1 Introduction

The proof is straightforward, and thus omitted

Ja-Chen Lin and Wen-Hsiang Tsai[1]

1.1 Concerning this book

This is an important observation: This book does NOT have enough information to tell you how to implement significant large systems. It teaches general principles. You MUST make use of the literature when you get down to the gnitty gritty.

We have written this book at two levels, the principal level being introductory. "Introductory" does not mean "easy" or "simple" or "doesn't require math." Rather, the introductory topics are those which need to be mastered before the advanced topics can be understood.

In addition, the book is intended to be useful as a reference. When you have to study a topic in more detail than is covered here, in order, for example, to implement a practical system, we have tried to provide adequate citations to the relevant literature to get you off to a good start.

We have tried to write in a style aimed directly toward the student and in a conversational tone.

We have also tried to make the text readable and entertaining. Words which are deluberately missppelled for humorous affects should be ubvious. Some of the humor runs to exaggeration and to puns; we hope you forgive us.

We did not attempt to cover every topic in the machine vision area. In particular, nearly all papers in the general areas of optical character recognition and face recognition have been omitted; not to slight these very important and very successful application areas, but rather because the papers tend to be rather specialized; in addition, we simply cannot cover everything.

There are two themes which run through this book: ***consistency*** and ***optimization***. Consistency is a conceptual tool, implemented as a variety of algorithms, which helps machines to recognize images – they fuse information from local measurements to make global conclusions about the image. Optimization is the mathematical mechanism used in virtually every chapter to accomplish the objectives of that chapter, be they pattern classification or image matching.

[1] Ja-Chen Lin and Wen-Hsiang Tsai, "Feature-preserving Clustering of 2-D Data for Two-class Problems Using Analytical Formulas: An Automatic and Fast Approach," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **16**(5), 1994.

These two topics, consistency and optimization, are so important and so pervasive, that we point out to the student, in the conclusion of each chapter, exactly where those concepts turned up in that chapter. So read the chapter conclusions. Who knows, it might be on a test.

1.2 Concerning prerequisites

To find out if you meet this criterion, answer the following question: What do the following words mean? "transpose," "inverse," "determinant," "eigenvalue." If you do not have any idea, do not take this course!

The target audience for this book is graduate students or advanced undergraduates in electrical engineering, computer engineering, computer science, math, statistics, or physics. To do the work in this book, you must have had a graduate-level course in advanced calculus, and in statistics and/or probability. You need either a formal course or experience in linear algebra.

Many of the homeworks will be projects of sorts, and will be computer-based. To complete these assignments, you will need a hardware and software environment capable of

You will have to write programs in C (yes, C or C++, not Matlab) to complete this course.

(1) declaring large arrays (256×256) in C
(2) displaying an image
(3) printing an image.

Software and data used in the book can be found at www.cambridge.org/9780521830461.

We are going to insist that you write programs, and that you write them at a relatively low level. Some of the functionality that you will be coding is available in software packages like Matlab. However, while you learn something by simply calling a function, you learn more by writing and debugging the code yourself. Exceptions to this occur, of course, when the coding is so extensive that the programming gets in the way of the image analysis. For that reason, we provide the student with a library of subroutines which allow the student to ignore details like data type, byteswapping, file access, and platform dependencies, and instead focus on the logic of making image analysis algorithms work.

You should have an instructor, and if you do, we strongly recommend that you GO to class, even though all the information you really need is in this book. Read the assigned material in the text, then go to class, then read the text material again. Remember:

A hacker hermit named Dave
Tapped in to this course in his cave.
He had to admit
He learned not a bit.
But look at the money he saved.

And now, on to the technical stuff.

1.3 Some terminology

Students usually confuse machine vision with image processing. In this section, we define some terminology that will clarify the differences between the contents and objectives of these two topics.

1.3.1 Image processing

Many people consider the content of this course as part of the discipline of image processing. However, a better use of the term is to distinguish between image processing and machine vision by the intent. "Image processing" strives to make images look better, and the output of an image processing system is an image. The output of a "machine vision" system is information about the content of the image. The functions of an image processing system may include enhancement, coding, compression, restoration, and reconstruction.

Enhancement

Enhancement systems perform operations which make the image look better, as perceived by a human observer. Typical operations include contrast stretching (including functions like histogram equalization), brightness scaling, edge sharpening, etc.

Coding

Coding is the process of finding efficient and effective ways to represent the information in an image. These include quantization methods and redundancy removal. Coding may also include methods for making the representation robust to bit-errors which occur when the image is transmitted or stored.

Compression

Compression includes many of the same techniques as coding, but with the specific objective of reducing the number of bits required to store and/or transmit the image.

Restoration

Restoration concerns itself with fixing what is wrong with the image. It is unlike enhancement, which is just concerned with making images look better. In order to "correct" an image, there must be some model of the image degradation. It is common in restoration applications to assume a deterministic blur operator, followed by additive random noise.

Reconstruction

Reconstruction usually refers to the process of constructing an image from several partial images. For example, in computed tomography (CT),[2] we make a large number, say 360, of x-ray projections through the subject. From this set of one-dimensional signals, we can compute the actual x-ray absorption at each point in the two-dimensional image. Similar methods are used in positron emission tomography (PET), magnetic resonance imagery (MRI), and in several shape-from-X algorithms which we will discuss later in this course.

1.3.2 Machine vision

Machine vision is the process whereby a machine, usually a digital computer, automatically processes an image and reports "what is in the image." That is, it recognizes the content of the image. Often the content may be a machined part, and the objective is not only to locate the part, but to inspect it as well. We will in this book discuss several applications of machine vision in detail, such as automatic target recognition (ATR), and industrial inspection. There are a wide variety of other applications, such as determining the flow equations from observations of fluid flow [1.1], which time and space do not allow us to cover.

The terms "computer vision" and "image understanding" are often also used to denote machine vision.

Machine vision includes two components – measurement of features and pattern classification based on those features.

Measurement of features

The measurement of features is the principal focus of this book. Except for Chapters 14 and 15, in this book, we focus on processing the elements of images (pixels) and from those pixels and collections of pixels, extract sets of measurements which characterize either the entire image or some component thereof.

Pattern classification

Pattern classification may be defined as the process of making a decision about a measurement. That is, we are **given** a measurement or set of measurements made on an unknown object. From that set of measurements with knowledge about the possible *classes* to which that unknown might belong, we make a decision. For

[2] Sometimes, CT is referred to as "CAT scanning." In that case, CAT stands for "computed axial tomography." There are other types of tomography as well.

example, the set of possible classes might be men and women and one measurement which we could make to distinguish men from women would be height (clearly, height is not a very good measurement to use to distinguish men from women, for if our decision is that anyone over five foot six is male we will surely be wrong in many instances).

Pattern recognition

Pattern recognition may be defined as the process of assigning unknowns to classes just as in the definition of pattern classification. However, the definition is extended to include the process of making the measurements.

1.4 Organization of a machine vision system

Fig. 1.1 shows schematically, at the most basic level, the organization of a machine vision system. The unknown is first measured and the values of a number of *features* are determined. In an industrial application, such features might include the length, width, and area of the image of the part being measured. Once the features are measured, their numerical values are passed to a process which implements a *decision rule*. This decision rule is typically implemented by a subroutine which performs calculations to determine to which class the unknown is most likely to belong based on the measurements made.

