# Chapter 5

# Loss functions

The last three chapters described linear regression, shallow neural networks, and deep neural networks. Each represents a family of functions that map input to output, where the particular member of the family is determined by the model parameters $\boldsymbol{\phi}$. When we train these models, we seek the parameters that produce the best possible mapping from input to output for the task we are considering. This chapter defines what is meant by the "best possible" mapping.

That definition requires a training dataset $\{\mathbf{x}_i, \mathbf{y}_i\}$ of input/output pairs. A *loss function* or *cost function* $L[\boldsymbol{\phi}]$ returns a single number that describes the mismatch between the model predictions $\mathbf{f}[\mathbf{x}_i, \boldsymbol{\phi}]$ and their corresponding ground-truth outputs $\mathbf{y}_i$. During training, we seek parameter values $\boldsymbol{\phi}$ that minimize the loss and hence map the training inputs to the outputs as closely as possible. We saw one example of a loss function in chapter 2; the least squares loss function is suitable for univariate regression problems for which the target is a real number $y \in \mathbb{R}$. It computes the sum of the squares of the deviations between the model predictions $\mathrm{f}[\mathbf{x}_i, \boldsymbol{\phi}]$ and the true values $y_i$.

Appendix A Number sets

This chapter provides a framework that both justifies the choice of the least squares criterion for real-valued outputs and allows us to build loss functions for other prediction types. We consider *binary classification*, where the prediction $y \in \{0, 1\}$ is one of two categories, *multiclass classification*, where the prediction $y \in \{1, 2, \ldots, K\}$ is one of $K$ categories, and more complex cases. In the following two chapters, we address model training, where the goal is to find the parameter values that minimize these loss functions.

## 5.1 Maximum likelihood

In this section, we develop a recipe for constructing loss functions. Consider a model $\mathbf{f}[\mathbf{x}, \boldsymbol{\phi}]$ with parameters $\boldsymbol{\phi}$ that computes an output from input $\mathbf{x}$. Until now, we have implied that the model directly computes a prediction $\mathbf{y}$. We now shift perspective and consider the model as computing a conditional probability distribution $Pr(\mathbf{y}|\mathbf{x})$ over possible outputs $\mathbf{y}$ given input $\mathbf{x}$. The loss encourages each training output $\mathbf{y}_i$ to have a high probability under the distribution $Pr(\mathbf{y}_i|\mathbf{x}_i)$ computed from the corresponding input $\mathbf{x}_i$ (figure 5.1).

Appendix C.1.3 Conditional probability

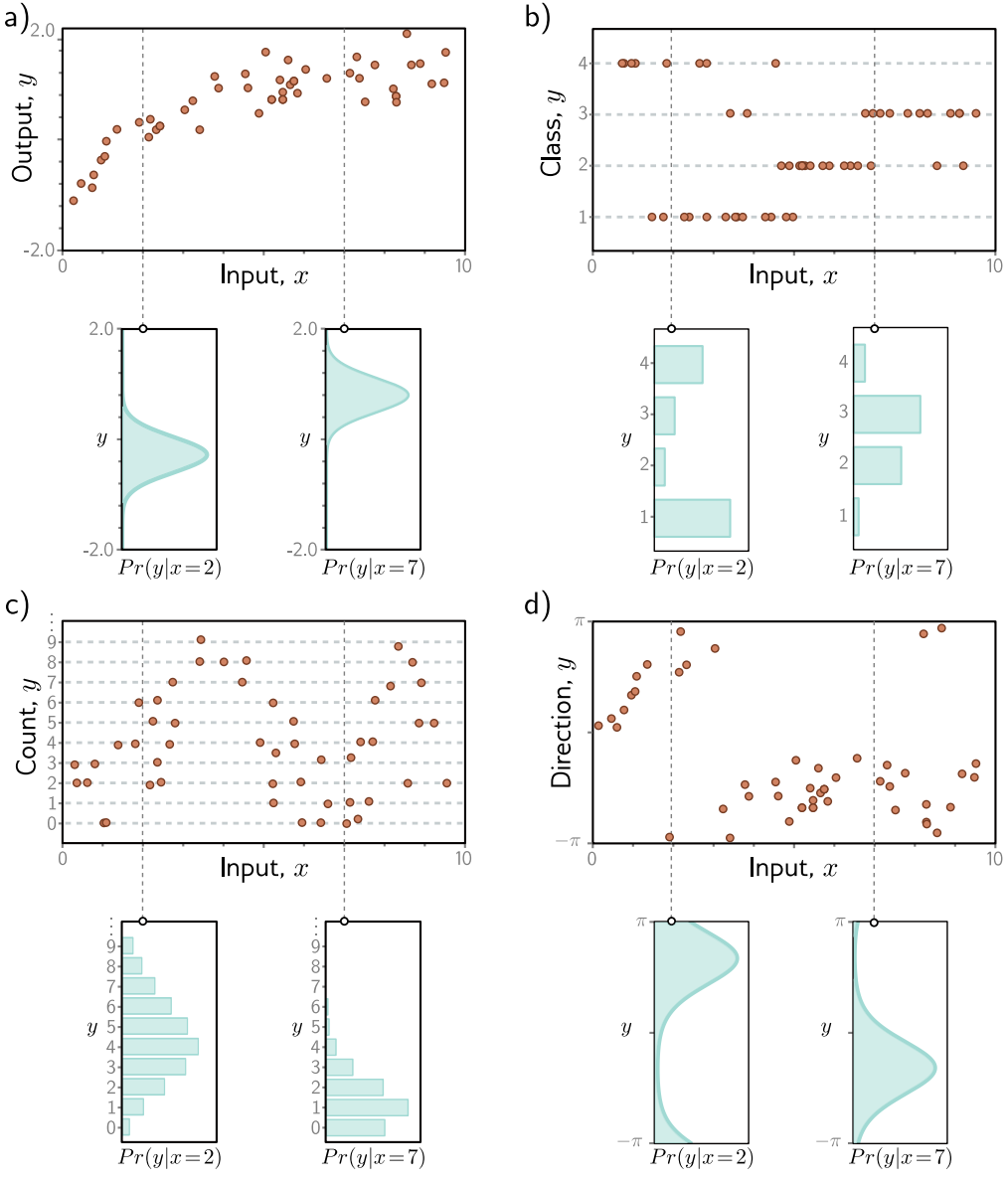


**Figure 5.1** Predicting distributions over outputs. a) Regression task, where the goal is to predict a real-valued output $y$ from the input $x$ based on training data $\{x_i, y_i\}$ (orange points). For each input value $x$, the machine learning model predicts a distribution $Pr(y|x)$ over the output $y \in \mathbb{R}$ (cyan curves show distributions for $x = 2.0$ and $x = 7.0$). The loss function aims to maximize the probability of the observed training outputs $y_i$ under the distribution predicted from the corresponding inputs $x_i$. b) To predict discrete classes $y \in \{1, 2, 3, 4\}$ in a classification task, we use a discrete probability distribution, so the model predicts a different histogram over the four possible values of $y_i$ for each value of $x_i$. c) To predict counts $y \in \{0, 1, 2, \ldots\}$ and d) direction $y \in (-\pi, \pi]$, we use distributions defined over positive integers and circular domains, respectively.

### 5.1.1 Computing a distribution over outputs

This raises the question of exactly how a model $\mathbf{f}[\mathbf{x}, \boldsymbol{\phi}]$ can be adapted to compute a probability distribution. The solution is simple. First, we choose a parametric distribution $Pr(\mathbf{y}|\boldsymbol{\theta})$ defined on the output domain $\mathbf{y}$. Then we use the network to compute one or more of the parameters $\boldsymbol{\theta}$ of this distribution.

For example, suppose the prediction domain is the set of real numbers, so $y \in \mathbb{R}$. Here, we might choose the univariate normal distribution, which is defined on $\mathbb{R}$. This distribution is defined by the mean $\mu$ and variance $\sigma^2$, so $\boldsymbol{\theta} = \{\mu, \sigma^2\}$. The machine learning model might predict the mean $\mu$, and the variance $\sigma^2$ could be treated as an unknown constant.

