input x. Then we develop a loss function, and finally, we discuss model training.

2.2.1 1D linear regression model

A *1D linear regression model* describes the relationship between input x and output y as a straight line:

$$\begin{aligned} y &= \mathrm{f}[x, \boldsymbol{\phi}] \\ &= \phi_0 + \phi_1 x. \end{aligned} \tag{2.4}$$

[1] More properly, the loss function also depends on the training data $\{\mathbf{x}_i, \mathbf{y}_i\}$, so we should write $L\left[\{\mathbf{x}_i, \mathbf{y}_i\}, \boldsymbol{\phi}\right]$, but this is rather cumbersome.

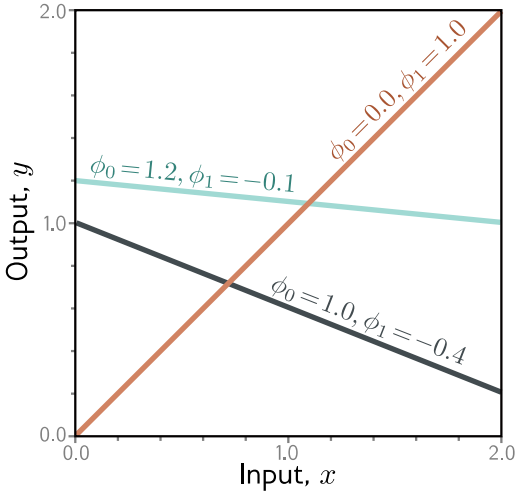

Figure 2.1 Linear regression model. For a given choice of parameters $\boldsymbol{\phi} = [\phi_0, \phi_1]$, the model makes a prediction for the output (y-axis) based on the input (x-axis). Different choices for the y-intercept ϕ_0 and the slope ϕ_1 change these predictions (cyan, orange, and gray lines). The linear regression model (equation 2.4) defines a family of input/output relations (lines) and the parameters determine the member of the family (the particular line).

This model has two parameters $\boldsymbol{\phi} = [\phi_0, \phi_1]$, where ϕ_0 is the y-intercept of the line and ϕ_1 is the slope. Different choices for the y-intercept and slope result in different relations between input and output (figure 2.1). Hence, equation 2.4 defines a family of possible input-output relations (all possible lines), and the choice of parameters determines the member of this family (the particular line).

2.2.2 Loss

For this model, the training dataset (figure 2.2a) consists of I input/output pairs $\{x_i, y_i\}$. Figures 2.2b–d show three lines defined by three sets of parameters. The green line in figure 2.2d describes the data more accurately than the other two since it is much closer to the data points. However, we need a principled approach for deciding which parameters $\boldsymbol{\phi}$ are better than others. To this end, we assign a numerical value to each choice of parameters that quantifies the degree of mismatch between the model and the data. We term this value the *loss*; a lower loss means a better fit.

The mismatch is captured by the deviation between the model predictions $\mathrm{f}[x_i, \boldsymbol{\phi}]$ (height of the line at x_i) and the ground truth outputs y_i. These deviations are depicted as orange dashed lines in figures 2.2b–d. We quantify the total mismatch, *training error*, or *loss* as the sum of the squares of these deviations for all I training pairs:

$$\begin{aligned} L[\phi] &= \sum_{i=1}^{I} \left(\mathrm{f}[x_i, \boldsymbol{\phi}] - y_i\right)^2 \\ &= \sum_{i=1}^{I} \left(\phi_0 + \phi_1 x_i - y_i\right)^2. \end{aligned} \tag{2.5}$$

Since the best parameters minimize this expression, we call this a *least-squares* loss. The squaring operation means that the direction of the deviation (i.e., whether the line is