As Fig. 1.1 illustrates, a machine vision system is really a fairly simple architectural structure. The details of each module may be quite complex, however, and many different options exist for designing the classifier and the feature measuring system. In this book, we mention the process of classifier design. However, the process of determining and measuring features is the principal topic of this book.

The "feature measurement" box can be further broken down into more detailed operations as illustrated in Fig. 1.2. At that level, the organization chart becomes more complex because the specific operations to be performed vary with the type of image and the objective of the tasks. Not every operation is performed in every application.

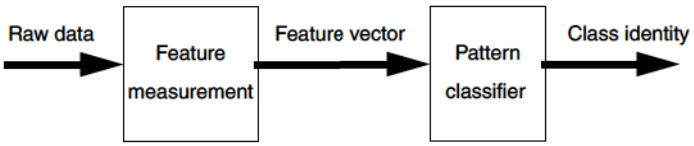

Fig. 1.1. Organization of a machine vision system.

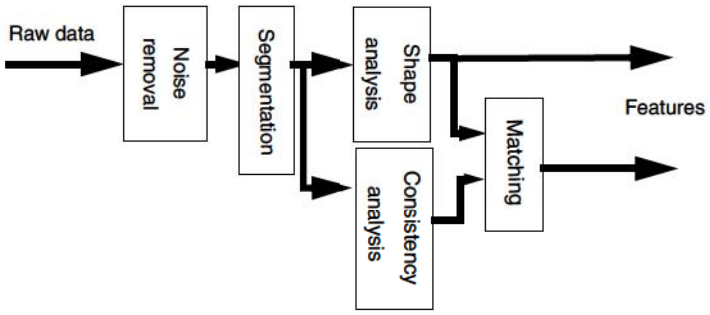

Fig. 1.2. Some components of a feature characterization system. Many machine vision applications do not use every block, and information often flows in other ways. For example, it is possible to perform matching directly on the image data.

1.5 The nature of images

We will pay much more attention to the nature of images in Chapter 4. We will observe that there are several different types of images as well as several different ways to represent images. The types of images include what we call "pictures," that is, two-dimensional images. In addition, however, we will discuss three-dimensional images and range images. We will also consider different representations for images, including iconic, functional, linear, and relational representations.

1.6 Images: Operations and analysis

Some equivalent words.

We will learn many different operations to perform on images. The emphasis in this course is "image analysis," or "computer vision," or "machine vision," or "image understanding." All these phrases mean the same thing. We are interested in making measurements on images with the objective of providing our machine (usually, but not always, a computer) with the ability to recognize what is in the image. This process includes several steps:

- *denoising* – all images are noisy, most are blurred, many have other distortions as well. These distortions need to be removed or reduced before any further operations can be carried out. We discuss two general approaches for denoising in Chapters 6 and 7.
- *segmentation* – we must segment the image into meaningful regions. Segmentation is covered in Chapter 8.
- *feature extraction* – making measurements, geometric or otherwise, on those regions is discussed in Chapter 9.

- *consistency* – interpreting the entire image from local measurements is covered in Chapters 10 and 11.
- *classification and matching* – recognizing the object is covered in Chapter 12 through Chapter 16.

So turn to the next chapter. (Did you notice? No homework assignments in this chapter? Don't worry. We'll fix that in future chapters.)

Reference

[1.1] C. Shu and R. Jain, "Vector Field Analysis for Oriented Patterns," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **16**(9), 1994.

2 Review of mathematical principles

Everything, once understood, is trivial

W. Snyder

2.1 A brief review of probability

Let us imagine a statistical experiment: rolling two dice. It is possible to roll any number between two and twelve (inclusive), but as we know, some numbers are more likely than others. To see this, consider the possible ways to roll a five.

We see from Fig. 2.1 that there are four possible ways to roll a five with two dice. Each event is **independent**. That is, the chance of rolling a two with the second die (1 in 6) does not depend at all on what is rolled with die number 1.

Independence of events has an important implication. It means that the *joint probability* of the two events is equal to the product of their individual probabilities and the conditional probabilities:

$$Pr(a|b)P(b) = Pr(a)Pr(b) = Pr(b|a)Pr(a) = Pr(a,b). \tag{2.1}$$

In Eq. (2.1), the symbols a and b represent **events**, e.g., the rolling of a six. $Pr(b)$ is the probability of such an event occurring, and $Pr(a \mid b)$ is the **conditional probability** of event a occurring, given that event b has occurred.

In Fig. 2.1, we tabulate all the possible ways of rolling two dice, and show the resulting number of different ways that the numbers from 2 to 12 can occur. We note that 6 different events can lead to a 7 being rolled. Since each of these events is equally probable (1 in 36), then a 7 is the most likely roll of two dice. In Fig. 2.2 the information from Fig. 2.1 is presented in graphical form.

In pattern classification, we are most often interested in the probability of a particular measurement occurring. We have a problem, however, when we try to plot a graph such as Fig. 2.2 for a continuously-valued function. For example, how do we ask the question: "What is the probability that a man is six feet tall?" Clearly, the answer is zero, for an infinite number of possibilities could occur (we might equally well ask, "What is the probability that a man is (exactly) 6.314 159 267 feet tall?"). Still, we know intuitively that the likelihood of a man being six feet tall is higher than the likelihood of his being ten feet tall. We need some way of quantifying this intuitive notion of likelihood.

Sum		Number of ways
0		0
1		0
2	1–1	1
3	2–1, 1–2	2
4	1–3, 3–1, 2–2	3
5	2–3, 3–2, 4–1, 1–4	4
6	1–5, 5–1, 2–4, 4–2, 3–3	5
7	3–4, 4–3, 2–5, 5–2, 1–6, 6–1	6
8	2–6, 6–2, 3–5, 5–3, 4–4	5
9	3–6, 6–3, 4–5, 5–4	4
10	4–6, 6–4, 5–5	3
11	6–5, 5–6	2
12	6–6	1

Fig. 2.1. The possible ways to roll two dice.

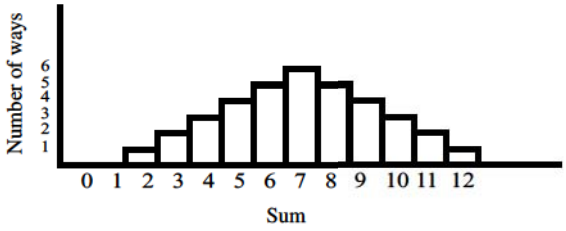

Fig. 2.2. The information of Fig. 2.1, in graphical form.

One question that does make sense is, "What is the probability that a man is **less than** six feet tall?" Such a function is referred to as a **probability distribution** function

$$P(x) = Pr(z < x) \tag{2.2}$$

for some measurement, z.

Fig. 2.3 illustrates the probability distribution function for the result of rolling two dice.