### 5.1.2 Maximum likelihood criterion

The model now computes different distribution parameters $\boldsymbol{\theta}_i = \mathbf{f}[\mathbf{x}_i, \boldsymbol{\phi}]$ for each training input $\mathbf{x}_i$. Each observed training output $\mathbf{y}_i$ should have high probability under its corresponding distribution $Pr(\mathbf{y}_i|\boldsymbol{\theta}_i)$. Hence, we choose the model parameters $\boldsymbol{\phi}$ so that they maximize the combined probability across all $I$ training examples:

$$\begin{aligned}\hat{\boldsymbol{\phi}} &= \underset{\boldsymbol{\phi}}{\operatorname{argmax}} \left[\prod_{i=1}^{I} Pr(\mathbf{y}_i|\mathbf{x}_i)\right] \\ &= \underset{\boldsymbol{\phi}}{\operatorname{argmax}} \left[\prod_{i=1}^{I} Pr(\mathbf{y}_i|\boldsymbol{\theta}_i)\right] \\ &= \underset{\boldsymbol{\phi}}{\operatorname{argmax}} \left[\prod_{i=1}^{I} Pr(\mathbf{y}_i|\mathbf{f}[\mathbf{x}_i, \boldsymbol{\phi}])\right]. \end{aligned} \tag{5.1}$$

The combined probability term is the *likelihood* of the parameters, and hence equation 5.1 is known as the *maximum likelihood* criterion.[1]

Here we are implicitly making two assumptions. First, we assume that the data are identically distributed (the form of the probability distribution over the outputs $\mathbf{y}_i$ is the same for each data point). Second, we assume that the conditional distributions $Pr(\mathbf{y}_i|\mathbf{x}_i)$ of the output given the input are independent, so the total likelihood of the training data decomposes as:

Appendix C.1.5 Independence

$$Pr(\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_I|\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_I) = \prod_{i=1}^{I} Pr(\mathbf{y}_i|\mathbf{x}_i). \tag{5.2}$$

In other words, we assume the data are *independent and identically distributed* (*i.i.d.*).

[1] A conditional probability $Pr(z|\psi)$ can be considered in two ways. As a function of $z$, it is a probability distribution that sums to one. As a function of $\psi$, it is known as a *likelihood* and does not generally sum to one.

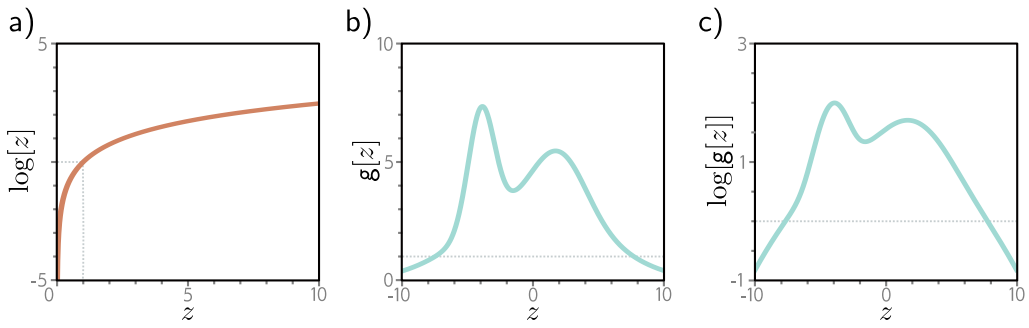


**Figure 5.2** The log transform. a) The log function is monotonically increasing. If $z > z'$, then $\log[z] > \log[z']$. It follows that the maximum of any function $g[z]$ will be at the same position as the maximum of $\log[g[z]]$. b) A function $g[z]$. c) The logarithm of this function $\log[g[z]]$. All positions on $g[z]$ with a positive slope retain a positive slope after the log transform, and those with a negative slope retain a negative slope. The position of the maximum remains the same.

### 5.1.3 Maximizing log-likelihood

The maximum likelihood criterion (equation 5.1) is not very practical. Each term $Pr(\mathbf{y}_i|\mathbf{f}[\mathbf{x}_i, \boldsymbol{\phi}])$ can be small, so the product of many of these terms can be tiny. It may be difficult to represent this quantity with finite precision arithmetic. Fortunately, we can equivalently maximize the logarithm of the likelihood:

$$\begin{aligned}\hat{\boldsymbol{\phi}} &= \underset{\boldsymbol{\phi}}{\text{argmax}}\left[\prod_{i=1}^{I} Pr(\mathbf{y}_i|\mathbf{f}[\mathbf{x}_i, \boldsymbol{\phi}])\right] \\ &= \underset{\boldsymbol{\phi}}{\text{argmax}}\left[\log\left[\prod_{i=1}^{I} Pr(\mathbf{y}_i|\mathbf{f}[\mathbf{x}_i, \boldsymbol{\phi}])\right]\right] \\ &= \underset{\boldsymbol{\phi}}{\text{argmax}}\left[\sum_{i=1}^{I} \log\Big[Pr(\mathbf{y}_i|\mathbf{f}[\mathbf{x}_i, \boldsymbol{\phi}])\Big]\right]. \end{aligned} \tag{5.3}$$

This *log-likelihood* criterion is equivalent because the logarithm is a monotonically increasing function: if $z > z'$, then $\log[z] > \log[z']$ and vice versa (figure 5.2). It follows that when we change the model parameters $\boldsymbol{\phi}$ to improve the log-likelihood criterion, we also improve the original maximum likelihood criterion. It also follows that the overall maxima of the two criteria must be in the same place, so the best model parameters $\hat{\boldsymbol{\phi}}$ are the same in both cases. However, the log-likelihood criterion has the practical advantage of using a sum of terms, not a product, so representing it with finite precision isn't problematic.

### 5.1.4 Minimizing negative log-likelihood

Finally, we note that, by convention, model fitting problems are framed in terms of minimizing a loss. To convert the maximum log-likelihood criterion to a minimization problem, we multiply by minus one, which gives us the *negative log-likelihood criterion*:

$$\begin{aligned}\hat{\boldsymbol{\phi}} &= \underset{\boldsymbol{\phi}}{\operatorname{argmin}}\left[-\sum_{i=1}^{I}\log\Big[Pr(\mathbf{y}_i|\mathbf{f}[\mathbf{x}_i,\boldsymbol{\phi}])\Big]\right] \\ &= \underset{\boldsymbol{\phi}}{\operatorname{argmin}}\Big[L[\boldsymbol{\phi}]\Big], \end{aligned} \tag{5.4}$$

which is what forms the final loss function $L[\boldsymbol{\phi}]$.

### 5.1.5 Inference

The network no longer directly predicts the outputs $\mathbf{y}$ but instead determines a probability distribution over $\mathbf{y}$. When we perform inference, we often want a point estimate rather than a distribution, so we return the maximum of the distribution:

$$\hat{\mathbf{y}} = \underset{\mathbf{y}}{\operatorname{argmax}}\Big[Pr(\mathbf{y}|\mathbf{f}[\mathbf{x},\hat{\boldsymbol{\phi}}])\Big]. \tag{5.5}$$

It is usually possible to find an expression for this in terms of the distribution parameters $\boldsymbol{\theta}$ predicted by the model. For example, in the univariate normal distribution, the maximum occurs at the mean $\mu$.

## 5.2 Recipe for constructing loss functions

The recipe for constructing loss functions for training data $\{\mathbf{x}_i,\mathbf{y}_i\}$ using the maximum likelihood approach is hence:

1. Choose a suitable probability distribution $Pr(\mathbf{y}|\boldsymbol{\theta})$ defined over the domain of the predictions $\mathbf{y}$ with distribution parameters $\boldsymbol{\theta}$.
2. Set the machine learning model $\mathbf{f}[\mathbf{x},\boldsymbol{\phi}]$ to predict one or more of these parameters, so $\boldsymbol{\theta}=\mathbf{f}[\mathbf{x},\boldsymbol{\phi}]$ and $Pr(\mathbf{y}|\boldsymbol{\theta}) = Pr(\mathbf{y}|\mathbf{f}[\mathbf{x},\boldsymbol{\phi}])$.
3. To train the model, find the network parameters $\hat{\boldsymbol{\phi}}$ that minimize the negative log-likelihood loss function over the training dataset pairs $\{\mathbf{x}_i,\mathbf{y}_i\}$:

$$\hat{\boldsymbol{\phi}} = \underset{\boldsymbol{\phi}}{\operatorname{argmin}}\Big[L[\boldsymbol{\phi}]\Big] = \underset{\boldsymbol{\phi}}{\operatorname{argmin}}\left[-\sum_{i=1}^{I}\log\Big[Pr(\mathbf{y}_i|\mathbf{f}[\mathbf{x}_i,\boldsymbol{\phi}])\Big]\right]. \tag{5.6}$$

4. To perform inference for a new test example $\mathbf{x}$, return either the full distribution $Pr(\mathbf{y}|\mathbf{f}[\mathbf{x},\hat{\boldsymbol{\phi}}])$ or the value where this distribution is maximized.

We devote most of the rest of this chapter to constructing loss functions for common prediction types using this recipe.

