Unsupervised Machine Learning

Lecture notes

Aapo Hyvärinen
University of Helsinki

Exercices and solutions by:
Michael Gutmann and Doris Entner

Version for spring semester 2015

Contents

Preface

These are the lecture notes for the course Unsupervised Machine Learning, lectured by Aapo Hyvärinen at the University of Helsinki. These lecture notes contain all the material in the lectures (except for parts of some computer demos) and are hopefully suitable for self-study as well. Exercices with solutions are given in the appendix. Computer assignments are given separately, and they also contain some essential material.

Acknowledgments

Thanks to Olli Tapiola for helping with the latexing of the exercice solutions, and Henrik Nyman for extensive corrections on the manuscript.

Chapter 1

Introduction

Machine learning typically means a situation where a computer or a robot learns from its experiences and observations of the outside world. The goal of such learning is behaviour which one would call "intelligent", although intelligence is notoriously difficult to define precisely.

It is widely believed that learning is necessary for developing an intelligent system. It seems to be too difficult to program intelligent behaviour in detail, and it is believed that letting the system learn from its experience and observing the world makes things easier. The outside world contains implicitly a huge amount of information, and if the system can successfully extract that information (or rather, as some would say, it is able to extract knowledge from that information), it may be able to learn complex intelligent behaviour based on some relatively simple learning rules.

Learning regularities in the environment is closely related to the goals of statistical science, and that's why modern machine learning is heavily using the theory of statistical estimation and modelling. In fact, this course is at the same time a course in statistics, where it might be called *computational data analysis* or, historically, *multivariate statistics*.

Machine learning is usually divided into three parts:

- Supervised learning: we observe input x and output y and the goal is to learn the connection. This includes such topics as regression, prediction, and classification.
- Reinforcement learning: we have an agent which is in state s, takes action a, receives reward r (or none). The goal is to learn to take action a in state s to maximize reward. This is rather different from classical statistical theory.
- Unsupervised learning: Observe only x, nothing else. This is closely related to topics such as exploratory data analysis and data mining. We don't have separate "inputs" and "outputs", just a lot of observations of one variable or vector, x.

Some goals of unsupervised learning are the following:

- Visualization: You have 100 variables and 1000 observations. How to plot the data so that you see something? (Figure 1.1)
- Preprocessing for supervised learning: How to find the "relevant" features of the data to improve regression/classification? (Figure 1.2)
- Noise reduction and feature extraction for e.g. computer vision: You have 1,000,000 variables and your computer cannot handle it. How to reduce their number? (Again, Figure 1.2)
- Finding interesting components: You have 100 variables and 1000 observations. Can you extract the underlying phenomena in the data, which the scientists are really interested in? (Figure 1.3)
- Dividing you data into groups: You have data on 1,000 items (patients, cell types, genes, or something). Can you divide them into two distinct groups? (Figure 1.4)
- (Not in this course:) Bayesian/causal networks. How does x_1 affect x_2?

There are different theoretical approaches one might use in machine learning in general, and unsupervised learning in particular. Historically, the oldest is *rule-based* learning. One tries to learn rules such as "if $x_1 > 0$ and $x_2 < -2$, data point belongs to cluster #2". This uses classic algorithmics as developed in computer science. More modern approaches then to emphasize the *statistical* or *probabilistic* framework (the two words have basically the same meaning here). In the simplest case, this might mean computing covariances of the variables and analyzing them. Recently many researchers have advocated a more strictly probabilistic approach in which we formulate the problem as the estimation of a statistical model, and use the theory of statistical estimation (often Bayesian) to estimate the model. Thus, unsupervised machine learning is reduce to estimating a parametric statistical model $p(x|\theta)$ of the observed data, where θ is a vector of parameters.

In this course we use a probabilistic approach (more or less strict), which is why the course is, as already mentioned, essentially a statistics course as well. We also consider real-valued data, not binary or discrete-valued.

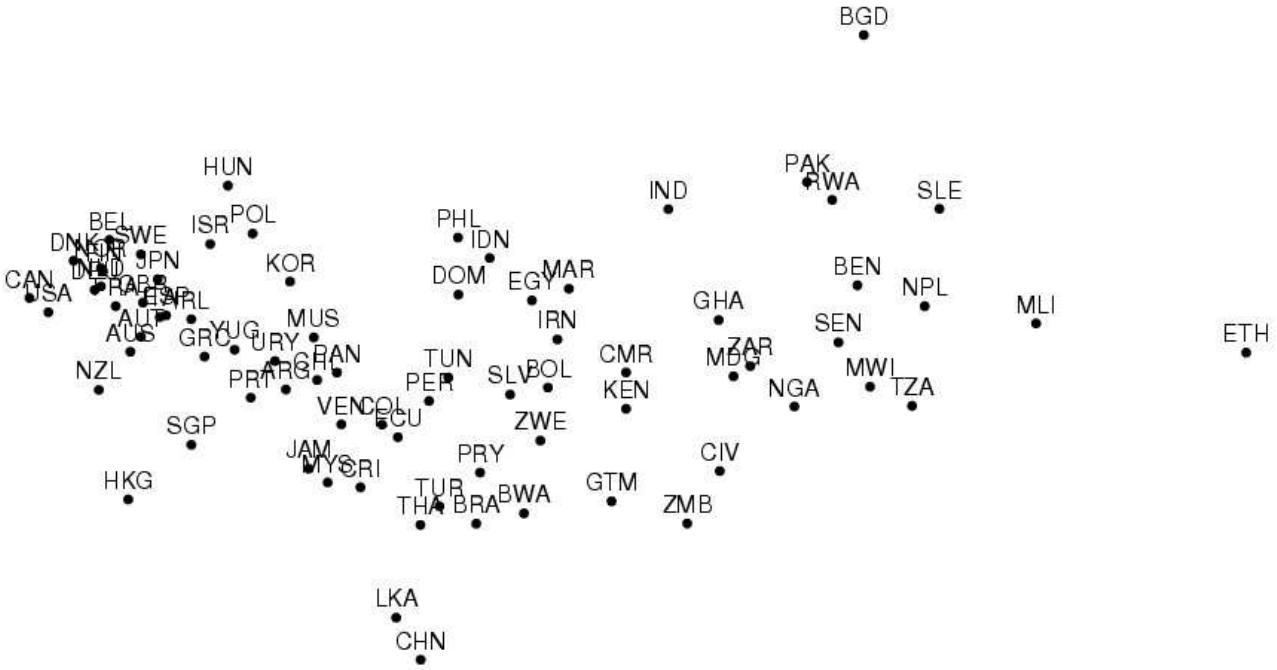

Figure 1.1: An illustration of visualization by projection onto a low-dimensional subspace (S. Kaski, PhD thesis, 1997). The data consisted of various socioeconomical statistics of different countries, identified by their three-letter acronyms. The visualization method learned to assign each country to a location on the two-dimensional plane, so that countries with similar socioeconomic conditions are close to each other.

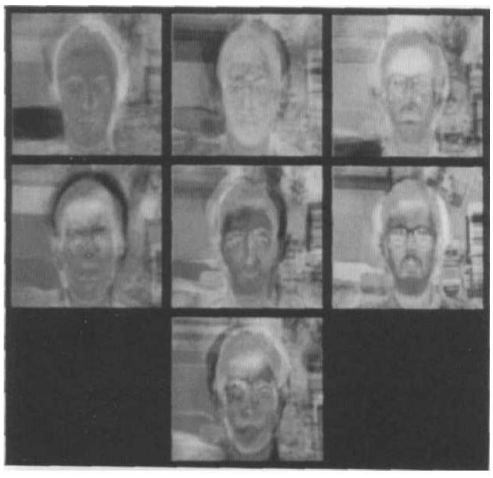

Figure 1.2: An illustration of dimension reduction as preprocessing for supervised learning (Turk and Pentland, J. Cogn. Neurosci., 1991). The data consisted of pictures of faces. The number of pixels in the pictures is very large. Using a method called principal component analysis, the authors found seven features which define a seven-dimensional subspace. Such a seven-dimensional representation can then be used for face recognition.

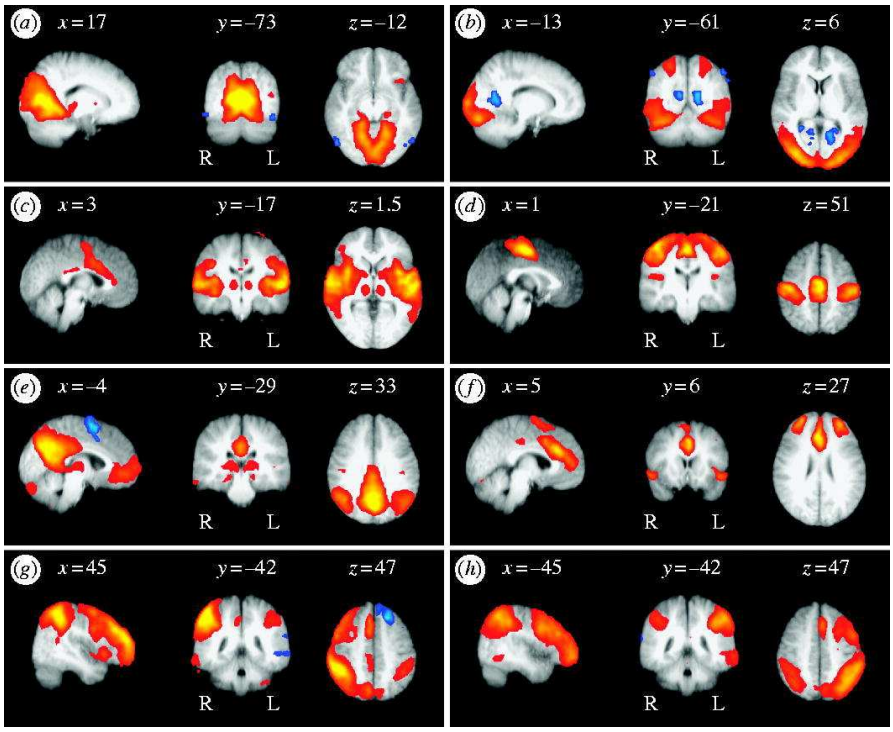

Figure 1.3: An illustration of finding interesting underlying phenomena in high-dimensional data (Beckmann et al, Phil. Trans. Royal Soc. B, 2005). The data consisted of a few hundred brain scans by functional magnetic resonance imaging. Using a method called independent component analysis, the authors found eight activation patterns. These activation patterns were proposed to play an essential role in brain activity during rest.

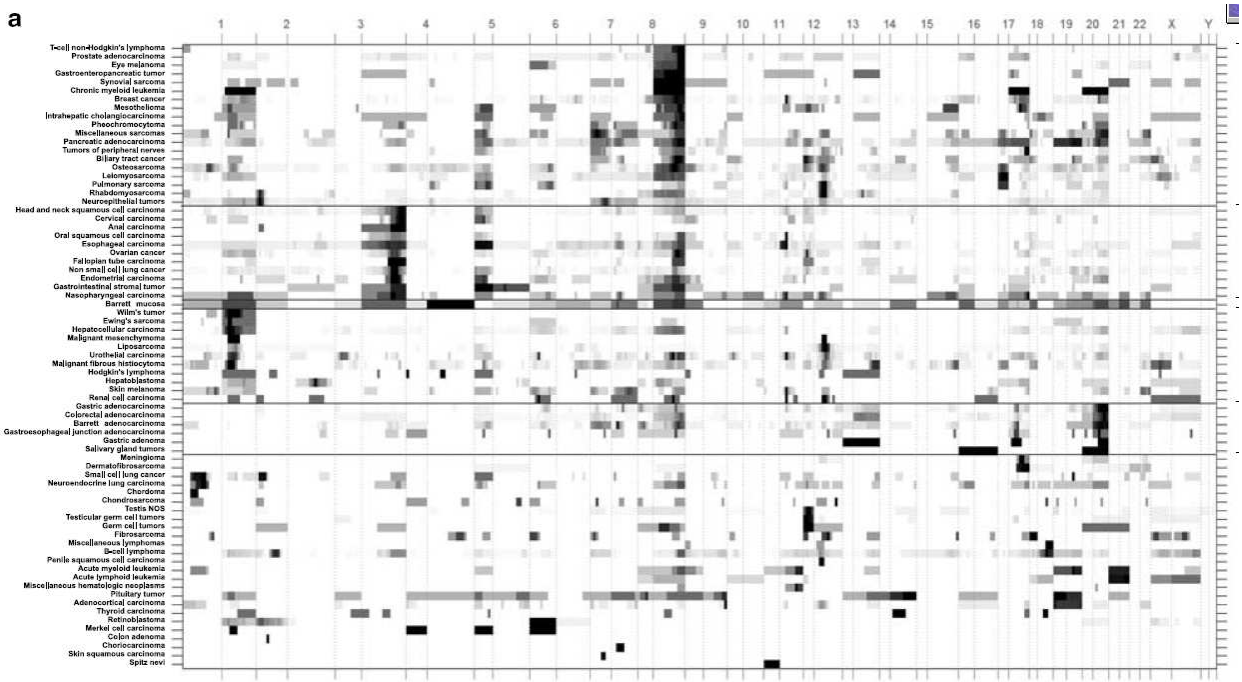

Figure 1.4: An illustration of grouping or clustering (Myllykangas et al, Oncogene, 2006). Each row in this plot corresponds to one cancer cell type, and the values on each row are measurements of what is called "DNA copy number amplification" of different chromosomes. Using a clustering algorithm, five groups of cells with similar properties were found. The rows have been ordered to reflect this grouping; the clusters are separated by horizontal lines. The rows at the bottom didn't fit any groups.

