Chapter 5

Machine Learning Basics

Deep learning is a specific kind of machine learning. To understan
well, one must have a solid understanding of the basic principles of m
This chapter provides a brief course in the most important general
are applied throughout the rest of the book. Novice readers or th
wider perspective are encouraged to consider machine learning te
more comprehensive coverage of the fundamentals, such as Murphy (
(2006). If you are already familiar with machine learning basics,
ahead to section 5.11. That section covers some perspectives on trad
learning techniques that have strongly influenced the development
algorithms.

We begin with a definition of what a learning algorithm is
example: the linear regression algorithm. We then proceed to de
challenge of fitting the training data differs from the challenge of f
that generalize to new data. Most machine learning algorithm
called *hyperparameters*, which must be determined outside the lear

itself; we discuss how to set these using additional data. Mach
essentially a form of applied statistics with increased emphasis
computers to statistically estimate complicated functions and a decr
on proving confidence intervals around these functions; we theref
two central approaches to statistics: frequentist estimators and Ba
Most machine learning algorithms can be divided into the categori
learning and unsupervised learning; we describe these categories
examples of simple learning algorithms from each category. Mos
algorithms are based on an optimization algorithm called stoc

descent. We describe how to combine various algorithm compo
an optimization algorithm, a cost function, a model, and a data
machine learning algorithm. Finally, in section 5.11, we descri
factors that have limited the ability of traditional machine learnin
These challenges have motivated the development of deep learning
overcome these obstacles.

5.1 Learning Algorithms

A machine learning algorithm is an algorithm that is able to le
But what do we mean by learning? Mitchell (1997) provides a suc
"A computer program is said to learn from experience E with r
class of tasks T and performance measure P, if its performance a
measured by P, improves with experience E." One can imagine a
experiences E, tasks T, and performance measures P, and we do
this book to formally define what may be used for each of these e
in the following sections, we provide intuitive descriptions and e
different kinds of tasks, performance measures, and experiences th
to construct machine learning algorithms.

5.1.1 The Task, T

Machine learning enables us to tackle tasks that are too difficul
fixed programs written and designed by human beings. From a
philosophical point of view, machine learning is interesting because
understanding of it entails developing our understanding of the
underlie intelligence.

In this relatively formal definition of the word "task," the pro
itself is not the task. Learning is our means of attaining the ability

task. For example, if we want a robot to be able to walk, then wal
We could program the robot to learn to walk, or we could attempt t
a program that specifies how to walk manually.

Machine learning tasks are usually described in terms of ho
learning system should process an **example**. An example is a collect
that have been quantitatively measured from some object or even
the machine learning system to process. We typically represent a
vector $\boldsymbol{x} \in \mathbb{R}^n$ where each entry x_i of the vector is another feature
the features of an image are usually the values of the pixels in the

Many kinds of tasks can be solved with machine learning. So
common machine learning tasks include the following:

- **Classification**: In this type of task, the computer program is
  which of k categories some input belongs to. To solve this ta
  algorithm is usually asked to produce a function $f : \mathbb{R}^n \to \{1$
  $y = f(\boldsymbol{x})$, the model assigns an input described by vector $\boldsymbol{x}$
  identified by numeric code y. There are other variants of th
  task, for example, where f outputs a probability distributi
  An example of a classification task is object recognition, w
  is an image (usually described as a set of pixel brightness v
  output is a numeric code identifying the object in the image
  the Willow Garage PR2 robot is able to act as a waiter tha
  different kinds of drinks and deliver them to people on co
  fellow *et al.*, 2010). Modern object recognition is best acc
  deep learning (Krizhevsky *et al.*, 2012; Ioffe and Szegedy,
  recognition is the same basic technology that enables comput
  faces (Taigman *et al.*, 2014), which can be used to automatic
  in photo collections and for computers to interact more natu
  users.

- **Classification with missing inputs**: Classification beco
  lenging if the computer program is not guaranteed that every
  its input vector will always be provided. To solve the classifi
  learning algorithm only has to define a *single* function mappin
  input to a categorical output. When some of the inputs m
  rather than providing a single classification function, the lea
  must learn a *set* of functions. Each function corresponds to cl
  a different subset of its inputs missing. This kind of situation a
  in medical diagnosis, because many kinds of medical tests a

  invasive. One way to efficiently define such a large set of

invasive. One way to efficiently define such a large set of
learn a probability distribution over all the relevant variables
classification task by marginalizing out the missing variables
variables, we can now obtain all 2^n different classification fu
for each possible set of missing inputs, but the computer
to learn only a single function describing the joint probabili
See Goodfellow *et al.* (2013b) for an example of a deep prob
applied to such a task in this way. Many of the other tasks d
section can also be generalized to work with missing input
with missing inputs is just one example of what machine lea

- **Regression**: In this type of task, the computer program is as
numerical value given some input. To solve this task, the lea
is asked to output a function $f : \mathbb{R}^n \to \mathbb{R}$. This type of ta
classification, except that the format of output is different.
a regression task is the prediction of the expected claim a
insured person will make (used to set insurance premiums), o
of future prices of securities. These kinds of predictions ar
algorithmic trading.

- **Transcription**: In this type of task, the machine learning
to observe a relatively unstructured representation of som
and transcribe the information into discrete textual form. I
optical character recognition, the computer program is show
containing an image of text and is asked to return this text
a sequence of characters (e.g., in ASCII or Unicode format)
View uses deep learning to process address numbers in this w
et al., 2014d). Another example is speech recognition, wher
program is provided an audio waveform and emits a sequence
word ID codes describing the words that were spoken in the
Deep learning is a crucial component of modern speech recog
used at major companies, including Microsoft, IBM and G
et al., 2012b).

- **Machine translation**: In a machine translation task, the
consists of a sequence of symbols in some language, and the co
must convert this into a sequence of symbols in another lan
commonly applied to natural languages, such as translating f
French. Deep learning has recently begun to have an impor
this kind of task (Sutskever *et al.*, 2014; Bahdanau *et al.*, 20

- **Structured output** : Structured output tasks involve any

output is a vector (or other data structure containing multip
important relationships between the different elements. T
category and subsumes the transcription and translation t
above, as well as many other tasks. One example is parsi
natural language sentence into a tree that describes its gramm
by tagging nodes of the trees as being verbs, nouns, adve
See Collobert (2011) for an example of deep learning appli
task. Another example is pixel-wise segmentation of ima
computer program assigns every pixel in an image to a sp

For example, deep learning can be used to annotate the loc
in aerial photographs (Mnih and Hinton, 2010). The out
not mirror the structure of the input as closely as in these a
tasks. For example, in image captioning, the computer progr
image and outputs a natural language sentence describing th
et al., 2014a,b; Mao *et al.*, 2015; Vinyals *et al.*, 2015b; Dona
Karpathy and Li, 2015; Fang *et al.*, 2015; Xu *et al.*, 2015
are called *structured output tasks* because the program must
values that are all tightly interrelated. For example, the wor
an image captioning program must form a valid sentence.

- **Anomaly detection**: In this type of task, the computer
through a set of events or objects and flags some of them as
or atypical. An example of an anomaly detection task is cr
detection. By modeling your purchasing habits, a credit car
detect misuse of your cards. If a thief steals your credit card
information, the thief's purchases will often come from a diffe
distribution over purchase types than your own. The credit
can prevent fraud by placing a hold on an account as soon a
been used for an uncharacteristic purchase. See Chandola *et*
survey of anomaly detection methods.

- **Synthesis and sampling**: In this type of task, the mach
gorithm is asked to generate new examples that are similar
training data. Synthesis and sampling via machine learning
for media applications when generating large volumes of co
would be expensive, boring, or require too much time. For
games can automatically generate textures for large objects
rather than requiring an artist to manually label each pixel (L
In some cases, we want the sampling or synthesis procedur
specific kind of output given the input. For example, in a sp

task, we provide a written sentence and ask the program to
waveform containing a spoken version of that sentence. T
structured output task, but with the added qualification t
single correct output for each input, and we explicitly desire
of variation in the output, in order for the output to seem m
realistic.

- **Imputation of missing values**: In this type of task, the m
algorithm is given a new example $\boldsymbol{x} \in \mathbb{R}^n$, but with some

missing. The algorithm must provide a prediction of the value
entries.

- **Denoising**: In this type of task, the machine learning algori
input a *corrupted example* $\tilde{\boldsymbol{x}} \in \mathbb{R}^n$ obtained by an unknown co
from a *clean example* $\boldsymbol{x} \in \mathbb{R}^n$. The learner must predict the
$\boldsymbol{x}$ from its corrupted version $\tilde{\boldsymbol{x}}$, or more generally predict
probability distribution $p(\boldsymbol{x} \mid \tilde{\boldsymbol{x}})$.

- **Density estimation** or **probability mass function estim**
density estimation problem, the machine learning algorithm is
function $p_{\text{model}} : \mathbb{R}^n \rightarrow \mathbb{R}$, where $p_{\text{model}}(\boldsymbol{x})$ can be interpreted
density function (if $\mathbf{x}$ is continuous) or a probability mass f
discrete) on the space that the examples were drawn from. To
well (we will specify exactly what that means when we discu
measures P), the algorithm needs to learn the structure of the
It must know where examples cluster tightly and where they
occur. Most of the tasks described above require the learning
least implicitly capture the structure of the probability distri
estimation enables us to explicitly capture that distributio
we can then perform computations on that distribution to
tasks as well. For example, if we have performed density estim
a probability distribution $p(\boldsymbol{x})$, we can use that distributi
missing value imputation task. If a value x_i is missing, an
values, denoted $\boldsymbol{x}_{-i}$, are given, then we know the distribution
by $p(x_i \mid \boldsymbol{x}_{-i})$. In practice, density estimation does not alwa
solve all these related tasks, because in many cases the requ
on $p(\boldsymbol{x})$ are computationally intractable.

Of course, many other tasks and types of tasks are possible. Th
we list here are intended only to provide examples of what machi

do, not to define a rigid taxonomy of tasks.

5.1.2 The Performance Measure, P

To evaluate the abilities of a machine learning algorithm, we
quantitative measure of its performance. Usually this performanc
specific to the task T being carried out by the system.

For tasks such as classification, classification with missing in
scription, we often measure the **accuracy** of the model. Accu

proportion of examples for which the model produces the correct o
also obtain equivalent information by measuring the **error rate**,
of examples for which the model produces an incorrect output. W
the error rate as the expected 0-1 loss. The 0-1 loss on a particul
if it is correctly classified and 1 if it is not. For tasks such as den
it does not make sense to measure accuracy, error rate, or any ot
loss. Instead, we must use a different performance metric that g
a continuous-valued score for each example. The most common
report the average log-probability the model assigns to some exam

Usually we are interested in how well the machine learning algo
on data that it has not seen before, since this determines how well it
deployed in the real world. We therefore evaluate these performance
a **test set** of data that is separate from the data used for traini
learning system.

The choice of performance measure may seem straightforward
but it is often difficult to choose a performance measure that corr
the desired behavior of the system.

In some cases, this is because it is difficult to decide what shou
For example, when performing a transcription task, should we measu
of the system at transcribing entire sequences, or should we use a m
performance measure that gives partial credit for getting some e
sequence correct? When performing a regression task, should w
system more if it frequently makes medium-sized mistakes or if
very large mistakes? These kinds of design choices depend on the

In other cases, we know what quantity we would ideally like t
measuring it is impractical. For example, this arises frequently in
density estimation. Many of the best probabilistic models repres
distributions only implicitly. Computing the actual probability va
a specific point in space in many such models is intractable. In t

must design an alternative criterion that still corresponds to the de
or design a good approximation to the desired criterion.

5.1.3 The Experience, E

Machine learning algorithms can be broadly categorized as **un**
supervised by what kind of experience they are allowed to h
learning process.

Most of the learning algorithms in this book can be understood a

to experience an entire **dataset**. A dataset is a collection of mar
defined in section 5.1.1. Sometimes we call examples **data points**

One of the oldest datasets studied by statisticians and mach
searchers is the Iris dataset (Fisher, 1936). It is a collection of
of different parts of 150 iris plants. Each individual plant corre
example. The features within each example are the measuremen
of the plant: the sepal length, sepal width, petal length and pe
dataset also records which species each plant belonged to. Three
are represented in the dataset.

Unsupervised learning algorithms experience a dataset co
features, then learn useful properties of the structure of this dataset
of deep learning, we usually want to learn the entire probability d
generated a dataset, whether explicitly, as in density estimation, o
tasks like synthesis or denoising. Some other unsupervised learn
perform other roles, like clustering, which consists of dividing th
clusters of similar examples.

Supervised learning algorithms experience a dataset cont
but each example is also associated with a **label** or **target**. For ex
dataset is annotated with the species of each iris plant. A supe
algorithm can study the Iris dataset and learn to classify iris pl
different species based on their measurements.

Roughly speaking, unsupervised learning involves observing se
of a random vector $\mathbf{x}$ and attempting to implicitly or explicitly l
bility distribution $p(\mathbf{x})$, or some interesting properties of that dist
supervised learning involves observing several examples of a rando
an associated value or vector $\mathbf{y}$, then learning to predict $\mathbf{y}$ from
estimating $p(\mathbf{y} \mid \mathbf{x})$. The term **supervised learning** originates fr
the target $\mathbf{y}$ being provided by an instructor or teacher who show
learning system what to do. In unsupervised learning, there is n

teacher, and the algorithm must learn to make sense of the data wit

Unsupervised learning and supervised learning are not formally
The lines between them are often blurred. Many machine learning t
be used to perform both tasks. For example, the chain rule of pr
that for a vector $\mathbf{x} \in \mathbb{R}^n$, the joint distribution can be decomposed

$$p(\mathbf{x}) = \prod_{i=1}^{n} p(\mathrm{x}_i \mid \mathrm{x}_1, \ldots, \mathrm{x}_{i-1}).$$

This decomposition means that we can solve the ostensibly unsuperv

modeling $p(\mathbf{x})$ by splitting it into n supervised learning problems. A
can solve the supervised learning problem of learning $p(y \mid \mathbf{x})$ by u
unsupervised learning technologies to learn the joint distributio
inferring

$$p(y \mid \mathbf{x}) = \frac{p(\mathbf{x}, y)}{\sum_{y'} p(\mathbf{x}, y')}.$$

Though unsupervised learning and supervised learning are not co
or distinct concepts, they do help roughly categorize some of the th
machine learning algorithms. Traditionally, people refer to regressio
and structured output problems as supervised learning. Densit
support of other tasks is usually considered unsupervised learning

Other variants of the learning paradigm are possible. For ex
supervised learning, some examples include a supervision target
not. In multi-instance learning, an entire collection of example
containing or not containing an example of a class, but the indiv
of the collection are not labeled. For a recent example of multi-in
with deep models, see Kotzias *et al.* (2015).

Some machine learning algorithms do not just experience a fix
example, **reinforcement learning** algorithms interact with an e
there is a feedback loop between the learning system and its exp
algorithms are beyond the scope of this book. Please see Sutton a
or Bertsekas and Tsitsiklis (1996) for information about reinforc
and Mnih *et al.* (2013) for the deep learning approach to reinforc

Most machine learning algorithms simply experience a dataset
be described in many ways. In all cases, a dataset is a collectio
which are in turn collections of features.

One common way of describing a dataset is with a **design ma**
matrix is a matrix containing a different example in each row. Eac
matrix corresponds to a different feature. For instance, the Iris d

150 examples with four features for each example. This means w
the dataset with a design matrix $\boldsymbol{X} \in \mathbb{R}^{150 \times 4}$, where $X_{i,1}$ is the
plant i, $X_{i,2}$ is the sepal width of plant i, etc. We describe most
algorithms in this book in terms of how they operate on design m

Of course, to describe a dataset as a design matrix, it must
describe each example as a vector, and each of these vectors must b
This is not always possible. For example, if you have a collection
with different widths and heights, then different photographs will
numbers of pixels, so not all the photographs may be described

length of vector. In Section 9.7 and chapter 10, we describe how to
types of such heterogeneous data. In cases like these, rather than
dataset as a matrix with m rows, we describe it as a set containi
$\{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(m)}\}$. This notation does not imply that any two e
$\boldsymbol{x}^{(i)}$ and $\boldsymbol{x}^{(j)}$ have the same size.

In the case of supervised learning, the example contains a lab
well as a collection of features. For example, if we want to use a lea
to perform object recognition from photographs, we need to speci
appears in each of the photos. We might do this with a numer
signifying a person, 1 signifying a car, 2 signifying a cat, and so for
working with a dataset containing a design matrix of feature obse
also provide a vector of labels $\boldsymbol{y}$, with y_i providing the label for ex

Of course, sometimes the label may be more than just a sing
example, if we want to train a speech recognition system to tr
sentences, then the label for each example sentence is a sequence

Just as there is no formal definition of supervised and unsupe
there is no rigid taxonomy of datasets or experiences. The structures
cover most cases, but it is always possible to design new ones for n

5.1.4 Example: Linear Regression

Our definition of a machine learning algorithm as an algorithm
of improving a computer program's performance at some task vi
somewhat abstract. To make this more concrete, we present an
simple machine learning algorithm: **linear regression**. We will
example repeatedly as we introduce more machine learning concep
understand the algorithm's behavior.

As the name implies, linear regression solves a regression pro
words, the goal is to build a system that can take a vector

$\boldsymbol{x} \in \mathbb{R}$

predict the value of a scalar $y \in \mathbb{R}$ as its output. The output of lin
a linear function of the input. Let $\hat{y}$ be the value that our model pr
take on. We define the output to be

$$\hat{y} = \boldsymbol{w}^\top \boldsymbol{x},$$

where $\boldsymbol{w} \in \mathbb{R}^n$ is a vector of **parameters**.

Parameters are values that control the behavior of the system. In
the coefficient that we multiply by feature x_i before summing up th
from all the features. We can think of $\boldsymbol{w}$ as a set of **weights** that

each feature affects the prediction. If a feature x_i receives a pos
then increasing the value of that feature increases the value of o
If a feature receives a negative weight, then increasing the value
decreases the value of our prediction. If a feature's weight is larg
then it has a large effect on the prediction. If a feature's weight is
effect on the prediction.

We thus have a definition of our task T: to predict y from $\boldsymbol{x}$
$\hat{y} = \boldsymbol{w}^\top \boldsymbol{x}$. Next we need a definition of our performance measure,

Suppose that we have a design matrix of m example inputs t
use for training, only for evaluating how well the model performs
a vector of regression targets providing the correct value of y fo
examples. Because this dataset will only be used for evaluation, we
set. We refer to the design matrix of inputs as $\boldsymbol{X}^{(\text{test})}$ and the vect
targets as $\boldsymbol{y}^{(\text{test})}$.