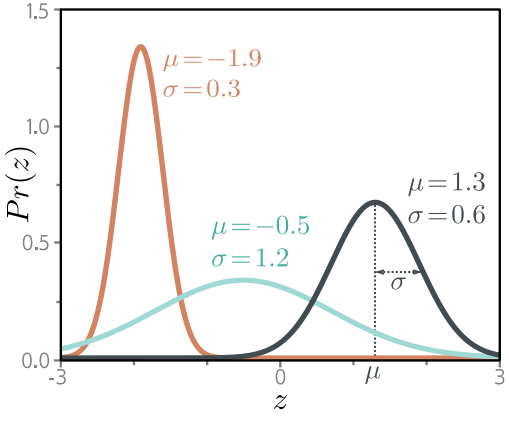


**Figure 5.3** The univariate normal distribution (also known as the Gaussian distribution) is defined on the real line $z \in \mathbb{R}$ and has parameters $\mu$ and $\sigma^2$. The mean $\mu$ determines the position of the peak. The positive root of the variance $\sigma^2$ (the standard deviation) determines the width of the distribution. Since the total probability density sums to one, the peak becomes higher as the variance decreases and the distribution becomes narrower.

## 5.3 Example 1: univariate regression

We start by considering univariate regression models. Here the goal is to predict a single scalar output $y \in \mathbb{R}$ from input $\mathbf{x}$ using a model $\mathrm{f}[\mathbf{x}, \boldsymbol{\phi}]$ with parameters $\boldsymbol{\phi}$. Following the recipe, we choose a probability distribution over the output domain $y$. We select the univariate normal (figure 5.3), which is defined over $y \in \mathbb{R}$. This distribution has two parameters (mean $\mu$ and variance $\sigma^2$) and has a probability density function:

$$Pr(y|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{(y-\mu)^2}{2\sigma^2}\right]. \tag{5.7}$$

Second, we set the machine learning model $\mathrm{f}[\mathbf{x}, \boldsymbol{\phi}]$ to compute one or more of the parameters of this distribution. Here, we just compute the mean so $\mu = \mathrm{f}[\mathbf{x}, \boldsymbol{\phi}]$:

$$Pr(y|\mathrm{f}[\mathbf{x}, \boldsymbol{\phi}], \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{(y-\mathrm{f}[\mathbf{x}, \boldsymbol{\phi}])^2}{2\sigma^2}\right]. \tag{5.8}$$

We aim to find the parameters $\boldsymbol{\phi}$ that make the training data $\{\mathbf{x}_i, y_i\}$ most probable under this distribution (figure 5.4). To accomplish this, we choose a loss function $L[\boldsymbol{\phi}]$ based on the negative log-likelihood:

$$\begin{aligned} L[\boldsymbol{\phi}] &= -\sum_{i=1}^{I} \log\left[Pr(y_i|\mathrm{f}[\mathbf{x}_i, \boldsymbol{\phi}], \sigma^2)\right] \\ &= -\sum_{i=1}^{I} \log\left[\frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{(y_i - \mathrm{f}[\mathbf{x}_i, \boldsymbol{\phi}])^2}{2\sigma^2}\right]\right]. \end{aligned} \tag{5.9}$$

When we train the model, we seek parameters $\hat{\boldsymbol{\phi}}$ that minimize this loss.

### 5.3.1 Least squares loss function

Now let's perform some algebraic manipulations on the loss function. We seek:

$$\begin{aligned}\hat{\boldsymbol{\phi}} &= \underset{\boldsymbol{\phi}}{\operatorname{argmin}}\left[-\sum_{i=1}^{I}\log\left[\frac{1}{\sqrt{2\pi\sigma^2}}\exp\left[-\frac{(y_i-\mathrm{f}[\mathbf{x}_i,\boldsymbol{\phi}])^2}{2\sigma^2}\right]\right]\right]\\ &= \underset{\boldsymbol{\phi}}{\operatorname{argmin}}\left[-\sum_{i=1}^{I}\left(\log\left[\frac{1}{\sqrt{2\pi\sigma^2}}\right]-\frac{(y_i-\mathrm{f}[\mathbf{x}_i,\boldsymbol{\phi}])^2}{2\sigma^2}\right)\right]\\ &= \underset{\boldsymbol{\phi}}{\operatorname{argmin}}\left[-\sum_{i=1}^{I}-\frac{(y_i-\mathrm{f}[\mathbf{x}_i,\boldsymbol{\phi}])^2}{2\sigma^2}\right]\\ &= \underset{\boldsymbol{\phi}}{\operatorname{argmin}}\left[\sum_{i=1}^{I}(y_i-\mathrm{f}[\mathbf{x}_i,\boldsymbol{\phi}])^2\right], \end{aligned} \tag{5.10}$$

where we removed the first term between the second and third lines because it doesn't depend on $\boldsymbol{\phi}$. We removed the denominator between the third and fourth lines, as this is just a constant positive scaling factor that does not affect the position of the minimum.

The result of these manipulations is the least squares loss function that we originally introduced when we discussed linear regression in chapter 2:

$$L[\boldsymbol{\phi}] = \sum_{i=1}^{I}\big(y_i-\mathrm{f}[\mathbf{x}_i,\boldsymbol{\phi}]\big)^2. \tag{5.11}$$

Notebook 5.1 Least squares loss

We see that the least squares loss function follows naturally from the assumptions that the predictions are (i) independent and (ii) drawn from a normal distribution with mean $\mu = \mathrm{f}[\mathbf{x}_i,\boldsymbol{\phi}]$ (figure 5.4).

### 5.3.2 Inference

The network no longer directly predicts $y$ but instead predicts the mean $\mu = \mathrm{f}[\mathbf{x},\boldsymbol{\phi}]$ of the normal distribution over $y$. When we perform inference, we usually want a single "best" point estimate $\hat{y}$, so we take the maximum of the predicted distribution:

$$\hat{y} = \underset{y}{\operatorname{argmax}}\left[Pr(y|\mathrm{f}[\mathbf{x},\hat{\boldsymbol{\phi}},\sigma^2])\right]. \tag{5.12}$$

For the univariate normal, the maximum position is determined by the mean parameter $\mu$ (figure 5.3). This is precisely what the model computed, so $\hat{y} = \mathrm{f}[\mathbf{x},\hat{\boldsymbol{\phi}}]$.

### 5.3.3 Estimating variance

To formulate the least squares loss function, we assumed that the network predicted the mean of a normal distribution. The final expression in equation 5.11 (perhaps surpris-