Another important point is that we take *computations* seriously. Sometimes in statistical modelling we only formulate the statistical principles, give an objective function (such as likelihood) which should be maximized, and ignore the problem of how it is computationally maximized. In machine learning, we are not satisfied with that but we also develop methods which do the required computations. This is why optimization theory is essential, and we will develop computational methods for estimating/optimizing each statistical model considered.

Chapter 2

Review of some basic mathematics

2.1 Linear algebra

2.1.1 Matrices

In matrix algebra, linear transformations and linear systems of equations can be succinctly expressed by matrices. A matrix $\mathbf{M}$ of size $n_1 \times n_2$ is a collection of real numbers arranged into n_1 rows and n_2 columns. The single entries are denoted by m_{ij} where i is the row and j is the column. Thus,

$$\mathbf{M} = \begin{bmatrix} m_{11} & m_{12} & \dots & m_{1n_2} \\ \vdots & & & \vdots \\ m_{n_1 1} & m_{n_1 2} & \dots & m_{n_1 n_2} \end{bmatrix} \tag{2.1}$$

Consider a vector $\mathbf{z}$ in an n_2 dimensional real space. A linear transformation of the vector $\mathbf{z}$, as defined by the matrix $\mathbf{M}$, is then denoted by

$$\mathbf{y} = \mathbf{M}\mathbf{z} \tag{2.2}$$

which is basically a short-cut notation for

$$y_i = \sum_{j=1}^{n_2} m_{ij} z_j, \text{ for all } i \tag{2.3}$$

This operation is also the definition of the product of a matrix and a vector.

If we concatenate two linear transformations, defining

$$\mathbf{s} = \mathbf{N}\mathbf{y} \tag{2.4}$$

we get another linear transformation. The matrix $\mathbf{P}$ that expresses this linear transformation is obtained by

$$p_{ij} = \sum_{k=1}^{n_1} n_{ik} m_{kj} \tag{2.5}$$

This is the definition of the product of two matrices: the new matrix $\mathbf{P}$ is denoted by

$$\mathbf{P} = \mathbf{N}\mathbf{M} \tag{2.6}$$

The definition is quite useful, because it means we can multiply matrices and vectors in any order when we compute $\mathbf{s}$. In fact, we have

$$\mathbf{s} = \mathbf{N}\mathbf{y} = \mathbf{N}(\mathbf{M}\mathbf{z}) = (\mathbf{N}\mathbf{M})\mathbf{z} \tag{2.7}$$

Another important operation with matrices is the transpose. The transpose $\mathbf{M}^T$ of a matrix $\mathbf{M}$ is the matrix where the indices are exchanged: the i, j-th entry of $\mathbf{M}^T$ is m_{ji}. A matrix $\mathbf{M}$ is called symmetric if $m_{ij} = m_{ji}$, i.e., if $\mathbf{M}$ equals its transpose.

2.1.2 Determinant

The determinant answers the question: how are volumes changed when the data space is transformed by the linear transformation $\mathbf{M}$? That is, if $\mathbf{z}$ takes values in a cube whose edges are all of length one, what is the volume of the set of the values $\mathbf{y}$ in Equation (2.2)?. The answer is given by the absolute value of the determinant, denoted by $|\det(\mathbf{M})|$, or sometimes simply as $|\mathbf{M}|$.

Two basic properties of the determinant are very useful.

1. The determinant of a product is the product of the determinants: $\det(\mathbf{MN}) = \det(\mathbf{M})\det(\mathbf{N})$. If you think that the first transformation changes the volume by a factor or 2 and the second by a factor of 3, it is obvious that when you do both transformation, the change in volume is by a factor of $2 \times 3 = 6$.
2. The determinant of a diagonal matrix equals the product of the diagonal elements. If you think in two dimensions, a diagonal matrix simply stretches one coordinate by a factor of, say 2, and the other coordinate by a factor of, say 3, so the volume of a square of area equal to 1 then becomes $2 \times 3 = 6$.

(In Section 2.1.4 we will see a further important result on the determinant of an orthogonal matrix).

2.1.3 Inverse

If a linear transformation in Equation (2.2) does not change the dimension of the data the transformation can usually be inverted. That is, Equation (2.2) can usually be solved for $\mathbf{z}$: if we know $\mathbf{M}$ and $\mathbf{y}$, we can compute what was the original $\mathbf{z}$. This is the case if the linear transformation is invertible — a technical condition that is almost always true.

In matrix algebra, the coefficients needed to solve an equation can be obtained by computing the inverse of the matrix $\mathbf{M}$, denoted by $\mathbf{M}^{-1}$. So, solving for $\mathbf{z}$ in (2.2) we have

$$\mathbf{z} = \mathbf{M}^{-1}\mathbf{y} \tag{2.8}$$

A multitude of numerical methods for computing the inverse of the matrix exist.

Note that the determinant of the inverse matrix is simply the inverse of the determinant: $\det(\mathbf{M}^{-1}) = 1/\det(\mathbf{M})$. Logically, if the transformation changes the volume by a factor of 5 (say), then the inverse must change the volume by a factor of 1/5.

The product of a matrix with its inverse equals the *identity matrix* $\mathbf{I}$:

$$\mathbf{M}\mathbf{M}^{-1} = \mathbf{M}^{-1}\mathbf{M} = \mathbf{I} \tag{2.9}$$

The identity matrix is a matrix whose diagonal elements are all ones and the off-diagonal elements are all zero. It corresponds to the identity transformation, i.e., a transformation which does not change the vector. This means we have

$$\mathbf{I}\mathbf{z} = \mathbf{z} \tag{2.10}$$

for any $\mathbf{z}$.

2.1.4 Orthogonality

A linear transformation, or equivalently a matrix, is called *orthogonal* if it does not change the norm of the vector. Likewise, a matrix $\mathbf{A}$ is called orthogonal if the corresponding transformation is orthogonal. An equivalent condition for orthogonality is

$$\mathbf{A}^T\mathbf{A} = \mathbf{I} \tag{2.11}$$

If you think about the meaning of this equation in detail, you will realize that it says two things: the column vectors of the matrix $\mathbf{A}$ are orthogonal, and all normalized to unit norm. This is because the entries in the matrix $\mathbf{A}^T\mathbf{A}$ are the dot-products $\mathbf{a}_i^T\mathbf{a}_j$ between the column vectors of the matrix $\mathbf{A}$.

Equation (2.11) shows that the inverse of an orthogonal matrix (or an orthogonal transformation) is trivial to compute: we just need to rearrange the entries by taking the transpose.

The compound transformation of two orthogonal transformation is orthogonal. This is natural since if neither of the transformations changes the norm of the vector, then doing one transformation after the other does not change the norm either.

The determinant of an orthogonal matrix is equal to plus or minus one. This is because because an orthogonal transformation does not change volumes, so the absolute value has to be one. The change in sign is related to reflections. Think of multiplying one-dimensional data by -1: This does not change the "volumes", but "reflects" the data with respect to 0, and corresponds to a determinant of -1.

2.1.5 Eigenvalues and eigenvectors

If $\mathbf{z}$ fulfills the equation

$$\mathbf{M}\mathbf{z} = \lambda\mathbf{z} \tag{2.12}$$

for some scalar quantity λ, $\mathbf{z}$ is called an eigenvector of $\mathbf{M}$, and λ is called the corresponding eigenvalue. To each eigenvector corresponds one eigenvalue by definition, but the same eigenvalue can correspond to many eigenvectors.

Typically, an $n \times n$ matrix will have n linearly independent eigenvectors, although some complicated conditions are necessary for this to be exactly true. In general, eigenvectors can be complex-valued, but we will in this course always consider eigenvectors of symmetric matrices, which are guaranteed to be real-valued, and furthermore, orthogonal to each other.

Note that if $\mathbf{z}$ is an eigenvector of $\mathbf{M}$, $\alpha\mathbf{z}$ is also, for any scalar α. Thus, the eigenvectors are defined only up to a scaling. Typically, they are scaled to have unit norm in practical computations, and this is what will always be done in these lecture notes.

One intuitive definition of an eigenvector is that its "direction" is not changed by the linear transformation $\mathbf{M}$, it is only rescaled. However, in this course this interpretation is not very useful. Here, the relevant intuitive interpretation of eigenvalues and vectors is related to the eigenvalue decomposition of a matrix, which will be treated next.

Let us assume that the matrix $\mathbf{M}$ is symmetric. Then, we can decompose the matrix into the following product, called the eigenvalue decomposition:

$$\mathbf{M} = \mathbf{U}\mathbf{D}\mathbf{U}^T \tag{2.13}$$

where $\mathbf{U}$ is an orthogonal matrix , and $\mathbf{D} = \text{diag}(\lambda_1, \ldots, \lambda_m)$ is diagonal. The columns of $\mathbf{U}$ are the eigenvectors of $\mathbf{M}$, and the λ_i are the corresponding eigenvalues. Very efficient numerical algorithms exist for computing the eigenvalue decomposition of a matrix.

The meaning of the eigenvalue decomposition is that by changing the coordinate frame (by $\mathbf{U}$), or rotating the space, any symmetric matrix can be extremely simplified, making it diagonal.

2.2 Probability theory and statistics

2.2.1 Multivariate probability distributions

In this chapter, we will denote random variables by $z_1, z_2, \ldots, z_n$ and $s_1, s_2, \ldots, s_n$ for some number n. Taken together, the random variables $z_1, z_2, \ldots, z_n$ form an n-dimensional random vector which we denote by $\mathbf{z}$:

$$\mathbf{z} = \begin{pmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{pmatrix} \tag{2.14}$$

Likewise, the variables $s_1, s_2, \ldots, s_n$ can be collected to a random vector, denoted by $\mathbf{s}$.

A probability distribution of a random vector such as $\mathbf{z}$ is usually represented using a *probability density function* (pdf). The pdf at a point in the n-dimensional space is denoted by $p_{\mathbf{z}}$.

The definition of the pdf of a multidimensional random vector is a simple generalization of the definition of the pdf of a random variable in one dimension. Let us first recall that definition. Denote by z a random variable. The idea is that we take a small number v, and look at the probability that z takes a value in the interval $[a, a+v]$ for any given a. Then we divide that probability by v, and that is the value of the probability density function at the point a. That is

$$p_z(a) = \frac{P(z \text{ is in } [a, a+v])}{v} \tag{2.15}$$

This principle is illustrated in Fig. 2.1. Rigorously speaking, we should take the limit of an infinitely small v in this definition.

This principle is simple to generalize to the case of an n-dimensional random vector. The value of the pdf function at a point, say $\mathbf{a} = (a_1, a_2, \ldots, a_n)$, gives the probability that an observation of $\mathbf{z}$ belongs to a small neighbourhood of the point $\mathbf{a}$, divided by the volume of the neighbourhood. Computing the probability that the values of each z_i are between the values of a_i and $a_i + v$, we obtain

$$p_{\mathbf{z}}(\mathbf{a}) = \frac{P(z_i \text{ is in } [a_i, a_i+v] \text{ for all } i)}{v^n} \tag{2.16}$$

where v^n is the volume of the n-dimensional cube whose edges all have length v. Again, rigorously speaking, this equation is true only in the limit of infinitely small v.

A most important property of a pdf is that it is normalized: its integral is equal to one

$$\int p_{\mathbf{z}}(\mathbf{a}) d\mathbf{a} = 1 \tag{2.17}$$

This constraint means that you cannot just take any non-negative function and say that it is a pdf: you have to normalize the function by dividing it by its integral. (Calculating such an integral can actually be quite difficult and sometimes leads to serious computational problems).

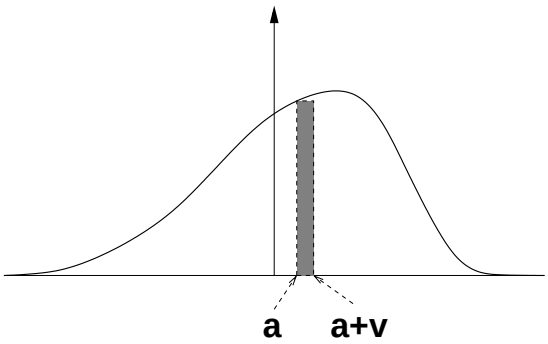

Figure 2.1: The pdf of a random variable at a point a gives the probability that the random variable takes a value in a small interval $[a, a+v]$, divided by the length of that interval, i.e. v. In other words, the shaded area, equal to $p(a)v$, gives the probability that that the variable takes a value in that interval.

For notational simplicity, we often omit the subscript $\mathbf{z}$. We often also write $p(\mathbf{z})$ which means the value of $p_{\mathbf{z}}$ at the point $\mathbf{z}$. This simplified notation is rather ambiguous because now $\mathbf{z}$ is used as an ordinary vector (like $\mathbf{a}$ above) instead of a random vector. However, often it can be used without any confusion.