One way of measuring the performance of the model is to com
squared error of the model on the test set. If $\hat{\boldsymbol{y}}^{(\text{test})}$ gives the pr
model on the test set, then the mean squared error is given by

$$\text{MSE}_{\text{test}} = \frac{1}{m} \sum_i (\hat{\boldsymbol{y}}^{(\text{test})} - \boldsymbol{y}^{(\text{test})})_i^2.$$

Intuitively, one can see that this error measure decreases to 0 when
We can also see that

$$\text{MSE}_{\text{test}} = \frac{1}{m} ||\hat{\boldsymbol{y}}^{(\text{test})} - \boldsymbol{y}^{(\text{test})}||_2^2,$$

so the error increases whenever the Euclidean distance between
and the targets increases.

To make a machine learning algorithm, we need to design an

will improve the weights $\boldsymbol{w}$ in a way that reduces MSE^{test} when

will improve the weights $\boldsymbol{w}$ in a way that reduces MSE^{test} when
is allowed to gain experience by observing a training set ($\boldsymbol{X}^{(\text{train}}$
intuitive way of doing this (which we justify later, in section 5
minimize the mean squared error on the training set, $\text{MSE}_{\text{train}}$.

To minimize $\text{MSE}_{\text{train}}$, we can simply solve for where its gradi

$$\nabla_{\boldsymbol{w}} \text{MSE}_{\text{train}} = 0$$

$$\Rightarrow \nabla_{\boldsymbol{w}} \frac{1}{m} ||\hat{\boldsymbol{y}}^{(\text{train})} - \boldsymbol{y}^{(\text{train})}||_2^2 = 0$$

$$\Rightarrow \frac{1}{m}\nabla_{\boldsymbol{w}}||\boldsymbol{X}^{(\text{train})}\boldsymbol{w} - \boldsymbol{y}^{(\text{train})}||_2^2 = 0$$

$$\Rightarrow \nabla_{\boldsymbol{w}}\left(\boldsymbol{X}^{(\text{train})}\boldsymbol{w} - \boldsymbol{y}^{(\text{train})}\right)^\top\left(\boldsymbol{X}^{(\text{train})}\boldsymbol{w} - \boldsymbol{y}^{(\text{train})}\right)$$

$$\Rightarrow \nabla_{\boldsymbol{w}}\left(\boldsymbol{w}^\top\boldsymbol{X}^{(\text{train})\top}\boldsymbol{X}^{(\text{train})}\boldsymbol{w} - 2\boldsymbol{w}^\top\boldsymbol{X}^{(\text{train})\top}\boldsymbol{y}^{(\text{train})} + \boldsymbol{y}^{(\text{train})\top}\right.$$

$$\Rightarrow 2\boldsymbol{X}^{(\text{train})\top}\boldsymbol{X}^{(\text{train})}\boldsymbol{w} - 2\boldsymbol{X}^{(\text{train})\top}\boldsymbol{y}^{(\text{train})} = 0$$

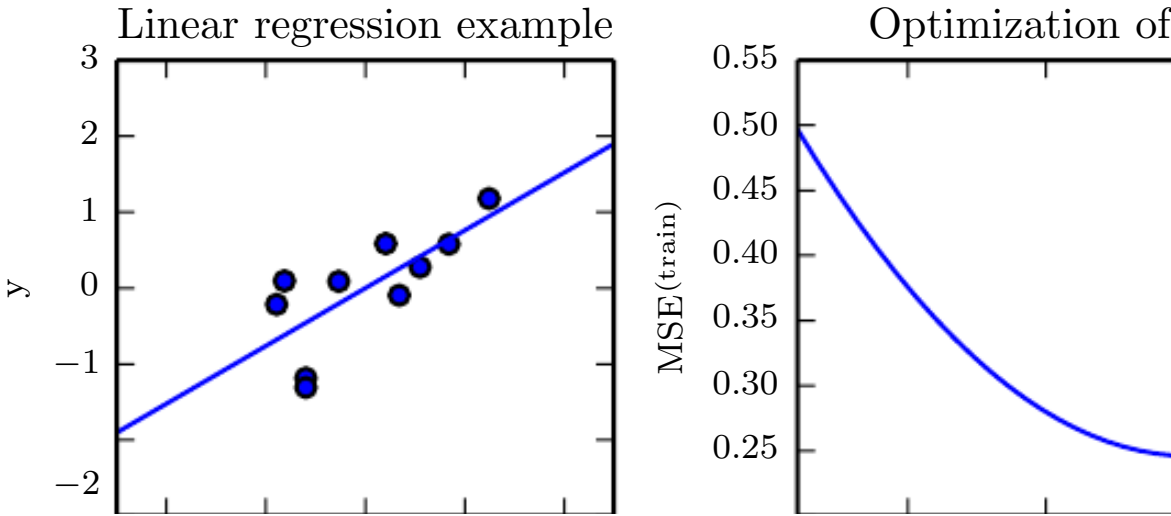
$$\Rightarrow \boldsymbol{w} = \left(\boldsymbol{X}^{(\text{train})\top}\boldsymbol{X}^{(\text{train})}\right)^{-1}\boldsymbol{X}^{(\text{train})\top}\boldsymbol{y}^{(\text{train})}$$

The system of equations whose solution is given by equation 5
the **normal equations**. Evaluating equation 5.12 constitutes a
algorithm. For an example of the linear regression learning algo
see figure 5.1.

It is worth noting that the term **linear regression** is often u
a slightly more sophisticated model with one additional paramete
term b. In this model

$$\hat{y} = \boldsymbol{w}^\top\boldsymbol{x} + b,$$

so the mapping from parameters to predictions is still a linear fu
mapping from features to predictions is now an affine function. Th



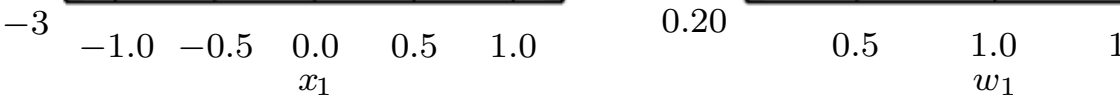


Figure 5.1: A linear regression problem, with a training set consisting of
each containing one feature. Because there is only one feature, the
contains only a single parameter to learn, w_1. *(Left)*Observe that linear
to set w_1 such that the line $y = w_1 x$ comes as close as possible to passin
training points. *(Right)*The plotted point indicates the value of w_1 foun
equations, which we can see minimizes the mean squared error on the t

affine functions means that the plot of the model's predictions s
line, but it need not pass through the origin. Instead of adding the
b, one can continue to use the model with only weights but augr
extra entry that is always set to 1. The weight corresponding to th
plays the role of the bias parameter. We frequently use the term
referring to affine functions throughout this book.

The intercept term b is often called the **bias** parameter of the
mation. This terminology derives from the point of view that th
transformation is biased toward being b in the absence of any in
is different from the idea of a statistical bias, in which a statist
algorithm's expected estimate of a quantity is not equal to the tru

Linear regression is of course an extremely simple and limited lea
but it provides an example of how a learning algorithm can work.
sections we describe some of the basic principles underlying lear
design and demonstrate how these principles can be used to build m
learning algorithms.

5.2 Capacity, Overfitting and Underfitting

The central challenge in machine learning is that our algorithm
well on *new, previously unseen* inputs—not just those on which
trained. The ability to perform well on previously unobserved i
generalization.

Typically, when training a machine learning model, we have acce
set; we can compute some error measure on the training set, calle
error; and we reduce this training error. So far, what we have desc
an optimization problem. What separates machine learning from
that we want the **generalization error**, also called the **test erro**

well. The generalization error is defined as the expected value of

well. The generalization error is defined as the expected value o
new input. Here the expectation is taken across different possible
from the distribution of inputs we expect the system to encounter

We typically estimate the generalization error of a machine lea
measuring its performance on a **test set** of examples that were colle
from the training set.

In our linear regression example, we trained the model by
training error,

$$\frac{1}{m^{(\text{train})}}||\boldsymbol{X}^{(\text{train})}\boldsymbol{w} - \boldsymbol{y}^{(\text{train})}||_2^2,$$

but we actually care about the test error, $\frac{1}{m^{(\text{test})}}||\boldsymbol{X}^{(\text{test})}\boldsymbol{w} - \boldsymbol{y}^{(\text{test}}$

How can we affect performance on the test set when we can o
training set? The field of **statistical learning theory** provides s
the training and the test set are collected arbitrarily, there is inde
do. If we are allowed to make some assumptions about how the tr
set are collected, then we can make some progress.

The training and test data are generated by a probability di
datasets called the **data-generating process**. We typically ma
sumptions known collectively as the **i.i.d. assumptions**. These a
that the examples in each dataset are **independent** from each o
the training set and test set are **identically distributed**, drawn
probability distribution as each other. This assumption enables
the data-generating process with a probability distribution over a
The same distribution is then used to generate every train example
example. We call that shared underlying distribution the **data-g**
tribution, denoted p_{data}. This probabilistic framework and the i.i
enables us to mathematically study the relationship between tra
test error.

One immediate connection we can observe between training erro
is that the expected training error of a randomly selected model
expected test error of that model. Suppose we have a probabili
$p(\boldsymbol{x}, y)$ and we sample from it repeatedly to generate the training s
set. For some fixed value $\boldsymbol{w}$, the expected training set error is exac
the expected test set error, because both expectations are formed
dataset sampling process. The only difference between the two co
name we assign to the dataset we sample.

Of course, when we use a machine learning algorithm, we
parameters ahead of time, then sample both datasets. We sample t
then use it to choose the parameters to reduce training set error, t

test set. Under this process, the expected test error is greater th
the expected value of training error. The factors determining how
learning algorithm will perform are its ability to

1. Make the training error small.

2. Make the gap between training and test error small.

These two factors correspond to the two central challenges in m
underfitting and **overfitting**. Underfitting occurs when the mod

obtain a sufficiently low error value on the training set. Overfitti
the gap between the training error and test error is too large.

We can control whether a model is more likely to overfit or und
its **capacity**. Informally, a model's capacity is its ability to fit a
functions. Models with low capacity may struggle to fit the traini
with high capacity can overfit by memorizing properties of the trai
not serve them well on the test set.

One way to control the capacity of a learning algorithm is
hypothesis space, the set of functions that the learning algorith
select as being the solution. For example, the linear regression alg
set of all linear functions of its input as its hypothesis space. We
linear regression to include polynomials, rather than just linear f
hypothesis space. Doing so increases the model's capacity.

A polynomial of degree 1 gives us the linear regression model
are already familiar, with the prediction

$$\hat{y} = b + wx.$$

By introducing x^2 as another feature provided to the linear regres
can learn a model that is quadratic as a function of x:

$$\hat{y} = b + w_1 x + w_2 x^2 .$$

Though this model implements a quadratic function of its *input*
still a linear function of the *parameters*, so we can still use the no
to train the model in closed form. We can continue to add more
additional features, for example, to obtain a polynomial of degree

$$\hat{y} = b + \sum_{i=1}^{9} w_i x^i .$$

Machine learning algorithms will generally perform best when
is appropriate for the true complexity of the task they need to p
amount of training data they are provided with. Models with insu
are unable to solve complex tasks. Models with high capacity can
tasks, but when their capacity is higher than needed to solve the pr
may overfit.

Figure 5.2 shows this principle in action. We compare a lin
and degree-9 predictor attempting to fit a problem where the t

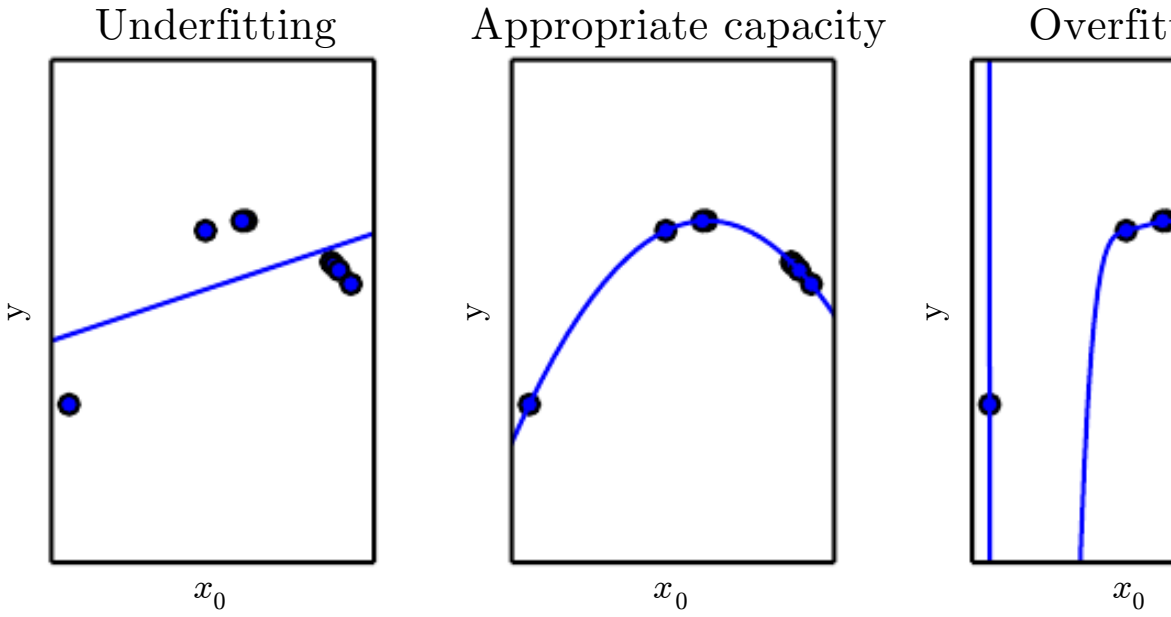


Figure 5.2: We fit three models to this example training set. The tr
generated synthetically, by randomly sampling x values and choosing y
by evaluating a quadratic function. *(Left)*A linear function fit to the d
underfitting—it cannot capture the curvature that is present in the d
quadratic function fit to the data generalizes well to unseen points. It doe
a significant amount of overfitting or underfitting. *(Right)*A polynomi
to the data suffers from overfitting. Here we used the Moore-Penrose
solve the underdetermined normal equations. The solution passes throug
points exactly, but we have not been lucky enough for it to extract the
It now has a deep valley between two training points that does not ap
underlying function. It also increases sharply on the left side of the dat
function decreases in this area.

function is quadratic. The linear function is unable to capture th
the true underlying problem, so it underfits. The degree-9 predict
representing the correct function, but it is also capable of represe
many other functions that pass exactly through the training poi
have more parameters than training examples. We have little cha
a solution that generalizes well when so many wildly different sol

this example, the quadratic model is perfectly matched to the tr
the task, so it generalizes well to new data.

So far we have described only one way of changing a model
changing the number of input features it has, and simultaneous
parameters associated with those features. There are in fact many
a model's capacity. Capacity is not determined only by the choice
model specifies which family of functions the learning algorithm c
when varying the parameters in order to reduce a training objectiv
the **representational capacity** of the model. In many cases, fi

function within this family is a difficult optimization problem. I
learning algorithm does not actually find the best function, but n
significantly reduces the training error. These additional limitatio
imperfection of the optimization algorithm, mean that the learni
effective capacity may be less than the representational capacit
family.

Our modern ideas about improving the generalization of ma
models are refinements of thought dating back to philosophers at l
Ptolemy. Many early scholars invoke a principle of parsimony th
widely known as **Occam's razor** (c. 1287–1347). This principle sta
competing hypotheses that explain known observations equally w
choose the "simplest" one. This idea was formalized and made m
the twentieth century by the founders of statistical learning theo
Chervonenkis, 1971; Vapnik, 1982; Blumer *et al.*, 1989; Vapnik, 19

Statistical learning theory provides various means of quantifying
Among these, the most well known is the **Vapnik-Chervonenkis**
VC dimension. The VC dimension measures the capacity of a binar
VC dimension is defined as being the largest possible value of m
exists a training set of m different $\boldsymbol{x}$ points that the classifier can

Quantifying the capacity of the model enables statistical lear
make quantitative predictions. The most important results in stat
theory show that the discrepancy between training error and gene
is bounded from above by a quantity that grows as the model cap
shrinks as the number of training examples increases (Vapnik and
1971; Vapnik, 1982; Blumer *et al.*, 1989; Vapnik, 1995). These b
intellectual justification that machine learning algorithms can wor
rarely used in practice when working with deep learning algorith
part because the bounds are often quite loose and in part because
difficult to determine the capacity of deep learning algorithms.

determining the capacity of a deep learning model is especially d
the effective capacity is limited by the capabilities of the optimiza
and we have little theoretical understanding of the general nonconv
problems involved in deep learning.

We must remember that while simpler functions are more like
(to have a small gap between training and test error), we mus
sufficiently complex hypothesis to achieve low training error. Typ
error decreases until it asymptotes to the minimum possible error
capacity increases (assuming the error measure has a minimum va

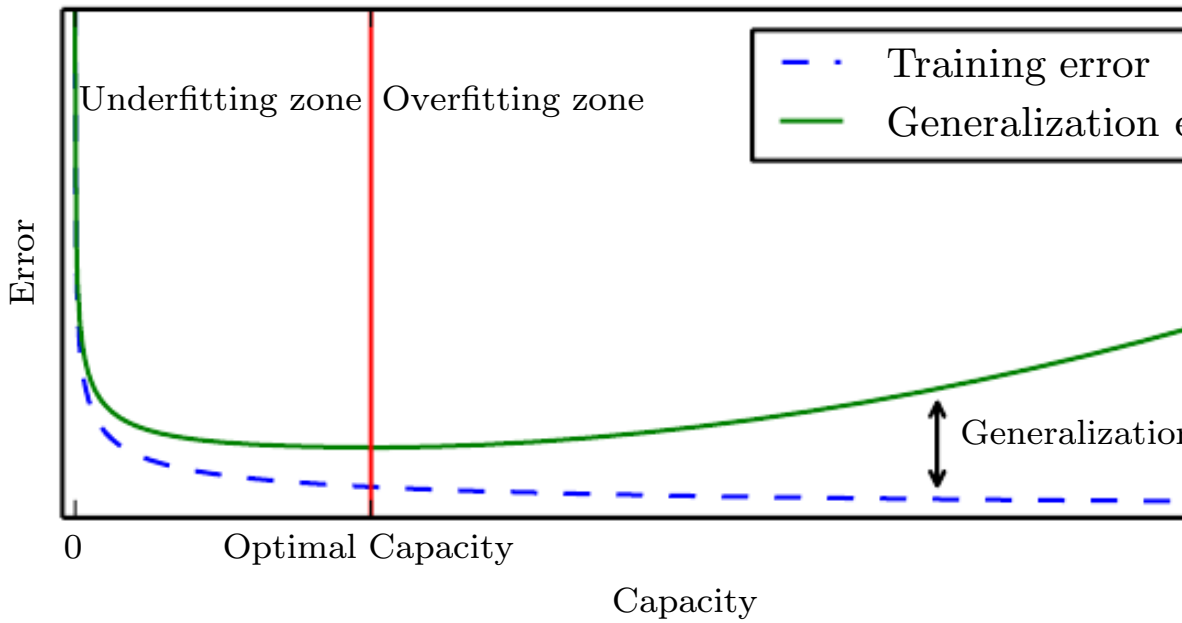


Figure 5.3: Typical relationship between capacity and error. Trainin
behave differently. At the left end of the graph, training error and gen
are both high. This is the **underfitting regime**. As we increase capaci
decreases, but the gap between training and generalization error incre
the size of this gap outweighs the decrease in training error, and we enter
regime, where capacity is too large, above the **optimal capacity**.

generalization error has a U-shaped curve as a function of model c
illustrated in figure 5.3.