When we asked "what is the probability that a man is less than x feet tall?" we obtained the probability distribution function. Another well-formed question would be "what is the probability that a man's height is between x and $x + \Delta x$?" Such a question is easily answered in terms of the density function:

$$Pr(x \leq h < x + \Delta x) = Pr(h < x + \Delta x) - Pr(h < x) = P(x + \Delta x) - P(x)$$

Dividing by Δx and taking the limit as $\Delta x \to 0$, we see that we may define the **probability density function** as the derivative of the distribution function:

$$p(x) = \frac{d}{dx} P(x). \tag{2.3}$$

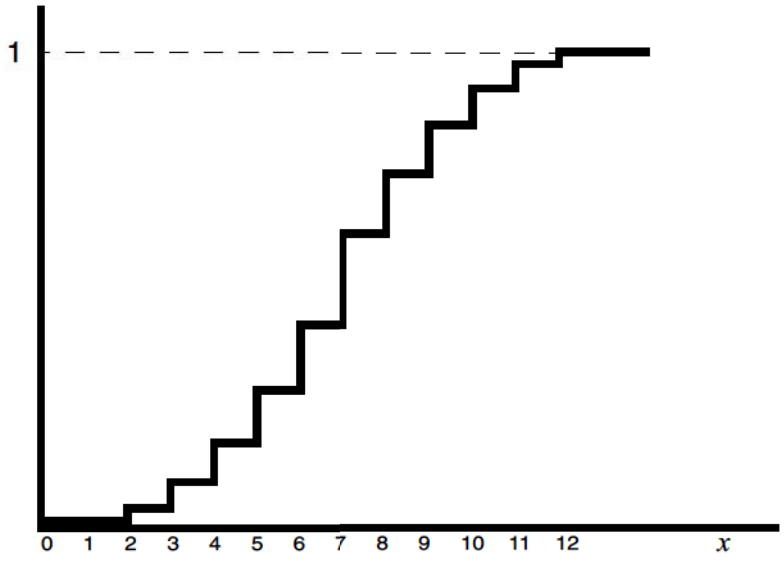

Fig. 2.3. The probability distribution of Fig. 2.2, showing the probability of rolling two dice to get a number LESS than *x*. Note that the curve is steeper at the more likely numbers.

$p(x)$ has all the properties that we desire. It is well defined for continuously-valued measurements and it has a maximum value for those values of the measurement which are intuitively most likely.

Furthermore:

$$\int_{-\infty}^{\infty} p(x)\,dx = 1, \tag{2.4}$$

which we must require, since **some** value will certainly occur.

2.2 A review of linear algebra

This section will serve more as a reference than a teaching aid, since you should know this material already.

In this section, we very briefly review vector and matrix operations. Generally, we denote vectors in boldface, scalars in lowercase Roman, and matrices in uppercase Roman.

Vectors are always considered to be column vectors. If we need to write one horizontally for the purpose of saving space in a document, we use transpose notation. For example, we denote a vector which consists of three scalar elements as:

$$\boldsymbol{v} = [x_1 \quad x_2 \quad x_3]^{\mathrm{T}}.$$

The inner product of two vectors is a scalar, $\boldsymbol{v} = \boldsymbol{a}^{\mathrm{T}}\boldsymbol{b}$. Its value is the sum of products

of the corresponding elements of the two vectors:

$$\boldsymbol{a}^{\mathrm{T}}\boldsymbol{b} = \sum_i a_i b_i.$$

You will also sometimes see the notation $\langle x, y \rangle$ used for inner product. We do not like this because it looks like an expected value of a random variable. One sometimes also sees the "dot product" notation $\boldsymbol{x} \cdot \boldsymbol{y}$ for inner product.

The magnitude of a vector is $|\boldsymbol{x}| = \sqrt{\boldsymbol{x}^{\mathrm{T}}\boldsymbol{x}}$. If $|\boldsymbol{x}| = 1, \boldsymbol{x}$ is said to be a "unit vector." If $\boldsymbol{x}^{\mathrm{T}}\boldsymbol{y} = 0$, then $\boldsymbol{x}$ and $\boldsymbol{y}$ are "orthogonal." If $\boldsymbol{x}$ and $\boldsymbol{y}$ are orthogonal unit vectors, they are "orthonormal."

The concept of orthogonality can easily be extended to continuous functions by simply thinking of a function as an infinite-dimensional vector. Just list all the values of $f(x)$ as x varies between, say, a and b. If x is continuous, then there are an infinite number of possible values of x between a and b. But that should not stop us – we cannot enumerate them, but we can still think of a vector containing all the values of $f(x)$. Now, the concept of summation which we defined for finite-dimensional vectors turns into integration, and an inner product may be written

$$\langle f(x), g(x) \rangle = \int_a^b f(x)g(x)\,dx. \tag{2.5}$$

The concepts of orthogonality and orthonormality hold for this definition of the inner product as well. If the integral is equal to zero, we say the two functions are orthogonal. So the transition from orthogonal vectors to orthogonal functions is not that difficult. With an infinite number of dimensions, it is impossible to visualize orthogonal as "perpendicular," of course, so you need to give up on thinking about things being perpendicular. Just recall the definition and use it.

Suppose we have n vectors $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots \boldsymbol{x}_n$; if we can write $\boldsymbol{v} = a_1\boldsymbol{x}_1 + a_2\boldsymbol{x}_2 + \cdots a_n\boldsymbol{x}_n$, then $\boldsymbol{v}$ is said to be a "linear combination" of $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots \boldsymbol{x}_n$.

A set of vectors $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots \boldsymbol{x}_n$ is said to be "linearly independent" if it is impossible to write any of the vectors as a linear combination of the others.

Given d linearly independent vectors, of d dimensions, $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots \boldsymbol{x}_d$ defined on $\Re^d$, then any vector $\boldsymbol{y}$ in the space may be written $\boldsymbol{y} = a_1\boldsymbol{x}_1 + a_2\boldsymbol{x}_2 + \cdots a_d\boldsymbol{x}_d$.

Since any d-dimensional real-valued vector $\boldsymbol{y}$ may be written as a linear combination of $\boldsymbol{x}_1, \ldots \boldsymbol{x}_d$, then the set $\{\boldsymbol{x}_i\}$ is called a "basis" set and the vectors are said to "span the space" $\Re^d$. Any linearly independent set of vectors can be used as a basis (necessary and sufficient). It is often particularly convenient to choose basis sets which are orthonormal.

For example, the following two vectors form a basis for $\Re^2$

$$\boldsymbol{x}_1 = [0 \quad 1]^{\mathrm{T}} \quad \text{and} \quad \boldsymbol{x}_2 = [1 \quad 0]^{\mathrm{T}}.$$

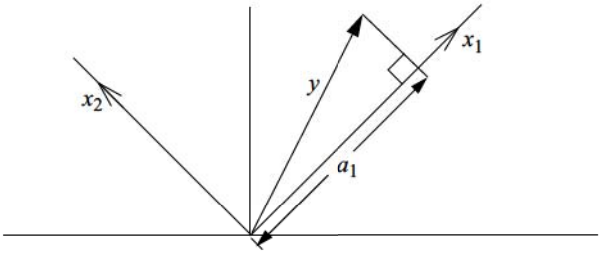

Fig. 2.4. x_1 and x_2 are orthonormal bases. The projection of y onto x_1 has length a_1.