Example 1 The most classic probability density function for two variables is the gaussian, or normal, distribution. Let us first recall the one-dimensional gaussian distribution, which in the basic case is given by

$$p(z) = \frac{1}{\sqrt{2\pi}} \exp(-\frac{1}{2}z^2) \tag{2.18}$$

It is plotted in Fig. 2.2 b). This is the "standardized" version (mean is zero and variance is one), as explained below. The most basic case of a two-dimensional gaussian distribution is obtained by taking this one-dimensional pdf separately for each variables, and multiplying them together. (The meaning of such multiplication is that the variables are independent, as will be explained below.) Thus, the pdf is given by

$$p(z_1, z_2) = \frac{1}{2\pi} \exp(-\frac{1}{2}(z_1^2 + z_2^2)) \tag{2.19}$$

A scatter plot of the distribution is shown in Fig. 2.2 a). The two-dimensional pdf itself is plotted in Fig. 2.2 c).

Example 2 Let us next consider the following two-dimensional pdf:

$$p(z_1, z_2) = \begin{cases} 1, & \text{if } |z_1| + |z_2| < 1 \\ 0, & \text{otherwise} \end{cases} \tag{2.20}$$

This means that the data is uniformly distributed inside a square which has been rotated 45 degrees. A scatter plot of data from this distribution is shown in Figure 2.3 a).

2.2.2 Marginal and joint probabilities

Consider the random vector $\mathbf{z}$ whose pdf is denoted by $p_{\mathbf{z}}$. It is important to make a clear distinction between the *joint* pdf and the *marginal* pdf's. The joint pdf is just what we called pdf above. The marginal pdf's are what you might call the "individual" pdf's of z_i, i.e. the pdf's of those variables, $p_{z_1}(z_1), p_{z_2}(z_2), \ldots$ when we just consider one of the variables and ignore the existence of the other variables.

There is actually a simple connection between marginal and joint pdf's. We can obtain a marginal pdf by integrating the joint pdf over one of the variables. This is sometimes called "integrating out". Consider for simplicity the case where we only have two variables, z_1 and z_2. Then, the marginal pdf of z_1 is obtained by

$$p_{z_1}(z_1) = \int p_{\mathbf{z}}(z_1, z_2) dz_2 \tag{2.21}$$

This is a continuous-space version of the intuitive idea that for a given value of z_1, we "count" how many observations we have with that value, going through all the possible values of z_2.[1] (In this continuous-valued case, no observed values of z_1 are likely to be exactly equal to the specified value, but we can use the idea of a small interval centred around that value as in the definition of the pdf above.)

[1]Note again that the notation in Eq. (2.21) is sloppy, because now z_1 in the parentheses, both on the left and the right-hand side,

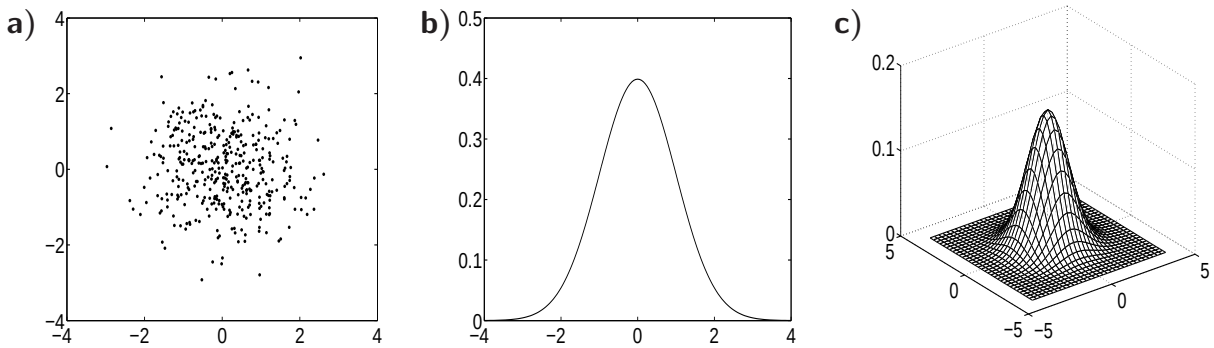

Figure 2.2: **a)** Scatter plot of the two-dimensional gaussian distribution in Equation (2.19). **b)** The one-dimensional standardized gaussian pdf. As explained in Section 2.2.2, it is also the marginal distribution of one of the variables in a), and furthermore, turns out to be equal to the conditional distribution of one variable given the other variable. **c)** The probability density function of the two-dimensional gaussian distribution.

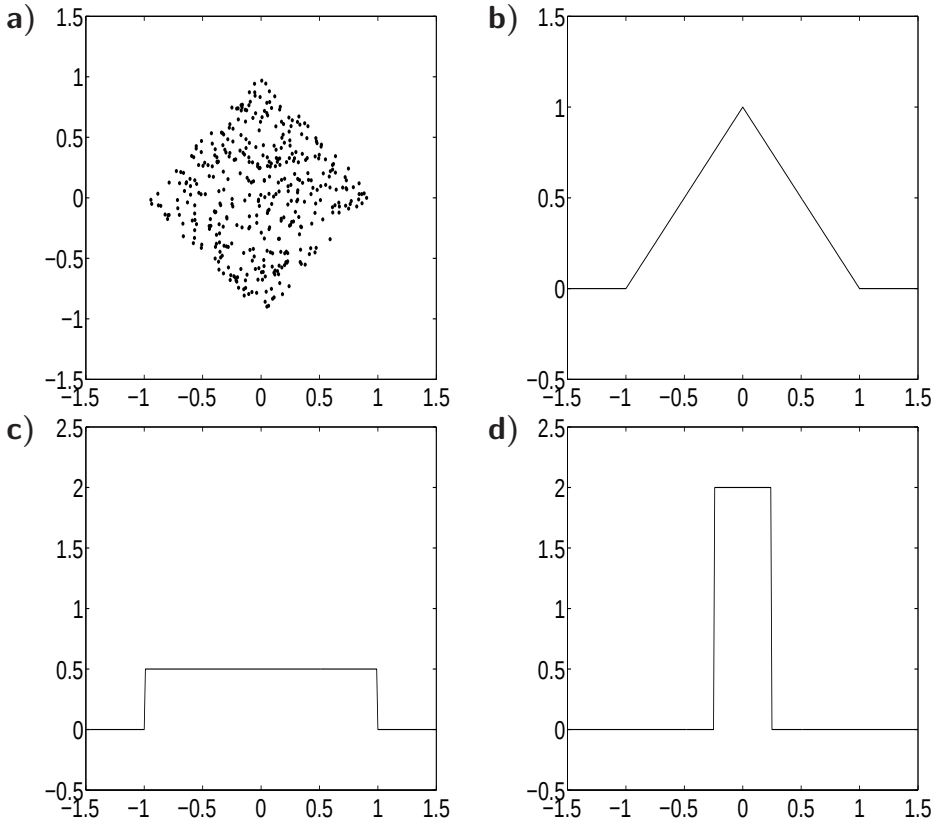

Figure 2.3: **a)** scatter plot of data obtained from the pdf in Eq. (2.20). **b)** marginal pdf of one of the variables in a). **c)** conditional pdf of z_2 given $z_1 = 0$. **d)** conditional pdf of z_2 given $z_1 = .75$

Example 3 In the case of the gaussian distribution in Equation (2.19), we have

$$p(z_1) = \int p(z_1, z_2)dz_2 = \int \frac{1}{2\pi}\exp(-\frac{1}{2}(z_1^2 + z_2^2))dz_2 = \frac{1}{\sqrt{2\pi}}\exp(-\frac{1}{2}z_1^2)\int \frac{1}{\sqrt{2\pi}}\exp(-\frac{1}{2}z_2^2)dz_2 \tag{2.23}$$

Here, we used the fact that the pdf is factorizable since $\exp(a+b) = \exp(a)\exp(b)$. In the last integral, we recognize the pdf of the one-dimensional gaussian distribution of zero mean and unit variance given in Equation (2.18). Thus, that integral is one, because the integral of any pdf is equal to one. This means that the marginal distribution $p(z_1)$ is just the the classic one-dimensional standardized gaussian pdf.

Example 4 Going back to our example in Eq. (2.20), we can calculate the marginal pdf of z_1 to equal

$$p_{z_1}(z_1) = \begin{cases} 1 - |z_1|, \text{ if } |z_1| < 1 \\ 0, \text{ otherwise} \end{cases} \tag{2.24}$$

which is plotted in Fig. 2.3 b), and shows the fact that there is more "stuff" near the origin, and no observation can have an absolute value larger than one. Due to symmetry, the marginal pdf of z_2 has exactly the same form.

2.2.3 Conditional probabilities

Another important concept is the *conditional* pdf of z_2 given z_1. This means the pdf of z_2 when we have observed the value of z_1. Let us denote the observed value of z_1 by a. The conditional pdf is basically obtained by just fixing the value of z_1 to a in the pdf, which gives $p_{\mathbf{z}}(a, z_2)$. However, this is not enough because a pdf must have an integral equal to one. Therefore, we must normalize $p_{\mathbf{z}}(a, z_2)$ by dividing it by its integral. Thus, we obtain the conditional pdf, denoted by $p(z_2 \,|\, z_1 = a)$ as

$$p(z_2 \,|\, z_1 = a) = \frac{p_{\mathbf{z}}(a, z_2)}{\int p_{\mathbf{z}}(a, z_2)dz_2} \tag{2.25}$$

Note that the integral in the denominator equals the marginal pdf of z_1 at point a, so we can also write

$$p(z_2 \,|\, z_1 = a) = \frac{p_{\mathbf{z}}(a, z_2)}{p_{z_1}(a)} \tag{2.26}$$

Again, for notational simplicity, we can omit the subscripts and just write

$$p(z_2 \,|\, z_1 = a) = \frac{p(a, z_2)}{p(a)} \tag{2.27}$$

or, we can even avoid introducing the new quantity a and write

$$p(z_2 \,|\, z_1) = \frac{p(z_1, z_2)}{p(z_1)} \tag{2.28}$$

Example 5 For the gaussian density in Equation (2.19), the computation of the conditional pdf is quite simple, if we use the same factorization as in Equation (2.23):

$$p(z_2|z_1) = \frac{p(z_1, z_2)}{p(z_1)} = \frac{\frac{1}{\sqrt{2\pi}}\exp(-\frac{1}{2}z_1^2)\frac{1}{\sqrt{2\pi}}\exp(-\frac{1}{2}z_2^2)}{\frac{1}{\sqrt{2\pi}}\exp(-\frac{1}{2}z_1^2)} = \frac{1}{\sqrt{2\pi}}\exp(-\frac{1}{2}z_2^2) \tag{2.29}$$

which turns out to be the same as the marginal distribution of z_2. (This kind of situation where $p(z_2|z_1) = p(z_2)$ is related to independence as discussed in Section 2.2.4 below.)

---

stands for any value z_1 might obtain, although the same notation is used for the random quantity itself. A more rigorous notation would be something like:

$$p_{z_1}(v_1) = \int p_{\mathbf{z}}(v_1, v_2)dv_2 \tag{2.22}$$

where we have used two new variables, v_1 to denote the point where we want to evaluate the marginal density, and v_2 which is the integration variable. However, in practice we often do not want to introduce new variable names in order to keep things simple, so we use the notation in Eq. (2.21).

Example 6 In our example pdf in Eq. (2.20), the conditional pdf changes quite a lot as a function of the value a of z_1. If z_1 is zero (i.e. $a = 0$), the conditional pdf of z_2 is the uniform density in the interval $[-1, 1]$. In contrast, if z_1 is close to 1 (or -1), the values that can be taken by z_2 are quite small. Simply fixing $z_1 = a$ in the pdf, we have

$$p(a, z_2) = \begin{cases} 1, \text{ if } |z_2| < 1 - |a| \\ 0 \text{ otherwise} \end{cases} \tag{2.30}$$

which can be easily integrated:

$$\int p(a, z_2) dz_2 = 2(1 - |a|) \tag{2.31}$$

(This is just the length of the segment in which z_2 is allowed to take values.) So, we get

$$p(z_2|z_1) = \begin{cases} \frac{1}{2-2|z_1|}, \text{ if } |z_2| < 1 - |z_1| \\ 0 \text{ otherwise} \end{cases} \tag{2.32}$$

where we have replaced a by z_1. This pdf is plotted for $z_1 = 0$ and $z_1 = 0.75$ in Fig. 2.3 a) and b), respectively.

Generalization to many dimensions The concepts of marginal and conditional pdf's extend naturally to the case where we have n random variables instead of just two. The point is that instead of two random variables, z_1 and z_2, we can have two random vectors, say $\mathbf{z}_1$ and $\mathbf{z}_2$, and use exactly the same formulas as for the two random variables. So, starting with a random vector $\mathbf{z}$, we take some of its variables and put them in the vector $\mathbf{z}_1$, and leave the rest in the vector $\mathbf{z}_2$

$$\mathbf{z} = \begin{pmatrix} \mathbf{z}_1 \\ \mathbf{z}_2 \end{pmatrix} \tag{2.33}$$

Now, the marginal pdf of $\mathbf{z}_1$ is obtained by the same integral formula as above:

$$p_{\mathbf{z}_1}(\mathbf{z}_1) = \int p_{\mathbf{z}}(\mathbf{z}_1, \mathbf{z}_2) d\mathbf{z}_2 \tag{2.34}$$

and, likewise, the conditional pdf of $\mathbf{z}_2$ given $\mathbf{z}_1$ is given by:

$$p(\mathbf{z}_2 \,|\, \mathbf{z}_1) = \frac{p(\mathbf{z}_1, \mathbf{z}_2)}{p(\mathbf{z}_1)} \tag{2.35}$$

Both of these are, naturally, multidimensional pdf's.