To reach the most extreme case of arbitrarily high capacity
the concept of **nonparametric models**. So far, we have seen o
models, such as linear regression. Parametric models learn a fun
by a parameter vector whose size is finite and fixed before any d
Nonparametric models have no such limitation.

Sometimes, nonparametric models are just theoretical abstra
an algorithm that searches over all possible probability distributio
be implemented in practice. However, we can also design practical
models by making their complexity a function of the training set siz

of such an algorithm is **nearest neighbor regression**. Unlike li
which has a fixed-length vector of weights, the nearest neighbor re
simply stores the $\boldsymbol{X}$ and $\boldsymbol{y}$ from the training set. When asked to
point $\boldsymbol{x}$, the model looks up the nearest entry in the training set a
associated regression target. In other words, $\hat{y} = y_i$ where $i = \arg n$
The algorithm can also be generalized to distance metrics other tha
such as learned distance metrics (Goldberger *et al.*, 2005). If th
allowed to break ties by averaging the y_i values for all $\boldsymbol{X}_{i,:}$ that are
then this algorithm is able to achieve the minimum possible traini

might be greater than zero, if two identical inputs are associated outputs) on any regression dataset.

Finally, we can also create a nonparametric learning algorithm parametric learning algorithm inside another algorithm that increa of parameters as needed. For example, we could imagine an outer that changes the degree of the polynomial learned by linear regress polynomial expansion of the input.

The ideal model is an oracle that simply knows the true probabi that generates the data. Even such a model will still incur some problems, because there may still be some noise in the distributi of supervised learning, the mapping from $\boldsymbol{x}$ to y may be inhere or y may be a deterministic function that involves other variable included in $\boldsymbol{x}$. The error incurred by an oracle making prediction distribution $p(\boldsymbol{x}, y)$ is called the **Bayes error**.

Training and generalization error vary as the size of the trai Expected generalization error can never increase as the number of tr increases. For nonparametric models, more data yield better gene the best possible error is achieved. Any fixed parametric model optimal capacity will asymptote to an error value that exceeds t See figure 5.4 for an illustration. Note that it is possible for the optimal capacity and yet still have a large gap between training an errors. In this situation, we may be able to reduce this gap by g training examples.

5.2.1 The No Free Lunch Theorem

Learning theory claims that a machine learning algorithm can gene a finite training set of examples. This seems to contradict some ba logic. Inductive reasoning, or inferring general rules from a limited

is not logically valid. To logically infer a rule describing every m
one must have information about every member of that set.

In part, machine learning avoids this problem by offering only pr
rather than the entirely certain rules used in purely logical reasc
learning promises to find rules that are *probably* correct about *m*
the set they concern.

Unfortunately, even this does not resolve the entire problem
lunch theorem for machine learning (Wolpert, 1996) states that
all possible data-generating distributions, every classification alg

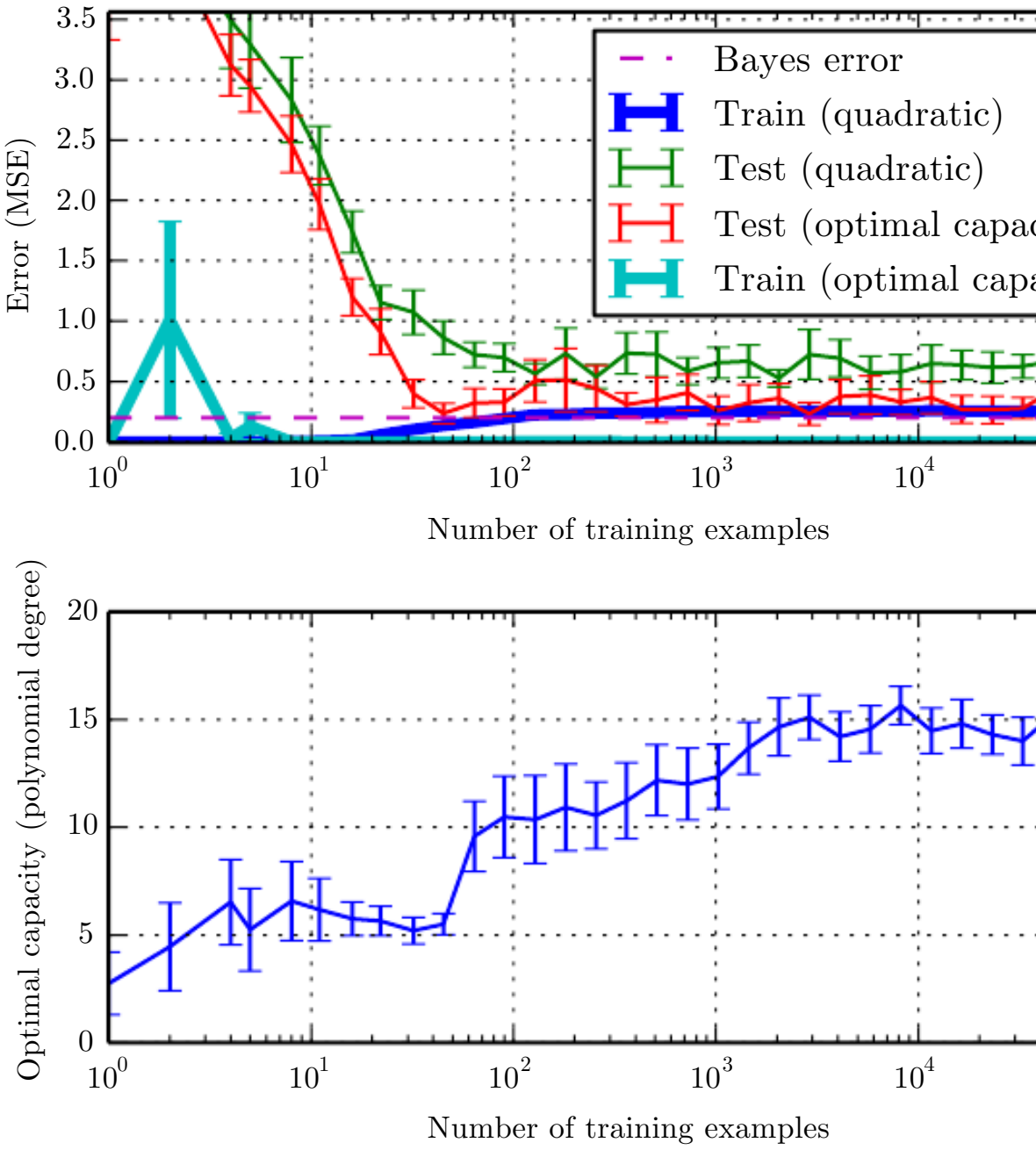


Figure 5.4: The effect of the training dataset size on the train and test
on the optimal model capacity. We constructed a synthetic regression p
adding a moderate amount of noise to a degree-5 polynomial, generated
and then generated several different sizes of training set. For each size,
different training sets in order to plot error bars showing 95 percent co
*(Top)*The MSE on the training and test set for two different models: a
and a model with degree chosen to minimize the test error. Both are fit in

the quadratic model, the training error increases as the size of the train

the quadratic model, the training error increases as the size of the train
This is because larger datasets are harder to fit. Simultaneously, the tes
because fewer incorrect hypotheses are consistent with the training data
model does not have enough capacity to solve the task, so its test erro
a high value. The test error at optimal capacity asymptotes to the B
training error can fall below the Bayes error, due to the ability of the tr
to memorize specific instances of the training set. As the training size inc
the training error of any fixed-capacity model (here, the quadratic model
least the Bayes error. *(Bottom)*As the training set size increases, the c
(shown here as the degree of the optimal polynomial regressor) increas
capacity plateaus after reaching sufficient complexity to solve the task.

same error rate when classifying previously unobserved points. I
in some sense, no machine learning algorithm is universally any b
other. The most sophisticated algorithm we can conceive of has th
performance (over all possible tasks) as merely predicting that ever
to the same class.

Fortunately, these results hold only when we average over *al*
generating distributions. If we make assumptions about the kind
distributions we encounter in real-world applications, then we can
algorithms that perform well on these distributions.

This means that the goal of machine learning research is not to
learning algorithm or the absolute best learning algorithm. Instead
understand what kinds of distributions are relevant to the "real wo
agent experiences, and what kinds of machine learning algorithms
data drawn from the kinds of data-generating distributions we can

5.2.2 Regularization

The no free lunch theorem implies that we must design our ma
algorithms to perform well on a specific task. We do so by bu
preferences into the learning algorithm. When these preferences a
the learning problems that we ask the algorithm to solve, it perfo

So far, the only method of modifying a learning algorithm that we
concretely is to increase or decrease the model's representational cap
or removing functions from the hypothesis space of solutions the lea
is able to choose from. We gave the specific example of increasin
the degree of a polynomial for a regression problem. The view we
so far is oversimplified.

The behavior of our algorithm is strongly affected not just b
make the set of functions allowed in its hypothesis space, but by the

of those functions. The learning algorithm we have studied so far, li
has a hypothesis space consisting of the set of linear functions of i
linear functions can be useful for problems where the relationship
and outputs truly is close to linear. They are less useful for problems
a very nonlinear fashion. For example, linear regression would not
we tried to use it to predict $\sin(x)$ from x. We can thus control the
our algorithms by choosing what kind of functions we allow them to
from, as well as by controlling the amount of these functions.

We can also give a learning algorithm a preference for one soluti

in its hypothesis space. This means that both functions are elig
preferred. The unpreferred solution will be chosen only if it fits th
significantly better than the preferred solution.

For example, we can modify the training criterion for linear regre
weight decay. To perform linear regression with weight decay, we
$J(\boldsymbol{w})$ comprising both the mean squared error on the training and
expresses a preference for the weights to have smaller squared L^2 no

$$J(\boldsymbol{w}) = \text{MSE}_{\text{train}} + \lambda \boldsymbol{w}^\top \boldsymbol{w},$$

where λ is a value chosen ahead of time that controls the strength o
for smaller weights. When $\lambda = 0$, we impose no preference, and lar
weights to become smaller. Minimizing $J(\boldsymbol{w})$ results in a choice
make a tradeoff between fitting the training data and being small
solutions that have a smaller slope, or that put weight on fewer
As an example of how we can control a model's tendency to ove
via weight decay, we can train a high-degree polynomial regressi
different values of λ. See figure 5.5 for the results.

More generally, we can regularize a model that learns a func
adding a penalty called a **regularizer** to the cost function. In the
decay, the regularizer is $\Omega(\boldsymbol{w}) = \boldsymbol{w}^\top \boldsymbol{w}$. In chapter 7, we will see t
regularizers are possible.

Expressing preferences for one function over another is a mo
of controlling a model's capacity than including or excluding mem
hypothesis space. We can think of excluding a function from a hyp
expressing an infinitely strong preference against that function.

In our weight decay example, we expressed our preference for
defined with smaller weights explicitly, via an extra term in th
minimize. There are many other ways of expressing preferenc
solutions, both implicitly and explicitly. Together, these differ

are known as **regularization**. *Regularization is any modificatio*
learning algorithm that is intended to reduce its generalization e
training error. Regularization is one of the central concerns of the f
learning, rivaled in its importance only by optimization.

The no free lunch theorem has made it clear that there is n
learning algorithm, and, in particular, no best form of regulariz
we must choose a form of regularization that is well suited to the
we want to solve. The philosophy of deep learning in general a
particular is that a wide range of tasks (such as all the intellec

people can do) may all be solved effectively using very general-pu
regularization.

5.3 Hyperparameters and Validation Sets

Most machine learning algorithms have hyperparameters, settin
use to control the algorithm's behavior. The values of hyperpara
adapted by the learning algorithm itself (though we can design a
procedure in which one learning algorithm learns the best hyper
another learning algorithm).

The polynomial regression example in figure 5.2 has a single h
the degree of the polynomial, which acts as a **capacity** hyperp
λ value used to control the strength of weight decay is anothe
hyperparameter.

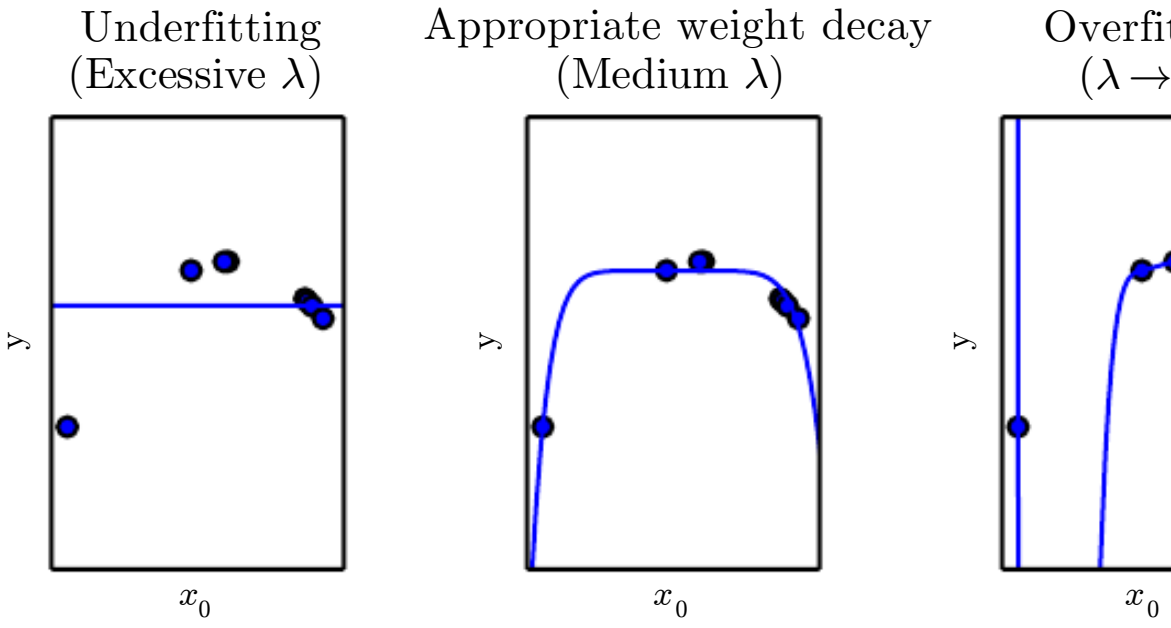


Figure 5.5: We fit a high-degree polynomial regression model to our exa
from figure 5.2. The true function is quadratic, but here we use only mod

We vary the amount of weight decay to prevent these high-degree models
*(Left)*With very large λ, we can force the model to learn a function
all. This underfits because it can only represent a constant function.
medium value of λ, the learning algorithm recovers a curve with the rig
Even though the model is capable of representing functions with much
shapes, weight decay has encouraged it to use a simpler function desc
coefficients. *(Right)*With weight decay approaching zero (i.e., using the
pseudoinverse to solve the underdetermined problem with minimal reg
degree-9 polynomial overfits significantly, as we saw in figure 5.2.

Sometimes a setting is chosen to be a hyperparameter that th
rithm does not learn because the setting is difficult to optimize. M
the setting must be a hyperparameter because it is not appropria
hyperparameter on the training set. This applies to all hyperp
control model capacity. If learned on the training set, such hyperpa
always choose the maximum possible model capacity, resulting in o
to figure 5.3). For example, we can always fit the training set
higher-degree polynomial and a weight decay setting of $\lambda = 0$ than
a lower-degree polynomial and a positive weight decay setting.

To solve this problem, we need a **validation set** of examples t
algorithm does not observe.

Earlier we discussed how a held-out test set, composed of examp
the same distribution as the training set, can be used to estimate th
error of a learner, after the learning process has completed. It is imp
test examples are not used in any way to make choices about the m
its hyperparameters. For this reason, no example from the test s
in the validation set. Therefore, we always construct the validati
training data. Specifically, we split the training data into two d
One of these subsets is used to learn the parameters. The othe
validation set, used to estimate the generalization error during or
allowing for the hyperparameters to be updated accordingly. The
used to learn the parameters is still typically called the training se
this may be confused with the larger pool of data used for the
process. The subset of data used to guide the selection of hype
called the validation set. Typically, one uses about 80 percent
data for training and 20 percent for validation. Since the valida
to "train" the hyperparameters, the validation set error will und
generalization error, though typically by a smaller amount than th
does. After all hyperparameter optimization is complete, the gene
may be estimated using the test set.

In practice, when the same test set has been used repeated
performance of different algorithms over many years, and especiall
all the attempts from the scientific community at beating the re
the-art performance on that test set, we end up having optimistic e
the test set as well. Benchmarks can thus become stale and then do
true field performance of a trained system. Thankfully, the comm
move on to new (and usually more ambitious and larger) benchma

5.3.1 Cross-Validation

Dividing the dataset into a fixed training set and a fixed test set can
if it results in the test set being small. A small test set implies statist
around the estimated average test error, making it difficult to claim
A works better than algorithm B on the given task.

When the dataset has hundreds of thousands of examples or m
a serious issue. When the dataset is too small, alternative proced
to use all the examples in the estimation of the mean test error,
increased computational cost. These procedures are based on the id
the training and testing computation on different randomly chosen s
of the original dataset. The most common of these is the k-fold c
procedure, shown in algorithm 5.1, in which a partition of the datas
splitting it into k nonoverlapping subsets. The test error may the
by taking the average test error across k trials. On trial i, the i-t
data is used as the test set, and the rest of the data is used as t
One problem is that no unbiased estimators of the variance of suc
estimators exist (Bengio and Grandvalet, 2004), but approximatio
used.

5.4 Estimators, Bias and Variance

The field of statistics gives us many tools to achieve the machine l
solving a task not only on the training set but also to generalize
concepts such as parameter estimation, bias and variance are use
characterize notions of generalization, underfitting and overfitting

5.4.1 Point Estimation

Point estimation is the attempt to provide the single "best" prec
quantity of interest. In general the quantity of interest can be a si
or a vector of parameters in some parametric model, such as the
linear regression example in section 5.1.4, but it can also be a wh

To distinguish estimates of parameters from their true value,
will be to denote a point estimate of a parameter $\boldsymbol{\theta}$ by $\hat{\boldsymbol{\theta}}$.

Let $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ be a set of m independent and identica

Algorithm 5.1 The k-fold cross-validation algorithm. It can be u
generalization error of a learning algorithm A when the given d
small for a simple train/test or train/valid split to yield accurat
generalization error, because the mean of a loss L on a small test se
high a variance. The dataset $\mathbb{D}$ contains as elements the abstract ex
the i-th example), which could stand for an (input,target) pair $\boldsymbol{z}$
in the case of supervised learning, or for just an input $\boldsymbol{z}^{(i)} = \boldsymbol{x}$
of unsupervised learning. The algorithm returns the vector of er
example in $\mathbb{D}$, whose mean is the estimated generalization error.
individual examples can be used to compute a confidence inter
mean (equation 5.47). Though these confidence intervals are no
after the use of cross-validation, it is still common practice to use t
that algorithm A is better than algorithm B only if the confidence
error of algorithm A lies below and does not intersect the confid
algorithm B.