This is the familiar Cartesian coordinate system. Here's another basis set for $\Re^2$

Is this set orthonormal?

$$\boldsymbol{x}_1 = [1 \quad 1]^{\mathrm{T}} \; \boldsymbol{x}_2 = [-1 \quad 1]^{\mathrm{T}}.$$

If $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots \boldsymbol{x}_d$ span $\Re^d$, and $\boldsymbol{y} = a_1\boldsymbol{x}_1 + a_2\boldsymbol{x}_2 + \cdots a_d\boldsymbol{x}_d$, then the "components" of $\boldsymbol{y}$ may be found by

$$a_i = \boldsymbol{y}^{\mathrm{T}}\boldsymbol{x}_i \tag{2.6}$$

and a_i is said to be the "projection" of $\boldsymbol{y}$ onto $\boldsymbol{x_i}$. In a simple Cartesian geometric interpretation, the inner product of Eq. (2.6) is literally a projection as illustrated in Fig. 2.4. However, whenever Eq. (2.6) is used, the term "projection" may be used as well, even in a more general sense (e.g. the coefficients of a Fourier series).

The only vector spaces which concern us here are those in which the vectors are real-valued.

2.2.1 Linear transformations

What does this say about m and d?

A "linear transformation," A, is simply a matrix. Suppose A is $m \times d$. If applied to a vector $\boldsymbol{x} \in \Re^d$, $\boldsymbol{y} = A\boldsymbol{x}$, then $\boldsymbol{y} \in \Re^m$. So A took a vector from one vector space, $\Re^d$, and produced a vector in $\Re^m$. If that vector $\boldsymbol{y}$ could have been produced by applying A to one and only one vector in $\Re^d$, then A is said to be "one-to-one." Now suppose that there are no vectors in $\Re^m$ that can not be produced by applying A to some vector in $\Re^d$. In that case, A is said to be "onto." If A is one-to-one and onto, then A^{-1} exists. Two matrices A and B are "conformable" if the matrix multiplication $C = AB$ makes sense.

We assume you know the meanings of transpose, inverse, determinant, and trace. If you do not, look them up.

Some important (and often forgotten) properties: If A and B are conformable, then

$$(AB)^{\mathrm{T}} = B^{\mathrm{T}}A^{\mathrm{T}} \tag{2.7}$$

and

$$(AB)^{-1} = B^{-1}A^{-1} \tag{2.8}$$

if A and B are invertible at all.

A couple of other useful properties are

$$\det(AB) = \det(BA) \quad \text{and} \quad \text{tr}(AB) = \text{tr}(BA)$$

which only is true, of course, if A and B are square. If a matrix A satisfies

$$AA^{\mathrm{T}} = A^{\mathrm{T}}A = I \tag{2.9}$$

then obviously, the transpose of the matrix is the inverse as well, and A is said to be an "orthonormal transformation" (OT), which will correspond geometrically to a rotation. If A is a $d \times d$ orthonormal transformation, then the columns of A are orthonormal, linearly independent, and form a basis spanning the space of $\Re^d$. For $\Re^3$, three convenient OTs are the rotations about the Cartesian axes:

Some example orthonormal transformations.

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} R_y = \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix} R_z = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Suppose R is an OT, and $\boldsymbol{y} = R\boldsymbol{x}$, then

$$|\boldsymbol{y}| = |\boldsymbol{x}|. \tag{2.10}$$

A matrix A is "positive definite" if

$$\boldsymbol{y} = \boldsymbol{x}^{\mathrm{T}}A\boldsymbol{x} > 0 \; \forall \boldsymbol{x} \in \Re^d, \boldsymbol{x} \neq 0$$

$\boldsymbol{x}^{\mathrm{T}}A\boldsymbol{x}$ is called a *quadratic form*.

The derivative of a quadratic form is particularly useful:

What happens here if A is symmetric?

$$\frac{d}{dx}(\boldsymbol{x}^{\mathrm{T}}A\boldsymbol{x}) = (A + A^{\mathrm{T}})\boldsymbol{x}.$$

Since we mentioned derivatives, we might as well mention a couple of other vector calculus things:

Suppose f is a scalar function of $\boldsymbol{x}$, $\boldsymbol{x} \in \Re^d$, then

$$\frac{df}{d\boldsymbol{x}} = \left[\frac{\partial f}{\partial x_1} \quad \frac{\partial f}{\partial x_2} \quad \cdots \quad \frac{\partial f}{\partial x_d}\right]^{\mathrm{T}}, \tag{2.11}$$

and is called the "gradient." This will be often used when we talk about edges in images, and $f(\boldsymbol{x})$ will be the brightness as a function of the two spatial directions.

If $\boldsymbol{f}$ is vector-valued, then the derivative is a matrix

$$\frac{d\boldsymbol{f}^{\mathrm{T}}}{d\boldsymbol{x}} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_2}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_1} \\ \cdots & \cdot & \cdots & \cdots \\ \cdots & \cdot & \cdots & \cdots \\ \cdots & \cdot & \cdots & \cdots \\ \frac{\partial f_1}{\partial x_d} & \frac{\partial f_2}{\partial x_d} & \cdots & \frac{\partial f_m}{\partial x_d} \end{bmatrix}, \tag{2.12}$$

and is called the "Jacobian."

One more: If f is scalar-valued, the matrix of second derivatives

$$\begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_d} \\ \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial^2 f}{\partial x_d \partial x_1} & \frac{\partial^2 f_2}{\partial x_d \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_d^2} \end{bmatrix} \tag{2.13}$$

is called the "Hessian."

2.2.2 Derivative operators

Here, we introduce a new notation, a vector containing only derivative operators,

$$\nabla = \begin{bmatrix} \frac{\partial}{\partial x_1} & \frac{\partial}{\partial x_2} & \cdots & \frac{\partial}{\partial x_d} \end{bmatrix}^{\mathrm{T}}. \tag{2.14}$$

It is important to note that this is an OPERATOR, not a vector. We will do linear algebra sorts of things with it, but by itself, it has no value, not even really any meaning – it must be applied to something to have any meaning. For most of this book, we will deal with two-dimensional images, and with the two-dimensional form of this operator,

$$\nabla = \begin{bmatrix} \frac{\partial}{\partial x} & \frac{\partial}{\partial y} \end{bmatrix}^{\mathrm{T}}. \tag{2.15}$$

Apply this operator to a scalar, f, and we get a vector which does have meaning, the gradient of f:

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x} & \frac{\partial f}{\partial y} \end{bmatrix}^{\mathrm{T}}. \tag{2.16}$$

Similarly, we may define the divergence using the inner (dot) product (in all the following definitions, only the two-dimensional form of the del operator defined in

Eq. (2.16) is used. However, remember that the same concepts apply to operators of arbitrary dimension):

$$\text{div}\boldsymbol{f} = \nabla \boldsymbol{f} = \begin{bmatrix} \frac{\partial}{\partial x} & \frac{\partial}{\partial y} \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \end{bmatrix} = \frac{\partial f_1}{\partial x} + \frac{\partial f_2}{\partial y}. \tag{2.17}$$

We will also have opportunity to use the outer product of the del operator with a matrix:

$$\nabla \times \boldsymbol{f} = \begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{bmatrix} [f_1 \quad f_2] = \begin{bmatrix} \frac{\partial f_1}{\partial x} & \frac{\partial f_2}{\partial x} \\ \frac{\partial f_1}{\partial y} & \frac{\partial f_2}{\partial y} \end{bmatrix}. \tag{2.18}$$

2.2.3 Eigenvalues and eigenvectors

If matrix A and vector $\boldsymbol{x}$ are conformable, then one may write the "characteristic equation"

$$A\boldsymbol{x} = \lambda \boldsymbol{x}, \lambda \in \Re. \tag{2.19}$$

Since $A\boldsymbol{x}$ is a linear operation, A may be considered as mapping $\boldsymbol{x}$ onto itself with only a change in length. There may be more than one "eigenvalue[1]" λ, which satisfies Eq. (2.19). For $\boldsymbol{x} \in \Re^d$, A will have exactly d eigenvalues (which are not, however, necessarily distinct). These may be found by solving $\det(A - \lambda I) = 0$. (But for $d > 2$, we do not recommend this method. Use a numerical package instead.)