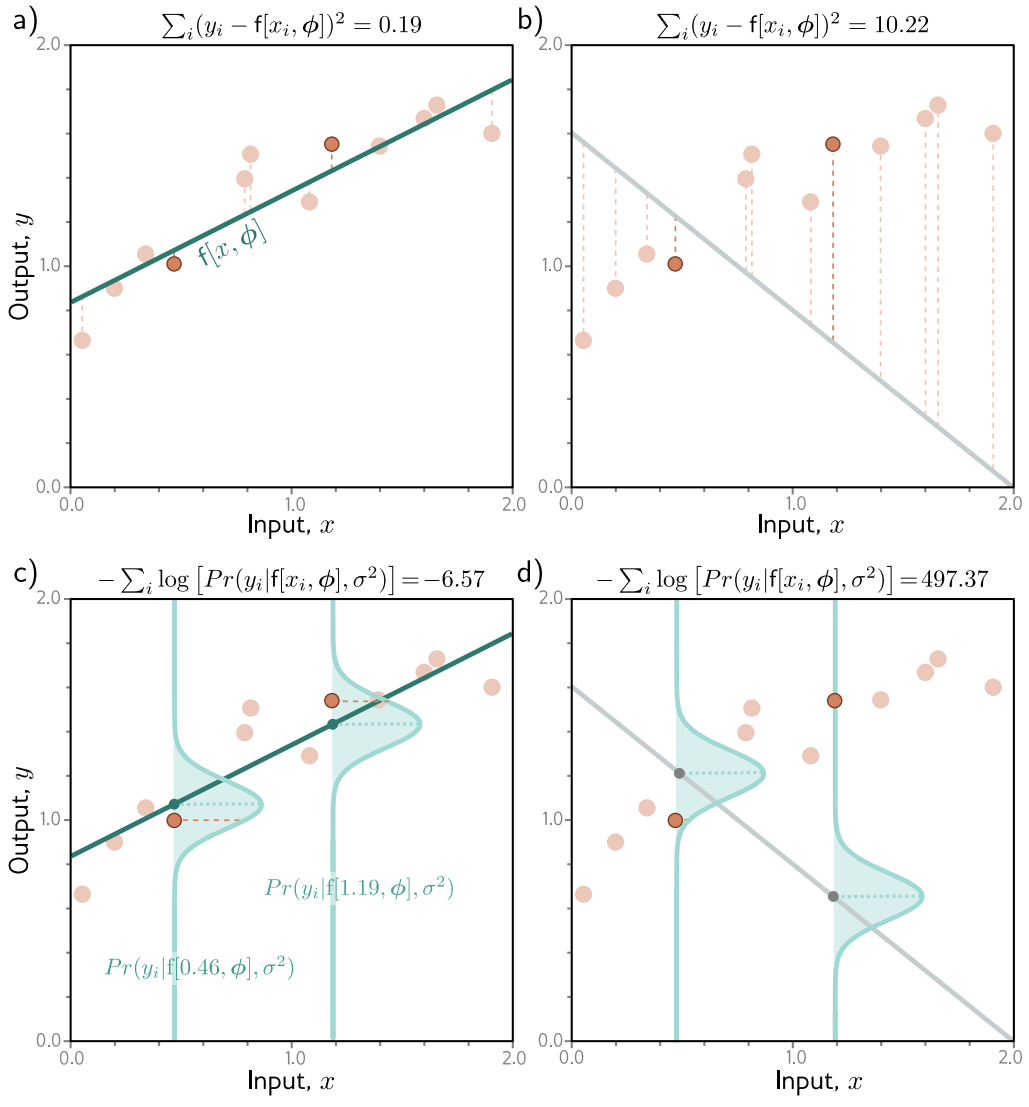


**Figure 5.4** Equivalence of least squares and maximum likelihood loss for the normal distribution. a) Consider the linear model from figure 2.2. The least squares criterion minimizes the sum of the squares of the deviations (dashed lines) between the model prediction $\mathrm{f}[x_i, \phi]$ (green line) and the true output values $y_i$ (orange points). Here the fit is good, so these deviations are small (e.g., for the two highlighted points). b) For these parameters, the fit is bad, and the squared deviations are large. c) The least squares criterion follows from the assumption that the model predicts the mean of a normal distribution over the outputs and that we maximize the probability. For the first case, the model fits well, so the probability $Pr(y_i|x_i)$ of the data (horizontal orange dashed lines) is large (and the negative log probability is small). d) For the second case, the model fits badly, so the probability is small and the negative log probability is large.

ingly) does not depend on the variance $\sigma^2$. However, there is nothing to stop us from treating $\sigma^2$ as a learned parameter and minimizing equation 5.9 with respect to both the model parameters $\boldsymbol{\phi}$ and the distribution variance $\sigma^2$:

$$\hat{\boldsymbol{\phi}}, \hat{\sigma}^2 = \underset{\boldsymbol{\phi}, \sigma^2}{\operatorname{argmin}} \left[ -\sum_{i=1}^{I} \log \left[ \frac{1}{\sqrt{2\pi\sigma^2}} \exp \left[ -\frac{(y_i - \mathrm{f}[\mathbf{x}_i, \boldsymbol{\phi}])^2}{2\sigma^2} \right] \right] \right]. \tag{5.13}$$

In inference, the model predicts the mean $\mu = \mathrm{f}[\mathbf{x}, \hat{\boldsymbol{\phi}}]$ from the input, and we learned the variance $\hat{\sigma}^2$ during the training process. The former is the best prediction. The latter tells us about the uncertainty of the prediction.

### 5.3.4 Heteroscedastic regression

The model above assumes that the variance of the data is constant everywhere. However, this might be unrealistic. When the uncertainty of the model varies as a function of the input data, we refer to this as *heteroscedastic* (as opposed to *homoscedastic*, where the uncertainty is constant).

A simple way to model this is to train a neural network $\mathbf{f}[\mathbf{x}, \boldsymbol{\phi}]$ that computes both the mean and the variance. For example, consider a shallow network with two outputs. We denote the first output as $\mathrm{f}_1[\mathbf{x}, \boldsymbol{\phi}]$ and use this to predict the mean, and we denote the second output as $\mathrm{f}_2[\mathbf{x}, \boldsymbol{\phi}]$ and use it to predict the variance.

There is one complication; the variance must be positive, but we can't guarantee that the network will always produce a positive output. To ensure that the computed variance is positive, we pass the second network output through a function that maps an arbitrary value to a positive one. A suitable choice is the squaring function, giving:

$$\begin{aligned} \mu &= \mathrm{f}_1[\mathbf{x}, \boldsymbol{\phi}] \\ \sigma^2 &= \mathrm{f}_2[\mathbf{x}, \boldsymbol{\phi}]^2, \end{aligned} \tag{5.14}$$

which results in the loss function:

$$\hat{\boldsymbol{\phi}} = \underset{\boldsymbol{\phi}}{\operatorname{argmin}} \left[ -\sum_{i=1}^{I} \left( \log \left[ \frac{1}{\sqrt{2\pi \mathrm{f}_2[\mathbf{x}_i, \boldsymbol{\phi}]^2}} \right] - \frac{(y_i - \mathrm{f}_1[\mathbf{x}_i, \boldsymbol{\phi}])^2}{2\mathrm{f}_2[\mathbf{x}_i, \boldsymbol{\phi}]^2} \right) \right]. \tag{5.15}$$

Homoscedastic and heteroscedastic models are compared in figure 5.5.

## 5.4 Example 2: binary classification

In *binary classification*, the goal is to assign the data $\mathbf{x}$ to one of two discrete classes $y \in \{0, 1\}$. In this context, we refer to $y$ as a *label*. Examples of binary classification include (i) predicting whether a restaurant review is positive ($y = 1$) or negative ($y = 0$) from text data $\mathbf{x}$ and (ii) predicting whether a tumor is present ($y = 1$) or absent ($y = 0$) from an MRI scan $\mathbf{x}$.

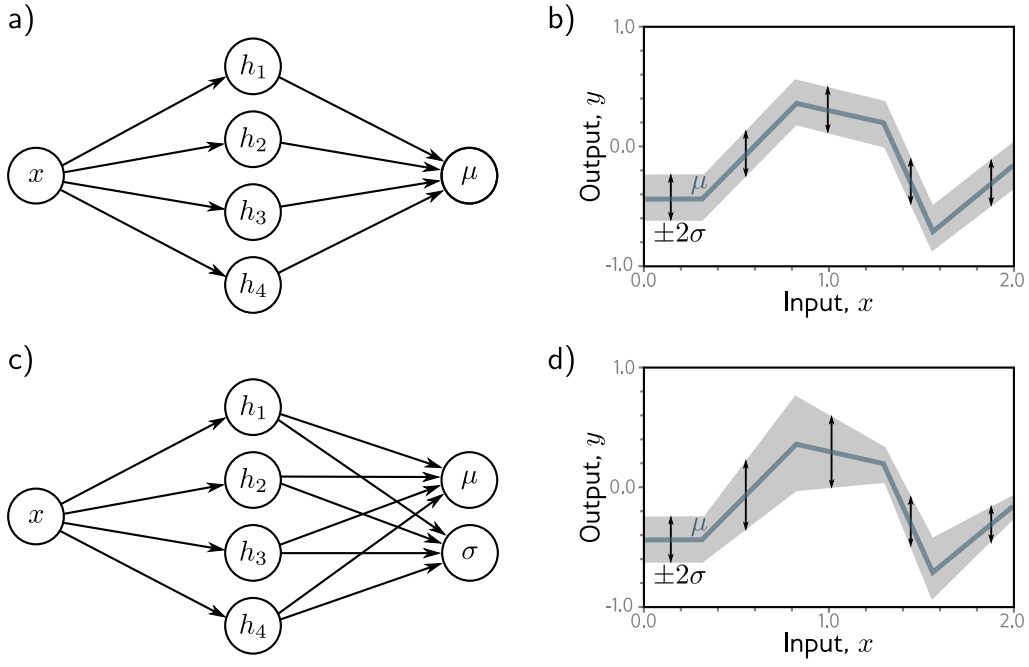


**Figure 5.5** Homoscedastic vs. heteroscedastic regression. a) A shallow neural network for homoscedastic regression predicts just the mean $\mu$ of the output distribution from the input $x$. b) The result is that while the mean (blue line) is a piecewise linear function of the input $x$, the variance is constant everywhere (arrows and gray region show $\pm 2$ standard deviations). c) A shallow neural network for heteroscedastic regression also predicts the variance $\sigma^2$ (or, more precisely, computes its square root, which we then square). d) The standard deviation now also becomes a piecewise linear function of the input $x$.

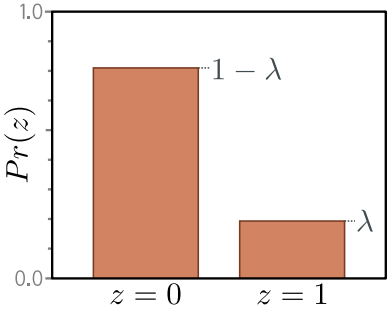


**Figure 5.6** Bernoulli distribution. The Bernoulli distribution is defined on the domain $z \in \{0, 1\}$ and has a single parameter $\lambda$ that denotes the probability of observing $z = 1$. It follows that the probability of observing $z = 0$ is $1 - \lambda$.

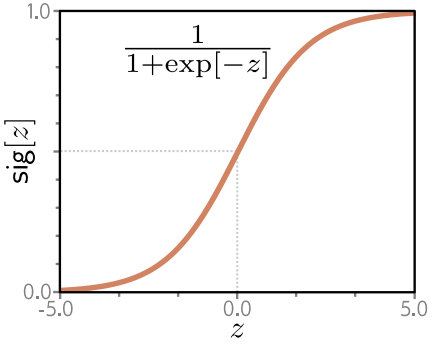


**Figure 5.7** Logistic sigmoid function. This function maps the real line $z \in \mathbb{R}$ to numbers between zero and one, so $\text{sig}[z] \in [0, 1]$. An input of 0 is mapped to 0.5. Negative inputs are mapped to numbers below 0.5, and positive inputs to numbers above 0.5.