Define `KFoldXV`$(\mathbb{D}, A, L, k)$:
Require: $\mathbb{D}$, the given dataset, with elements $\boldsymbol{z}^{(i)}$
Require: A, the learning algorithm, seen as a function that tak
input and outputs a learned function
Require: L, the loss function, seen as a function from a learned
an example $\boldsymbol{z}^{(i)} \in \mathbb{D}$ to a scalar $\in \mathbb{R}$
Require: k, the number of folds
Split $\mathbb{D}$ into k mutually exclusive subsets $\mathbb{D}_i$, whose union is $\mathbb{D}$
for i from 1 to k **do**
$f_i = A(\mathbb{D} \backslash \mathbb{D}_i)$
for $\boldsymbol{z}^{(j)}$ in $\mathbb{D}_i$ **do**
$e_j = L(f_i, \boldsymbol{z}^{(j)})$
end for
end for
Return $\boldsymbol{e}$

(i.i.d.) data points. A **point estimator** or **statistic** is any functi

$$\hat{\boldsymbol{\theta}}_m = g(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}).$$

The definition does not require that g return a value that is clos
or even that the range of g be the same as the set of allowable va
definition of a point estimator is very general and would enable th
estimator great flexibility. While almost any function thus qualifies

a good estimator is a function whose output is close to the true un
generated the training data.

For now, we take the frequentist perspective on statistics. Tha
that the true parameter value $\boldsymbol{\theta}$ is fixed but unknown, while the
$\hat{\boldsymbol{\theta}}$ is a function of the data. Since the data is drawn from a rando
function of the data is random. Therefore $\hat{\boldsymbol{\theta}}$ is a random variable.

Point estimation can also refer to the estimation of the relatio
input and target variables. We refer to these types of point estima
estimators.

Function Estimation Sometimes we are interested in perfo
estimation (or function approximation). Here, we are trying to pr
$\boldsymbol{y}$ given an input vector $\boldsymbol{x}$. We assume that there is a function $f(\boldsymbol{x}$
the approximate relationship between $\boldsymbol{y}$ and $\boldsymbol{x}$. For example, w
that $\boldsymbol{y} = f(\boldsymbol{x}) + \boldsymbol{\epsilon}$, where $\boldsymbol{\epsilon}$ stands for the part of $\boldsymbol{y}$ that is not p
$\boldsymbol{x}$. In function estimation, we are interested in approximating f w
estimate $\hat{f}$. Function estimation is really just the same as estimati
$\boldsymbol{\theta}$; the function estimator $\hat{f}$ is simply a point estimator in functi
linear regression example (discussed in section 5.1.4) and the polyno
example (discussed in section 5.2) both illustrate scenarios that may
as either estimating a parameter $\boldsymbol{w}$ or estimating a function $\hat{f}$ m
to y.

We now review the most commonly studied properties of point
discuss what they tell us about these estimators.

5.4.2 Bias

The bias of an estimator is defined as

$$\text{bias}(\hat{\boldsymbol{\theta}}_m) = \mathbb{E}(\hat{\boldsymbol{\theta}}_m) - \boldsymbol{\theta}$$

$$\text{bias}(\hat{\boldsymbol{\theta}}_m) = \mathbb{E}(\hat{\boldsymbol{\theta}}_m) - \boldsymbol{\theta},$$

where the expectation is over the data (seen as samples from a ra
and $\boldsymbol{\theta}$ is the true underlying value of $\boldsymbol{\theta}$ used to define the data-ge
bution. An estimator $\hat{\boldsymbol{\theta}}_m$ is said to be **unbiased** if $\text{bias}(\hat{\boldsymbol{\theta}}_m) = \mathbf{0}$
that $\mathbb{E}(\hat{\boldsymbol{\theta}}_m) = \boldsymbol{\theta}$. An estimator $\hat{\boldsymbol{\theta}}_m$ is said to be **asymptotical**
$\lim_{m\to\infty} \text{bias}(\hat{\boldsymbol{\theta}}_m) = \mathbf{0}$, which implies that $\lim_{m\to\infty} \mathbb{E}(\hat{\boldsymbol{\theta}}_m) = \boldsymbol{\theta}$.

Example: Bernoulli Distribution Consider a set of samples
that are independently and identically distributed according to a

bution with mean θ:

$$P(x^{(i)}; \theta) = \theta^{x^{(i)}}(1-\theta)^{(1-x^{(i)})}.$$

A common estimator for the θ parameter of this distribution is t
training samples:

$$\hat{\theta}_m = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}.$$

To determine whether this estimator is biased, we can substitut
into equation 5.20:

$$\begin{aligned}
\text{bias}(\hat{\theta}_m) &= \mathbb{E}[\hat{\theta}_m] - \theta \\
&= \mathbb{E}\left[\frac{1}{m}\sum_{i=1}^{m} x^{(i)}\right] - \theta \\
&= \frac{1}{m}\sum_{i=1}^{m} \mathbb{E}\left[x^{(i)}\right] - \theta \\
&= \frac{1}{m}\sum_{i=1}^{m} \sum_{x^{(i)}=0}^{1} \left(x^{(i)}\theta^{x^{(i)}}(1-\theta)^{(1-x^{(i)})}\right) - \theta \\
&= \frac{1}{m}\sum_{i=1}^{m} (\theta) - \theta \\
&= \theta - \theta = 0
\end{aligned}$$

Since $\text{bias}(\hat{\theta}) = 0$, we say that our estimator $\hat{\theta}$ is unbiased.

Example: Gaussian Distribution Estimator of the Mean

a set of samples $\{x^{(1)}, \ldots, x^{(m)}\}$ that are independently and identic
according to a Gaussian distribution $p(x^{(i)}) = \mathcal{N}(x^{(i)}; \mu, \sigma^2)$, where

Recall that the Gaussian probability density function is given by

$$p(x^{(i)}; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2} \frac{(x^{(i)} - \mu)^2}{\sigma^2} \right).$$

A common estimator of the Gaussian mean parameter is known **mean**:

$$\hat{\mu}_m = \frac{1}{m} \sum_{i=1}^{m} x^{(i)}$$

To determine the bias of the sample mean, we are again intereste
its expectation:

$$\begin{aligned}
\text{bias}(\hat{\mu}_m) &= \mathbb{E}[\hat{\mu}_m] - \mu \\
&= \mathbb{E}\left[\frac{1}{m}\sum_{i=1}^{m} x^{(i)}\right] - \mu \\
&= \left(\frac{1}{m}\sum_{i=1}^{m} \mathbb{E}\left[x^{(i)}\right]\right) - \mu \\
&= \left(\frac{1}{m}\sum_{i=1}^{m} \mu\right) - \mu \\
&= \mu - \mu = 0
\end{aligned}$$

Thus we find that the sample mean is an unbiased estimator of
parameter.

Example: Estimators of the Variance of a Gaussian Dist
this example, we compare two different estimators of the variance
a Gaussian distribution. We are interested in knowing if either esti

The first estimator of σ^2 we consider is known as the **sample**

$$\hat{\sigma}_m^2 = \frac{1}{m}\sum_{i=1}^{m} \left(x^{(i)} - \hat{\mu}_m\right)^2,$$

where $\hat{\mu}_m$ is the sample mean. More formally, we are interested in

$$\text{bias}(\hat{\sigma}_m^2) = \mathbb{E}[\hat{\sigma}_m^2] - \sigma^2.$$

We begin by evaluating the term $\mathbb{E}[\hat{\sigma}_m^2]$:

$$\begin{aligned}\mathbb{E}[\hat{\sigma}_m^2] =& \mathbb{E}\left[\frac{1}{m}\sum_{i=1}^{m}\left(x^{(i)} - \hat{\mu}_m\right)^2\right] \\ =& \frac{m-1}{m}\sigma^2\end{aligned}$$

Returning to equation 5.37, we conclude that the bias of $\hat{\sigma}_m^2$ is $-\sigma$ the sample variance is a biased estimator.

The **unbiased sample variance** estimator

$$\tilde{\sigma}_m^2 = \frac{1}{m-1}\sum_{i=1}^{m}\left(x^{(i)} - \hat{\mu}_m\right)^2$$

provides an alternative approach. As the name suggests this estima
That is, we find that $\mathbb{E}[\tilde{\sigma}_m^2] = \sigma^2$:

$$\begin{aligned}
\mathbb{E}[\tilde{\sigma}_m^2] &= \mathbb{E}\left[\frac{1}{m-1}\sum_{i=1}^{m}\left(x^{(i)} - \hat{\mu}_m\right)^2\right] \\
&= \frac{m}{m-1}\mathbb{E}[\hat{\sigma}_m^2] \\
&= \frac{m}{m-1}\left(\frac{m-1}{m}\sigma^2\right) \\
&= \sigma^2.
\end{aligned}$$

We have two estimators: one is biased, and the other is not. V
estimators are clearly desirable, they are not always the "best" est
will see we often use biased estimators that possess other importa

5.4.3 Variance and Standard Error

Another property of the estimator that we might want to consid
we expect it to vary as a function of the data sample. Just as we
expectation of the estimator to determine its bias, we can compu
The **variance** of an estimator is simply the variance

$$\mathrm{Var}(\hat{\theta})$$

where the random variable is the training set. Alternately, the squ

variance is called the **standard error**, denoted $\text{SE}(\theta)$.

The variance, or the standard error, of an estimator provides a
we would expect the estimate we compute from data to vary as w
resample the dataset from the underlying data-generating proce
might like an estimator to exhibit low bias, we would also like it to
low variance.

When we compute any statistic using a finite number of sample
of the true underlying parameter is uncertain, in the sense that
obtained other samples from the same distribution and their statis

been different. The expected degree of variation in any estimato
error that we want to quantify.

The standard error of the mean is given by

$$\mathrm{SE}(\hat{\mu}_m) = \sqrt{\mathrm{Var}\left[\frac{1}{m}\sum_{i=1}^{m} x^{(i)}\right]} = \frac{\sigma}{\sqrt{m}},$$

where σ^2 is the true variance of the samples x^i. The standard
estimated by using an estimate of σ. Unfortunately, neither the
the sample variance nor the square root of the unbiased estimator
provide an unbiased estimate of the standard deviation. Both a
to underestimate the true standard deviation but are still used i
square root of the unbiased estimator of the variance is less of an
For large m, the approximation is quite reasonable.

The standard error of the mean is very useful in machine learni
We often estimate the generalization error by computing the samp
error on the test set. The number of examples in the test set
accuracy of this estimate. Taking advantage of the central limit
tells us that the mean will be approximately distributed with a norm
we can use the standard error to compute the probability that the t
falls in any chosen interval. For example, the 95 percent confidence i
on the mean $\hat{\mu}_m$ is

$$(\hat{\mu}_m - 1.96\mathrm{SE}(\hat{\mu}_m), \hat{\mu}_m + 1.96\mathrm{SE}(\hat{\mu}_m)),$$

under the normal distribution with mean $\hat{\mu}_m$ and variance $\mathrm{SE}(\hat{\mu}_m$
learning experiments, it is common to say that algorithm A is better
B if the upper bound of the 95 percent confidence interval for the er
A is less than the lower bound of the 95 percent confidence interv

of algorithm B

of algorithm B.

Example: Bernoulli Distribution We once again consider a
$\{x^{(1)}, \ldots, x^{(m)}\}$ drawn independently and identically from a Berno
(recall $P(x^{(i)}; \theta) = \theta^{x^{(i)}}(1-\theta)^{(1-x^{(i)})}$). This time we are intereste
the variance of the estimator $\hat{\theta}_m = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$.

$$\text{Var}\left(\hat{\theta}_m\right) = \text{Var}\left(\frac{1}{m}\sum_{i=1}^{m} x^{(i)}\right)$$

$$
\begin{aligned}
&= \frac{1}{m^2} \sum_{i=1}^{m} \text{Var}\left(x^{(i)}\right) \\
&= \frac{1}{m^2} \sum_{i=1}^{m} \theta(1-\theta) \\
&= \frac{1}{m^2} m\theta(1-\theta) \\
&= \frac{1}{m} \theta(1-\theta)
\end{aligned}
$$

The variance of the estimator decreases as a function of m, the num
in the dataset. This is a common property of popular estimato
return to when we discuss consistency (see section 5.4.5).

5.4.4 Trading off Bias and Variance to Minimize M Error

Bias and variance measure two different sources of error in an e
measures the expected deviation from the true value of the functio
Variance on the other hand, provides a measure of the deviation fro
estimator value that any particular sampling of the data is likely

What happens when we are given a choice between two estim
more bias and one with more variance? How do we choose betw
example, imagine that we are interested in approximating the fu
figure 5.2 and we are only offered the choice between a model with
one that suffers from large variance. How do we choose between t

The most common way to negotiate this trade-off is to use c
Empirically, cross-validation is highly successful on many real-wor
natively, we can also compare the **mean squared error** (MSE) o

$$
\text{MSE} = \mathbb{E}[(\hat{\theta}_m - \theta)^2]
$$

$$= \text{Bias}(\hat{\theta}_m)^2 + \text{Var}(\hat{\theta}_m)$$

The MSE measures the overall expected deviation—in a square between the estimator and the true value of the parameter θ. A equation 5.54, evaluating the MSE incorporates both the bias an Desirable estimators are those with small MSE and these are e manage to keep both their bias and variance somewhat in check.

The relationship between bias and variance is tightly linked learning concepts of capacity, underfitting and overfitting. When

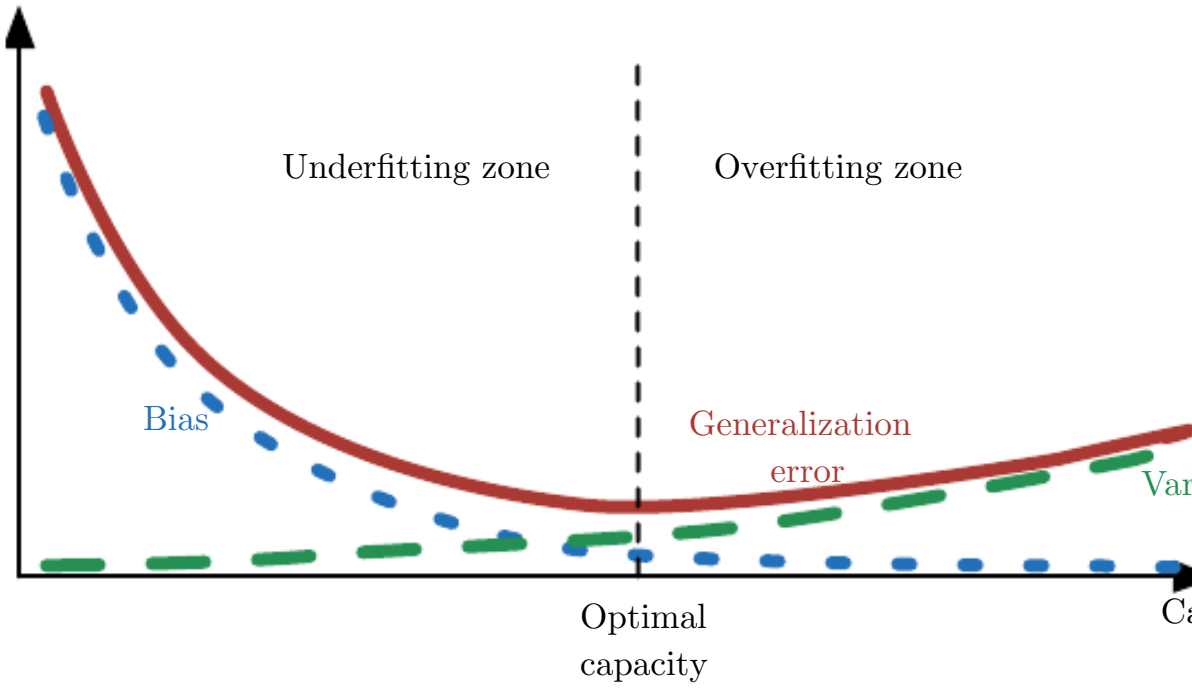


Figure 5.6: As capacity increases (x-axis), bias (dotted) tends to decre
(dashed) tends to increase, yielding another U-shaped curve for generaliz
curve). If we vary capacity along one axis, there is an optimal capacity,
when the capacity is below this optimum and overfitting when it is above.
is similar to the relationship between capacity, underfitting, and overfitt
section 5.2 and figure 5.3.

error is measured by the MSE (where bias and variance are meaning
of generalization error), increasing capacity tends to increase varian
bias. This is illustrated in figure 5.6, where we see again the U-s
generalization error as a function of capacity.

5.4.5 Consistency

So far we have discussed the properties of various estimators for a
fixed size. Usually, we are also concerned with the behavior of an e
amount of training data grows. In particular, we usually wish that

of data points m in our dataset increases, our point estimates conv
value of the corresponding parameters. More formally, we would l

$$\text{plim}_{m\to\infty}\hat{\theta}_m = \theta.$$

The symbol plim indicates convergence in probability, meaning tha
$P(|\hat{\theta}_m - \theta| > \epsilon) \to 0$ as $m \to \infty$. The condition described by e
known as **consistency**. It is sometimes referred to as weak co
strong consistency referring to the **almost sure** convergence of $\hat{\theta}$

sure convergence of a sequence of random variables $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, .$ occurs when $p(\lim_{m\to\infty} \mathbf{x}^{(m)} = \boldsymbol{x}) = 1$.

Consistency ensures that the bias induced by the estimator di number of data examples grows. However, the reverse is not tru unbiasedness does not imply consistency. For example, consider mean parameter μ of a normal distribution $\mathcal{N}(x; \mu, \sigma^2)$, with a da of m samples: $\{x^{(1)}, \ldots, x^{(m)}\}$. We could use the first sample $x^{(1}$ as an unbiased estimator: $\hat{\theta} = x^{(1)}$. In that case, $\mathbb{E}(\hat{\theta}_m) = \theta$, s is unbiased no matter how many data points are seen. This, of that the estimate is asymptotically unbiased. However, this is n estimator as it is *not* the case that $\hat{\theta}_m \to \theta$ as $m \to \infty$.

5.5 Maximum Likelihood Estimation

We have seen some definitions of common estimators and analyzed But where did these estimators come from? Rather than gues function might make a good estimator and then analyzing its bia we would like to have some principle from which we can derive sp that are good estimators for different models.

The most common such principle is the maximum likelihood p

Consider a set of m examples $\mathbb{X} = \{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ drawn inde the true but unknown data-generating distribution $p_{\text{data}}(\mathbf{x})$.

Let $p_{\text{model}}(\mathbf{x}; \boldsymbol{\theta})$ be a parametric family of probability distrib same space indexed by $\boldsymbol{\theta}$. In other words, $p_{\text{model}}(\boldsymbol{x}; \boldsymbol{\theta})$ maps any to a real number estimating the true probability $p_{\text{data}}(\boldsymbol{x})$.