2.2.4 Independence

Let us consider two random variables, z_1 and z_2. Basically, the variables z_1 and z_2 are said to be statistically independent if information on the value taken by z_1 does not give any information on the value of z_2, and vice versa.

The idea that z_1 gives no information on z_2 can be intuitively expressed using conditional probabilities: the conditional probability $p(z_2 \,|\, z_1)$ should be just the same as $p(z_2)$:

$$p(z_2 \,|\, z_1) = p(z_2) \tag{2.36}$$

for any observed value a of z_1. This implies

$$\frac{p(z_1, z_2)}{p(z_1)} = p(z_2) \tag{2.37}$$

or

$$p(z_1, z_2) = p(z_1)p(z_2) \tag{2.38}$$

for any values of z_1 and z_2. Equation (2.38) is usually taken as the definition of independence because it is mathematically so simple. It simply says that the joint pdf must be a product of the marginal pdf's. The joint pdf is then called factorizable.

The definition is easily generalized to n variables $z_1, z_2, \ldots, z_n$, in which case it is

$$p(z_1, z_2, \ldots, z_n) = p(z_1)p(z_2) \ldots p(z_n) \tag{2.39}$$

Example 7 For the gaussian distribution in Equation (2.19) and Fig. 2.2, we have

$$p(z_1, z_2) = \frac{1}{\sqrt{2\pi}} \exp(-\frac{1}{2} z_1^2) \times \frac{1}{\sqrt{2\pi}} \exp(-\frac{1}{2} z_2^2) \tag{2.40}$$

So, we have factorized the joint pdf as the product of two pdf's, each of which depends on only one of the variables. Thus, z_1 and z_2 are independent. This can also be seen in the form of the conditional pdf in Equation (2.29), which does not depend on the conditioning variable at all.

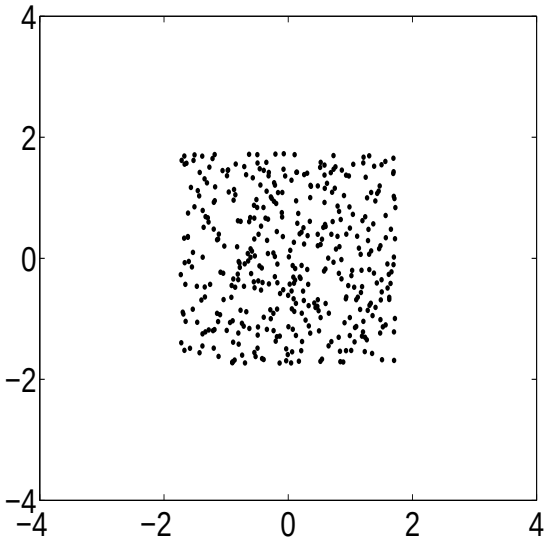

Figure 2.4: A scatter plot of the two-dimensional uniform distribution in Equation (2.41)

Example 8 For our second pdf in Eq. (2.20), we computed the conditional pdf $p(z_2|z_1)$ in Eq. (2.32). This is clearly not the same as the marginal pdf in Eq. (2.24); it depends on z_1. So the variables are not independent. (See the discussion just before Eq. (2.30) for an intuitive explanation of the dependencies.)

Example 9 Consider the uniform distribution on a square:

$$p(z_1, z_2) = \begin{cases} \frac{1}{12}, & \text{if } |z_1| \leq \sqrt{3} \text{ and } |z_2| \leq \sqrt{3} \\ 0, & \text{otherwise} \end{cases} \tag{2.41}$$

A scatter plot from this distribution is shown in Fig. 2.4. Now, z_1 and z_2 are independent because the pdf can be expressed as the product of the marginal distributions, which are

$$p(z_1) = \begin{cases} \frac{1}{2\sqrt{3}}, & \text{if } |z_1| \leq \sqrt{3} \\ 0, & \text{otherwise} \end{cases} \tag{2.42}$$

and the same for z_2.

2.2.5 Expectation

The expectation of a random vector, or its "mean" value, is, in theory, obtained by the same kind of integral as for a single random variable

$$E\{\mathbf{z}\} = \int p_{\mathbf{z}}(\mathbf{z})\, \mathbf{z}\, d\mathbf{z} \tag{2.43}$$

The expectation can be computed by taking the expectation of each variable separately, completely ignoring the existence of the other variables

$$E\{\mathbf{z}\} = \begin{pmatrix} E\{z_1\} \\ E\{z_2\} \\ \vdots \\ E\{z_n\} \end{pmatrix} = \begin{pmatrix} \int p_{z_1}(z_1) z_1\, dz_1 \\ \int p_{z_2}(z_2) z_2\, dz_2 \\ \vdots \\ \int p_{z_n}(z_n) z_n\, dz_n \end{pmatrix} \tag{2.44}$$

The expectation of any transformation $\mathbf{g}$, whether one- or multidimensional, can be computed as:

$$E\{\mathbf{g}(\mathbf{z})\} = \int p_{\mathbf{z}}(\mathbf{z}) \mathbf{g}(\mathbf{z})\, d\mathbf{z} \tag{2.45}$$

The expectation is a linear operation, which means

$$E\{a\mathbf{z} + b\mathbf{s}\} = aE\{\mathbf{z}\} + bE\{\mathbf{s}\} \tag{2.46}$$

for any constants a and b. In fact, this generalizes to any multiplication by a matrix $\mathbf{M}$:

$$E\{\mathbf{Mz}\} = \mathbf{M}E\{\mathbf{z}\} \tag{2.47}$$

The expectation is closely related to the average over a sample. Expectations are theoretical quantities which usually cannot be computed from an observed sample $\mathbf{z}_1, ...\mathbf{z}_N$. The expectation can be approximated, however, by the sample average

$$E\{\mathbf{g}(\mathbf{z})\}\frac{1}{N} \approx \sum_{i=1}^{n} \mathbf{g}(\mathbf{z}_i) \tag{2.48}$$

where the approximation becomes exact in the limit of an infinite sample, i.e. an infinite number of observations.

2.2.6 Parameter estimation and likelihood

A *statistical model* describes the pdf of the observed random vector using a number of parameters. The parameters typically have an intuitive interpretation. A model is basically a conditional density of the observed data variable, $p(z \,|\, \alpha)$, where α is the parameter. The parameter could be a multidimensional vector as well. Different values of the parameter imply different distributions for the data, which is why this can also be thought of as a conditional density in Bayesian theory.

For example, consider the one-dimensional gaussian pdf

$$p(z \,|\, \alpha) = \frac{1}{\sqrt{2\pi}} \exp(-\frac{1}{2}(z-\alpha)^2) \tag{2.49}$$

Here, the parameter α has an intuitive interpretation as the mean of the distribution. Given α, the observed data variable z then takes values around α, with variance equal to one.

Typically, we have a large number of observations of the random variable z, which might come from measuring some phenomenon n times, and these observations are independent. The set of observations is called a *sample* in statistics.[2] So, we want to use all the observations to better estimate the parameters. For example, in the model in (2.49), it is obviously not a very good idea to estimate the mean of the distribution based on just a single observation.

Estimation has a very boring mathematical definition, but basically it means that we want to find a reasonable approximation of the value of the parameter based on the observations in the sample. A method (a formula or an algorithm) that estimates α is called an estimator. The value given by the estimator for a particular sample is called an estimate. Both are usually denoted by a hat: $\hat{\alpha}$.

Assume we now have a sample of n observations. Let us denote the observed values by $z(1), z(2), \ldots, z(n)$. Because the observations are independent, the joint probability is simply obtained by multiplying the probabilities of the observations, so we have

$$p(z(1), z(2), \ldots, z(n) \,|\, \alpha) = p(z(1) \,|\, \alpha) \times p(z(2) \,|\, \alpha) \times \ldots \times p(z(n) \,|\, \alpha) \tag{2.50}$$

This conditional density is called the *likelihood*. It is often simpler to consider the logarithm, which transforms products into sums. If we take the logarithm, we have the log-likelihood as

$$\log p(z(1), z(2), \ldots, z(n) \,|\, \alpha) = \log p(z(1) \,|\, \alpha) + \log p(z(2) \,|\, \alpha) + \ldots + \log p(z(n) \,|\, \alpha) \tag{2.51}$$

The question is then, How can we estimate α?

One basic principle is *maximum likelihood estimation*. It means that we take the value of the parameters which gives the largest value for the likelihood, when likelihood is considered as a function of the parameters with the sample being fixed. The maximum likelihood estimator has thus the intuitive interpretation: it gives *the parameter value that gives the highest probability for the observed data*.

Sometimes the maximum likelihood estimator can be computed by a simple algebraic formula, but in most cases, the maximization has to be done numerically. In some cases it is computationally so difficult that other methods are preferred.

In general, estimation can be accomplished by finding some system of equations which the right parameter values fulfill and then solving it. Of course, this is a very general statement but we will see some examples in this course.

Example 10 In the case of the model in Eq. (2.49), we have

$$\log p(z \,|\, \alpha) = -\frac{1}{2}(z-\alpha)^2 + \text{const.} \tag{2.52}$$

[2] In signal processing, sampling refers the process of reducing a continuous signal to a discrete signal. For example, an image $I(x, y)$ with continuous-valued coordinates x and y is reduced to a finite-dimensional vector in which the coordinates x and y take only a limited number of values (e.g. as on a rectangular grid). These two meanings of the word "sample" need to be distinguished.

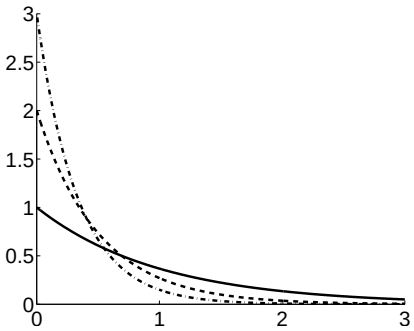

Figure 2.5: The exponential pdf in Equation (2.55) plotted for three different values of α, which is equal 1,2, or 3. The value of α is equal to the value of the pdf at zero.

where the constant is not important because it does not depend on α. So, we have for a sample

$$\log p(z(1), z(2), \ldots, z(n) \,|\, \alpha) = -\frac{1}{2}\sum_{i=1}^{n}(z(i) - \alpha)^2 + \text{const.} \tag{2.53}$$

It can be shown (this is left as an exercise) that this is maximized by

$$\hat{\alpha} = \frac{1}{n}\sum_{i=1}^{n} z(i) \tag{2.54}$$

Thus, the maximum likelihood estimator is given by the average of the observed values. This is not a trivial result: in some other models, the maximum likelihood estimator of such a location parameter is given by the median.

Example 11 Here's an example of maximum likelihood estimation with a less obvious result. Consider the exponential distribution

$$p(z|\alpha) = \alpha \exp(-\alpha z) \tag{2.55}$$

where z is constrained to be positive. The parameter α determines how likely large values are and what the mean is. Some examples of this pdf are shown in Fig. 2.5. The log-pdf is given by

$$\log p(z|\alpha) = \log \alpha - \alpha z \tag{2.56}$$

so the log-likelihood for a sample equals

$$\log p(z(1), z(2), \ldots, z(n) \,|\, \alpha) = n \log \alpha - \alpha \sum_{i=1}^{n} z(i) \tag{2.57}$$

To solve for the α which maximizes the likelihood, we take the derivative of this with respect to α and find the point where it is zero. This gives

$$\frac{n}{\alpha} - \sum_{i=1}^{n} z(i) = 0 \tag{2.58}$$

from which we obtain

$$\hat{\alpha} = \frac{1}{\frac{1}{n}\sum_{i=1}^{n} z(i)} \tag{2.59}$$

So, the estimate is the reciprocal of the mean of the z in the sample.

Chapter 3

Optimization in real spaces

We start by an outline of classical optimization theory which is needed in this course.

3.1 Definition and meaning of gradient

The gradient method is the most fundamental method for maximizing a continuous-valued, smooth function in a multidimensional space.

We consider the general problem of finding the maximum of a function that takes real values in an n-dimensional real space. Let us denote the function to be maximized by $f(\mathbf{w})$ where $\mathbf{w} = (w_1, \ldots, w_n)$ is just an n-dimensional vector. This is usually written as

$$\max_{\mathbf{w}} f(\mathbf{w}) \tag{3.1}$$

The function to be optimized is usually called the objective function.

Finding the minimum is just finding the maximum of the negative of the function, so the same theory is directly applicable to both cases. We consider here maximization because that is what we need mainly later.

The gradient of f, denoted by ∇f is defined as the vector of the partial derivatives:

$$\nabla f(\mathbf{w}) = \begin{pmatrix} \frac{\partial f(\mathbf{w})}{\partial w_1} \\ \vdots \\ \frac{\partial f(\mathbf{w})}{\partial w_n} \end{pmatrix} \tag{3.2}$$

The meaning of the gradient is that it points in the direction where the function *grows the fastest*. More precisely, suppose we want to find a vector $\mathbf{v}$ which is such that $f(\mathbf{w} + \mathbf{v})$ is as large as possible when we constrain the norm of $\mathbf{v}$ to be fixed and very small. Then the optimal $\mathbf{v}$ is given by a suitably short vector in the direction of the gradient vector. Likewise, the vector that reduces the value of f as much as possible is given by $-\nabla f(\mathbf{w})$, multiplied by a small constant. Thus, the gradient is the direction of "steepest ascent", and $-\nabla f(\mathbf{w})$ is the direction of steepest descent.