For any given matrix, there are only a few eigenvalue/eigenvector pairs.

Given some eigenvalue λ, which satisfies Eq. (2.19), the corresponding $\boldsymbol{x}$ is called the corresponding "eigenvector."

2.3 Introduction to function minimization

In this book, essentially EVERY machine vision topic will be discussed in terms of some sort of minimization, so get used to it!

Minimization of functions is a pervasive element of engineering: One is always trying to find the set of parameters which minimizes some function of those parameters. Notationally, we state the problem as: Find the vector $\mathbf{x}$ which produces a minimum of some function $H(\mathbf{x})$:

$$\widehat{H} = \min_{\mathbf{x}} H(\mathbf{x}) \tag{2.20}$$

where $\mathbf{x}$ is some d-dimensional parameter vector, and H is a scalar function of $\mathbf{x}$, often referred to as an "objective function." We denote the $\mathbf{x}$ which results in the

[1] "Eigen-" is the German prefix meaning "principal" or "most important." These are NOT named for Mr Eigen.

minimal H as $\mathbf{x}$

$$\mathbf{x} = \arg\min_{\mathbf{x}} H(\mathbf{x}). \tag{2.21}$$

The authors get VERY annoyed at improper use of the word "optimal." If you didn't solve a formal optimization problem to get your result, you didn't come up with the "optimal" anything.

The most straightforward way to minimize a function is to set its derivative to zero:

$$\nabla H(\mathbf{x}) = 0, \tag{2.22}$$

where ∇ is the gradient operator – the set of partial derivatives. Eq. (2.22) results in a set of equations, one for each element of $\mathbf{x}$, which must be solved simultaneously:

$$\begin{aligned} \frac{\partial}{\partial x_1} H(\mathbf{x}) &= 0 \\ \frac{\partial}{\partial x_2} H(\mathbf{x}) &= 0 \\ &\cdots \\ \frac{\partial}{\partial x_d} H(\mathbf{x}) &= 0. \end{aligned} \tag{2.23}$$

Such an approach is practical only if the system of Eq. (2.23) is solvable. This may be true if $d = 1$, or if H is at most quadratic in $\mathbf{x}$.

EXERCISE

Find the vector $\boldsymbol{x} = [x_1, x_2, x_3]^{\mathrm{T}}$ which minimizes

$$H = ax_1^2 + bx_1 + cx_2^2 + dx_3^2$$

where a, b, c, and d are known constants.

Solution

$$\begin{aligned} \frac{\partial H}{\partial x_1} &= 2ax_1 + b \\ \frac{\partial H}{\partial x_2} &= 2cx_2 \\ \frac{\partial H}{\partial x_3} &= 2dx_3 \end{aligned}$$

minimized by

$$x_3 = x_2 = 0, x_1 = \frac{-b}{2a}.$$

If H is some function of order higher than two, or is transcendental, the technique of setting the derivative equal to zero will not work (at least, not in general) and we must resort to numerical techniques. The first of these is gradient descent.

In one dimension, the utility of the gradient is easy to see. At a point $x^{(k)}$ (Fig. 2.5), the derivative points **AWAY FROM** the minimum. That is, in one dimension, its sign will be positive on an "uphill" slope.

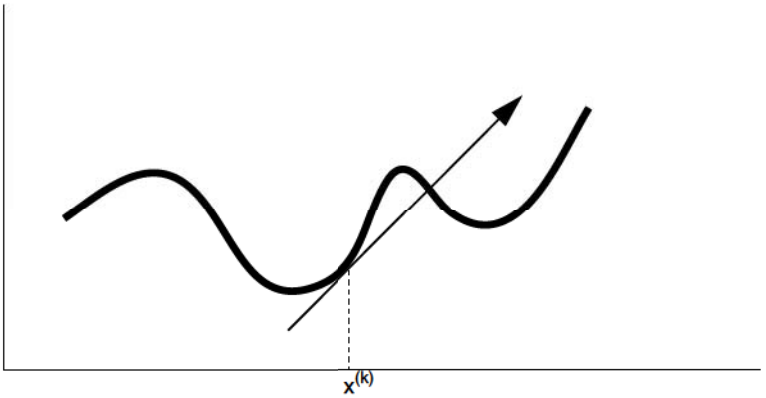

Fig. 2.5. The sign of the derivative is always away from the minimum.

Thus, to find a new point, x^{k+1}, we let

$$x^{(k+1)} = x^{(k)} - \alpha \left.\frac{\partial H}{\partial x}\right|_{x^{(k)}} \tag{2.24}$$

where α is some "small" constant.

In a problem with d variables, we write

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \alpha \nabla H(\mathbf{x})|_{\mathbf{x}^{(k)}}. \tag{2.25}$$

2.3.1 Newton–Raphson

It is not immediately obvious in Eq. (2.25) how to choose the variable α. If α is too small, the iteration of Eq. (2.25) will take too long to converge. If α is too large, the algorithm may become unstable and never find the minimum.

We can find an estimate for α by considering the well-known Newton–Raphson method for finding roots: (In one dimension), we expand the function $H(x)$ in a Taylor series about the point $x^{(k)}$ and truncate, assuming all higher order terms are zero,

$$H\big(x^{(k+1)}\big) = H\big(x^{(k)}\big) + \big(x^{(k+1)} - x^{(k)}\big)H'\big(x^{(k)}\big).$$

Since we want $x^{(k+1)}$ to be a zero of H, we set

$$H\big(x^{(k)}\big) + \big(x^{(k+1)} - x^{(k)}\big)H'\big(x^{(k)}\big) = 0, \tag{2.26}$$

and find that to estimate a root, we should use

$$x^{(k+1)} = x^{(k)} - \frac{H\big(x^{(k)}\big)}{H'\big(x^{(k)}\big)}. \tag{2.27}$$

In optimization however, we are not finding roots, but rather, we are minimizing a function, so how does knowing how to find roots help us? The minima of the function are the roots of its derivative, and our algorithm becomes

Algorithm: Gradient descent

$$x^{(k+1)} = x^{(k)} - \frac{H'\left(x^{(k)}\right)}{H''\left(x^{(k)}\right)}. \tag{2.28}$$

In higher dimensions, Equation (2.28) becomes

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \mathbf{H}^{-1}\nabla H, \tag{2.29}$$

where $\mathbf{H}$ is the Hessian matrix of second derivatives, which we mentioned earlier in this chapter:

$$\mathbf{H} = \left[\frac{\partial^2}{\partial x_i \partial x_j} H(\mathbf{x})\right]. \tag{2.30}$$