The maximum likelihood estimator for $\boldsymbol{\theta}$ is then defined as

$$\boldsymbol{\theta}_{\text{ML}} = \arg\max_{\boldsymbol{\theta}} p_{\text{model}}(\mathbb{X}; \boldsymbol{\theta}),$$

$$= \arg\max_{\boldsymbol{\theta}} \prod_{i=1}^{m} p^{\text{model}}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}).$$

This product over many probabilities can be inconvenient for
For example, it is prone to numerical underflow. To obtain a m
but equivalent optimization problem, we observe that taking the l
likelihood does not change its arg max but does conveniently trans

into a sum:

$$\boldsymbol{\theta}_{\text{ML}} = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{m} \log p_{\text{model}}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}).$$

Because the arg max does not change when we rescale the cost fu
divide by m to obtain a version of the criterion that is expressed as
with respect to the empirical distribution $\hat{p}_{\text{data}}$ defined by the trai

$$\boldsymbol{\theta}_{\text{ML}} = \arg\max_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{x} \sim \hat{p}_{\text{data}}} \log p_{\text{model}}(\boldsymbol{x}; \boldsymbol{\theta}).$$

One way to interpret maximum likelihood estimation is to view i
the dissimilarity between the empirical distribution $\hat{p}_{\text{data}}$, defined
set and the model distribution, with the degree of dissimilarity b
measured by the KL divergence. The KL divergence is given by

$$D_{\text{KL}}\left(\hat{p}_{\text{data}} \| p_{\text{model}}\right) = \mathbb{E}_{\mathbf{x} \sim \hat{p}_{\text{data}}} \left[\log \hat{p}_{\text{data}}(\boldsymbol{x}) - \log p_{\text{model}}(\boldsymbol{x})\right]$$

The term on the left is a function only of the data-generating p
model. This means when we train the model to minimize the KL
need only minimize

$$-\mathbb{E}_{\mathbf{x} \sim \hat{p}_{\text{data}}} \left[\log p_{\text{model}}(\boldsymbol{x})\right],$$

which is of course the same as the maximization in equation 5.59.

Minimizing this KL divergence corresponds exactly to minim
entropy between the distributions. Many authors use the term "cr
identify specifically the negative log-likelihood of a Bernoulli or softm
but that is a misnomer. Any loss consisting of a negative log-likeli
entropy between the empirical distribution defined by the traini
probability distribution defined by model. For example, mean squa
cross-entropy between the empirical distribution and a Gaussian m

We can thus see maximum likelihood as an attempt to make
tribution match the empirical distribution $\hat{p}_{\text{data}}$. Ideally, we woul
the true data-generating distribution p_{data}, but we have no direc

distribution.

While the optimal $\boldsymbol{\theta}$ is the same regardless of whether we are
likelihood or minimizing the KL divergence, the values of the obj
are different. In software, we often phrase both as minimizing
Maximum likelihood thus becomes minimization of the negativ
(NLL), or equivalently, minimization of the cross-entropy. The
maximum likelihood as minimum KL divergence becomes helpf
because the KL divergence has a known minimum value of zero
log-likelihood can actually become negative when $\boldsymbol{x}$ is real-valued

5.5.1 Conditional Log-Likelihood and Mean Squared

The maximum likelihood estimator can readily be generalized to es
tional probability $P(\mathbf{y} \mid \mathbf{x}; \boldsymbol{\theta})$ in order to predict $\mathbf{y}$ given $\mathbf{x}$. This
most common situation because it forms the basis for most supervi
$\boldsymbol{X}$ represents all our inputs and $\boldsymbol{Y}$ all our observed targets, then
maximum likelihood estimator is

$$\boldsymbol{\theta}_{\text{ML}} = \arg\max_{\boldsymbol{\theta}} P(\boldsymbol{Y} \mid \boldsymbol{X}; \boldsymbol{\theta}).$$

If the examples are assumed to be i.i.d., then this can be decompo

$$\boldsymbol{\theta}_{\text{ML}} = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{m} \log P(\boldsymbol{y}^{(i)} \mid \boldsymbol{x}^{(i)}; \boldsymbol{\theta}).$$

Example: Linear Regression as Maximum Likelihood Li
introduced in section 5.1.4, may be justified as a maximum likelih
Previously, we motivated linear regression as an algorithm that le
input $\boldsymbol{x}$ and produce an output value $\hat{y}$. The mapping from $\boldsymbol{x}$ to
minimize mean squared error, a criterion that we introduced more or
We now revisit linear regression from the point of view of maxim
estimation. Instead of producing a single prediction $\hat{y}$, we now thi
as producing a conditional distribution $p(y \mid \boldsymbol{x})$. We can imagin
infinitely large training set, we might see several training examples
input value $\boldsymbol{x}$ but different values of y. The goal of the learning al
to fit the distribution $p(y \mid \boldsymbol{x})$ to all those different y values that ar
with $\boldsymbol{x}$. To derive the same linear regression algorithm we obtai
define $p(y \mid \boldsymbol{x}) = \mathcal{N}(y; \hat{y}(\boldsymbol{x}; \boldsymbol{w}), \sigma^2)$. The function $\hat{y}(\boldsymbol{x}; \boldsymbol{w})$ gives th
the mean of the Gaussian. In this example, we assume that the var
some constant σ^2 chosen by the user. We will see that this choice o

form of $p(y \mid \boldsymbol{x})$ causes the maximum likelihood estimation proced
same learning algorithm as we developed before. Since the exampl
to be i.i.d., the conditional log-likelihood (equation 5.63) is given

$$\sum_{i=1}^{m} \log p(y^{(i)} \mid \boldsymbol{x}^{(i)}; \boldsymbol{\theta})$$

$$= -\, m \log \sigma - \frac{m}{2} \log(2\pi) - \sum_{i=1}^{m} \frac{\left\| \hat{y}^{(i)} - y^{(i)} \right\|^2}{2\sigma^2},$$

where $\hat{y}^{(i)}$ is the output of the linear regression on the i-th input $\boldsymbol{x}$ number of the training examples. Comparing the log-likelihood squared error,

$$\text{MSE}_{\text{train}} = \frac{1}{m}\sum_{i=1}^{m} ||\hat{y}^{(i)} - y^{(i)}||^2,$$

we immediately see that maximizing the log-likelihood with resp the same estimate of the parameters $\boldsymbol{w}$ as does minimizing the mea The two criteria have different values but the same location of the justifies the use of the MSE as a maximum likelihood estimation pr will see, the maximum likelihood estimator has several desirable p

5.5.2 Properties of Maximum Likelihood

The main appeal of the maximum likelihood estimator is that it c be the best estimator asymptotically, as the number of examples m of its rate of convergence as m increases.

Under appropriate conditions, the maximum likelihood esti property of consistency (see section 5.4.5), meaning that as the nur examples approaches infinity, the maximum likelihood estimate converges to the true value of the parameter. These conditions ar

- The true distribution p_{data} must lie within the model fam Otherwise, no estimator can recover p_{data}.
- The true distribution p_{data} must correspond to exactly one v erwise, maximum likelihood can recover the correct p_{data} but to determine which value of $\boldsymbol{\theta}$ was used by the data-generati

There are other inductive principles besides the maximum likeli many of which share the property of being consistent estimato

estimators can differ, however, in their **statistical efficiency**, me
consistent estimator may obtain lower generalization error for a f
samples m, or equivalently, may require fewer examples to obtain
generalization error.

Statistical efficiency is typically studied in the **parametric ca**
regression), where our goal is to estimate the value of a paramet
is possible to identify the true parameter), not the value of a func
measure how close we are to the true parameter is by the expected
error, computing the squared difference between the estimated and

values, where the expectation is over m training samples from the
distribution. That parametric mean squared error decreases as m
for m large, the Cramér-Rao lower bound (Rao, 1945; Cramér, 19
no consistent estimator has a lower MSE than the maximum likeli

For these reasons (consistency and efficiency), maximum like
considered the preferred estimator to use for machine learning. Wh
of examples is small enough to yield overfitting behavior, regulariz
such as weight decay may be used to obtain a biased version of maxi
that has less variance when training data is limited.

5.6 Bayesian Statistics

So far we have discussed **frequentist statistics** and approaches ba
ing a single value of $\boldsymbol{\theta}$, then making all predictions thereafter bas
estimate. Another approach is to consider all possible values of $\boldsymbol{\theta}$
prediction. The latter is the domain of **Bayesian statistics**.

As discussed in section 5.4.1, the frequentist perspective is
parameter value $\boldsymbol{\theta}$ is fixed but unknown, while the point estimate
variable on account of it being a function of the dataset (which is s

The Bayesian perspective on statistics is quite different. The
probability to reflect degrees of certainty in states of knowledge.
directly observed and so is not random. On the other hand, the tr
is unknown or uncertain and thus is represented as a random vari

Before observing the data, we represent our knowledge of $\boldsymbol{\theta}$ u
probability distribution, $p(\boldsymbol{\theta})$ (sometimes referred to as simp
Generally, the machine learning practitioner selects a prior distr
quite broad (i.e., with high entropy) to reflect a high degree of un
value of $\boldsymbol{\theta}$ before observing any data. For example, one might assu

$\boldsymbol{\theta}$ lies in some finite range or volume, with a uniform distributio
instead reflect a preference for "simpler" solutions (such as sma
coefficients, or a function that is closer to being constant).

Now consider that we have a set of data samples $\{x^{(1)}, \ldots,$
recover the effect of data on our belief about $\boldsymbol{\theta}$ by combining the
$p(x^{(1)}, \ldots, x^{(m)} \mid \boldsymbol{\theta})$ with the prior via Bayes' rule:

$$p(\boldsymbol{\theta} \mid x^{(1)}, \ldots, x^{(m)}) = \frac{p(x^{(1)}, \ldots, x^{(m)} \mid \boldsymbol{\theta}) p(\boldsymbol{\theta})}{p(x^{(1)}, \ldots, x^{(m)})}$$

In the scenarios where Bayesian estimation is typically used, the p
relatively uniform or Gaussian distribution with high entropy, and
of the data usually causes the posterior to lose entropy and concer
few highly likely values of the parameters.

Relative to maximum likelihood estimation, Bayesian estima
important differences. First, unlike the maximum likelihood appro
predictions using a point estimate of $\boldsymbol{\theta}$, the Bayesian approach is to m
using a full distribution over $\boldsymbol{\theta}$. For example, after observing m
predicted distribution over the next data sample, $x^{(m+1)}$, is given

$$p(x^{(m+1)} \mid x^{(1)}, \ldots, x^{(m)}) = \int p(x^{(m+1)} \mid \boldsymbol{\theta}) p(\boldsymbol{\theta} \mid x^{(1)}, \ldots, x^{(m)}$$

Here each value of $\boldsymbol{\theta}$ with positive probability density contributes t
of the next example, with the contribution weighted by the posteri
After having observed $\{x^{(1)}, \ldots, x^{(m)}\}$, if we are still quite uncer
value of $\boldsymbol{\theta}$, then this uncertainty is incorporated directly into any
might make.

In section 5.4, we discussed how the frequentist approach addr
tainty in a given point estimate of $\boldsymbol{\theta}$ by evaluating its variance. T
the estimator is an assessment of how the estimate might change v
samplings of the observed data. The Bayesian answer to the questio
with the uncertainty in the estimator is to simply integrate over it,
protect well against overfitting. This integral is of course just an
the laws of probability, making the Bayesian approach simple to ju
frequentist machinery for constructing an estimator is based on th
decision to summarize all knowledge contained in the dataset with
estimate.

The second important difference between the Bayesian approac
and the maximum likelihood approach is due to the contribution

prior distribution. The prior has an influence by shifting probabili
towards regions of the parameter space that are preferred a pric
the prior often expresses a preference for models that are simpler c
Critics of the Bayesian approach identify the prior as a source of su
judgment affecting the predictions.

Bayesian methods typically generalize much better when limite
is available but typically suffer from high computational cost when
training examples is large.

Example: Bayesian Linear Regression Here we consider the
mation approach to learning the linear regression parameters. In li
we learn a linear mapping from an input vector $\boldsymbol{x} \in \mathbb{R}^n$ to predic
scalar $y \in \mathbb{R}$. The prediction is parametrized by the vector $\boldsymbol{w} \in \mathbb{R}$

$$\hat{y} = \boldsymbol{w}^\top \boldsymbol{x}.$$

Given a set of m training samples $(\boldsymbol{X}^{(\text{train})}, \boldsymbol{y}^{(\text{train})})$, we can expres
of y over the entire training set as

$$\hat{\boldsymbol{y}}^{(\text{train})} = \boldsymbol{X}^{(\text{train})}\boldsymbol{w}.$$

Expressed as a Gaussian conditional distribution on $\boldsymbol{y}^{(\text{train})}$, w

$$p(\boldsymbol{y}^{(\text{train})} \mid \boldsymbol{X}^{(\text{train})}, \boldsymbol{w}) = \mathcal{N}(\boldsymbol{y}^{(\text{train})}; \boldsymbol{X}^{(\text{train})}\boldsymbol{w}, \boldsymbol{I})$$

$$\propto \exp\left(-\frac{1}{2}(\boldsymbol{y}^{(\text{train})} - \boldsymbol{X}^{(\text{train})}\boldsymbol{w})^\top (\boldsymbol{y}^{(\text{train})} -$$

where we follow the standard MSE formulation in assuming tha
variance on y is one. In what follows, to reduce the notational bur
$(\boldsymbol{X}^{(\text{train})}, \boldsymbol{y}^{(\text{train})})$ as simply $(\boldsymbol{X}, \boldsymbol{y})$.

To determine the posterior distribution over the model paramet
first need to specify a prior distribution. The prior should reflect
about the value of these parameters. While it is sometimes difficu
to express our prior beliefs in terms of the parameters of the model
typically assume a fairly broad distribution, expressing a high degre
about $\boldsymbol{\theta}$. For real-valued parameters it is common to use a Gaus
distribution,

$$p(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}; \boldsymbol{\mu}_0, \boldsymbol{\Lambda}_0) \propto \exp\left(-\frac{1}{2}(\boldsymbol{w} - \boldsymbol{\mu}_0)^\top \boldsymbol{\Lambda}_0^{-1}(\boldsymbol{w} - \boldsymbol{\mu}_0) \right)$$

where $\boldsymbol{\mu}^0$ and $\boldsymbol{\Lambda}^0$ are the prior distribution mean vector and cov
respectively.[1]

With the prior thus specified, we can now proceed in determining
distribution over the model parameters:

$$p(\boldsymbol{w} \mid \boldsymbol{X}, \boldsymbol{y}) \propto p(\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{w}) p(\boldsymbol{w})$$

[1]Unless there is a reason to use a particular covariance structure, we t
diagonal covariance matrix $\boldsymbol{\Lambda}_0 = \text{diag}(\boldsymbol{\lambda}_0)$.

$$\propto \exp\left(-\frac{1}{2}(\boldsymbol{y}-\boldsymbol{X}\boldsymbol{w})^\top(\boldsymbol{y}-\boldsymbol{X}\boldsymbol{w})\right)\exp\left(-\frac{1}{2}(\boldsymbol{w}-\boldsymbol{\mu}_0)\right.$$

$$\propto \exp\left(-\frac{1}{2}\left(-2\boldsymbol{y}^\top\boldsymbol{X}\boldsymbol{w}+\boldsymbol{w}^\top\boldsymbol{X}^\top\boldsymbol{X}\boldsymbol{w}+\boldsymbol{w}^\top\boldsymbol{\Lambda}_0^{-1}\boldsymbol{w}-\right.\right.$$

We now define $\boldsymbol{\Lambda}_m = \left(\boldsymbol{X}^\top\boldsymbol{X}+\boldsymbol{\Lambda}_0^{-1}\right)^{-1}$ and $\boldsymbol{\mu}_m = \boldsymbol{\Lambda}_m\left(\boldsymbol{X}^\top\boldsymbol{y}\right.$ ing these new variables, we find that the posterior may be rewritte distribution:

$$p(\boldsymbol{w}\mid\boldsymbol{X},\boldsymbol{y}) \propto \exp\left(-\frac{1}{2}(\boldsymbol{w}-\boldsymbol{\mu}_m)^\top\boldsymbol{\Lambda}_m^{-1}(\boldsymbol{w}-\boldsymbol{\mu}_m)+\frac{1}{2}\boldsymbol{\mu}_m^\top\boldsymbol{\Lambda}_m^{-1}\right.$$
$$\propto \exp\left(-\frac{1}{2}(\boldsymbol{w}-\boldsymbol{\mu}_m)^\top\boldsymbol{\Lambda}_m^{-1}(\boldsymbol{w}-\boldsymbol{\mu}_m)\right).$$

All terms that do not include the parameter vector $\boldsymbol{w}$ have beer are implied by the fact that the distribution must be normalized t Equation 3.23 shows how to normalize a multivariate Gaussian di

Examining this posterior distribution enables us to gain some i effect of Bayesian inference. In most situations, we set $\boldsymbol{\mu}_0$ to $\mathbf{0}$. If v then μ_m gives the same estimate of $\boldsymbol{w}$ as does frequentist linear re weight decay penalty of $\alpha\boldsymbol{w}^\top\boldsymbol{w}$. One difference is that the Bayes undefined if α is set to zero—we are not allowed to begin the Ba process with an infinitely wide prior on $\boldsymbol{w}$. The more important d the Bayesian estimate provides a covariance matrix, showing ho different values of $\boldsymbol{w}$ are, rather than providing only the estimate

5.6.1 Maximum a Posteriori (MAP) Estimation

While the most principled approach is to make predictions using th
posterior distribution over the parameter $\boldsymbol{\theta}$, it is still often desi
single point estimate. One common reason for desiring a point e
most operations involving the Bayesian posterior for most interes
intractable, and a point estimate offers a tractable approximatio
simply returning to the maximum likelihood estimate, we can sti
the benefit of the Bayesian approach by allowing the prior to influ
of the point estimate. One rational way to do this is to choose
a posteriori (MAP) point estimate. The MAP estimate choos

maximal posterior probability (or maximal probability density in th
case of continuous $\boldsymbol{\theta}$):

$$\boldsymbol{\theta}_{\text{MAP}} = \arg\max_{\boldsymbol{\theta}} p(\boldsymbol{\theta} \mid \boldsymbol{x}) = \arg\max_{\boldsymbol{\theta}} \log p(\boldsymbol{x} \mid \boldsymbol{\theta}) + \log p(\boldsymbol{\theta})$$

We recognize, on the righthand side, $\log p(\boldsymbol{x} \mid \boldsymbol{\theta})$, that is, th
likelihood term, and $\log p(\boldsymbol{\theta})$, corresponding to the prior distribut

As an example, consider a linear regression model with a Ga
the weights $\boldsymbol{w}$. If this prior is given by $\mathcal{N}(\boldsymbol{w}; \mathbf{0}, \frac{1}{\lambda}\boldsymbol{I}^2)$, then the lo
equation 5.79 is proportional to the familiar $\lambda \boldsymbol{w}^\top \boldsymbol{w}$ weight decay
term that does not depend on $\boldsymbol{w}$ and does not affect the learning
Bayesian inference with a Gaussian prior on the weights thus corres
decay.