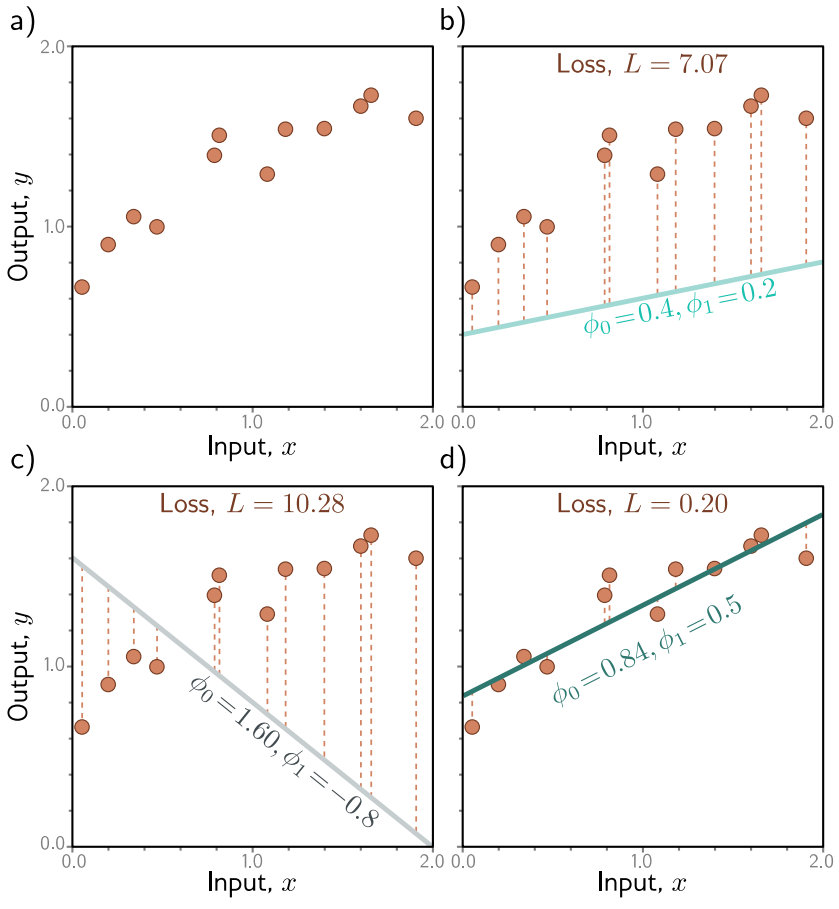

Figure 2.2 Linear regression training data, model, and loss. a) The training data (orange points) consist of $I = 12$ input/output pairs $\{x_i, y_i\}$. b–d) Each panel shows the linear regression model with different parameters. Depending on the choice of y-intercept and slope parameters $\boldsymbol{\phi} = [\phi_0, \phi_1]$, the model errors (orange dashed lines) may be larger or smaller. The loss L is the sum of the squares of these errors. The parameters that define the lines in panels (b) and (c) have large losses $L = 7.07$ and $L = 10.28$, respectively because the models fit badly. The loss $L = 0.20$ in panel (d) is smaller because the model fits well; in fact, this has the smallest loss of all possible lines, so these are the optimal parameters.

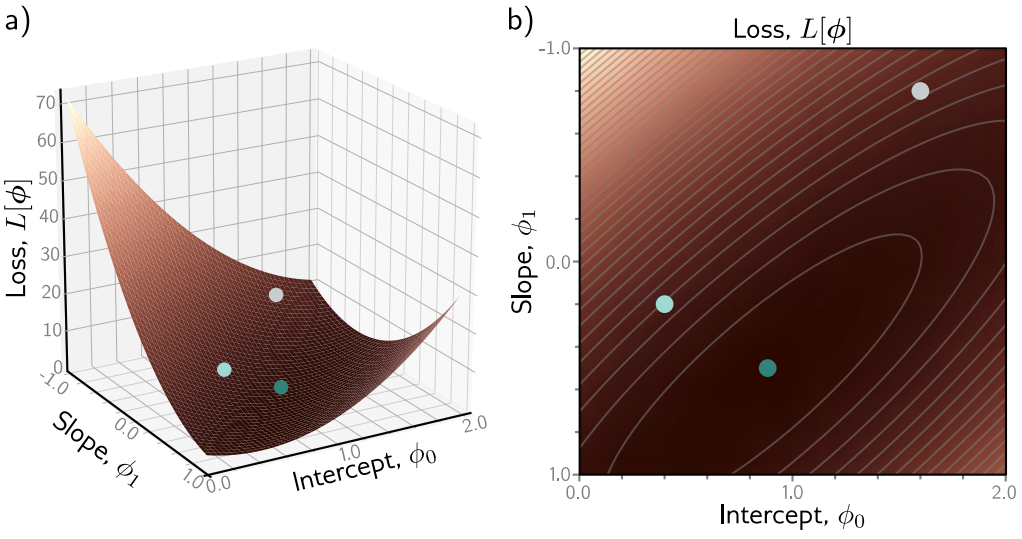

Figure 2.3 Loss function for linear regression model with the dataset in figure 2.2a. a) Each combination of parameters $\boldsymbol{\phi} = [\phi_0,\phi_1]$ has an associated loss. The resulting loss function $L[\boldsymbol{\phi}]$ can be visualized as a surface. The three circles represent the lines from figure 2.2b–d. b) The loss can also be visualized as a heatmap, where brighter regions represent larger losses; here we are looking straight down at the surface in (a) from above and gray ellipses represent isocontours. The best fitting line (figure 2.2d) has the parameters with the smallest loss (green circle).

above or below the data) is unimportant. There are also theoretical reasons for this choice which we return to in chapter 5.