These properties come from the fundamental first-order approximation property (basically, the definition of derivatives):

$$f(\mathbf{w} + \boldsymbol{\epsilon}) = f(\mathbf{w}) + (\nabla f(\mathbf{w}))^T \boldsymbol{\epsilon} + o(\boldsymbol{\epsilon}) \tag{3.3}$$

Geometrically, the gradient is always *orthogonal* to the curves in a contour plot of the function (i.e. to the curves that show where f has the same value), pointing in the direction of growing f. See Figure 3.1.

The basic principles of optimization theory tell that at the *maximizing points*, the gradient is zero; this is a generalization of the elementary calculus result which says that in one dimension, the minima or maxima of a function are obtained at those points where the derivative is zero.

For illustration, let us consider the following function:

$$f(\mathbf{w}) = \exp(-5(w_1 - 1)^2 - 10(w_2 - 1)^2) \tag{3.4}$$

which is, incidentally, like a gaussian pdf. The function is plotted in Figure 3.1 a). Its maximum is at the point (1,1). The gradient is equal to

$$\nabla f(\mathbf{w}) = \begin{pmatrix} -10(w_1 - 1)\exp(-5(w_1 - 1)^2 - 10(w_1 - 1)^2) \\ -20(w_2 - 1)\exp(-5(w_1 - 1)^2 - 10(w_2 - 1)^2) \end{pmatrix} \tag{3.5}$$

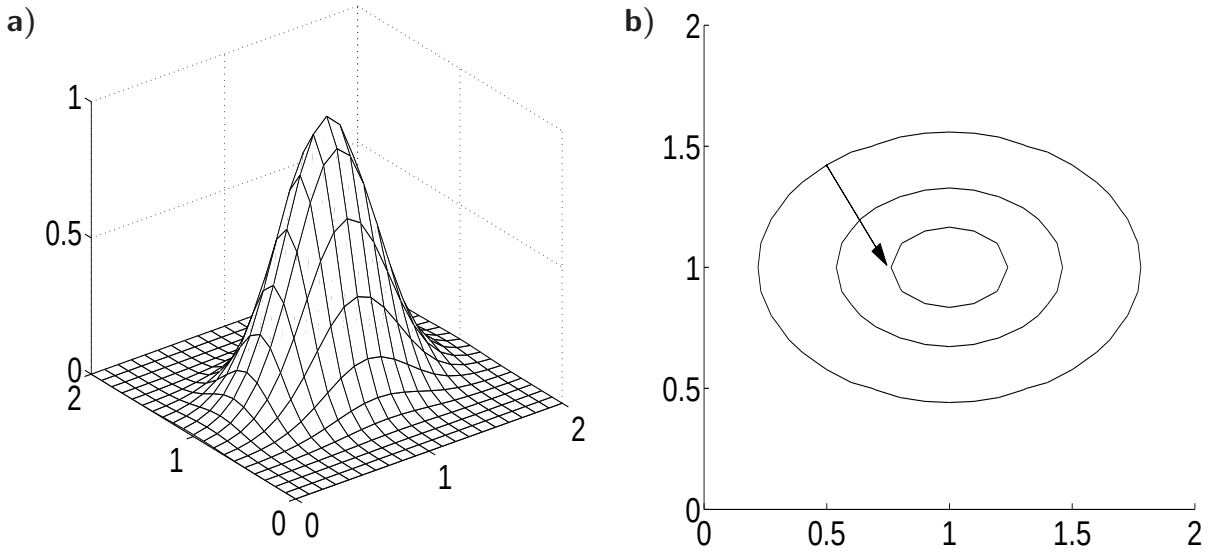

Figure 3.1: The geometrical meaning of the gradient. Consider the function in Equation (3.5), plotted in **a)**. In **b)**, the closed curves are the sets where the function f has a constant value. The gradient at point (0.5,1.42) is shown and it is orthogonal to the curve.

Some contours where the function is constant are shown in Fig. 3.1 b). Also, the gradient at one point is shown. We can see that taking a small step in the direction of the gradient, one gets closer to the maximizing point. However, if one takes a big enough step, one actually misses the maximizing point, so the step really has to be small.

3.2 Gradient and optimization

The *gradient method* for finding the maximum of a function consists of repeatedly taking small steps in the direction of the gradient, $\nabla f(\mathbf{w})$, recomputing the gradient at the current point after each step. We have to take small steps because we know that this direction leads to an increase in the value of f only locally — actually, we can be sure of this only when the steps are infinitely small. The direction of the gradient is, of course, different at each point and needs to be computed again in the new point. The method can then be expressed as

$$\mathbf{w} \leftarrow \mathbf{w} + \mu \nabla f(\mathbf{w}) \tag{3.6}$$

where the parameter μ is a small step size, typically much smaller than 1. The iteration in (3.6) is repeated over and over again until the algorithm "converges" to a point.

To get some insight to the algorithm, consider the (rather trivial) one-dimensional case. At points where the function is increasing, the derivative (i.e. gradient) is positive, so the gradient method will tell us to take steps to the right. At points where the function is decreasing, the derivative/gradient is negative, so it will tell us to move to the left.

When does such an algorithm *converge*? Obviously, if it arrives at a point where the gradient it zero, it will not move away from it. This is not surprising since, as pointed out above, optimization theory tells us that at the maximizing points, the gradient is zero.

In practice, convergence is usually tested by rather heuristic methods: e.g. by looking at the change in $\mathbf{w}$ between two subsequent iterations: if it is small enough, we assume the algorithm has converged.

If the gradient method is used for *minimization* of the function, as is more conventional in the literature, the sign of the increment in (3.6) is negative, i.e.

$$\mathbf{w} \leftarrow \mathbf{w} - \mu \nabla f(\mathbf{w}) \tag{3.7}$$

Choosing a *good step size parameter* μ is crucial. If it is too large, the algorithm will not work at all; if it is too small, the algorithm will be too slow. One possibility is to adapt the step size during the iterations. At each step, we consider the step size used in the previous iteration (say μ_0), and a larger one (say $2\mu_0$) and a smaller one ($\mu_0/2$). Then, we compute the value of the objective function that results from using any of these three step sizes in the current step, and choose the step size which gives the largest value for the objective function, and use it as the μ_0 in the next iteration. Of course, many different adaptation schemes for the step size can be developed.

3.3 Optimization of function of matrix

Many functions we want to maximize are actually functions of a matrix. However, in this context, such matrices are treated just like vectors. That is, a $n \times n$ matrix is treated as an ordinary n^2-dimensional vector; we consider the parameter matrices as if they had been vectorized. In practice, we don't need to concretely vectorize the parameter matrices we optimize, but that is the underlying idea.

For example, consider the following objective function:

$$f(\mathbf{W}) = \sum_{t=1}^{T} \sum_{i} g(\mathbf{w}_i^T \mathbf{z}_t) \tag{3.8}$$

where the vectors $\mathbf{w}_i$ are collected to a single matrix $\mathbf{W}$ with each $\mathbf{w}_i^T$ giving one row. Then, the gradient can be calculated to equal

$$\sum_{t=1}^{T} g'(\mathbf{W}\mathbf{z}_t)\mathbf{z}_t^T \tag{3.9}$$

where we use a short-cut notation where the function g' is applied on each element of $\mathbf{W}\mathbf{z}_t$ separately.

This gradient illustrates some fundamental properties of the gradient: linearity (we can take the gradient "inside" the sum), the chain rule, and the gradient of a linear function: $\nabla_{\mathbf{w}}(\mathbf{w}^T\mathbf{z}) = \mathbf{z}$.

Thus, a gradient method for maximizing f is given by

$$\mathbf{W} \leftarrow \mathbf{W} + \mu[\sum_{t=1}^{T} g'(\mathbf{W}\mathbf{z}_t)\mathbf{z}_t^T] \tag{3.10}$$

3.4 Constrained optimization

It is often necessary to maximize a function under some constraints. That is, the vector $\mathbf{w}$ is not allowed to take any value in the n-dimensional real space. The most common constraint that we will encounter is that the norm of $\mathbf{w}$ is fixed to be constant, typically equal to one: $\|\mathbf{w}\| = 1$. The set of allowed values is called the constraint set. Some changes are needed in the gradient method to take such constraints into account, but in the cases that we are interested in, the changes are actually quite simple.

3.4.1 Projecting back to constraint set

The basic idea of the gradient method can be used in the constrained case as well. Only a simple modification is needed: after each iteration of (3.6), we *project* the vector $\mathbf{w}$ onto the constraint set. Projecting means going to the point in the constraint set which is closest. Projection to the constraint set is illustrated in Figure 3.2 a).

In general, computing the projection can be very difficult, but in some special cases, it is a simple operation. For example, if the constraint set consists of vectors with norm equal to one, the projection is performed simply by the division:

$$\mathbf{w} \leftarrow \mathbf{w}/\|\mathbf{w}\| \tag{3.11}$$

Another common constraint is orthogonality of a matrix. In that case, the projection onto the constraint set is given by

$$\mathbf{W} \leftarrow (\mathbf{W}\mathbf{W}^T)^{-1/2}\mathbf{W} \tag{3.12}$$

Here, we see a rather involved operation: the inverse of the square root of the matrix. We shall not go here into details on how it can be computed; suffice it to say that most numerical software can compute it quite efficiently. The definition will be given later in Eq. (7.20).[1]

3.4.2 Projection of the gradient

Actually, an even better method is obtained if we first project the gradient onto the tangent space of the constraint set, and then take a step in that direction instead of the ordinary gradient direction. This means we find a vector which points in such a direction that an infinitely small step size does not take us out of the constraint set, yet points in the directions of maximum ascent among all such directions. What this means is that we compute a direction that is "inside" the constraint set in the sense that infinitely small changes along that direction do not get us out of the constraint set, yet the movement in that direction maximally increases the value of the objective function.

[1] For a proof that this is the orthogonal projection, see `http://people.csail.mit.edu/bkph/articles/Nearest_Orthonormal_Matrix.pdf`

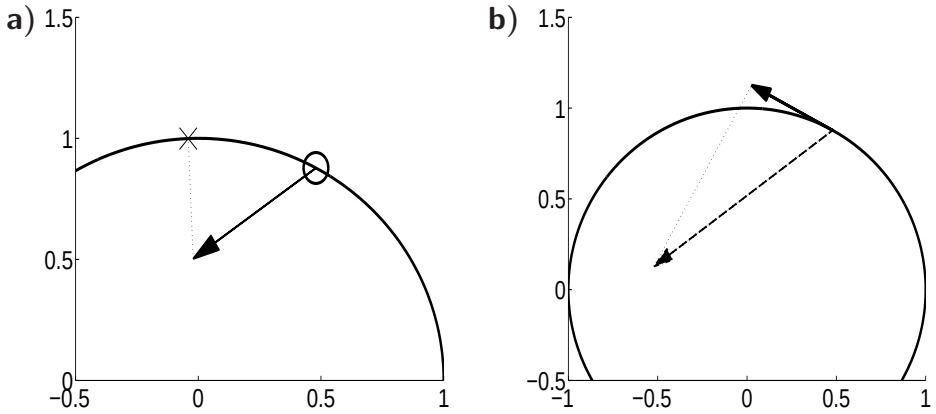

Figure 3.2: Projection onto the constraint set (a), and projection of the gradient (b). A function (not shown explicitly) is to be minimized on the unit sphere. **a)** Starting at the point marked with "o", a small gradient step is taken, as shown by the arrow. Then, the point is projected to the closest point on the unit sphere, which is marked by "x". This is one iteration of the method. **b)** The gradient (dashed arrow) points in a direction in which is it dangerous to take a big step. The projected gradient (solid arrow) points in a better direction, which is "tangential" to the constraint set. Then, a small step in this projected direction is taken (the step is not shown here).

This improves the method because then we can usually take larger step sizes and obtain larger increases in the objective function without going in a completely wrong direction as is always the danger when taking large steps. The projection onto the constraint set has to be done even in this case. Projection of the gradient is illustrated in Figure 3.2 b).

For example, in the case of the unit norm constraint, we have the projected gradient

$$\tilde{\nabla} f(\mathbf{w}) = \nabla f(\mathbf{w}) - \mathbf{w}^T \nabla f(\mathbf{w}) \mathbf{w} \tag{3.13}$$

To verify this, compute the dot-product of this gradient with $\mathbf{w}$, and you see it is zero.

When will such a projected gradient method *converge*? Consider the projected gradient. It can be zero only if either a) the gradient is zero, or b) the gradient is orthogonal to the tangent plane. In the former case, we are actually at an extremum even without taking the constraints into account. In the latter case, the conditions for a constrained maximum is usually fulfilled. They are given in the theory of Lagrangian multipliers, but we will not go into details. Intuitively, it shoudl be clear that at the optimum, the gradient should be orthogonal to the constraint surface, because that means a linear approximation to the function stays constant when moving in the constraint surface.

3.5 Global and local maxima

An important distinction is between global and local maxima. Consider the one-dimensional function in Figure 3.3. The global maximum of the function is at the point $x = 6$; this is the "real" maximum point where the function attains its very largest value. However, there are also two local maxima, at $x = 2$ and $x = 9$. A local maximum is a point in which the function obtains a value which is greater than the values in all neighbouring points close-by.