As with full Bayesian inference, MAP Bayesian inference has t
leveraging information that is brought by the prior and cannot b
training data. This additional information helps to reduce the
MAP point estimate (in comparison to the ML estimate). Howeve
the price of increased bias.

Many regularized estimation strategies, such as maximum like
regularized with weight decay, can be interpreted as making the M
tion to Bayesian inference. This view applies when the regulariza
adding an extra term to the objective function that corresponds t
all regularization penalties correspond to MAP Bayesian inferenc
some regularizer terms may not be the logarithm of a probabili
Other regularization terms depend on the data, which of course a p
distribution is not allowed to do.

MAP Bayesian inference provides a straightforward way to desi
yet interpretable regularization terms. For example, a more comp
term can be derived by using a mixture of Gaussians, rather than a

distribution, as the prior (Nowlan and Hinton, 1992).

5.7 Supervised Learning Algorithms

Recall from section 5.1.3 that supervised learning algorithms are, ro
learning algorithms that learn to associate some input with some
training set of examples of inputs $\boldsymbol{x}$ and outputs $\boldsymbol{y}$. In many cas
$\boldsymbol{y}$ may be difficult to collect automatically and must be provide

"supervisor," but the term still applies even when the training s
collected automatically.

5.7.1 Probabilistic Supervised Learning

Most supervised learning algorithms in this book are based o
probability distribution $p(y \mid \boldsymbol{x})$. We can do this simply by u
likelihood estimation to find the best parameter vector $\boldsymbol{\theta}$ for a pa
of distributions $p(y \mid \boldsymbol{x}; \boldsymbol{\theta})$.

We have already seen that linear regression corresponds to the

$$p(y \mid \boldsymbol{x}; \boldsymbol{\theta}) = \mathcal{N}(y; \boldsymbol{\theta}^\top \boldsymbol{x}, \boldsymbol{I}).$$

We can generalize linear regression to the classification scenario
different family of probability distributions. If we have two class
class 1, then we need only specify the probability of one of thes
probability of class 1 determines the probability of class 0, because t
must add up to 1.

The normal distribution over real-valued numbers that we
regression is parametrized in terms of a mean. Any value we suppl
is valid. A distribution over a binary variable is slightly more compl
its mean must always be between 0 and 1. One way to solve this pr
the logistic sigmoid function to squash the output of the linear fu
interval (0, 1) and interpret that value as a probability:

$$p(y = 1 \mid \boldsymbol{x}; \boldsymbol{\theta}) = \sigma(\boldsymbol{\theta}^\top \boldsymbol{x}).$$

This approach is known as **logistic regression** (a somewhat stra
we use the model for classification rather than regression).

In the case of linear regression, we were able to find the opti

solving the normal equations. Logistic regression is somewhat more
is no closed-form solution for its optimal weights. Instead, we n
them by maximizing the log-likelihood. We can do this by minimizi
log-likelihood using gradient descent.

This same strategy can be applied to essentially any supervised le
by writing down a parametric family of conditional probability dis
the right kind of input and output variables.

5.7.2 Support Vector Machines

One of the most influential approaches to supervised learning is the machine (Boser *et al.*, 1992; Cortes and Vapnik, 1995). This mod logistic regression in that it is driven by a linear function $\boldsymbol{w}^\top \boldsymbol{x} + b$. regression, the support vector machine does not provide probabi outputs a class identity. The SVM predicts that the positive class $\boldsymbol{w}^\top \boldsymbol{x} + b$ is positive. Likewise, it predicts that the negative class i $\boldsymbol{w}^\top \boldsymbol{x} + b$ is negative.

One key innovation associated with support vector machines **trick**. The kernel trick consists of observing that many machine lear can be written exclusively in terms of dot products between example it can be shown that the linear function used by the support vect be re-written as

$$\boldsymbol{w}^\top \boldsymbol{x} + b = b + \sum_{i=1}^{m} \alpha_i \boldsymbol{x}^\top \boldsymbol{x}^{(i)},$$

where $\boldsymbol{x}^{(i)}$ is a training example, and $\boldsymbol{\alpha}$ is a vector of coefficients. learning algorithm this way enables us to replace $\boldsymbol{x}$ with the output o function $\phi(\boldsymbol{x})$ and the dot product with a function $k(\boldsymbol{x}, \boldsymbol{x}^{(i)}) = \phi(\boldsymbol{x}$ a **kernel**. The $\cdot$ operator represents an inner product analogous t For some feature spaces, we may not use literally the vector inn some infinite dimensional spaces, we need to use other kinds of inn example, inner products based on integration rather than summati development of these kinds of inner products is beyond the scope

After replacing dot products with kernel evaluations, we can m using the function

$$f(\boldsymbol{x}) = b + \sum_{i}^{} \alpha_i k(\boldsymbol{x}, \boldsymbol{x}^{(i)}).$$

This function is nonlinear with respect to $\boldsymbol{x}$, but the relationshi

and $f(\boldsymbol{x})$ is linear. Also, the relationship between $\boldsymbol{\alpha}$ and $f(\boldsymbol{x})$
kernel-based function is exactly equivalent to preprocessing the da
$\phi(\boldsymbol{x})$ to all inputs, then learning a linear model in the new transfo

The kernel trick is powerful for two reasons. First, it enables us
that are nonlinear as a function of $\boldsymbol{x}$ using convex optimization t
are guaranteed to converge efficiently. This is possible because we c
and optimize only $\boldsymbol{\alpha}$, that is, the optimization algorithm can vie
function as being linear in a different space. Second, the kernel fu

admits an implementation that is significantly more computational naively constructing two $\phi(\boldsymbol{x})$ vectors and explicitly taking their

In some cases, $\phi(\boldsymbol{x})$ can even be infinite dimensional, which an infinite computational cost for the naive, explicit approach. $k(\boldsymbol{x}, \boldsymbol{x}')$ is a nonlinear, tractable function of $\boldsymbol{x}$ even when $\phi(\boldsymbol{x})$ is an example of an infinite-dimensional feature space with a tract construct a feature mapping $\phi(x)$ over the nonnegative integers x this mapping returns a vector containing x ones followed by infinit We can write a kernel function $k(x, x^{(i)}) = \min(x, x^{(i)})$ that is exa to the corresponding infinite-dimensional dot product.

The most commonly used kernel is the **Gaussian kernel**,

$$k(\boldsymbol{u}, \boldsymbol{v}) = \mathcal{N}(\boldsymbol{u} - \boldsymbol{v}; 0, \sigma^2 \boldsymbol{I}),$$

where $\mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$ is the standard normal density. This kernel is the **radial basis function** (RBF) kernel, because its value decre in $\boldsymbol{v}$ space radiating outward from $\boldsymbol{u}$. The Gaussian kernel corres product in an infinite-dimensional space, but the derivation of th straightforward than in our example of the min kernel over the in

We can think of the Gaussian kernel as performing a kind of **ter ing**. A training example $\boldsymbol{x}$ associated with training label y beco for class y. When a test point $\boldsymbol{x}'$ is near $\boldsymbol{x}$ according to Euclidea Gaussian kernel has a large response, indicating that $\boldsymbol{x}'$ is very s template. The model then puts a large weight on the associated t Overall, the prediction will combine many such training labels w similarity of the corresponding training examples.

Support vector machines are not the only algorithm that ca using the kernel trick. Many other linear models can be enhanced i category of algorithms that employ the kernel trick is known as **ker** or **kernel methods** (Williams and Rasmussen, 1996; Schölkopf *e*

A major drawback to kernel machines is that the cost of evaluat
function is linear in the number of training examples, because th
contributes a term $\alpha_i k(\boldsymbol{x}, \boldsymbol{x}^{(i)})$ to the decision function. Support v
are able to mitigate this by learning an $\boldsymbol{\alpha}$ vector that contain
Classifying a new example then requires evaluating the kernel fu
the training examples that have nonzero α_i. These training exam
as **support vectors**.

Kernel machines also suffer from a high computational cost o
the dataset is large. We revisit this idea in section 5.9. Kernel

generic kernels struggle to generalize well. We explain why in sec
modern incarnation of deep learning was designed to overcome thes
kernel machines. The current deep learning renaissance began wh
(2006) demonstrated that a neural network could outperform the R
on the MNIST benchmark.

5.7.3 Other Simple Supervised Learning Algorithms

We have already briefly encountered another nonprobabilistic supe
algorithm, nearest neighbor regression. More generally, k-neare
a family of techniques that can be used for classification or re
nonparametric learning algorithm, k-nearest neighbors is not restr
number of parameters. We usually think of the k-nearest neigh
as not having any parameters but rather implementing a simple
training data. In fact, there is not even really a training stage or l
Instead, at test time, when we want to produce an output y for a ne
we find the k-nearest neighbors to $\boldsymbol{x}$ in the training data $\boldsymbol{X}$. We
average of the corresponding y values in the training set. This work
any kind of supervised learning where we can define an average ov
the case of classification, we can average over one-hot code vector
and $c_i = 0$ for all other values of i. We can then interpret the ave
one-hot codes as giving a probability distribution over classes. As a
learning algorithm, k-nearest neighbor can achieve very high capacit
suppose we have a multiclass classification task and measure perfor
loss. In this setting, 1-nearest neighbor converges to double the Ba
number of training examples approaches infinity. The error in exce
error results from choosing a single neighbor by breaking ties b
distant neighbors randomly. When there is infinite training data, a
will have infinitely many training set neighbors at distance zero.
algorithm to use all these neighbors to vote, rather than randoml

of them, the procedure converges to the Bayes error rate. The h
k-nearest neighbors enables it to obtain high accuracy given a lar
It does so at high computational cost, however, and it may genera
given a small finite training set. One weakness of k-nearest neig
cannot learn that one feature is more discriminative than anothe
imagine we have a regression task with $\boldsymbol{x} \in \mathbb{R}^{100}$ drawn from an iso
distribution, but only a single variable x_1 is relevant to the ou
further that this feature simply encodes the output directly, tha
cases. Nearest neighbor regression will not be able to detect this

The nearest neighbor of most points $\boldsymbol{x}$ will be determined by the l features x_2 through x_{100}, not by the lone feature x_1. Thus the o training sets will essentially be random.

Another type of learning algorithm that also breaks the input sp and has separate parameters for each region is the **decision tree** 1984) and its many variants. As shown in figure 5.7, each node tree is associated with a region in the input space, and internal no region into one subregion for each child of the node (typically using cut). Space is thus subdivided into nonoverlapping regions, wit correspondence between leaf nodes and input regions. Each leaf noc every point in its input region to the same output. Decision tr trained with specialized algorithms that are beyond the scope of learning algorithm can be considered nonparametric if it is allowed of arbitrary size, though decision trees are usually regularized with that turn them into parametric models in practice. Decision tr typically used, with axis-aligned splits and constant outputs wit struggle to solve some problems that are easy even for logistic r example, if we have a two-class problem, and the positive class o $x_2 > x_1$, the decision boundary is not axis aligned. The decision need to approximate the decision boundary with many nodes, impl function that constantly walks back and forth across the true de with axis-aligned steps.

As we have seen, nearest neighbor predictors and decision tr limitations. Nonetheless, they are useful learning algorithms when resources are constrained. We can also build intuition for mor learning algorithms by thinking about the similarities and diffe sophisticated algorithms and k-nearest neighbors or decision tree

See Murphy (2012), Bishop (2006), Hastie *et al.* (2001) or learning textbooks for more material on traditional supervised learn

5.8 Unsupervised Learning Algorithms

Recall from section 5.1.3 that unsupervised algorithms are those t
only "features" but not a supervision signal. The distinction betw
and unsupervised algorithms is not formally and rigidly defined bec
objective test for distinguishing whether a value is a feature or a tar
a supervisor. Informally, unsupervised learning refers to most atte
information from a distribution that do not require human lab

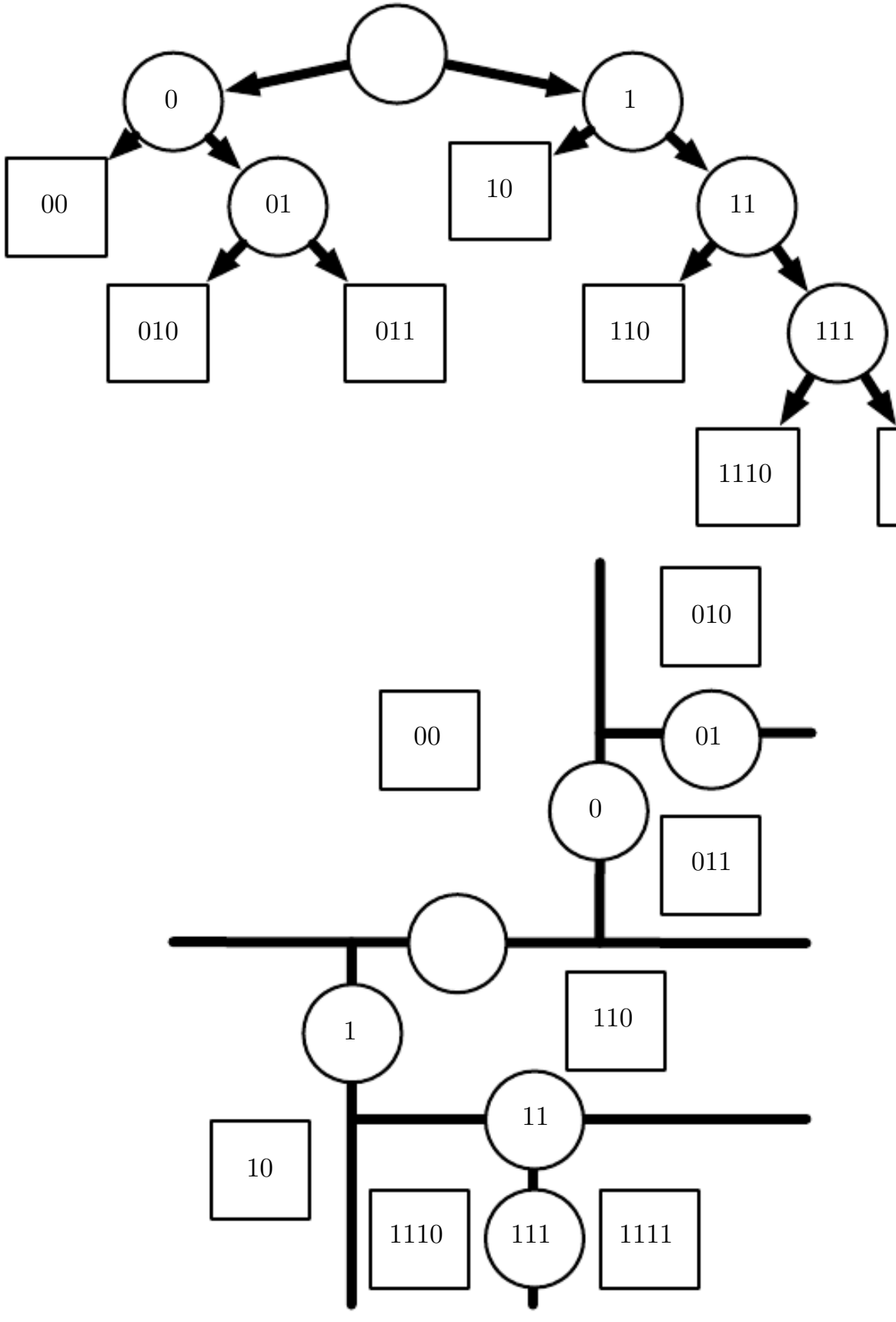


Figure 5.7: Diagrams describing how a decision tree works. *(Top)*Each

chooses to send the input example to the child node on the left (0) or t
on the right (1). Internal nodes are drawn as circles and leaf nodes a
node is displayed with a binary string identifier corresponding to its pos
obtained by appending a bit to its parent identifier (0 = choose left or
right or bottom). *(Bottom)*The tree divides space into regions. The
how a decision tree might divide $\mathbb{R}^2$. The nodes of the tree are plott
with each internal node drawn along the dividing line it uses to cate
and leaf nodes drawn in the center of the region of examples they rec
is a piecewise-constant function, with one piece per leaf. Each leaf requ
training example to define, so it is not possible for the decision tree to
that has more local maxima than the number of training examples.

examples. The term is usually associated with density estimati
draw samples from a distribution, learning to denoise data from so
finding a manifold that the data lies near, or clustering the data
related examples.

A classic unsupervised learning task is to find the "best" repres
data. By "best" we can mean different things, but generally speakin
for a representation that preserves as much information about $\boldsymbol{x}$ a
obeying some penalty or constraint aimed at keeping the representa
more accessible than $\boldsymbol{x}$ itself.

There are multiple ways of defining a simpler representation
most common include lower-dimensional representations, sparse
and independent representations. Low-dimensional representati
compress as much information about x as possible in a smaller
Sparse representations (Barlow, 1989; Olshausen and Field, 199
Ghahramani, 1997) embed the dataset into a representation wh
mostly zeros for most inputs. The use of sparse representations ty
increasing the dimensionality of the representation, so that the
becoming mostly zeros does not discard too much information. Th
overall structure of the representation that tends to distribute data
of the representation space. Independent representations attempt
the sources of variation underlying the data distribution such that
of the representation are statistically independent.

Of course these three criteria are certainly not mutually e
dimensional representations often yield elements that have fewe
pendencies than the original high-dimensional data. This is beca
reduce the size of a representation is to find and remove redundanc
and removing more redundancy enables the dimensionality reducti
achieve more compression while discarding less information.

The notion of representation is one of the central themes of de

therefore one of the central themes in this book. In this section, w
simple examples of representation learning algorithms. Together,
algorithms show how to operationalize all three of the criteria abo
remaining chapters introduce additional representation learning a
develop these criteria in different ways or introduce other criteria.