Once again, we follow the recipe from section 5.2 to construct the loss function. First, we choose a probability distribution over the output space $y \in \{0, 1\}$. A suitable choice is the Bernoulli distribution, which is defined on the domain $\{0, 1\}$. This has a single parameter $\lambda \in [0, 1]$ that represents the probability that $y$ takes the value one (figure 5.6):

$$Pr(y|\lambda) = \begin{cases} 1-\lambda & y = 0 \\ \lambda & y = 1 \end{cases}, \tag{5.16}$$

which can equivalently be written as:

$$Pr(y|\lambda) = (1-\lambda)^{1-y} \cdot \lambda^{y}. \tag{5.17}$$

Second, we set the machine learning model $\text{f}[\mathbf{x}, \boldsymbol{\phi}]$ to predict the single distribution parameter $\lambda$. However, $\lambda$ can only take values in the range $[0, 1]$, and we cannot guarantee that the network output will lie in this range. Consequently, we pass the network output through a function that maps the real numbers $\mathbb{R}$ to $[0, 1]$. A suitable function is the *logistic sigmoid* (figure 5.7):

Problem 5.1

$$\text{sig}[z] = \frac{1}{1+\exp[-z]}. \tag{5.18}$$

Hence, we predict the distribution parameter as $\lambda = \text{sig}[\text{f}[\mathbf{x}, \boldsymbol{\phi}]]$. The likelihood is now:

$$Pr(y|\mathbf{x}) = (1 - \text{sig}[\text{f}[\mathbf{x}, \boldsymbol{\phi}]])^{1-y} \cdot \text{sig}[\text{f}[\mathbf{x}, \boldsymbol{\phi}]]^{y}. \tag{5.19}$$

This is depicted in figure 5.8 for a shallow neural network model. The loss function is the negative log-likelihood of the training set:

$$L[\boldsymbol{\phi}] = \sum_{i=1}^{I} -(1-y_i)\log\Big[1 - \text{sig}[\text{f}[\mathbf{x}_i, \boldsymbol{\phi}]]\Big] - y_i \log\Big[\text{sig}[\text{f}[\mathbf{x}_i, \boldsymbol{\phi}]]\Big]. \tag{5.20}$$

For reasons to be explained in section 5.7, this is known as the *binary cross-entropy loss*.

Notebook 5.2 Binary cross-entropy loss

The transformed model output $\text{sig}[\text{f}[\mathbf{x}, \boldsymbol{\phi}]]$ predicts the parameter $\lambda$ of the Bernoulli distribution. This represents the probability that $y = 1$, and it follows that $1 - \lambda$ represents the probability that $y = 0$. When we perform inference, we may want a point estimate of $y$, so we set $y = 1$ if $\lambda > 0.5$ and $y = 0$ otherwise.

Problem 5.2

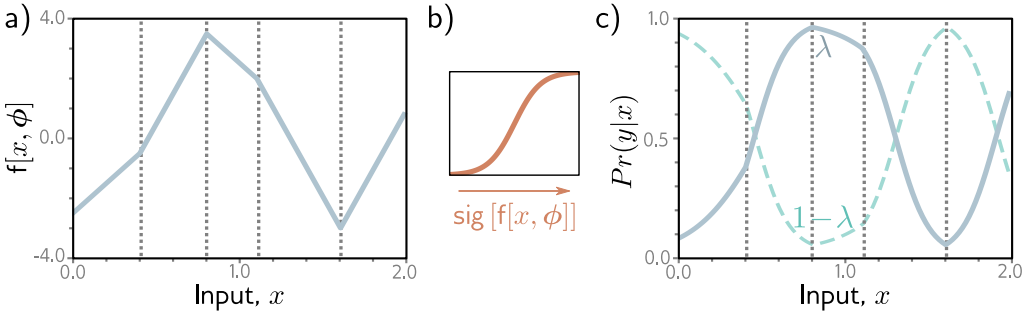


**Figure 5.8** Binary classification model. a) The network output is a piecewise linear function that can take arbitrary real values. b) This is transformed by the logistic sigmoid function, which compresses these values to the range $[0, 1]$. c) The transformed output predicts the probability $\lambda$ that $y = 1$ (solid line). The probability that $y = 0$ is hence $1 - \lambda$ (dashed line). For any fixed $x$ (vertical slice), we retrieve the two values of a Bernoulli distribution similar to that in figure 5.6. The loss function favors model parameters that produce large values of $\lambda$ at positions $x_i$ that are associated with positive examples $y_i = 1$ and small values of $\lambda$ at positions associated with negative examples $y_i = 0$.

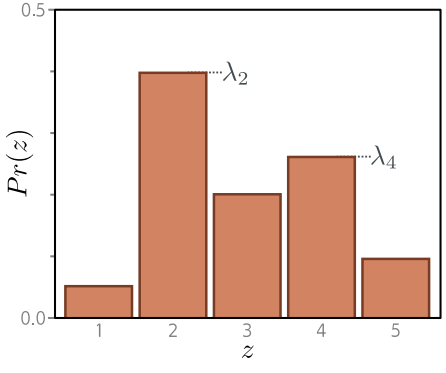


**Figure 5.9** Categorical distribution. The categorical distribution assigns probabilities to $K > 2$ categories, with associated probabilities $\lambda_1, \lambda_2, \ldots, \lambda_K$. Here, there are five categories, so $K = 5$. To ensure that this is a valid probability distribution, each parameter $\lambda_k$ must lie in the range $[0, 1]$, and all $K$ parameters must sum to one.

## 5.5 Example 3: multiclass classification

The goal of *multiclass classification* is to assign an input data example $\mathbf{x}$ to one of $K > 2$ classes, so $y \in \{1, 2, \ldots, K\}$. Real-world examples include (i) predicting which of $K = 10$ digits $y$ is present in an image $\mathbf{x}$ of a handwritten number and (ii) predicting which of $K$ possible words $y$ follows an incomplete sentence $\mathbf{x}$.

We once more follow the recipe from section 5.2. We first choose a distribution over the prediction space $y$. In this case, we have $y \in \{1, 2, \ldots, K\}$, so we choose the *categorical distribution* (figure 5.9), which is defined on this domain. This has $K$ parameters $\lambda_1, \lambda_2, \ldots, \lambda_K$, which determine the probability of each category:

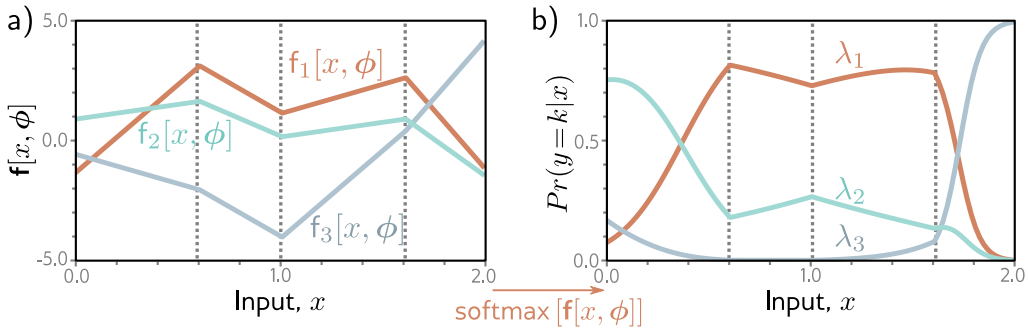


**Figure 5.10** Multiclass classification for $K=3$ classes. a) The network has three piecewise linear outputs, which can take arbitrary values. b) After the softmax function, these outputs are constrained to be non-negative and sum to one. Hence, for a given input $\mathbf{x}$, we compute valid parameters for the categorical distribution: any vertical slice of this plot produces three values that sum to one and would form the heights of the bars in a categorical distribution similar to figure 5.9.

$$Pr(y=k) = \lambda_k. \tag{5.21}$$

The parameters are constrained to take values between zero and one, and they must collectively sum to one to ensure a valid probability distribution.

Then we use a network $\mathbf{f}[\mathbf{x},\boldsymbol{\phi}]$ with $K$ outputs to compute these $K$ parameters from the input $\mathbf{x}$. Unfortunately, the network outputs will not necessarily obey the aforementioned constraints. Consequently, we pass the $K$ outputs of the network through a function that ensures these constraints are respected. A suitable choice is the *softmax* function (figure 5.10). This takes an arbitrary vector of length $K$ and returns a vector of the same length but where the elements are now in the range $[0,1]$ and sum to one. The $k^{th}$ output of the softmax function is:

$$\text{softmax}_k[\mathbf{z}] = \frac{\exp[z_k]}{\sum_{k'=1}^{K}\exp[z_{k'}]}, \tag{5.22}$$

Appendix B.1.3 Exponential function

where the exponential functions ensure positivity, and the sum in the denominator ensures that the $K$ numbers sum to one.