An important point to understand is that the result of a gradient algorithm depends on the *initial point*, that is, the point where the algorithm starts in the first iteration. The algorithm only sees the local behaviour of the function, so it will find the closest local maximum. Thus, if in Figure 3.3, the algorithm is started in the point marked by circles, it will find the global maximum. In contrast, if it is started in one of the points marked by crosses, it will converge to one of the local maxima.

In many cases, we are only interested in the global maximum. Then, the behaviour of the gradient method can be rather unsatisfactory, because it only finds a local optimum. This is, actually, the case with most optimization algorithms. So, when running optimization algorithms, we have to always keep in mind that that algorithm only gives a local optimum, not the global one.

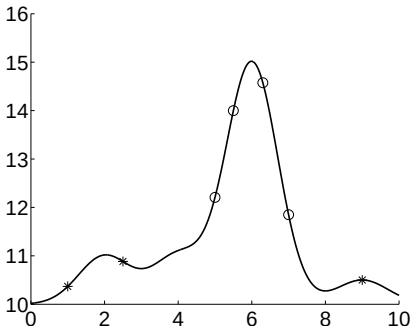

Figure 3.3: Local vs. global maxima. The function has a global maximum at $x = 6$ and two local maxima at $x = 2$ and $x = 9$. If a gradient algorithm starts near one of the local maxima (e.g. at the points marked by crosses), it will get stuck at one of the local maxima and it will not find the global maximum. Only if the algorithm starts sufficiently close to the global maximum (e.g. at the points marked by circles), it will find the global maximum.

3.6 Stochastic gradient methods

The idea in a stochastic gradient is simple and very general. Assume we want to maximize some expectation, say $E\{g(\mathbf{w}, \mathbf{x})\}$ where $\mathbf{x}$ is a random vector, and $\mathbf{w}$ is a parameter vector. The gradient method for maximizing this with respect to $\mathbf{w}$ gives

$$\mathbf{w} \leftarrow \mathbf{w} + \mu E\{\nabla_{\mathbf{w}} g(\mathbf{w}, \mathbf{x})\} \tag{3.14}$$

where the gradient is computed with respect to $\mathbf{w}$, as emphasized by the subscript in the ∇ operator. Note that we have taken the gradient inside the expectation operator, which is valid because expectation is basically a sum, and the derivative of a sum is the sum of the derivatives as noted above.

The stochastic gradient method now proposes that we don't need to compute the expectation before taking the gradient step. For *each observation* $\mathbf{x}$, we can use the gradient iteration given that single observation:

$$\mathbf{w} \leftarrow \mathbf{w} + \mu \nabla_{\mathbf{w}} g(\mathbf{w}, \mathbf{x}) \tag{3.15}$$

So, when given a sample of observations of $\mathbf{x}$, we compute the update in Equation (3.15) for each observation separately. This is reasonable because the update in the gradient will be, *on the average*, equal to the update in the original gradient with the expectation given in Equation (3.14). The step size has to be much smaller, though, because of the large random fluctuations in this "instantaneous" gradient.

For example, consider the iteration in (3.10). The sum over t can be considered an expectation. So, we can develop the stochastic gradient method

$$\mathbf{W} \leftarrow \mathbf{W} + \mu g'(\mathbf{W}\mathbf{z}_t)\mathbf{z}_t^T \tag{3.16}$$

where $\mathbf{z}_t$ is drawn randomly from our sample at each step.

3.7 Newton's method

A classic alternative to the gradient method is Newton's method.

As discussed above, the optima of an objective function are found in points where the gradient is zero. So, optimization can be approached as the problem of solving a system of equations given by

$$\frac{\partial f}{\partial w_1}(\mathbf{w}) = 0 \tag{3.17}$$

$$\vdots \tag{3.18}$$

$$\frac{\partial f}{\partial w_n}(\mathbf{w}) = 0 \tag{3.19}$$

A classic method for solving such a system of equations is Newton's method. It can be used to solve any system of equations, but we consider here the case of the gradient. This is why Newton's method is both a method for solving a system of equations, and an optimization method.

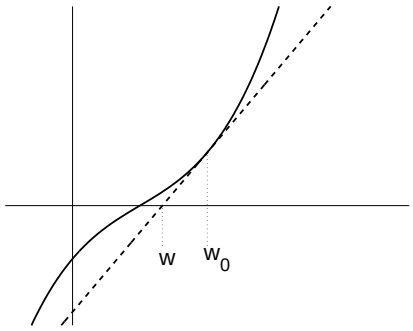

Figure 3.4: Illustration of Newton's method for solving an equation (which we use in optimization to solve the equation which says that the gradient is zero). The function is linearly approximated by its tangent. The point where the tangent intersects with the x-axis is taken as the next approximation of the point where the function is zero.

Basically, the idea is to approximate the function (here: gradient) linearly using its derivatives. In one dimension, the idea is simply to approximate the graph of the function using its tangent, whose slope is given by the derivative. That is, for a general function g:

$$g(w) \approx g(w_0) + g'(w_0)(w - w_0) \tag{3.20}$$

This very general idea of finding the point where the function attains the value zero is illustrated in Figure 3.4.

In our case, g corresponds to the gradient, so we use the derivatives of the gradients, which are second partial derivatives of the original function f. Also, we need to use a multidimensional version of this approximation. This gives

$$\nabla f(\mathbf{w}) \approx \nabla f(\mathbf{w}_0) + \mathbf{H}(\mathbf{w}_0)(\mathbf{w} - \mathbf{w}_0) \tag{3.21}$$

where the function $\mathbf{H}$, called the Hessian matrix, is the matrix of second partial derivatives:

$$\mathbf{H}_{ij} = \frac{\partial^2 f}{\partial w_i \partial w_j} \tag{3.22}$$

Now, we can at every step of the method, find the new point as the one for which this linear approximation is zero. Thus, we solve

$$\nabla f(\mathbf{w}_0) + \mathbf{H}(\mathbf{w}_0)(\mathbf{w} - \mathbf{w}_0) = \mathbf{0} \tag{3.23}$$

which gives

$$\mathbf{w} = \mathbf{w}_0 - \mathbf{H}(\mathbf{w}_0)^{-1}(\nabla f(\mathbf{w}_0)) \tag{3.24}$$

This is the idea in the Newton iteration. Starting from a random point, we iteratively update $\mathbf{w}$ according to Equation (3.24), i.e. compute the right-hand side for the current value of $\mathbf{w}$, and take that as the new value of $\mathbf{w}$. Using the same notation as with the gradient methods, we have the iteration

$$\mathbf{w} \leftarrow \mathbf{w} - \mathbf{H}(\mathbf{w})^{-1}(\nabla f(\mathbf{w})) \tag{3.25}$$

Note that this iteration is related to the gradient method. If the matrix $\mathbf{H}(\mathbf{w}_0)^{-1}$ in Equation (3.24) is replaced by a scalar step size μ, we actually get the gradient method. So, the difference between the methods is threefold:

1. In Newton's method the direction where $\mathbf{w}$ "moves" is not given by the gradient directly, but the gradient multiplied by the inverse of the Hessian.
2. This "step size" is not always very small: It is directly given by the inverse of the Hessian matrix, and can be quite large.
3. In the gradient method, one can choose between minimization and maximization of the objective function, by choosing the sign in the algorithm (cf. Equations (3.6) and (3.7)). In the Newton method, no such choice is possible. The algorithm just tries to find a local extremum in which the gradient is zero, and this can be either a minimum or a maximum.

The Newton iteration has some advantages and disadvantages compared to the basic gradient method. It usually requires a smaller number of steps to converge. However, the computations needed at each step are much more demanding, because one has to first compute the Hessian matrix, and then compute $\mathbf{H}(\mathbf{w})^{-1}(\nabla f(\mathbf{w}))$ (which is best obtained by solving the linear system $\mathbf{H}(\mathbf{w})\mathbf{v} = \nabla f(\mathbf{w})$).

In practice, however, the main problem with the Newton method is that its behaviour can be quite erratic. There is no guarantee any one iteration gives a $\mathbf{w}$ which increases $f(\mathbf{w})$. In fact, a typical empirical observation is that for some functions this does not happen, and the algorithm may completely diverge, i.e. go to arbitrary values of $\mathbf{w}$, eventually reaching infinity. This is

because the step size can be arbitrarily large, unlike in the gradient methods. This lack of robustness is why Newton's method is not often used in practice.

As an example of this phenomenon, consider the function

$$f(w) = \exp(-\frac{1}{2}w^2) \tag{3.26}$$

which has a single maximum as $w = 0$. The first and second derivatives, which are the one-dimensional equivalents of the gradient and the Hessian, can be easily calculated as

$$f'(w) = -w\exp(-\frac{1}{2}w^2) \tag{3.27}$$

$$f''(w) = (w^2 - 1)\exp(-\frac{1}{2}w^2) \tag{3.28}$$

which gives the Newton iteration as

$$w \leftarrow w + \frac{w}{w^2 - 1} \tag{3.29}$$

Now, assume that we start the iteration at any point where $w > 1$. Then, the change $\frac{w}{w^2-1}$ is positive, which means that w is increased and it moves further and further away from zero! In this case, the method fails completely and w goes to infinity without finding the maximum at zero. (In contrast, a gradient method, with a reasonably small step size, would find the maximum.)

However, different variants of the Newton method have proven useful. For example, methods which do something between the gradient method and Newton's method (e.g. the Levenberg-Marquardt algorithm) have proven useful in some applications. In independent component analysis, the FastICA algorithm uses the basic iteration of Newton's method but with a modification which takes the special structure of the objective function into account.

An alternative viewpoint is that in Newton's method, we approximated the original function by a quadratic one. In other words, we use a Taylor expansion

$$f(\mathbf{w}) = f(\mathbf{w}_0) + \nabla f(\mathbf{w})(\mathbf{w} - \mathbf{w}_0) + \frac{1}{2}(\mathbf{w} - \mathbf{w}_0)^T\mathbf{H}(\mathbf{w}_0)(\mathbf{w} - \mathbf{w}_0) \tag{3.30}$$

and then maximize this quadratic form to obtain the new $\mathbf{w}$, which is given by (3.24). How do we maximize this? The details are an interesting exercice, but you can just consider the following algebraic extensions of classic derivative formulas:

$$\nabla_{\mathbf{w}}(\mathbf{x}^T\mathbf{w}) = \mathbf{x} \tag{3.31}$$

$$\nabla_{\mathbf{w}}(\mathbf{w}^T\mathbf{M}\mathbf{w}) = 2\mathbf{M}\mathbf{w} \tag{3.32}$$

$$\tag{3.33}$$

from which you can find the point where the approximation in (3.30) is zero.

3.7.1 Conjugate gradient methods

Conjugate gradient methods are often considered as the most efficient general-purpose optimization methods. The theory is rather complicated and non-intuitive, so we do not try to explain it in detail.

Conjugate gradient methods try to find a direction which is better than the gradient direction. The idea is illustrated in Figure. 3.5. While the gradient direction is good for a very small step size (actually, it is still the best for an infinitely small step size), it is not very good for a moderately large step size. The conjugate gradient method tries to find a better direction based on information on the gradient directions in previous iterations. In this respect, the method is similar to Newton's method, which also modifies the gradient direction.

In fact, conjugate gradient methods do not just take a step of a fixed size in the direction they have found. An essential ingredient, which is actually necessary for the method to work, is a one-dimension *line search*. This means that once the direction, say $\mathbf{d}$, in which $\mathbf{w}$ should move has been chosen (using the complicated theory of conjugate gradient methods), many different step sizes μ are tried out, and the best one is chosen. In other words, a one-dimensional optimization is performed on the function $h(\mu) = f(\mathbf{w} + \mu\mathbf{d})$, and μ maximizing this function is chosen. (Such line search could also be used in the basic gradient method. However, in the conjugate gradient method it is completely necessary.)

Conjugate gradient methods are thus much more complicated than ordinary gradient methods. This is not a major problem if one uses a scientific computing environment in which the method is already programmed. Sometimes, the method is much more efficient than the ordinary gradient methods, but this is not always the case.

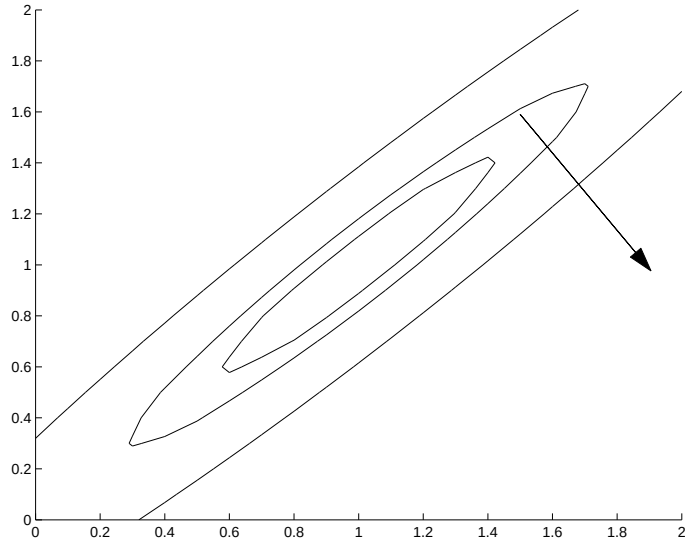

Figure 3.5: A problem with the gradient method. The gradient direction may be very bad for anything but the very smallest step sizes. Here, the gradient goes rather completely in the wrong direction due to the strongly non-circular (non-symmetric) structure of the objective function. The conjugate gradient method tries to find a better direction.