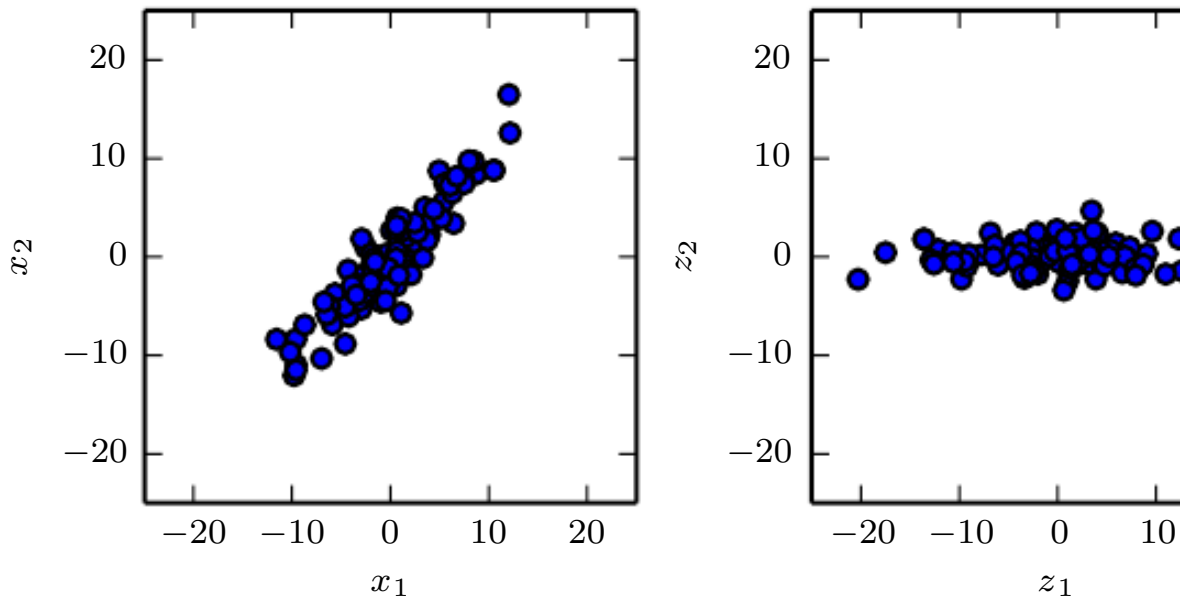


Figure 5.8: PCA learns a linear projection that aligns the direction of great
the axes of the new space. *(Left)*The original data consist of samples of $\boldsymbol{x}$.
variance might occur along directions that are not axis aligned. *(Right)*
data $\boldsymbol{z} = \boldsymbol{x}^\top \boldsymbol{W}$ now varies most along the axis z_1. The direction of seco
is now along z_2.

5.8.1 Principal Components Analysis

In section 2.12, we saw that the principal components analysis alg
a means of compressing data. We can also view PCA as an unsupe
algorithm that learns a representation of data. This representat
two of the criteria for a simple representation described above.
representation that has lower dimensionality than the original inpu
a representation whose elements have no linear correlation with e
is a first step toward the criterion of learning representations who
statistically independent. To achieve full independence, a represen
algorithm must also remove the nonlinear relationships between v

PCA learns an orthogonal, linear transformation of the data t

input $\boldsymbol{x}$ to a representation $\boldsymbol{z}$ as shown in figure 5.8. In section 2.
we could learn a one-dimensional representation that best reconstru
data (in the sense of mean squared error) and that this represen
corresponds to the first principal component of the data. Thus w
as a simple and effective dimensionality reduction method that pre
of the information in the data as possible (again, as measured b
reconstruction error). In the following, we will study how the PCA
decorrelates the original data representation $\boldsymbol{X}$.

Let us consider the $m \times n$ design matrix $\boldsymbol{X}$. We will assume th

a mean of zero, $\mathbb{E}[\boldsymbol{x}] = \mathbf{0}$. If this is not the case, the data can eas
by subtracting the mean from all examples in a preprocessing step

The unbiased sample covariance matrix associated with $\boldsymbol{X}$ is g

$$\text{Var}[\boldsymbol{x}] = \frac{1}{m-1}\boldsymbol{X}^\top \boldsymbol{X}.$$

PCA finds a representation (through linear transformation) $\boldsymbol{z}$ =
$\text{Var}[\boldsymbol{z}]$ is diagonal.

In section 2.12, we saw that the principal components of a de
are given by the eigenvectors of $\boldsymbol{X}^\top \boldsymbol{X}$. From this view,

$$\boldsymbol{X}^\top \boldsymbol{X} = \boldsymbol{W}\boldsymbol{\Lambda}\boldsymbol{W}^\top.$$

In this section, we exploit an alternative derivation of the princip
The principal components may also be obtained via singular value
(SVD). Specifically, they are the right singular vectors of $\boldsymbol{X}$. To see
the right singular vectors in the decomposition $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{W}^\top$. W
the original eigenvector equation with $\boldsymbol{W}$ as the eigenvector basis

$$\boldsymbol{X}^\top \boldsymbol{X} = \left(\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{W}^\top\right)^\top \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{W}^\top = \boldsymbol{W}\boldsymbol{\Sigma}^2\boldsymbol{W}^\top.$$

The SVD is helpful to show that PCA results in a diagonal Va
SVD of $\boldsymbol{X}$, we can express the variance of $\boldsymbol{X}$ as:

$$\begin{aligned}
\text{Var}[\boldsymbol{x}] &= \frac{1}{m-1}\boldsymbol{X}^\top \boldsymbol{X} \\
&= \frac{1}{m-1}(\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{W}^\top)^\top \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{W}^\top \\
&= \frac{1}{\quad}\boldsymbol{W}\boldsymbol{\Sigma}^\top \boldsymbol{U}^\top \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{W}^\top \\
&\quad \frac{m-1}{\quad}
\end{aligned}$$

$$= \frac{1}{m-1}\boldsymbol{W}\boldsymbol{\Sigma}^2\boldsymbol{W}^\top,$$

where we use the fact that $\boldsymbol{U}^\top\boldsymbol{U} = \boldsymbol{I}$ because the $\boldsymbol{U}$ matrix of the decomposition is defined to be orthogonal. This shows that the co diagonal as required:

$$\begin{aligned}\text{Var}[\boldsymbol{z}] &= \frac{1}{m-1}\boldsymbol{Z}^\top\boldsymbol{Z} \\ &= \frac{1}{m-1}\boldsymbol{W}^\top\boldsymbol{X}^\top\boldsymbol{X}\boldsymbol{W}\end{aligned}$$

$$= \frac{1}{m-1}\boldsymbol{W}^\top \boldsymbol{W}\boldsymbol{\Sigma}^2\boldsymbol{W}^\top \boldsymbol{W}$$
$$= \frac{1}{m-1}\boldsymbol{\Sigma}^2,$$

where this time we use the fact that $\boldsymbol{W}^\top \boldsymbol{W} = \boldsymbol{I}$, again from the d
SVD.

The above analysis shows that when we project the data $\boldsymbol{x}$ to $\boldsymbol{z}$
transformation $\boldsymbol{W}$, the resulting representation has a diagonal co
(as given by $\boldsymbol{\Sigma}^2$), which immediately implies that the individual el
mutually uncorrelated.

This ability of PCA to transform data into a representation whe
are mutually uncorrelated is a very important property of PCA
example of a representation that attempts to *disentangle the unkn*
variation underlying the data. In the case of PCA, this disentan
form of finding a rotation of the input space (described by $\boldsymbol{W}$)
principal axes of variance with the basis of the new representation s
with $\boldsymbol{z}$.

While correlation is an important category of dependency betw
the data, we are also interested in learning representations that di
complicated forms of feature dependencies. For this, we will need m
can be done with a simple linear transformation.

5.8.2 k-means Clustering

Another example of a simple representation learning algorithm is k-m
The k-means clustering algorithm divides the training set into k d
of examples that are near each other. We can thus think of th
providing a k-dimensional one-hot code vector $\boldsymbol{h}$ representing an
belongs to cluster i, then $h_i = 1$, and all other entries of the repre

zero.

The one-hot code provided by k-means clustering is an exam
representation, because the majority of its entries are zero for eve
we develop other algorithms that learn more flexible sparse represe
more than one entry can be nonzero for each input $\boldsymbol{x}$. One-hot codes
example of sparse representations that lose many of the benefits
representation. The one-hot code still confers some statistical
naturally conveys the idea that all examples in the same cluster are
other), and it confers the computational advantage that the entire

may be captured by a single integer.

The k-means algorithm works by initializing k different centroids
to different values, then alternating between two different steps un
In one step, each training example is assigned to cluster i, where i
the nearest centroid $\boldsymbol{\mu}^{(i)}$. In the other step, each centroid $\boldsymbol{\mu}^{(i)}$ is
mean of all training examples $\boldsymbol{x}^{(j)}$ assigned to cluster i.

One difficulty pertaining to clustering is that the clustering probl
ill posed, in the sense that there is no single criterion that measu
clustering of the data corresponds to the real world. We can mea
of the clustering, such as the average Euclidean distance from a c
to the members of the cluster. This enables us to tell how well
reconstruct the training data from the cluster assignments. We do
well the cluster assignments correspond to properties of the real w
there may be many different clusterings that all correspond well to
of the real world. We may hope to find a clustering that relates to
obtain a different, equally valid clustering that is not relevant to
example, suppose that we run two clustering algorithms on a datas
images of red trucks, images of red cars, images of gray trucks, and
cars. If we ask each clustering algorithm to find two clusters, one
find a cluster of cars and a cluster of trucks, while another may f
red vehicles and a cluster of gray vehicles. Suppose we also run a
algorithm, which is allowed to determine the number of clusters. T
the examples to four clusters, red cars, red trucks, gray cars, and gr
new clustering now at least captures information about both attrib
lost information about similarity. Red cars are in a different clu
cars, just as they are in a different cluster from gray trucks. The
clustering algorithm does not tell us that red cars are more simil
than they are to gray trucks. They are different from both things,
we know.

These issues illustrate some of the reasons that we may prefe
representation to a one-hot representation. A distributed representa
two attributes for each vehicle—one representing its color and o
whether it is a car or a truck. It is still not entirely clear wh
distributed representation is (how can the learning algorithm kn
two attributes we are interested in are color and car-versus-tru
manufacturer and age?), but having many attributes reduces the
algorithm to guess which single attribute we care about, and give
to measure similarity between objects in a fine-grained way by c

attributes instead of just testing whether one attribute matches.

5.9 Stochastic Gradient Descent

Nearly all of deep learning is powered by one very important algorith
gradient descent (SGD). Stochastic gradient descent is an ex
gradient descent algorithm introduced in section 4.3.

A recurring problem in machine learning is that large training se
for good generalization, but large training sets are also more c
expensive.

The cost function used by a machine learning algorithm often d
sum over training examples of some per-example loss function. Fo
negative conditional log-likelihood of the training data can be wri

$$J(\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{x},\mathrm{y}\sim\hat{p}_{\mathrm{data}}} L(\boldsymbol{x}, y, \boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m} L(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta}),$$

where L is the per-example loss $L(\boldsymbol{x}, y, \boldsymbol{\theta}) = -\log p(y \mid \boldsymbol{x}; \boldsymbol{\theta})$.

For these additive cost functions, gradient descent requires co

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} L(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta}).$$

The computational cost of this operation is $O(m)$. As the training s
billions of examples, the time to take a single gradient step becom
long.

The insight of SGD is that the gradient is an expectation. The e
be approximately estimated using a small set of samples. Specifical
of the algorithm, we can sample a **minibatch** of examples $\mathbb{B} =$

drawn uniformly from the training set. The minibatch size m' is t
to be a relatively small number of examples, ranging from one to
Crucially, m' is usually held fixed as the training set size m grows
training set with billions of examples using updates computed on
examples.

The estimate of the gradient is formed as

$$\boldsymbol{g} = \frac{1}{m'} \nabla_{\boldsymbol{\theta}} \sum_{i=1}^{m'} L(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta})$$

using examples from the minibatch $\mathbb{B}$. The stochastic gradient des
then follows the estimated gradient downhill:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \boldsymbol{g},$$

where ϵ is the learning rate.

Gradient descent in general has often been regarded as slow o
the past, the application of gradient descent to nonconvex optimiz
was regarded as foolhardy or unprincipled. Today, we know tha
learning models described in part II work very well when trained
descent. The optimization algorithm may not be guaranteed to a
local minimum in a reasonable amount of time, but it often finds a
of the cost function quickly enough to be useful.

Stochastic gradient descent has many important uses outside
deep learning. It is the main way to train large linear models
datasets. For a fixed model size, the cost per SGD update does not
training set size m. In practice, we often use a larger model as the
increases, but we are not forced to do so. The number of updates re
convergence usually increases with training set size. However, as
infinity, the model will eventually converge to its best possible te
SGD has sampled every example in the training set. Increasing m
extend the amount of training time needed to reach the model's be
error. From this point of view, one can argue that the asymptotic
a model with SGD is $O(1)$ as a function of m.

Prior to the advent of deep learning, the main way to learn no
was to use the kernel trick in combination with a linear model. Many
algorithms require constructing an $m \times m$ matrix $G_{i,j} = k(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}$
this matrix has computational cost $O(m^2)$, which is clearly undesira
with billions of examples. In academia, starting in 2006, deep
initially interesting because it was able to generalize to new e

than competing algorithms when trained on medium-sized datase
thousands of examples. Soon after, deep learning garnered additi
industry because it provided a scalable way of training nonlinear m
datasets.

Stochastic gradient descent and many enhancements to it are d
in chapter 8.

5.10 Building a Machine Learning Algorithm

Nearly all deep learning algorithms can be described as particul
a fairly simple recipe: combine a specification of a dataset, a co
optimization procedure and a model.

For example, the linear regression algorithm combines a datas
$\boldsymbol{X}$ and $\boldsymbol{y}$, the cost function

$$J(\boldsymbol{w}, b) = -\mathbb{E}_{\mathbf{x},\mathrm{y}\sim\hat{p}_{\mathrm{data}}} \log p_{\mathrm{model}}(y \mid \boldsymbol{x}),$$

the model specification $p_{\mathrm{model}}(y \mid \boldsymbol{x}) = \mathcal{N}(y; \boldsymbol{x}^\top \boldsymbol{w} + b, 1)$, and, in
optimization algorithm defined by solving for where the gradient of
using the normal equations.

By realizing that we can replace any of these components mostly
from the others, we can obtain a wide range of algorithms.

The cost function typically includes at least one term that caus
process to perform statistical estimation. The most common cost
negative log-likelihood, so that minimizing the cost function ca
likelihood estimation.

The cost function may also include additional terms, such as
terms. For example, we can add weight decay to the linear regressio
to obtain

$$J(\boldsymbol{w}, b) = \lambda||\boldsymbol{w}||_2^2 - \mathbb{E}_{\mathbf{x},\mathrm{y}\sim\hat{p}_{\mathrm{data}}} \log p_{\mathrm{model}}(y \mid \boldsymbol{x}).$$

This still allows closed form optimization.

If we change the model to be nonlinear, then most cost function
be optimized in closed form. This requires us to choose an itera
optimization procedure, such as gradient descent.

The recipe for constructing a learning algorithm by combining mo

optimization algorithms supports both supervised and unsupervise

optimization algorithms supports both supervised and unsupervise
linear regression example shows how to support supervised learning
learning can be supported by defining a dataset that contains only $\boldsymbol{X}$
an appropriate unsupervised cost and model. For example, we can
PCA vector by specifying that our loss function is

$$J(\boldsymbol{w}) = \mathbb{E}_{\mathbf{x} \sim \hat{p}_{\text{data}}} ||\boldsymbol{x} - r(\boldsymbol{x}; \boldsymbol{w})||_2^2$$

while our model is defined to have $\boldsymbol{w}$ with norm one and reconstr
$r(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x} \boldsymbol{w}$.

In some cases, the cost function may be a function that we c
evaluate, for computational reasons. In these cases, we can still
minimize it using iterative numerical optimization, as long as we ha
approximating its gradients.

Most machine learning algorithms make use of this recipe, tho
be immediately obvious. If a machine learning algorithm seems espe
hand designed, it can usually be understood as using a special-case o
models, such as decision trees and k-means, require special-case opt
their cost functions have flat regions that make them inappropriate f
by gradient-based optimizers. Recognizing that most machine lear
can be described using this recipe helps to see the different algorith
taxonomy of methods for doing related tasks that work for similar
than as a long list of algorithms that each have separate justificat

5.11 Challenges Motivating Deep Learning

The simple machine learning algorithms described in this chapter
wide variety of important problems. They have not succeeded, how
the central problems in AI, such as recognizing speech or recogniz

The development of deep learning was motivated in part by
traditional algorithms to generalize well on such AI tasks.

This section is about how the challenge of generalizing to new exa
exponentially more difficult when working with high-dimensional
the mechanisms used to achieve generalization in traditional ma
are insufficient to learn complicated functions in high-dimensiona
spaces also often impose high computational costs. Deep learning
overcome these and other obstacles.

5.11.1 The Curse of Dimensionality

Many machine learning problems become exceedingly difficult wh
of dimensions in the data is high. This phenomenon is known a
dimensionality. Of particular concern is that the number of p
configurations of a set of variables increases exponentially as the num
increases.

The curse of dimensionality arises in many places in computer sci
in machine learning.

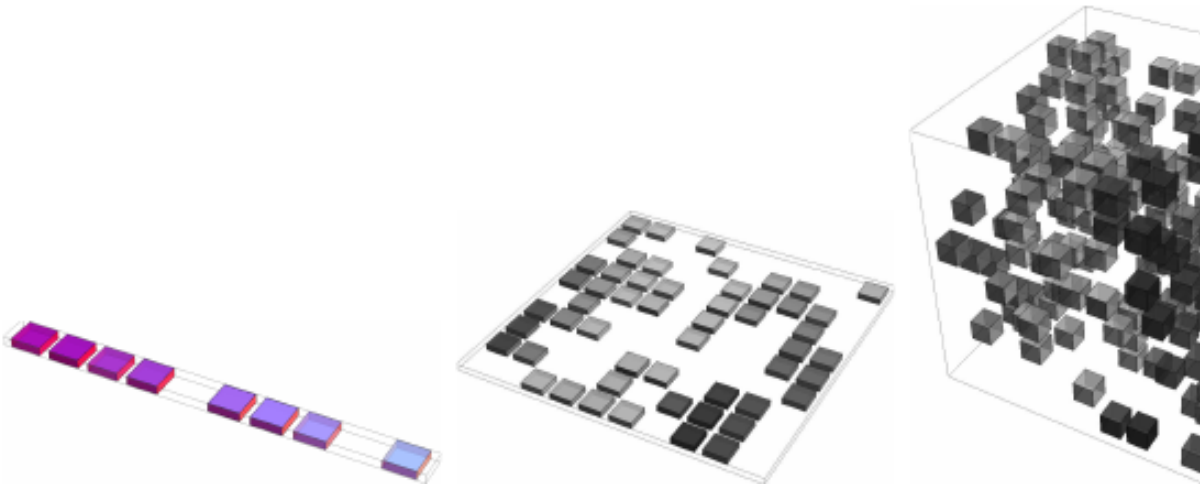


Figure 5.9: As the number of relevant dimensions of the data increa
right), the number of configurations of interest may grow exponential
one-dimensional example, we have one variable for which we only care
regions of interest. With enough examples falling within each of these reg
corresponds to a cell in the illustration), learning algorithms can easily ge
A straightforward way to generalize is to estimate the value of the targe
each region (and possibly interpolate between neighboring regions). *(*
dimensions, it is more difficult to distinguish 10 different values of each v
to keep track of up to $10 \times 10 = 100$ regions, and we need at least that n
cover all those regions. *(Right)*With three dimensions, this grows to 10^3
and at least that many examples. For d dimensions and v values to be dis
each axis, we seem to need $O(v^d)$ regions and examples. This is an insta
of dimensionality. Figure graciously provided by Nicolas Chapados.