EXAMPLE

Given a set of x, y data pairs $\{(x_i, y_i)\}$ and a function of the form

$$y = ae^{bx}, \tag{2.31}$$

find the parameters a and b which minimize

$$H(a, b) = \sum_i (y_i - ae^{bx_i})^2. \tag{2.32}$$

Solution

We can solve this problem with the linear approach by observing that $\ln y = \ln a + bx$ and re-defining variables $g = \ln y$ and $r = \ln a$.
With these substitutions, Eq. (2.32) becomes

$$H(r, b) = \sum_i (g_i - r - bx_i)^2 \tag{2.33}$$

$$\frac{\partial H}{\partial b} = 2\sum_i (g_i - r - bx_i)(-x_i) \tag{2.34}$$

$$\frac{\partial H}{\partial r} = 2\sum_i (g_i - r - bx_i)(-1). \tag{2.35}$$

Setting Eq. (2.34) to zero, we have

$$\sum_i g_i x_i - \sum_i r x_i - \sum_i b x_i^2 = 0 \tag{2.36}$$

or

$$r\sum_i x_i + b\sum_i x_i^2 = \sum_i g_i x_i \tag{2.37}$$

and from Eq. (2.35)

$$\sum_i g_i - r\sum_i 1 - b\sum_i x_i = 0 \tag{2.38}$$

or

$$Nr + b\sum_i x_i = \sum_i g_i \tag{2.39}$$

where N is the number of data points. Eqs. (2.37) and (2.39) are two simultaneous linear equations in two unknowns which are readily solved. (See [2.2, 2.3, 2.4] for more sophisticated descent techniques such as the conjugate gradient method.)

2.3.2 Local vs. global minima

Gradient descent suffers from a serious problem: Its solution is strongly dependent on the starting point. If started in a "valley," it will find the bottom of *that* valley. We have no assurance that this particular minimum is the lowest, or "global," minimum.

Before continuing, we will find it useful to distinguish two kinds of nonlinear optimization problems.

- **Combinatorial optimization**. In this case, the variables have discrete values, typically 0 and 1. With $\mathbf{x}$ consisting of d binary-valued variables, 2^d possible values exist for $\mathbf{x}$. Minimization of $H(\mathbf{x})$ then (in principle) consists of simply generating each possible value for $\mathbf{x}$ and consequently of $H(\mathbf{x})$, and choosing the minimum. Such "exhaustive search" is in general not practical due to the exponential explosion of possible values. We will find that simulated annealing provides an excellent approach to solving combinatorial optimization problems.
- **Image optimization**. Images have a particular property: Each pixel is influenced only by its neighborhood (this will be explained in more detail later), however, the pixel values are continuously-valued, and there are typically many thousand such variables. We will find that mean field annealing is most appropriate for the solution of these problems.

2.3.3 Simulated annealing

We will base much of the following discussion of minimization techniques on an algorithm known as "simulated annealing" (SA) which proceeds as follows. (See the book by Aarts and Van Laarhoven for more detail [2.1].)

Algorithm: Simulated annealing

Choose (at random) an initial value of $\mathbf{x}$, and an initial value of $T > 0$.

While $T > T_{\min}$, do

(1) Generate a point $\mathbf{y}$ which is a neighbor of $\mathbf{x}$. (The exact definition of neighbor will be discussed soon.)

(2) If $H(\boldsymbol{y}) < H(\boldsymbol{x})$ then replace $\mathbf{x}$ with $\boldsymbol{y}$.

(3) Else compute $P_{\mathbf{y}} = \exp(-\frac{(H(\mathbf{y})-H(\mathbf{x}))}{T})$. If $P_{\mathbf{y}} \geq R$ then replace $\mathbf{x}$ with $\boldsymbol{y}$, where R is a random number uniformly distributed between 0 and 1.
(4) Decrease T slightly and go to step 1.

How simulated annealing works

Simulated annealing is most easily understood in the context of combinatorial optimization. In this case, the "neighbor" of a vector $\mathbf{x}$ is another vector $\mathbf{x}_2$, such that only one of the elements of $\mathbf{x}$ is changed (discretely) to create $\mathbf{x}_2$.[2] Thus, if $\mathbf{x}$ is binary and of dimension d, one may choose a neighboring $\mathbf{y} = \mathbf{x} \oplus \mathbf{z}$, where $\mathbf{z}$ is a binary vector in which exactly one element is nonzero, and that element is chosen at random, and $\oplus$ represents exclusive OR.

In step 2 of the algorithm, we perform a descent. Thus we "always fall down hill."

In step 3, we provide a mechanism for sometimes making uphill moves. Initially, we ignore the parameter T and note that if $\mathbf{y}$ represents an uphill move, the probability of accepting $\mathbf{y}$ is proportional to $e^{-(H(\mathbf{y})-H(\mathbf{x}))}$. Thus, uphill moves can occur, but are exponentially less likely to occur as the size of the uphill move becomes larger. The likelihood of an uphill move is, however, strongly influenced by T. Consider the case that T is very large. Then $\frac{H(\mathbf{y})-H(\mathbf{x})}{T} \ll 1$ and $P_{\mathbf{y}} \approx 1$. Thus, all moves will be accepted. As T is gradually reduced, uphill moves become gradually less likely until for low values of $T (T \ll (H(\mathbf{y}) - H(\mathbf{x})))$, such moves are essentially impossible.

One may consider an analogy to physical processes in which the state of each variable (one or zero) is analogous to the spin of a particle (up or down). At high temperatures, particles randomly change state, and if temperature is gradually reduced, minimum energy states are achieved. The parameter T in step 4 is thus analogous to (and often referred to as) temperature, and this minimization technique is therefore called "simulated annealing."

2.4 Markov models

A Markov process is most easily described in terms of time, although in machine vision we are primarily interested in interactions over spatial distances. The concept is that the probability of something happening is dependent on a thing that *just recently* happened. We will use Markov processes primarily in the noise removal and segmentation issues in Chapter 6, however, they find applications in a wide variety of problems, including character recognition [16.1].

We start by introducing the simplest kind of Markov model, the Markov chain. This type of model is appropriate whenever a sequence of things can be identified,

[2] Thus the set of neighbors of $\mathbf{x}$ consists of all $\mathbf{x}$'s of Hamming distance = 1.

for example, when a string of symbols is received over a computer network, or a sequence of words is input to a natural language processor. Let the symbols which are received be denoted by $y(t)$, where the argument t denotes the (discrete) time instant at which that symbol is received. Thus, $y(1)$ is received before $y(2)$. Let w denote the class to which a symbol belongs, $w \in \{w_1, w_2, \ldots w_c\}$, if there are c possible classes. As an example the signals could represent 0 or 1, as in a communications system.