3.8 Alternating variables

In some cases, the objective function is such that it depends on two "different" groups of variables, let's denote them by the vectors $\mathbf{x}$ and $\mathbf{y}$. For example, it could be that $\mathbf{x}$ contains parameters whereas $\mathbf{y}$ contains latent variables. Then, it may happen that it is easy to optimize the objective with respect to $\mathbf{x}$ when *keeping* $\mathbf{y}$ *fixed*, or vice versa. This can lead to an alternating variables method, where we alternate between maximization with respect to $\mathbf{x}$ and with respect to $\mathbf{y}$. In other words:

$$\mathbf{x}(i+1) \leftarrow \arg\max_{\mathbf{x}} g(\mathbf{x}(i), \mathbf{y}(i)) \tag{3.34}$$

$$\mathbf{y}(i+1) \leftarrow \arg\max_{\mathbf{y}} g(\mathbf{x}(i+1), \mathbf{y}(i)) \tag{3.35}$$

where i is the iteration count, and $\arg\max$ denotes the argument which maximizes the objective function.

A more sophisticated version of this idea is the Expectation-Maximization (EM) algorithm, in which the alternating maximization is combined with integration over the posterior of the latent variables. We will see this in Chapter 12.

Chapter 4

Principal component analysis

Assume we have a random vector $\mathbf{x}$. How to characterize the distribution in simple terms? The first, obvious answer is the *mean*. To characterize the remaining distribution, you might think of computing the variances of the variables. However, this does not really characterize a multivariate distribution well in general. Look at Fig. 4.1 and you see that what we would (intuitively) like to describe is the how the data "cloud" is elongated or directed. Principal component analysis is one answer to this problem.

Let us actually subtract the mean from the random vector, so that each x_i has zero mean. Thus, we can better concentrate on the structure which is there in addition to the mean. In the following, it is assumed that all the variables have zero mean. Actually, we are dealing with linear combinations of the observed variables, so all these linear combinations also have zero mean.

4.1 Definition of maximization of variance

Principal components are linear combinations $s = \sum_i w_i x_i$ that contain (or "explain") as much of the variance of the input data as possible. It turns out that the amount of variance explained is directly related to the variance of the component, as will be discussed in Section 4.3 below. Thus, the first principal component is intuitively defined as the linear combination of observed variables, which has the maximum variance. The idea is to find the "main axis" of the data cloud, which is illustrated in Fig. 4.1.

Some constraint on the weights w_i, which we call the *principal component weights*, must be imposed as well. If no constraint were imposed, the maximum of the variance would be attained when w_i becomes infinitely large (and the minimum would be attained when all the w_i are zero). In fact, just multiplying all the weights in w_i by a factor of two, we would get a variance that is four times as large, and by dividing all the coefficients by two the variance decreases to one quarter.

A natural thing to do is to constrain the norm of the vector $\mathbf{w} = (w_1, \ldots, w_n)$:

$$\|\mathbf{w}\| = \sqrt{\sum_i w_i^2} = 1 \tag{4.1}$$

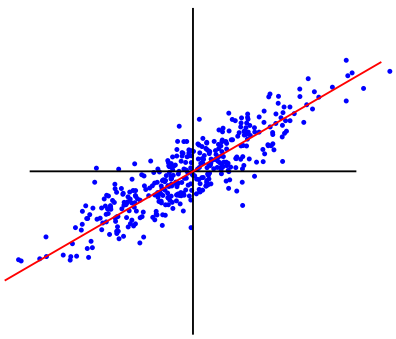

Figure 4.1: Illustration of PCA. The principal component of this (artificial) two-dimensional data is the oblique axis plotted. Projection on the principal axis explains more of the variance of the data than projection on any other axis. It describes the data in some way better than just looking at the variances.

For simplicity, we constrain the norm to be equal to one, but any other value would give the same results.

This definition gives only one principal component. Typically, the way to obtain many principal components is by a "deflation" approach: After estimating the first principal component, we want to find the linear combination of maximum variance *under the constraint* that the new combination must be *orthogonal* to the first one (i.e. the dot-product is zero, as in Equation 4.4). This will then be called the second principal component. This procedure can be repeated to obtain as many components as there are dimensions in the data space. To put this formally, assume that we have estimated k principal components, given by the weight vectors $\mathbf{w}_1, \mathbf{w}_2 \ldots, \mathbf{w}_k$. Then the $k+1$-th principal component weight vector is defined by

$$\max_{\mathbf{w}} \operatorname{var}\left(\sum_i w_i x_i\right) \tag{4.2}$$

under the constraints

$$\|\mathbf{w}\| = \sqrt{\sum_i w_i^2} = 1 \tag{4.3}$$

$$\sum_i w_{ji} w_i = 0 \text{ for all } j = 1, \ldots, k \tag{4.4}$$

4.2 Solution of PCA using eigenvalue decomposition

Here we present the fundamental proof of how PCA is related to eigenvalues of the covariance matrix.

4.2.1 Definitions

The variance of a random variable is defined as

$$\operatorname{var}(x_1) = E\{x_1^2\} - (E\{x_1\})^2 \tag{4.5}$$

This can also be written $\operatorname{var}(x_1) = E\{(x_1 - E\{x_1\})^2\}$, which more clearly shows how variance measures average deviation from the mean value.

When we have more than one random variable, it is useful to analyze the *covariance*:

$$\operatorname{cov}(x_1, x_2) = E\{x_1 x_2\} - E\{x_1\}E\{x_2\} \tag{4.6}$$

which measures how well we can predict the value of one of the variables using a simple linear predictor, as will be seen below.

If the covariance is zero, which is equivalent to saying that the correlation coefficient is zero, the variables are said to be *uncorrelated*.

4.2.2 Covariance matrix

The variances and covariances of the elements of a random vector $\mathbf{x}$ are often collected to a *covariance matrix* whose i,j-th element is simply the covariance of x_i and x_j:

$$\mathbf{C}(\mathbf{x}) = \begin{pmatrix} \operatorname{cov}(x_1, x_1) & \operatorname{cov}(x_1, x_2) & \ldots & \operatorname{cov}(x_1, x_n) \\ \operatorname{cov}(x_2, x_1) & \operatorname{cov}(x_2, x_2) & \ldots & \operatorname{cov}(x_2, x_n) \\ \vdots & & \ddots & \vdots \\ \operatorname{cov}(x_n, x_1) & \operatorname{cov}(x_n, x_2) & \ldots & \operatorname{cov}(x_n, x_n) \end{pmatrix} \tag{4.7}$$

Note that the covariance of x_i with itself is the same as the variance of x_i. So, the diagonal of the covariance matrix gives the variances. The covariance matrix is basically a generalization of variance to random vectors: in many cases, when moving from a single random variable to random vectors, the covariance matrix takes the place of variance.

In matrix notation, the covariance matrix is simply obtained as a generalization of the one-dimensional definitions in Equations (4.6) and (4.5) as

$$\mathbf{C}(\mathbf{x}) = E\{\mathbf{x}\mathbf{x}^T\} - E\{\mathbf{x}\}E\{\mathbf{x}\}^T \tag{4.8}$$

where taking the transposes in the correct places is essential. In most of this book, we will be dealing with random variables whose means are zero, in which case the second term in Equation (4.8) is zero.

If the variables are uncorrelated, the covariance matrix is diagonal. If they are all further standardized to unit variance, the covariance matrix equals the identity matrix.

4.2.3 Eigenvalues of covariance matrix

The important point to note is that the variance of any linear combination can be computed using the *covariance matrix* of the data. That is, considering any linear combination $\mathbf{w}^T\mathbf{x} = \sum_i w_i x_i$ we can compute its variance simply by:

$$E\{(\mathbf{w}^T\mathbf{x})^2\} = E\{(\mathbf{w}^T\mathbf{x})(\mathbf{x}^T\mathbf{w})\} = E\{\mathbf{w}^T(\mathbf{x}\mathbf{x}^T)\mathbf{w}\} = \mathbf{w}^T E\{\mathbf{x}\mathbf{x}^T\}\mathbf{w} = \mathbf{w}^T\mathbf{C}\mathbf{w} \tag{4.9}$$

where we denote the covariance matrix by $\mathbf{C} = E\{\mathbf{x}\mathbf{x}^T\}$. So, the basic PCA problem can be formulated as

$$\max_{\mathbf{w}:\|\mathbf{w}\|=1} \mathbf{w}^T\mathbf{C}\mathbf{w} \tag{4.10}$$

(We assume here that the mean is zero: $E\{\mathbf{x}\} = \mathbf{0}$).

A basic concept in linear algebra is the eigenvalue decomposition. The starting point is that $\mathbf{C}$ is a symmetric matrix, because $\mathrm{cov}(x_i, x_j) = \mathrm{cov}(x_j, x_i)$. In linear algebra, it is shown that any symmetric matrix can be expressed as a product of the form:

$$\mathbf{C} = \mathbf{U}\mathbf{D}\mathbf{U}^T \tag{4.11}$$

where $\mathbf{U}$ is an orthogonal matrix , and $\mathbf{D} = \mathrm{diag}(\lambda_1, \ldots, \lambda_m)$ is diagonal. The columns of $\mathbf{U}$ are the *eigenvectors* of $\mathbf{C}$, and the λ_i are the corresponding *eigenvalues*. (See Section 2.1.5 for the general definition of eigenvalues and eigenvectors.)

Now, we can solve PCA easily. Lets us make the change of variables $\mathbf{v} = \mathbf{U}^T\mathbf{w}$. Then we have

$$\mathbf{w}^T\mathbf{C}\mathbf{w} = \mathbf{w}^T\mathbf{U}\mathbf{D}\mathbf{U}^T\mathbf{w} = \mathbf{v}^T\mathbf{D}\mathbf{v} = \sum_i v_i^2\lambda_i \tag{4.12}$$

Because $\mathbf{U}$ is orthogonal, $\|\mathbf{v}\| = \|\mathbf{w}\|$, so the constraint is the same for $\mathbf{v}$ as it was for $\mathbf{w}$. Let us make the further change of variables to $m_i = v_i^2$. The constraint of unit norm of $\mathbf{v}$ is now equivalent to the constraints that the sum of the m_i must equal one (they must also be positive because they are squares). Then, the problem is transformed to

$$\max_{m_i \geq 0, \sum m_i = 1} \sum_i m_i\lambda_i \tag{4.13}$$

It is rather obvious that the maximum is found when the m_i corresponding to the largest λ_i is one and the others are zero. Let us denote by i^* the index of the maximum eigenvalue. Going back to the $\mathbf{w}$, this corresponds to $\mathbf{w}$ being equal to the i^*-th eigenvector, that is, the i^*-th column of $\mathbf{U}$. Thus, we see how the first principal component is easily computed by the eigenvalue decomposition.

Since the eigenvectors of a symmetric matrix are orthogonal, finding the second principal component means maximizing the variance so that v_{i^*} is kept zero. This is actually equivalent to making the new $\mathbf{w}$ orthogonal to the first eigenvector. Thus, in terms of m_i, we have exactly the same optimization problem, but with the extra constraint that $m_{i^*} = 0$. Obviously, the optimum is obtained when $\mathbf{w}$ is equal to the eigenvector corresponding to the *second* largest eigenvalue. This logic applies to the k-th principal component.

Thus, all the principal components can be found by ordering the eigenvectors $\mathbf{u}_i, i = 1, \ldots, m$ in $\mathbf{U}$ so that the corresponding eigenvalues are in decreasing order. Let us assume that $\mathbf{U}$ is ordered so. Then the i-th principal component s_i is equal to

$$s_i = \mathbf{u}_i^T\mathbf{x} \tag{4.14}$$

Note that it can be proven that the λ_i are all non-negative for a covariance matrix.

4.2.4 Some properties of principal components

Uncorrelatedness Using the eigenvalue decomposition, we can prove some interesting properties of PCA. First, the principal components are *uncorrelated*, because for the vector of the principal components

$$\mathbf{s} = \mathbf{U}^T\mathbf{x} \tag{4.15}$$

we have

$$E\{\mathbf{s}\mathbf{s}^T\} = E\{\mathbf{U}^T\mathbf{x}\mathbf{x}^T\mathbf{U}\} = \mathbf{U}^T E\{\mathbf{x}\mathbf{x}^T\}\mathbf{U} = \mathbf{U}^T(\mathbf{U}\mathbf{D}\mathbf{U}^T)\mathbf{U} = (\mathbf{U}^T\mathbf{U})\mathbf{D}(\mathbf{U}^T\mathbf{U}) = \mathbf{D} \tag{4.16}$$

because of the orthogonality of $\mathbf{U}$. Thus, the covariance matrix is diagonal, which shows that the principal components are uncorrelated.

Variances Second, we see that the variances of the principal components are equal to the λ_i.