The loss L is a function of the parameters $\boldsymbol{\phi}$; it will be larger when the model fit is poor (figure 2.2b,c) and smaller when it is good (figure 2.2d). Considered in this light, we term $L[\boldsymbol{\phi}]$ the *loss function* or *cost function*. The goal is to find the parameters $\hat{\boldsymbol{\phi}}$ that minimize this quantity:

Notebook 2.1 Supervised learning

$$\begin{aligned}\hat{\boldsymbol{\phi}} &= \underset{\boldsymbol{\phi}}{\operatorname{argmin}}\Big[L[\boldsymbol{\phi}]\Big] \\ &= \underset{\boldsymbol{\phi}}{\operatorname{argmin}}\left[\sum_{i=1}^{I}\left(\mathrm{f}[x_i,\boldsymbol{\phi}]-y_i\right)^2\right] \\ &= \underset{\boldsymbol{\phi}}{\operatorname{argmin}}\left[\sum_{i=1}^{I}\left(\phi_0+\phi_1 x_i-y_i\right)^2\right]. \end{aligned} \tag{2.6}$$

There are only two parameters (the y-intercept ϕ_0 and slope ϕ_1), so we can calculate the loss for every combination of values and visualize the loss function as a surface (figure 2.3). The "best" parameters are at the minimum of this surface.

Problems 2.1–2.2

2.2.3 Training

The process of finding parameters that minimize the loss is termed *model fitting*, *training*, or *learning*. The basic method is to choose the initial parameters randomly and then improve them by "walking down" the loss function until we reach the bottom (figure 2.4). One way to do this is to measure the gradient of the surface at the current position and take a step in the direction that is most steeply downhill. Then we repeat this process until the gradient is flat and we can improve no further.[2]

2.2.4 Testing

Having trained the model, we want to know how it will perform in the real world. We do this by computing the loss on a separate set of *test data*. The degree to which the prediction accuracy *generalizes* to the test data depends in part on how representative and complete the training data is. However, it also depends on how expressive the model is. A simple model like a line might not be able to capture the true relationship between input and output. This is known as *underfitting*. Conversely, a very expressive model may describe statistical peculiarities of the training data that are atypical and lead to unusual predictions. This is known as *overfitting*.

2.3 Summary

A supervised learning model is a function $\mathbf{y} = \mathbf{f}[\mathbf{x}, \boldsymbol{\phi}]$ that relates inputs $\mathbf{x}$ to outputs $\mathbf{y}$. The particular relationship is determined by parameters $\boldsymbol{\phi}$. To train the model, we define a loss function $L[\boldsymbol{\phi}]$ over a training dataset $\{\mathbf{x}_i, \mathbf{y}_i\}$. This quantifies the mismatch between the model predictions $\mathbf{f}[\mathbf{x}_i, \boldsymbol{\phi}]$ and observed outputs $\mathbf{y}_i$ as a function of the parameters $\boldsymbol{\phi}$. Then we search for the parameters that minimize the loss. We evaluate the model on a different set of test data to see how well it generalizes to new inputs.

Chapters 3–9 expand on these ideas. First, we tackle the model itself; 1D linear regression has the obvious drawback that it can only describe the relationship between the input and output as a straight line. Shallow neural networks (chapter 3) are only slightly more complex than linear regression but describe a much larger family of input/output relationships. Deep neural networks (chapter 4) are just as expressive but can describe complex functions with fewer parameters and work better in practice.

Chapter 5 investigates loss functions for different tasks and reveals the theoretical underpinnings of the least-squares loss. Chapters 6 and 7 discuss the training process. Chapter 8 discusses how to measure model performance. Chapter 9 considers *regularization* techniques, which aim to improve that performance.

[2]This iterative approach is not actually necessary for the linear regression model. Here, it's possible to find closed-form expressions for the parameters. However, this *gradient descent* approach works for more complex models where there is no closed-form solution and where there are too many parameters to evaluate the loss for every combination of values.