The likelihood that input $\mathbf{x}$ has label $y=k$ (figure 5.10) is hence:

$$Pr(y=k|\mathbf{x}) = \text{softmax}_k\Big[\mathbf{f}[\mathbf{x},\boldsymbol{\phi}]\Big]. \tag{5.23}$$

The loss function is the negative log-likelihood of the training data:

$$\begin{aligned} L[\boldsymbol{\phi}] &= -\sum_{i=1}^{I} \log\left[\text{softmax}_{y_i}\left[\mathbf{f}[\mathbf{x}_i, \boldsymbol{\phi}]\right]\right] \\ &= -\sum_{i=1}^{I}\left(\text{f}_{y_i}\left[\mathbf{x}_i, \boldsymbol{\phi}\right] - \log\left[\sum_{k'=1}^{K}\exp\left[\,\text{f}_{k'}\left[\mathbf{x}_i, \boldsymbol{\phi}\right]\right]\right]\right), \end{aligned} \tag{5.24}$$

where $\text{f}_k[\mathbf{x}, \boldsymbol{\phi}]$ denotes the $k^{th}$ output of the neural network. For reasons that will be explained in section 5.7, this is known as the *multiclass cross-entropy loss*.

The transformed model output represents a categorical distribution over possible classes $y \in \{1, 2, \ldots, K\}$. For a point estimate, we take the most probable category $\hat{y} = \text{argmax}_k[Pr(y = k|\mathbf{f}[\mathbf{x}, \hat{\boldsymbol{\phi}}])]$. This corresponds to whichever curve is highest for that value of $\mathbf{x}$ in figure 5.10.

Notebook 5.3 Multiclass cross-entropy loss

### 5.5.1 Predicting other data types

In this chapter, we have focused on regression and classification because these problems are widespread. However, to make different types of predictions, we simply choose an appropriate distribution over that domain and apply the recipe in section 5.2. Figure 5.11 enumerates a series of probability distributions and their prediction domains. Some of these are explored in the problems at the end of the chapter.

Problems 5.3–5.6

## 5.6 Multiple outputs

Often, we wish to make more than one prediction with the same model, so the target output $\mathbf{y}$ is a vector. For example, we might want to predict a molecule's melting and boiling point (a multivariate regression problem, figure 1.2b) or the object class at every point in an image (a multivariate classification problem, figure 1.4a). While it is possible to define multivariate probability distributions and use a neural network to model their parameters as a function of the input, it is more usual to treat each prediction as *independent*.

Independence implies that we treat the probability $Pr(\mathbf{y}|\mathbf{f}[\mathbf{x}, \boldsymbol{\phi}])$ as a product of univariate terms for each element $y_d \in \mathbf{y}$:

Appendix C.1.5 Independence

$$Pr(\mathbf{y}|\mathbf{f}[\mathbf{x}, \boldsymbol{\phi}]) = \prod_{d} Pr(y_d|\mathbf{f}_d[\mathbf{x}, \boldsymbol{\phi}]), \tag{5.25}$$

where $\mathbf{f}_d[\mathbf{x}, \boldsymbol{\phi}]$ is the $d^{th}$ set of network outputs, which describe the parameters of the distribution over $y_d$. For example, to predict multiple continuous variables $y_d \in \mathbb{R}$, we use a normal distribution for each $y_d$, and the network outputs $\text{f}_d[\mathbf{x}, \boldsymbol{\phi}]$ predict the means of these distributions. To predict multiple discrete variables $y_d \in \{1, 2, \ldots, K\}$, we use a categorical distribution for each $y_d$. Here, each set of network outputs $\mathbf{f}_d[\mathbf{x}, \boldsymbol{\phi}]$ predicts the $K$ values that contribute to the categorical distribution for $y_d$.

| Data Type | Domain | Distribution | Use |
|---|---|---|---|
| univariate, continuous, unbounded | $y \in \mathbb{R}$ | univariate normal | regression |
| univariate, continuous, unbounded | $y \in \mathbb{R}$ | Laplace or t-distribution | robust regression |
| univariate, continuous, unbounded | $y \in \mathbb{R}$ | mixture of Gaussians | multimodal regression |
| univariate, continuous, bounded below | $y \in \mathbb{R}^+$ | exponential or gamma | predicting magnitude |
| univariate, continuous, bounded | $y \in [0, 1]$ | beta | predicting proportions |
| multivariate, continuous, unbounded | $\mathbf{y} \in \mathbb{R}^K$ | multivariate normal | multivariate regression |
| univariate, continuous, circular | $y \in (-\pi, \pi]$ | von Mises | predicting direction |
| univariate, discrete, binary | $y \in \{0, 1\}$ | Bernoulli | binary classification |
| univariate, discrete, bounded | $y \in \{1, 2, \ldots, K\}$ | categorical | multiclass classification |
| univariate, discrete, bounded below | $y \in [0, 1, 2, 3, \ldots]$ | Poisson | predicting event counts |
| multivariate, discrete, permutation | $\mathbf{y} \in \text{Perm}[1, 2, \ldots, K]$ | Plackett-Luce | ranking |

**Figure 5.11** Distributions for loss functions for different prediction types.

When we minimize the negative log probability, this product becomes a sum of terms:

$$L[\boldsymbol{\phi}] = -\sum_{i=1}^{I} \log\Big[Pr(\mathbf{y}_i|\mathbf{f}[\mathbf{x}_i, \boldsymbol{\phi}])\Big] = -\sum_{i=1}^{I}\sum_{d} \log\Big[Pr(y_{id}|\mathbf{f}_d[\mathbf{x}_i, \boldsymbol{\phi}])\Big]. \tag{5.26}$$

where $y_{id}$ is the $d^{th}$ output from the $i^{th}$ training example.

Problems 5.7–5.10

To make two or more prediction types simultaneously, we similarly assume the errors in each are independent. For example, to predict wind direction and strength, we might choose the von Mises distribution (defined on circular domains) for the direction and the exponential distribution (defined on positive real numbers) for the strength. The independence assumption implies that the joint likelihood of the two predictions is the product of individual likelihoods. These terms will become additive when we compute the negative log-likelihood.

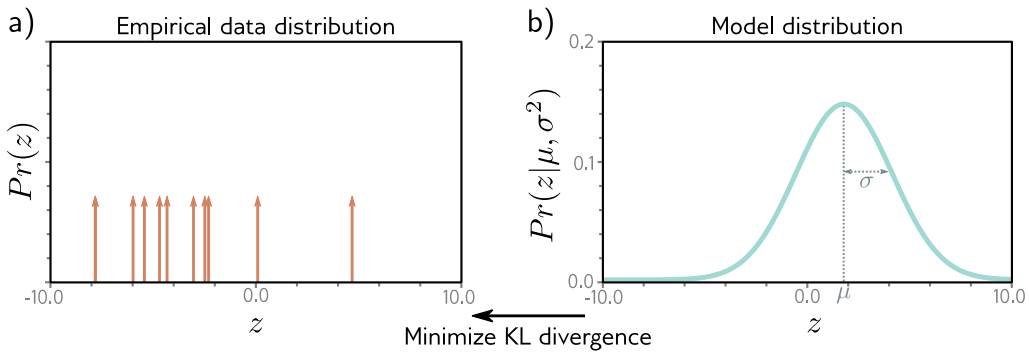


**Figure 5.12** Cross-entropy method. a) Empirical distribution of training samples (arrows denote Dirac delta functions). b) Model distribution (a normal distribution with parameters $\boldsymbol{\theta} = \{\mu, \sigma^2\}$). In the cross-entropy approach, we minimize the distance (KL divergence) between these two distributions as a function of the model parameters $\boldsymbol{\theta}$.

## 5.7 Cross-entropy loss

In this chapter, we developed loss functions that minimize negative log-likelihood. However, the term *cross-entropy* loss is also commonplace. In this section, we describe the cross-entropy loss and show that it is equivalent to using negative log-likelihood.