We are interested in the probability that $y(t)$ belongs to some particular class. For example, what is the probability that the kth symbol will be a one? We are interested in that probability as a function of history. For example, let w_1 represent the class 1. We formalize our previous discussion by asking what is the probability that $y(t)$ is a 1, given the last N symbols received:

DONT PANIC! It's just notation, and it is explained, right after the equation.

$$P(y(t) \in w_1 | y(t-1) \in w_{t-1}, y(t-2) \in w_{t-2}, \ldots, y(t-N) \in w_{t-N}). \quad (2.40)$$

This is an awkward bit of notation, but here is what it means. When you see the term "$y(t)$" think "the symbol received at time t." When you see the "$\in$," think "is." When you see "w_k" think "1" or think "0" or think "whatever might be received at time k." For example, we might ask "what is the probability that you receive at time k the symbol 1, when the last four symbols received previously were 0110?" Which, in our notation is to ask what is

$$P(y(k) \in w_1 | (y(k-1) \in w_0, y(k-2) \in w_1, y(k-3) \in w_1, y(k-4) \in w_0)).$$

It is possible that in order to compute this probability, we must know all of the history, or it is possible that we need only know the class of the last few symbols. One particularly interesting case is when we need only know the class of the last symbol received. In that case, we could say that the probability of class assignments for symbol $y(t)$, given all of the history, is precisely the same as the probability knowing only the last symbol:

$$P(y(k)|y(k-1)) = P(y(k)|(y(k-1), y(k-2), \ldots)) \quad (2.41)$$

where we have simplified the notation slightly by omitting the set element symbols. That is, $y(k)$ does not denote the fact that the kth symbol was received, but rather that the kth symbol belongs to some particular class. If this is the case – that the probability conditioned on all of history is identical to the probability conditioned on the last symbol received – we refer to this as a *Markov process*.[3]

This relationship implies that

$$P(y(N) \in w_N, \ldots y(1) \in w_1) = \left\{ \prod_{t=2}^{N} P(y(t) \in w_t | (y(t-1) \in w_{t-1})) \right\} P(y(1) \in w_1).$$

[3] To be perfectly correct, this is a *first-order* Markov process, but we will not be dealing with any other types in this chapter.

Suppose there are only two classes possible, say 0 and 1. Then we need to know only four possible "transition probabilities," which we define using subscripts as follows:

$$P(y(t) = 0|y(t-1) = 0) \equiv P_{00}$$
$$P(y(t) = 0|y(t-1) = 1) \equiv P_{01}$$
$$P(y(t) = 1|y(t-1) = 0) \equiv P_{10}$$
$$P(y(t) = 1|y(t-1) = 1) \equiv P_{11}.$$

In general, there could be more than two classes, so we denote the transition probabilities by P_{ij}, and can therefore describe a Markov chain by a $c \times c$ matrix $\mathbf{P}$ whose elements are P_{ij}.

We will take another look at Markov processes when we think about Markov random fields in Chapter 6.

Assignment 2.1

Is the matrix **P** symmetric? Why or why not? Does **P** have any interesting properties? Do its rows (or columns) add up to anything interesting?

2.4.1 Hidden Markov models

Hidden Markov models (HMMs) occur in many applications, including in particular recent research in speech recognition. In a hidden Markov model, we assume that there may be more than one transition matrix, and that there is an unmeasurable (hidden) process which switches between transition matrices. Furthermore, that switching process itself may be statistical in nature, and we normally assume it is a Markov process. This is illustrated in Fig. 2.6, where the position of the switch determines whether the output $y(t)$ is connected to the output of Markov Process 1 or Markov Process 2. The switch may be thought of as being controlled by a finite

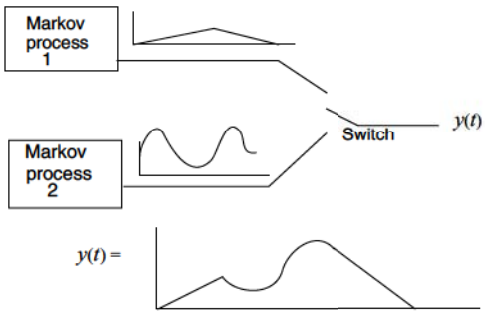

Fig. 2.6. A hidden Markov model may be viewed as a process which switches randomly between two signals.

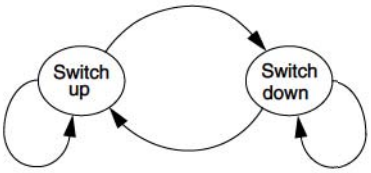

Fig. 2.7. The finite state machine of switches.

state machine (FSM), which at each time instant may stay in the same state or may switch, as shown in Fig. 2.7.

Here is our problem: We observe a sequence of symbols

$$Y = [y(t = 1), y(t = 2), \ldots] = [y(1), y(2), \ldots].$$

What can we infer? The transition probabilities? The state sequence? The structure of the FSM? The rules governing the FSM? Let's begin by estimating the state sequence.

Estimating the state sequence

Let $s(t)\, t = 1, \ldots, N$ denote the state associated with measurement $y(t)$, and denote the sequence of states $S = [s(1), s(2), \ldots s(N)]$, where each $s(t) \in \{s_1, s_2, \ldots s_m\}$. We seek a sequence of states, S, which maximizes the conditional probability that the sequence is correct, given the measurements; $P(S|Y)$.

Using Bayes' rule

$$P(S|Y) = \frac{p(Y|S)P(S)}{p(Y)}. \tag{2.42}$$

We assume the states form a Markov chain, so

$$P(S) = \left[\prod_{t=2}^{N} P_{s(t),s(t-1)}\right] P_{s(0)}. \tag{2.43}$$

Now, let's make a temporarily unbelievable assumption, that the probability density of the output depends only on the state. Denote that relationship by $p(y(t)|s(t))$. Then the posterior conditional probability of the sequence can be written:

$$p(Y|S)P(S) = \left[\prod_{t=1}^{N} p(y(t)|s(t))\right]\left[\prod_{t=2}^{N} P_{s(t),s(t-1)}\right] P_{s(0)}. \tag{2.44}$$

Define $P_{s(1),s(0)} \equiv P_{s(0)}$, and Eq. (2.44) simplifies to

$$p(Y|S)P(S) = \prod_{t=1}^{N} p(y(t)|s(t))P_{s(t),s(t-1)}. \tag{2.45}$$

Now look back at Eq. (2.42). The choice of S does not affect the denominator, so all we need to do is find the sequence S which maximizes

$$E = \prod_{t=1}^{N} p(y(t)|s(t))P_{s(t),s(t-1)}. \tag{2.46}$$

Surprisingly, there is an algorithm which will solve this maximization problem. It is called the Viterbi algorithm. It has many, many applications. We explain it in the next section, since this seems like a good place to motivate it.

2.4.2 The Viterbi algorithm

This is used for a particular kind of optimization problem, one where each state $s(t)$ has only two neighboring states, $s(t+1)$ and $s(t-1)$. Imposing this "neighboring state" requirement allows us to use an efficient algorithm.

First, it is (almost) always easier to work with a sum than with a product, so let's define a new objective function by taking logs.

$$L \equiv \ln E \equiv \sum_{t=1}^{N} (\Psi(t) + \Theta_{i,j}) \tag{2.47}$$

where $\Psi(t) = \ln p(y(t)|s(t))$ and $\Theta_{i,j} = \ln P_{i,j}$.

Pictorially, we illustrate the set of all possible sequences as a graph as illustrated in Fig. 2.8.