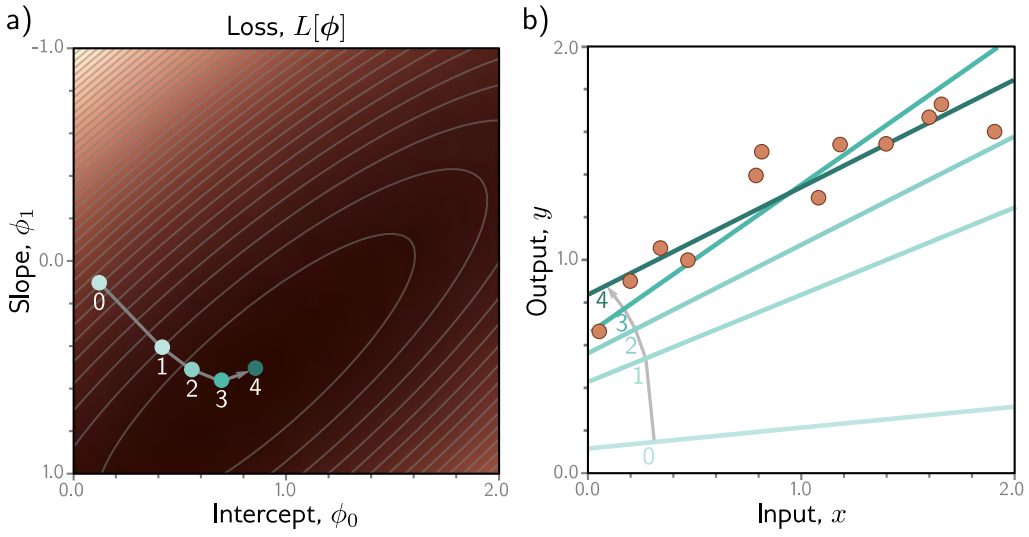

Figure 2.4 Linear regression training. The goal is to find the y-intercept and slope parameters that correspond to the smallest loss. a) Iterative training algorithms initialize the parameters randomly and then improve them by "walking downhill" until no further improvement can be made. Here, we start at position 0 and move a certain distance downhill (perpendicular to the contours) to position 1. Then we re-calculate the downhill direction and move to position 2. Eventually, we reach the minimum of the function (position 4). b) Each position 0–4 from panel (a) corresponds to a different y-intercept and slope and so represents a different line. As the loss decreases, the lines fit the data more closely.

Notes

Loss functions vs. cost functions: In much of machine learning and in this book, the terms loss function and cost function are used interchangeably. However, more properly, a loss function is the individual term associated with a data point (i.e., each of the squared terms on the right-hand side of equation 2.5), and the cost function is the overall quantity that is minimized (i.e., the entire right-hand side of equation 2.5). A cost function can contain additional terms that are not associated with individual data points (see section 9.1). More generally, an *objective function* is any function that is to be maximized or minimized.

Generative vs. discriminative models: The models $\mathbf{y} = \mathbf{f}[\mathbf{x}, \boldsymbol{\phi}]$ in this chapter are *discriminative models*. These make an output prediction $\mathbf{y}$ from real-world measurements $\mathbf{x}$. Another approach is to build a *generative model* $\mathbf{x} = \mathbf{g}[\mathbf{y}, \boldsymbol{\phi}]$, in which the real-world measurements $\mathbf{x}$ are computed as a function of the output $\mathbf{y}$.

Problem 2.3

The generative approach has the disadvantage that it doesn't directly predict $\mathbf{y}$. To perform inference, we must invert the generative equation as $\mathbf{y} = \mathbf{g}^{-1}[\mathbf{x}, \boldsymbol{\phi}]$, and this may be difficult. However, generative models have the advantage that we can build in prior knowledge about how the data were created. For example, if we wanted to predict the 3D position and orientation $\mathbf{y}$

of a car in an image $\mathbf{x}$, then we could build knowledge about car shape, 3D geometry, and light transport into the function $\mathbf{x} = \mathbf{g}[\mathbf{y}, \boldsymbol{\phi}]$.

This seems like a good idea, but in fact, discriminative models dominate modern machine learning; the advantage gained from exploiting prior knowledge in generative models is usually trumped by learning very flexible discriminative models with large amounts of training data.

Problems

Problem 2.1 To walk "downhill" on the loss function (equation 2.5), we measure its gradient with respect to the parameters ϕ_0 and ϕ_1. Calculate expressions for the slopes $\partial L/\partial\phi_0$ and $\partial L/\partial\phi_1$.

Problem 2.2 Show that we can find the minimum of the loss function in closed form by setting the expression for the derivatives from problem 2.1 to zero and solving for ϕ_0 and ϕ_1. Note that this works for linear regression but not for more complex models; this is why we use iterative model fitting methods like gradient descent (figure 2.4).

Problem 2.3* Consider reformulating linear regression as a generative model, so we have $x = \mathrm{g}[y, \boldsymbol{\phi}] = \phi_0 + \phi_1 y$. What is the new loss function? Find an expression for the inverse function $y = \mathrm{g}^{-1}[x, \boldsymbol{\phi}]$ that we would use to perform inference. Will this model make the same predictions as the discriminative version for a given training dataset $\{x_i, y_i\}$? One way to establish this is to write code that fits a line to three data points using both methods and see if the result is the same.