The cross-entropy loss is based on the idea of finding parameters $\boldsymbol{\theta}$ that minimize the distance between the empirical distribution $q(y)$ of the observed data $y$ and a model distribution $Pr(y|\boldsymbol{\theta})$ (figure 5.12). The distance between two probability distributions $q(z)$ and $p(z)$ can be evaluated using the Kullback-Leibler (KL) divergence:

Appendix C.5.1 KL Divergence

$$D_{KL}\big[q||p\big] = \int_{-\infty}^{\infty} q(z)\log\big[q(z)\big]dz - \int_{-\infty}^{\infty} q(z)\log\big[p(z)\big]dz. \tag{5.27}$$

Now consider that we observe an empirical data distribution at points $\{y_i\}_{i=1}^{I}$. We can describe this as a weighted sum of point masses:

$$q(y) = \frac{1}{I}\sum_{i=1}^{I}\delta[y - y_i], \tag{5.28}$$

where $\delta[\bullet]$ is the Dirac delta function. We want to minimize the KL divergence between the model distribution $Pr(y|\boldsymbol{\theta})$ and this empirical distribution:

Appendix B.1.3 Dirac delta function

$$\begin{aligned}\hat{\boldsymbol{\theta}} &= \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\left[\int_{-\infty}^{\infty} q(y)\log\big[q(y)\big]dy - \int_{-\infty}^{\infty} q(y)\log\big[Pr(y|\boldsymbol{\theta})\big]dy\right] \\ &= \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\left[-\int_{-\infty}^{\infty} q(y)\log\big[Pr(y|\boldsymbol{\theta})\big]dy\right],\end{aligned} \tag{5.29}$$

where the first term disappears, as it has no dependence on $\boldsymbol{\theta}$. The remaining second term is known as the *cross-entropy*. It can be interpreted as the amount of uncertainty that remains in one distribution after taking into account what we already know from the other. Now, we substitute in the definition of $q(y)$ from equation 5.28:

$$\begin{aligned}\hat{\boldsymbol{\theta}} &= \underset{\theta}{\operatorname{argmin}}\left[-\int_{-\infty}^{\infty}\left(\frac{1}{I}\sum_{i=1}^{I}\delta[y-y_i]\right)\log\big[Pr(y|\boldsymbol{\theta})\big]dy\right] \\ &= \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\left[-\frac{1}{I}\sum_{i=1}^{I}\log\big[Pr(y_i|\boldsymbol{\theta})\big]\right] \\ &= \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\left[-\sum_{i=1}^{I}\log\big[Pr(y_i|\boldsymbol{\theta})\big]\right]. \qquad (5.30)\end{aligned}$$

The product of the two terms in the first line corresponds to pointwise multiplying the point masses in figure 5.12a with the logarithm of the distribution in figure 5.12b. We are left with a finite set of weighted probability masses centered on the data points. In the last line, we have eliminated the constant scaling factor $1/I$, as this does not affect the position of the minimum.

In machine learning, the distribution parameters $\boldsymbol{\theta}$ are computed by the model $\mathbf{f}[\mathbf{x}_i, \boldsymbol{\phi}]$, so we have:

$$\hat{\boldsymbol{\phi}} = \underset{\boldsymbol{\phi}}{\operatorname{argmin}}\left[-\sum_{i=1}^{I}\log\big[Pr(y_i|\mathbf{f}[\mathbf{x}_i, \boldsymbol{\phi}])\big]\right]. \qquad (5.31)$$

This is precisely the negative log-likelihood criterion from the recipe in section 5.2.

It follows that the negative log-likelihood criterion (from maximizing the data likelihood) and the cross-entropy criterion (from minimizing the distance between the model and empirical data distributions) are equivalent.

## 5.8 Summary

We previously considered neural networks as directly predicting outputs $\mathbf{y}$ from data $\mathbf{x}$. In this chapter, we shifted perspective to think about neural networks as computing the parameters $\boldsymbol{\theta}$ of probability distributions $Pr(\mathbf{y}|\boldsymbol{\theta})$ over the output space. This led to a principled approach to building loss functions. We selected model parameters $\boldsymbol{\phi}$ that maximized the likelihood of the observed data under these distributions. We saw that this is equivalent to minimizing the negative log-likelihood.

The least squares criterion for regression is a natural consequence of this approach; it follows from the assumption that $y$ is normally distributed and that we are predicting the mean. We also saw how the regression model could be (i) extended to estimate the uncertainty over the prediction and (ii) extended to make that uncertainty dependent on the input (the heteroscedastic model). We applied the same approach to both binary and multiclass classification and derived loss functions for each. We discussed how to

tackle more complex data types and how to deal with multiple outputs. Finally, we argued that cross-entropy is an equivalent way to think about fitting models.

In previous chapters, we developed neural network models. In this chapter, we developed loss functions for deciding how well a model describes the training data for a given set of parameters. The next chapter considers model training, in which we aim to find the model parameters that minimize this loss.

## Notes

**Losses based on the normal distribution:** Nix & Weigend (1994) and Williams (1996) investigated heteroscedastic nonlinear regression in which both the mean and the variance of the output are functions of the input. In the context of unsupervised learning, Burda et al. (2016) use a loss function based on a multivariate normal distribution with diagonal covariance, and Dorta et al. (2018) use a loss function based on a normal distribution with full covariance.

**Robust regression:** Qi et al. (2020) investigate the properties of regression models that minimize mean absolute error rather than mean squared error. This loss function follows from assuming a Laplace distribution over the outputs and estimates the median output for a given input rather than the mean. Barron (2019) presents a loss function that parameterizes the degree of robustness. When interpreted in a probabilistic context, it yields a family of univariate probability distributions that includes the normal and Cauchy distributions as special cases.

**Estimating quantiles:** Sometimes, we may not want to estimate the mean or median in a regression task but may instead want to predict a quantile. For example, this is useful for risk models, where we want to know that the true value will be less than the predicted value 90% of the time. This is known as *quantile regression* (Koenker & Hallock, 2001). This could be done by fitting a heteroscedastic regression model and then estimating the quantile based on the predicted normal distribution. Alternatively, the quantiles can be estimated directly using *quantile loss* (also known as *pinball loss*). In practice, this minimizes the absolute deviations of the data from the model but weights the deviations in one direction more than the other. Recent work has investigated simultaneously predicting multiple quantiles to get an idea of the overall distribution shape (Rodrigues & Pereira, 2020).

**Class imbalance and focal loss:** Lin et al. (2017c) address data imbalance in classification problems. If the number of examples for some classes is much greater than for others, then the standard maximum likelihood loss does not work well; the model may concentrate on becoming more confident about well-classified examples from the dominant classes and classify less well-represented classes poorly. Lin et al. (2017c) introduce *focal loss*, which adds a single extra parameter that down-weights the effect of well-classified examples to improve performance.

**Learning to rank:** Cao et al. (2007), Xia et al. (2008), and Chen et al. (2009) all used the Plackett-Luce model in loss functions for learning to rank data. This is the *listwise* approach to learning to rank as the model ingests an entire list of objects to be ranked at once. Alternative approaches are the *pointwise* approach, in which the model ingests a single object, and the *pairwise* approach, where the model ingests pairs of objects. Chen et al. (2009) summarize different approaches for learning to rank.

**Other data types:** Fan et al. (2020) use a loss based on the beta distribution for predicting values between zero and one. Jacobs et al. (1991) and Bishop (1994) investigated *mixture density networks* for multimodal data. These model the output as a mixture of Gaussians

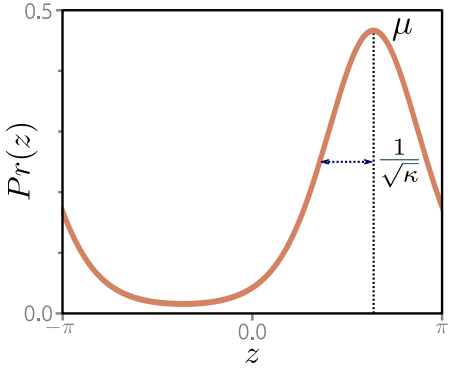


**Figure 5.13** The von Mises distribution is defined over the circular domain $(-\pi, \pi]$. It has two parameters. The mean $\mu$ determines the position of the peak. The concentration $\kappa > 0$ acts like the inverse of the variance. Hence $1/\sqrt{\kappa}$ is roughly equivalent to the standard deviation in a normal distribution.

(see figure 5.14) that is conditional on the input. Prokudin et al. (2018) used the von Mises distribution to predict direction (see figure 5.13). Fallah et al. (2009) constructed loss functions for prediction counts using the Poisson distribution (see figure 5.15). Ng et al. (2017) used loss functions based on the gamma distribution to predict duration.

**Non-probabilistic approaches:** It is not strictly necessary to adopt the probabilistic approach discussed in this chapter, but this has become the default in recent years; any loss function that aims to reduce the distance between the model output and the training outputs will suffice, and distance can be defined in any way that seems sensible. There are several well-known non-probabilistic machine learning models for classification, including support vector machines (Vapnik, 1995; Cristianini & Shawe-Taylor, 2000), which use *hinge loss*, and AdaBoost (Freund & Schapire, 1997), which uses *exponential loss*.