A particular sequence of states describes a *path* through this graph. For example, for a graph like this with $N = 4$, $m = 3$, the path $[s_1, s_2, s_3, s_1]$ is illustrated in Fig. 2.9.

A path like this implies a set of values for the functions. For each node in the graph, we associate a value of Ψ. Suppose we measured $y(1) = 2$, $y(2) = 1$, $y(3) = 2.2$,

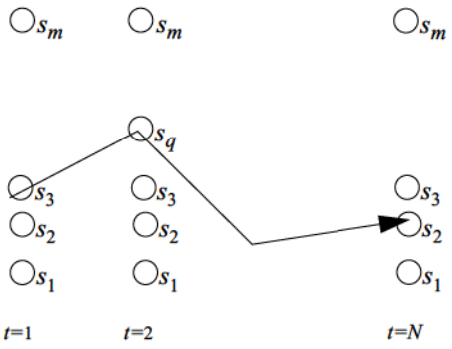

Fig. 2.8. Every possible sequence of states can be thought of as a path through such a graph.

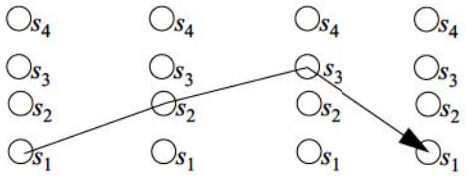

Fig. 2.9. A path through a problem with four states and four time values.

$y(4) = 1$. The function Ψ has value

$$\Psi = \ln p(y(1) = 2|s(1) = s_1) + \ln p(y(2) = 1|s(2) = s_2) + \ln p(y(3) = 2.2|s(3) = s_3) + \ln p(y(4) = 1|s(4) = s_1).$$

There is a value associated with each edge in the graph as well, the function Θ determined by the associated transition probability. So every possible path through the graph has a corresponding value of the objective function L. We describe the algorithm to find the best path as an induction: Suppose at time t we have already found the best path to each node, with cost denoted $\mathrm{LB}_i(t)$, $i = 1, \ldots, m$. Then we can compute the cost of going from each node at time t to each node at time $t + 1$ (m^2 calculations) by

$$L'_{ij}(t + 1) = LB_i(t) + \Psi(y(t + 1)|s_j(t + 1)) + \Theta_{i,j}. \tag{2.48}$$

The best path to node j at time $t + 1$ is the maximum of these. When we finally reach time step N, the node which terminates the best path is the final node.

The computational complexity of this algorithm is thus Nm^2, which is a **lot** less than m^N, the complexity of a simple exhaustive search of all possible paths.

2.4.3 Markov outputs

In the description above, we assumed that the probability of a particular output depended only on the state. We do not have to be that restrictive; we could allow the outputs themselves to be the Markov processes.

Assume if the state changes, the first output depends only on the state, as before. But afterward, if the state remains the same, the outputs obey a Markov chain. We can formulate this problem in the same way, and solve it with the Viterbi algorithm.

2.4.4 Estimating model parameters

One final detail still eludes us: Given an observation Y, how do we estimate the conditional output probabilities $p(y(k)|s(k))$, and those transition probabilities for the states, $P_{i,j}$?

To do this, first, we need to go from a continuous to a discrete representation for the values of $y(t)$, in order to be able to use probabilities rather than densities. Assume $y(t) \in \{y_1, y_2, \ldots y_r\}$. Then, we may define an output probability matrix $\Pi = [\pi_{k,l}], k = 1, \ldots m; l = 1, \ldots r$, which represents the probability of observing output y_l if in state s_k.

Define

$$P_{i,j|Y}(t) = Pr((s(t-1) = i, s(t) = j)|Y). \tag{2.49}$$

That is, given the observation sequence, what is the probability that we went from state i to state j at time t? We can compute that quantity using the methods of section 2.4.2 if we know the transition probabilities $P_{i,j}$ and the output probabilities $\pi_{k,l}$. Suppose we do know those. Then, we estimate the transition probability by averaging the probabilities over all the inputs.

$$P_{i,j} = \frac{\sum_{t=2}^{N} P_{i,j|Y}(t)}{\sum_{t=2}^{N} P_{j|Y}(t)} \tag{2.50}$$

where, since in order to go into state j, the system had to go there from somewhere,

$$P_{j|Y}(t) = \sum_{i=1}^{N} P_{ij|Y}(t). \tag{2.51}$$

Then we estimate the probability of the observation by again averaging all the observations.

$$\pi_{k,l} = \frac{\sum_{t=1,y(t)=j}^{N} P_{i|Y}(t)}{\sum_{t=1}^{N} P_{i|Y}(t)}. \tag{2.52}$$

At each iteration, we use Eqs. (2.50) and (2.52) to update the parameters. We then use Eqs. (2.49) and (2.51) to update the conditional probabilities. The process then repeats until it converges.

2.4.5 Applications of HMMs

Hidden Markov models have found many applications in speech recognition and document content recognition [17.29].

Assignment 2.2

(Trivia question) In what novel did a character named Markov Chaney occur?

Assignment 2.3

Find the OT corresponding to a rotation of 30° about the z axis. Prove the columns of the resulting matrix are a basis for $\Re^3$.

Assignment 2.4

Prove Eq. (2.10). Hint: Use Eq. (2.7); Eq. (2.9) might be useful as well.

Assignment 2.5

A positive definite matrix has positive eigenvalues. Prove this. (For that matter, is it even true?)

Assignment 2.6

Does the function $y = xe^{-x}$ have a unique value x which minimizes y? If so, can you find it by taking a derivative and setting it equal to zero? Suppose this problem requires gradient descent to solve. Write the algorithm you would use to find the x which minimizes y.

Assignment 2.7

We need to solve a minimization problem using gradient descent. The function we are minimizing is $\sin x + \ln y$. Which of the following is the expression for the gradient which you need in order to do gradient descent?

$$\text{(a)}\ \cos x + \frac{1}{y} \quad \text{(b)}\ y = -\frac{1}{\cos x} \quad \text{(c)}\ -\infty$$

$$\text{(d)}\ \begin{bmatrix} \cos x \\ 1/y \end{bmatrix} \quad \text{(e)}\ \frac{\partial}{\partial y}\sin x + \frac{\partial}{\partial x}\ln y$$

Assignment 2.8

(a) Write the algorithm which uses gradient descent to find the vector $[x, y]^T$ which minimizes the function

$z = x\exp(-(x^2 + y))$. (b) Write a computer program which finds the minimizing x, y pair.

Assignment 2.9

Determine whether the functions $\sin x$ and $\sin 2x$ might be orthonormal or orthogonal functions.

References

[2.1] E.H.L. Aarts and P.J.M. van Laarhoven. *Simulated Annealing: Theory and Applications*, Dordrecht, Holland, Reidel, 1987.

[2.2] R.L. Burden, J.D. Faires, and A.C. Reynolds. *Numerical Analysis*, Boston, MA, Prindle, Weber and Schmidt, 1981.

[2.3] G. Dahlquist and A. Bjorck. *Numerical Methods*, Englewood Cliffs, NJ, Prentice-Hall, 1974.

[2.4] B. Gottfried and J. Weisman. *Introduction to Optimization Theory*, Englewood Cliffs, NJ, Prentice-Hall, 1973.