One challenge posed by the curse of dimensionality is a statis
As illustrated in figure 5.9, a statistical challenge arises because
possible configurations of $\boldsymbol{x}$ is much larger than the number of tra
To understand the issue, let us consider that the input space is o
grid, as in the figure. We can describe low-dimensional space with
of grid cells that are mostly occupied by the data. When generalizin
point, we can usually tell what to do simply by inspecting the tra
that lie in the same cell as the new input. For example, if estimating

density at some point $\boldsymbol{x}$, we can just return the number of traini
the same unit volume cell as $\boldsymbol{x}$, divided by the total number of tra
If we wish to classify an example, we can return the most common
examples in the same cell. If we are doing regression, we can ave
values observed over the examples in that cell. But what about the
we have seen no example? Because in high-dimensional spaces,
configurations is huge, much larger than our number of examples, a
has no training example associated with it. How could we possibly
meaningful about these new configurations? Many traditional m

algorithms simply assume that the output at a new point should be the same as the output at the nearest training point.

5.11.2 Local Constancy and Smoothness Regulariza

To generalize well, machine learning algorithms need to be guided about what kind of function they should learn. We have seen these rated as explicit beliefs in the form of probability distributions ove the model. More informally, we may also discuss prior beliefs as dire the *function* itself and influencing the parameters only indirectly, as relationship between the parameters and the function. Additionally discuss prior beliefs as being expressed implicitly by choosing a are biased toward choosing some class of functions over anothe these biases may not be expressed (or even be possible to express probability distribution representing our degree of belief in variou

Among the most widely used of these implicit "priors" is th **prior**, or **local constancy prior**. This prior states that the fun should not change very much within a small region.

Many simpler algorithms rely exclusively on this prior to gene as a result, they fail to scale to the statistical challenges involved level tasks. Throughout this book, we describe how deep learn additional (explicit and implicit) priors in order to reduce the error on sophisticated tasks. Here, we explain why the smoothnes insufficient for these tasks.

There are many different ways to implicitly or explicitly expres that the learned function should be smooth or locally constant. Al methods are designed to encourage the learning process to learn a f satisfies the condition

$$f^*(\boldsymbol{x}) \approx f^*(\boldsymbol{x} + \epsilon)$$

for most configurations $\boldsymbol{x}$ and small change ϵ. In other words, if w
answer for an input $\boldsymbol{x}$ (for example, if $\boldsymbol{x}$ is a labeled training exam
answer is probably good in the neighborhood of $\boldsymbol{x}$. If we have sever
in some neighborhood, we would combine them (by some form
interpolation) to produce an answer that agrees with as many of th
possible.

An extreme example of the local constancy approach is the k-ne
family of learning algorithms. These predictors are literally cons
region containing all the points $\boldsymbol{x}$ that have the same set of k near

the training set. For $k = 1$, the number of distinguishable regions c
than the number of training examples.

While the k-nearest neighbors algorithm copies the output from
examples, most kernel machines interpolate between training set out
with nearby training examples. An important class of kernels is the
kernels, where $k(\boldsymbol{u}, \boldsymbol{v})$ is large when $\boldsymbol{u} = \boldsymbol{v}$ and decreases as $\boldsymbol{u}$ and
apart from each other. A local kernel can be thought of as a sim
that performs template matching, by measuring how closely a t
resembles each training example $\boldsymbol{x}^{(i)}$. Much of the modern moti
learning is derived from studying the limitations of local templat
how deep models are able to succeed in cases where local templat
(Bengio *et al.*, 2006b).

Decision trees also suffer from the limitations of exclusively sm
learning, because they break the input space into as many regio
leaves and use a separate parameter (or sometimes many parameter
of decision trees) in each region. If the target function requires
least n leaves to be represented accurately, then at least n trainin
required to fit the tree. A multiple of n is needed to achieve some le
confidence in the predicted output.

In general, to distinguish $O(k)$ regions in input space, all these r
$O(k)$ examples. Typically there are $O(k)$ parameters, with O
associated with each of the $O(k)$ regions. The nearest neighbor sc
each training example can be used to define at most one region,
figure 5.10.

Is there a way to represent a complex function that has many
be distinguished than the number of training examples? Clearly,
smoothness of the underlying function will not allow a learner t
example, imagine that the target function is a kind of checkerboard.
contains many variations, but there is a simple structure to them

happens when the number of training examples is substantially sm
number of black and white squares on the checkerboard. Based
generalization and the smoothness or local constancy prior, the le
guaranteed to correctly guess the color of a new point if it lay w
checkerboard square as a training example. There is no guarantee
the learner could correctly extend the checkerboard pattern to
squares that do not contain training examples. With this prior
information that an example tells us is the color of its square, and
get the colors of the entire checkerboard right is to cover each of

least one example.

The smoothness assumption and the associated nonparametri
rithms work extremely well as long as there are enough examples
algorithm to observe high points on most peaks and low points
of the true underlying function to be learned. This is generally
function to be learned is smooth enough and varies in few enou
In high dimensions, even a very smooth function can change sm
different way along each dimension. If the function additionally beh
in various regions, it can become extremely complicated to descri
training examples. If the function is complicated (we want to dis
number of regions compared to the number of examples), is the
generalize well?

The answer to both of these questions—whether it is possib
a complicated function efficiently, and whether it is possible for
function to generalize well to new inputs—is yes. The key insight is t
number of regions, such as $O(2^k)$, can be defined with $O(k)$ example

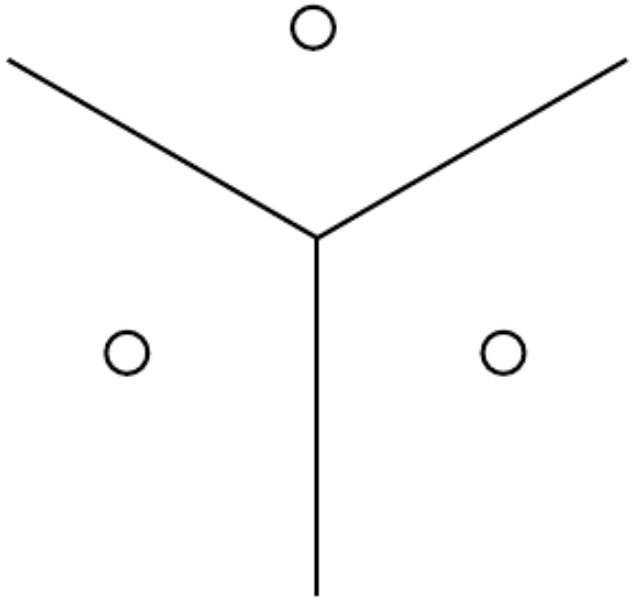


Figure 5.10: Illustration of how the nearest neighbor algorithm breaks up

into regions. An example (represented here by a circle) within each re
region boundary (represented here by the lines). The y value associated w
defines what the output should be for all points within the correspond
regions defined by nearest neighbor matching form a geometric pattern
diagram. The number of these contiguous regions cannot grow faster t
of training examples. While this figure illustrates the behavior of the
algorithm specifically, other machine learning algorithms that rely ex
local smoothness prior for generalization exhibit similar behaviors: each
only informs the learner about how to generalize in some neighborh
surrounding that example.

introduce some dependencies between the regions through additio
about the underlying data-generating distribution. In this way, v
generalize nonlocally (Bengio and Monperrus, 2005; Bengio *et al.*
different deep learning algorithms provide implicit or explicit assu
reasonable for a broad range of AI tasks in order to capture these

Other approaches to machine learning often make stronger, t
sumptions. For example, we could easily solve the checkerboard ta
the assumption that the target function is periodic. Usually we do n
strong, task-specific assumptions in neural networks so that they
to a much wider variety of structures. AI tasks have structure th
complex to be limited to simple, manually specified properties suc
so we want learning algorithms that embody more general-purpo
The core idea in deep learning is that we assume that the data wa
the *composition of factors*, or features, potentially at multiple le
chy. Many other similarly generic assumptions can further improv
algorithms. These apparently mild assumptions allow an exponen
relationship between the number of examples and the number of r
be distinguished. We describe these exponential gains more preci
6.4.1, 15.4 and 15.5. The exponential advantages conferred by t
distributed representations counter the exponential challenges pos
of dimensionality.

5.11.3 Manifold Learning

An important concept underlying many ideas in machine learni
manifold.

A **manifold** is a connected region. Mathematically, it is a
associated with a neighborhood around each point. From any g
manifold locally appears to be a Euclidean space. In everyday life

the surface of the world as a 2-D plane, but it is in fact a spheri

the surface of the world as a 2-D plane, but it is in fact a spheri
3-D space.

The concept of a neighborhood surrounding each point implies
transformations that can be applied to move on the manifold from
a neighboring one. In the example of the world's surface as a ma
walk north, south, east, or west.

Although there is a formal mathematical meaning to the term
machine learning it tends to be used more loosely to designate a
of points that can be approximated well by considering only a sr

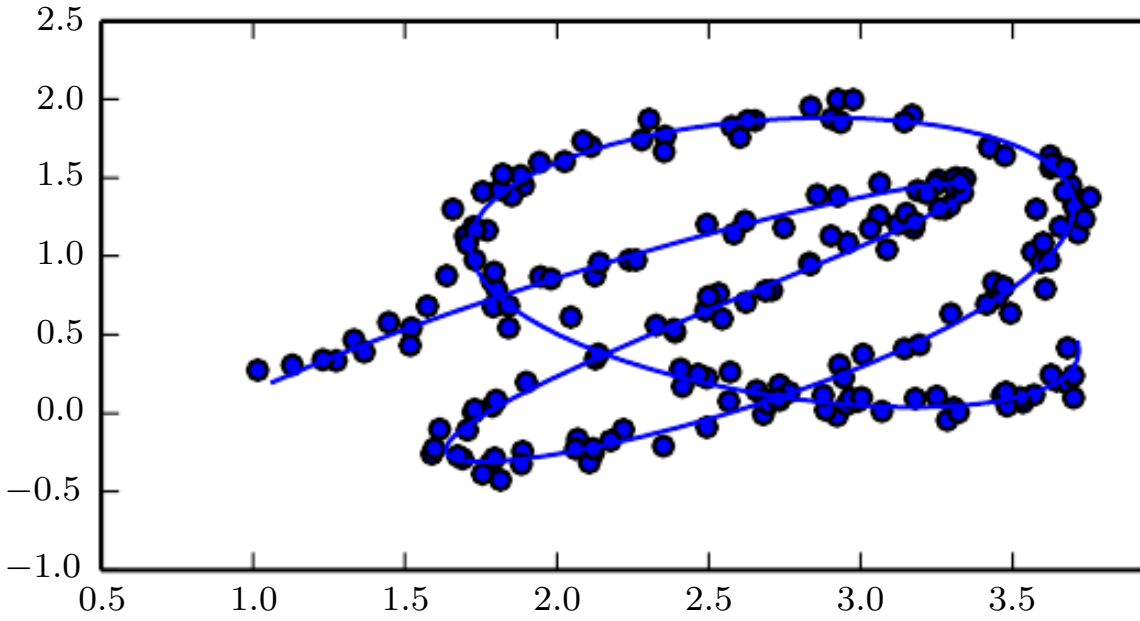


Figure 5.11: Data sampled from a distribution in a two-dimensional spac
concentrated near a one-dimensional manifold, like a twisted string. The s
the underlying manifold that the learner should infer.

degrees of freedom, or dimensions, embedded in a higher-dimensio
dimension corresponds to a local direction of variation. See fig
example of training data lying near a one-dimensional manifold em
dimensional space. In the context of machine learning, we allow the
of the manifold to vary from one point to another. This often h
manifold intersects itself. For example, a figure eight is a manifold t
dimension in most places but two dimensions at the intersection a

Many machine learning problems seem hopeless if we expec
learning algorithm to learn functions with interesting variations a
Manifold learning algorithms surmount this obstacle by assu
of $\mathbb{R}^n$ consists of invalid inputs, and that interesting inputs oc
a collection of manifolds containing a small subset of points, w
variations in the output of the learned function occurring only a

that lie on the manifold, or with interesting variations happening
move from one manifold to another. Manifold learning was introdu
of continuous-valued data and in the unsupervised learning setting
probability concentration idea can be generalized to both discret
supervised learning setting: the key assumption remains that prob
highly concentrated.

The assumption that the data lies along a low-dimensional ma
always be correct or useful. We argue that in the context of AI
those that involve processing images, sounds, or text, the manifol

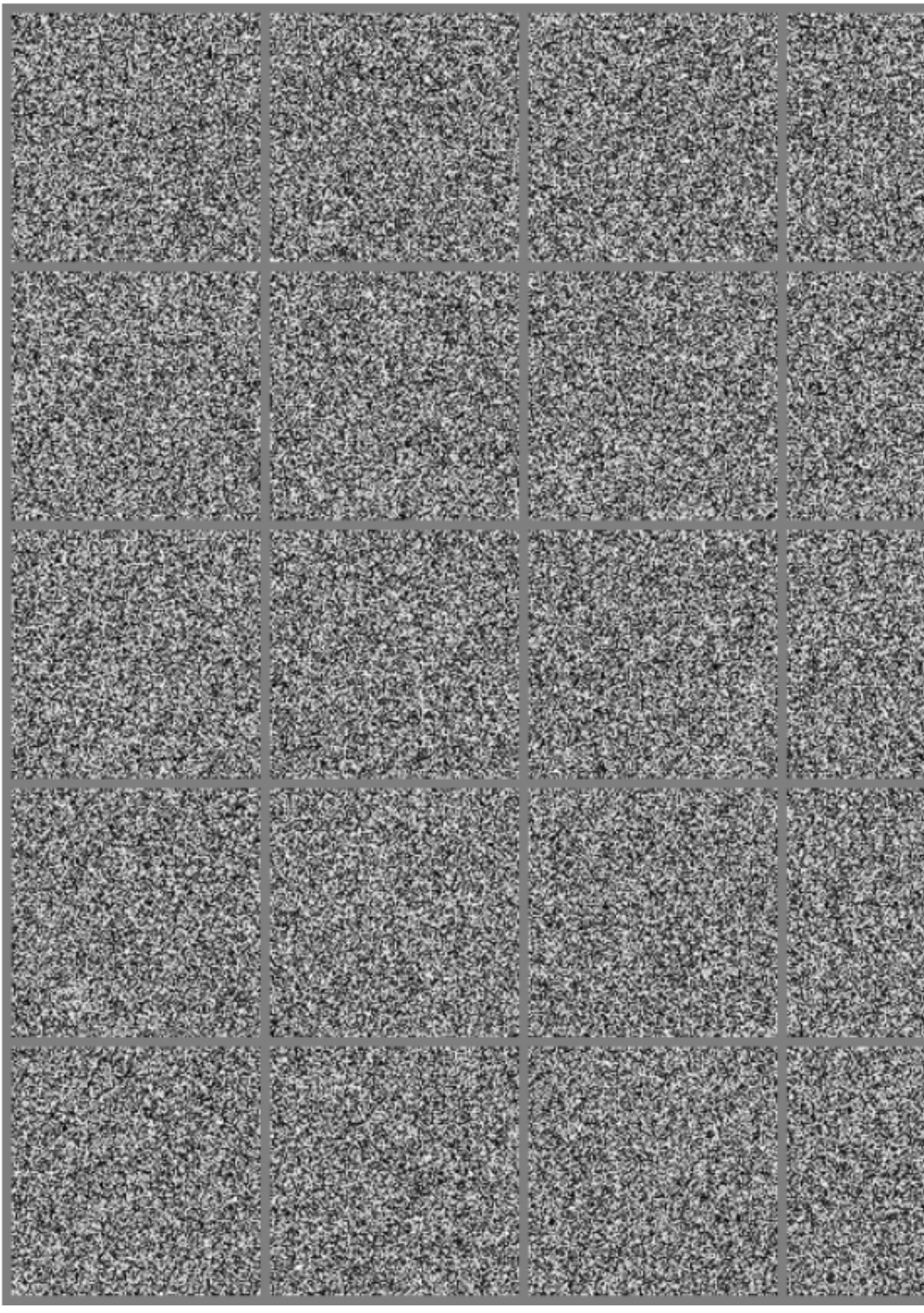


Figure 5.12: Sampling images uniformly at random (by randomly pi

according to a uniform distribution) gives rise to noisy images. Alth
nonzero probability of generating an image of a face or of any other c
encountered in AI applications, we never actually observe this happening
suggests that the images encountered in AI applications occupy a negligi
the volume of image space.

at least approximately correct. The evidence in favor of this assu
of two categories of observations.

The first observation in favor of the **manifold hypothesis** is

bility distribution over images, text strings, and sounds that occu
highly concentrated. Uniform noise essentially never resembles str
from these domains. Figure 5.12 shows how, instead, uniformly s
look like the patterns of static that appear on analog television sets
is available. Similarly, if you generate a document by picking lette
random, what is the probability that you will get a meaningful En
text? Almost zero, again, because most of the long sequences of
correspond to a natural language sequence: the distribution of na
sequences occupies a very little volume in the total space of seque

Of course, concentrated probability distributions are not sufficie
the data lies on a reasonably small number of manifolds. We mus
that the examples we encounter are connected to each other by o
with each example surrounded by other highly similar examples that
by applying transformations to traverse the manifold. The secon
favor of the manifold hypothesis is that we can imagine such neig
transformations, at least informally. In the case of images, we can
of many possible transformations that allow us to trace out a ma
space: we can gradually dim or brighten the lights, gradually m
objects in the image, gradually alter the colors on the surfaces of
forth. Multiple manifolds are likely involved in most applications
the manifold of human face images may not be connected to the
face images.

These thought experiments convey some intuitive reasons suppo
fold hypothesis. More rigorous experiments (Cayton, 2005; Naraya
2010; Schölkopf *et al.*, 1998; Roweis and Saul, 2000; Tenenbaum *et a*
2003; Belkin and Niyogi, 2003; Donoho and Grimes, 2003; Weinb
2004) clearly support the hypothesis for a large class of datasets o

When the data lies on a low-dimensional manifold, it can be m
machine learning algorithms to represent the data in terms of coo

$\mathbb{R}^n$

manifold, rather than in terms of coordinates in . In everyday li
of roads as 1-D manifolds embedded in 3-D space. We give direct
addresses in terms of address numbers along these 1-D roads, r
coordinates in 3-D space. Extracting these manifold coordinates is
holds the promise of improving many machine learning algorithm
principle is applied in many contexts. Figure 5.13 shows the manif
a dataset consisting of faces. By the end of this book, we will have
methods necessary to learn such a manifold structure. In figure 2
how a machine learning algorithm can successfully accomplish thi

This concludes part I, which has provided the basic concepts
and machine learning that are employed throughout the remaini
book. You are now prepared to embark on your study of deep lea

Figure 5.13: Training examples from the QMUL Multiview Face Data
2000), for which the subjects were asked to move in such a way as t
dimensional manifold corresponding to two angles of rotation. We wo
algorithms to be able to discover and disentangle such manifold coordin
illustrates such a feat.