## Problems

**Problem 5.1** Show that the logistic sigmoid function $\text{sig}[z]$ becomes 0 as $z \to -\infty$, is 0.5 when $z = 0$, and becomes 1 when $z \to \infty$, where:

$$\text{sig}[z] = \frac{1}{1+\exp[-z]}. \tag{5.32}$$

**Problem 5.2** The loss $L$ for binary classification for a single training pair $\{\mathbf{x}, y\}$ is:

$$L = -(1-y)\log\Big[1-\text{sig}[\text{f}[\mathbf{x},\boldsymbol{\phi}]]\Big] - y\log\Big[\text{sig}[\text{f}[\mathbf{x},\boldsymbol{\phi}]]\Big], \tag{5.33}$$

where $\text{sig}[\bullet]$ is defined in equation 5.32. Plot this loss as a function of the transformed network output $\text{sig}[\text{f}[\mathbf{x},\boldsymbol{\phi}]] \in [0,1]$ (i) when the training label $y = 0$ and (ii) when $y = 1$.

**Problem 5.3*** Suppose we want to build a model that predicts the direction $y$ in radians of the prevailing wind based on local measurements of barometric pressure $\mathbf{x}$. A suitable distribution over circular domains is the von Mises distribution (figure 5.13):

$$Pr(y|\mu,\kappa) = \frac{\exp\big[\kappa\cos[y-\mu]\big]}{2\pi\cdot\text{Bessel}_0[\kappa]}, \tag{5.34}$$

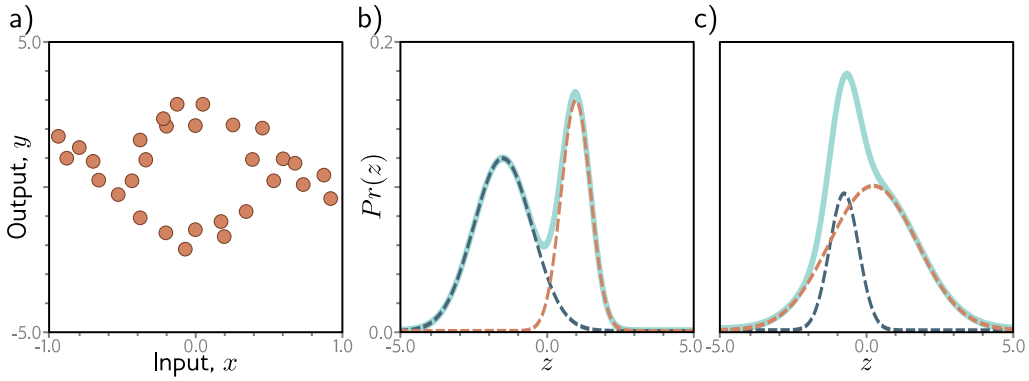


**Figure 5.14** Multimodal data and mixture of Gaussians density. a) Example training data where, for intermediate values of the input $x$, the corresponding output $y$ follows one of two paths. For example, at $x = 0$, the output $y$ might be roughly $-2$ or $+3$ but is unlikely to be between these values. b) The mixture of Gaussians is a probability model suited to this kind of data. As the name suggests, the model is a weighted sum (solid cyan curve) of two or more normal distributions with different means and variances (here, two weighted distributions, dashed blue and orange curves). When the means are far apart, this forms a multimodal distribution. c) When the means are close, the mixture can model unimodal but non-normal densities.

where $\mu$ is a measure of the mean direction and $\kappa$ is a measure of concentration (i.e., the inverse of the variance). The term $\text{Bessel}_0[\kappa]$ is a modified Bessel function of the first kind of order 0. Use the recipe from section 5.2 to develop a loss function for learning the parameter $\mu$ of a model $\text{f}[\mathbf{x}, \boldsymbol{\phi}]$ to predict the most likely wind direction. Your solution should treat the concentration $\kappa$ as constant. How would you perform inference?

**Problem 5.4*** Sometimes, the outputs $y$ for input $\mathbf{x}$ are multimodal (figure 5.14a); there is more than one valid prediction for a given input. Here, we might use a weighted sum of normal components as the distribution over the output. This is known as a *mixture of Gaussians* model. For example, a mixture of two Gaussians has parameters $\boldsymbol{\theta} = \{\lambda, \mu_1, \sigma_1^2, \mu_2, \sigma_2^2\}$:

$$Pr(y|\lambda, \mu_1, \mu_2, \sigma_1^2, \sigma_2^2) = \frac{\lambda}{\sqrt{2\pi\sigma_1^2}} \exp\left[\frac{-(y-\mu_1)^2}{2\sigma_1^2}\right] + \frac{1-\lambda}{\sqrt{2\pi\sigma_2^2}} \exp\left[\frac{-(y-\mu_2)^2}{2\sigma_2^2}\right], \tag{5.35}$$

where $\lambda \in [0, 1]$ controls the relative weight of the two components, which have means $\mu_1, \mu_2$ and variances $\sigma_1^2$, $\sigma_2^2$, respectively. This model can represent a distribution with two peaks (figure 5.14b) or a distribution with one peak but a more complex shape (figure 5.14c). Use the recipe from section 5.2 to construct a loss function for training a model $\mathbf{f}[x, \boldsymbol{\phi}]$ that takes input $x$, has parameters $\boldsymbol{\phi}$, and predicts a mixture of two Gaussians. The loss should be based on $I$ training data pairs $\{x_i, y_i\}$. What problems do you foresee when performing inference?

**Problem 5.5** Consider extending the model from problem 5.3 to predict the wind direction using a mixture of two von Mises distributions. Write an expression for the likelihood $Pr(y|\boldsymbol{\theta})$ for this model. How many outputs will the network need to produce?

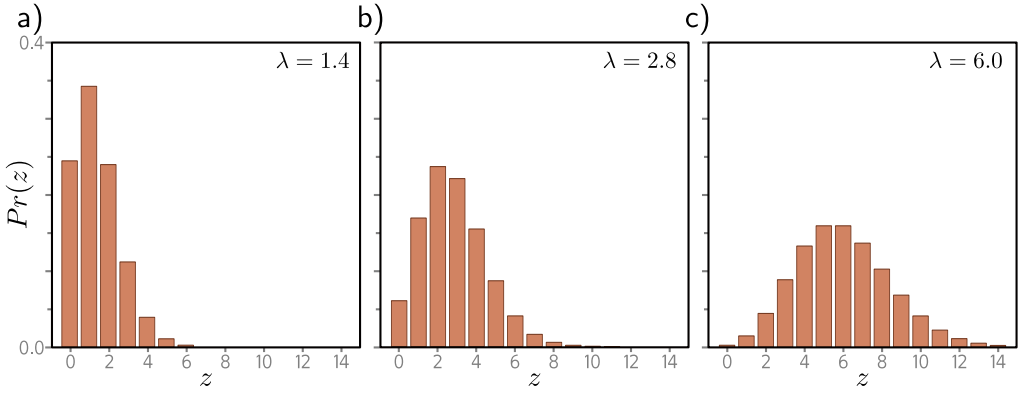


**Figure 5.15** Poisson distribution. This discrete distribution is defined over non-negative integers $z \in \{0, 1, 2, \ldots\}$. It has a single parameter $\lambda \in \mathbb{R}^+$, which is known as the rate and is the mean of the distribution. a–c) Poisson distributions with rates of 1.4, 2.8, and 6.0, respectively.

**Problem 5.6** Consider building a model to predict the number of pedestrians $y \in \{0, 1, 2, \ldots\}$ that will pass a given point in the city in the next minute, based on data $\mathbf{x}$ that contains information about the time of day, the longitude and latitude, and the type of neighborhood. A suitable distribution for modeling counts is the Poisson distribution (figure 5.15). This has a single parameter $\lambda > 0$ called the *rate* that represents the mean of the distribution. The distribution has probability density function:

$$Pr(y = k) = \frac{\lambda^k e^{-\lambda}}{k!}. \tag{5.36}$$

Design a loss function for this model assuming we have access to $I$ training pairs $\{\mathbf{x}_i, y_i\}$.

**Problem 5.7** Consider a multivariate regression problem where we predict ten outputs, so $\mathbf{y} \in \mathbb{R}^{10}$, and model each with an independent normal distribution where the means $\mu_d$ are predicted by the network, and variances $\sigma^2$ are constant. Write an expression for the likelihood $Pr(\mathbf{y}|\mathbf{f}[\mathbf{x}, \boldsymbol{\phi}])$. Show that minimizing the negative log-likelihood of this model is still equivalent to minimizing a sum of squared terms if we don't estimate the variance $\sigma^2$.

**Problem 5.8*** Construct a loss function for making multivariate predictions $\mathbf{y} \in \mathbb{R}^{D_o}$ based on independent normal distributions with different variances $\sigma_d^2$ for each dimension. Assume a heteroscedastic model so that both the means $\mu_d$ and variances $\sigma_d^2$ vary as a function of the data.

**Problem 5.9*** Consider a multivariate regression problem in which we predict the height of a person in meters and their weight in kilos from data $\mathbf{x}$. Here, the units take quite different ranges. What problems do you see this causing? Propose two solutions to these problems.

**Problem 5.10** Extend the model from problem 5.3 to predict both the wind direction and the wind speed and define the associated loss function.