Uniqueness The fact that the variances of the principal components are given by λ_i has an important implication for the *uniqueness* of PCA. If two of the eigenvalues are equal, then the variance of those principal components are equal. Then, the principal components are not well-defined anymore, because we can make a *rotation* of those principal components without affecting their variances. This is because if z_i and z_{i+1} have the same variance, then linear combinations such as $\sqrt{1/2}z_i + \sqrt{1/2}z_{i+1}$ and $\sqrt{1/2}z_i - \sqrt{1/2}z_{i+1}$ have the same variance as well; all the constraints (unit variance and orthogonality) are still fulfilled, so these are equally valid principal components. In fact, in linear algebra, it is well-known that the eigenvalue decomposition is uniquely defined only when the eigenvalues are all distinct.

4.3 Dimension reduction by PCA

4.3.1 Definition and uniqueness

One task where PCA is very useful is in reducing the dimension of the data so that the maximum amount of the variance is preserved.

Consider the following general problem. We have a very large number, say m, of random variables $x_1, \ldots, x_m$. Computations that use all the variables would be too burdensome. We want to reduce the dimension of the data by linearly transforming the variables into a smaller number, say n, of variables that we denote by $z_1, \ldots, z_n$:

$$z_i = \sum_{j=1}^{m} w_{ij} x_j, \text{ for all } i = 1, \ldots, n \tag{4.17}$$

The number of new variables n might be only 10% or 1% of the original number m. We want to find the new variables so that they preserve as much information on the original data as possible. This "preservation of information" has to be exactly defined. The most wide-spread definition is to look at the squared error that we get when we try to reconstruct the original data using the z_i. That is, we reconstruct x_j as a linear transformation $\tilde{x}_j = \sum_i a_{ji} z_i$, minimizing the average error

$$E\left\{\sum_j \left(x_j - \sum_i a_{ji} z_i\right)^2\right\} = E\left\{\|\mathbf{x} - \sum_i \mathbf{a}_i z_i\|^2\right\} \tag{4.18}$$

where the a_{ji} are also determined so that they minimize this error. For simplicity, let us consider only transformations for which the transforming weights are orthogonal and have unit norm:

$$\sum_j w_{ij}^2 = 1, \text{ for all } i \tag{4.19}$$

$$\sum_j w_{ij} w_{kj} = 0, \text{ for all } i \neq k \tag{4.20}$$

What is the best way of doing this dimension reduction? The solution is to take as the z_i the n first principal components! Furthermore, the optimal reconstruction weight vectors $\mathbf{a}_i$ in Equation (4.18) are given by the very same principal components weights which compute the z_i.

The solution is not uniquely defined, though, because any orthogonal transformation of the z_i is just as good. This is understandable because any such transformation of the z_i contains just the same information: we can make the inverse transformation to get the z_i from the transformed ones.

The lack of uniqueness is not very serious in the case of dimension reduction: What matters in the dimension reduction context is not so much the actual components themselves, but the *subspace* which they span. The *principal subspace* means the set of all possible linear combinations of the n first principal components. It corresponds to taking all possible linear combinations of the principal component weight vectors $\mathbf{w}_i$ associated with the n principal components.

The logic of the uniqueness result of PCA tells us when the principal subspace is unique. If the n-th and the $(n+1)$-th principal components have equal variances, we cannot decide which one to include in the subspace, and thus even the principal subspace is not unique. But the effect on the whole subspace is usually quite small and can be ignored in practice.

4.3.2 Proof of optimality of PCA

To prove the optimality property of the principal subspace, let us simplify the situation by assuming that at the optimum, the reconstructing weights $\mathbf{A}$ are equal to the transpose of $\mathbf{W}$. Then, the reconstruction $\tilde{\mathbf{x}}$ is really an orthogonal projection to a subspace:

$$\tilde{\mathbf{x}} = \mathbf{W}^T \mathbf{W} \mathbf{x} \tag{4.21}$$

The objective function can then be manipulated to give

$$E\|\mathbf{x} - \mathbf{W}^T\mathbf{W}\mathbf{x}\|^2 = E\left\{\|\mathbf{x}\|^2 - 2\mathbf{x}^T\mathbf{W}^T\mathbf{W}\mathbf{x} + \mathbf{x}\mathbf{W}^T\mathbf{W}\mathbf{W}^T\mathbf{W}\mathbf{x}\right\} = -E\{\mathbf{x}^T\mathbf{W}^T\mathbf{W}\mathbf{x}\} + \text{const.} \tag{4.22}$$

where the constant does not depend on $\mathbf{W}$. Denote by tr() the trace, i.e. the sum of diagonal elements of a matrix. Using the basic identity $\text{tr}(\mathbf{M}_1\mathbf{M}_2) = \text{tr}(\mathbf{M}_2\mathbf{M}_1)$, we obtain

$$E\{\mathbf{x}^T\mathbf{W}^T\mathbf{W}\mathbf{x}\} = \text{trace}(E\{\mathbf{x}^T\mathbf{W}^T\mathbf{W}\mathbf{x}\}) = \text{trace}(\mathbf{W}\mathbf{C}\mathbf{W}^T) = \sum_{i=1}^{n}\mathbf{w}_i^T\mathbf{C}\mathbf{w}_i \tag{4.23}$$

where $\mathbf{C} = E\{\mathbf{x}\mathbf{x}^T\}$ is the covariance matrix and $\mathbf{w}_i$ is the i-th row of $\mathbf{W}$. This is obviously very close to the PCA maximization problem, expect that we have a matrix $\mathbf{W}$ with orthogonal rows instead of a single vector. However, doing the same eigenvalue decomposition and the same kind of variable transformations $\mathbf{V} = \mathbf{W}\mathbf{U}$, we transform the objective to

$$\text{trace}(\mathbf{W}\mathbf{C}\mathbf{W}^T) = \text{trace}(\mathbf{W}\mathbf{U}\mathbf{D}\mathbf{U}^T\mathbf{W}^T) = \text{trace}(\mathbf{V}\mathbf{D}\mathbf{V}^T) = \sum_i\sum_j v_{ij}^2\lambda_j = \sum_j q_j\lambda_j \tag{4.24}$$

where we have defined $q_j = \sum_i v_{ij}^2$. By orthogonality of $\mathbf{V}$, we have $q_j \leq 1$ and $\sum_j q_j = n$. Note that in general, $q_j \neq 1$ because $\mathbf{V}$ has fewer rows than columns. Now, we have an optimization problem based on "weighting" as in (4.13). The difference is that while the total sum of the weight is constrained, also the maximum value for each weight is constrained:

$$\max_{0\leq q_i\leq 1\,,\,\sum q_i = n}\sum_i q_i\lambda_i \tag{4.25}$$

Intuitively, it should be clear that the optimum is obtained when we give weight equal to one to the n largest eigenvalues λ_i. Going back to the original variables, this means that we take as the rows of $\mathbf{W}$ the eigenvectors corresponding to the n largest eigenvalues of $\mathbf{C}$.

4.3.3 Proportion of variance explained by the components

Above, we saw that the eigenvalues λ_i of the covariance matrix give the variances of each principal component. Each eigenvalue is called the amount of the variance "explained" by that component. If we take the n first principal components, they together "explain" the variance of amount $\sum_{i=1}^n \lambda_i$. This terminology is motivated by the following connection to reconstruction error: Combining (4.22) with the solution of (4.25) gives

$$E\|\mathbf{x} - \mathbf{W}^T\mathbf{W}\mathbf{x}\|^2 = E\{\|\mathbf{x}\|^2\} - \sum_{i=1}^{n}\lambda_i \tag{4.26}$$

So we see that the reconstruction error is basically the "total variance" $E\{\|\mathbf{x}\|^2\}$ minus the sum of the n first eigenvalues. In fact, it is equal to the sum of the eigenvalues which were omitted from the PCA, since the total variance $\sum_{i=1}^m \text{var}(x_i)$ is equal to $\sum_{i=1}^m \lambda_i$ (prove this). Thus, each principal component taken into the low-dimensional representation reduces the reconstruction error by the amount given by the corresponding eigenvalue. It is this reduction in reconstruction error which is said to be "explained" by the component.

Typically, this is reported as a percentage of the total variance:

$$\text{proportion of variance explained} = \frac{\sum_{i=1}^n \lambda_i}{\sum_{i=1}^m \lambda_i} \times 100\% \tag{4.27}$$

where m is the dimension of the data. This percentage is often reported to show how well PCA represents the data, given the number of principal components n.

4.4 Illustration

We illustrate PCA on an artificial data set, given by the matrix

$$\mathbf{X} = \begin{pmatrix} 5 & 3 & 0 & 1 & -1 & -3 & 5 & 0 & -4 & -4 \\ -2 & -1 & 0 & 0 & 1 & 4 & -3 & 1 & 5 & 3 \\ 0 & 1 & 4 & -1 & 0 & 5 & 5 & -5 & -3 & -3 \\ 0 & 2 & 3 & 0 & -1 & 3 & 3 & -7 & -2 & 0 \\ 3 & 4 & -2 & 1 & 3 & -3 & -3 & 2 & 0 & 0 \end{pmatrix} \tag{4.28}$$

where each row is one variable and each column is one observation. The eigenvalues of the covariance matrix are:

$$\begin{pmatrix} 25.6351 & 16.1255 & 3.0215 & 0.9756 & 0.3201 \end{pmatrix} \tag{4.29}$$

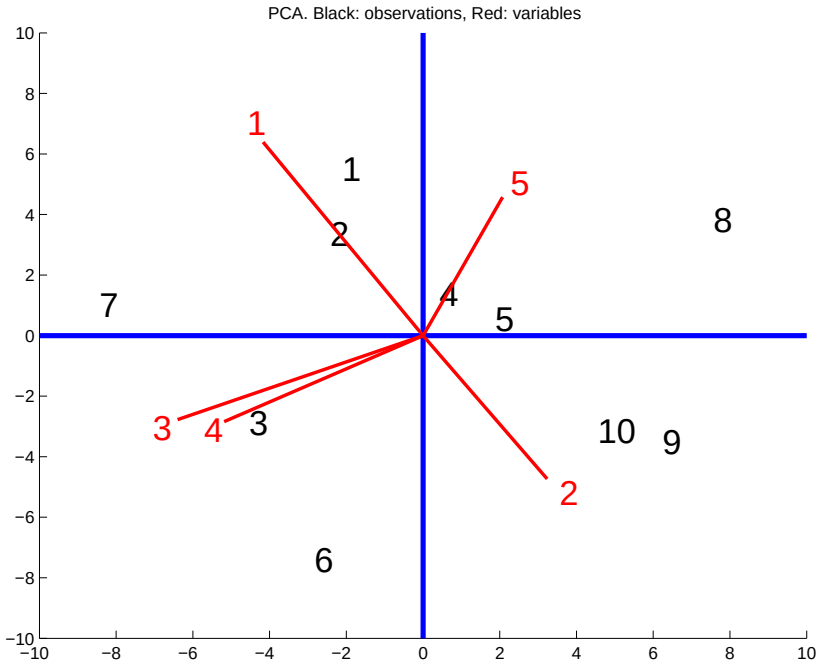

Figure 4.2: Illustration of application of PCA on an artificial data set. Each number in black gives the projection of one data point, and each axis and number in red gives the projection of one variable. Note that for the purpose of visibility, the axes representing the variables have been rescaled.

which shows that the first two principal components explain 90.6% of the variance. The corresponding two eigenvectors of the covariance matrix are

$$\mathbf{u}_1 = \begin{pmatrix} -0.4170 & 0.3237 & -0.6399 & -0.5184 & 0.2075 \end{pmatrix}^T \tag{4.30}$$

and

$$\mathbf{u}_2 = \begin{pmatrix} 0.6393 & -0.4736 & -0.2777 & -0.2841 & 0.4574 \end{pmatrix}^T \tag{4.31}$$

We can plot each observation by projecting it on the coordinate axes given by $\mathbf{u}_1$ and $\mathbf{u}_2$. That is, a data point $\mathbf{x}$ has coordinates $\mathbf{u}_1^T\mathbf{x}$ and $\mathbf{u}_2^T\mathbf{x}$ in the plot. See Figure 4.2.

Moreover, we can plot the original variables, simply by using the values in $\mathbf{u}_1$ and $\mathbf{u}_2$. For example, the first variable corresponds to an axis given by the vector (-0.4170,0.6393), where these numbers are the first entries in the vectors $\mathbf{u}_1$ and $\mathbf{u}_2$

We can basically see that similar observations are close to each other, and similar variables are close to each other (and variables with opposite signs are pointing in opposite directions).

4.5 Whitening

4.5.1 Whitening as normalized decorrelation

Another task for which PCA is quite useful is whitening. Whitening is an important preprocessing method where the variables x_i are transformed to a set of new variables $s_1, \ldots, s_n$ so that the s_i are uncorrelated and have unit variance:

$$E\{s_i s_j\} = \begin{cases} 0 \text{ if } i \neq j \\ 1 \text{ if } i = j \end{cases} \tag{4.32}$$

(It is assumed that all the variables have zero mean.) It is also said that the resulting vector $(s_1, \ldots, s_n)$ is white.

A central property of whitened data is that the variance is the same in all directions:

$$\text{var}(\mathbf{w}^T\mathbf{s}) = 1 \text{ for any } \mathbf{w} : \|\mathbf{w}\| = 1 \tag{4.33}$$

Related properties are considered below.

In addition to the principal components weights being orthogonal, the principal components themselves are uncorrelated, as was shown in Section 4.2.3. So, after PCA, the only thing we need to do to get whitened data is to normalize the variances of

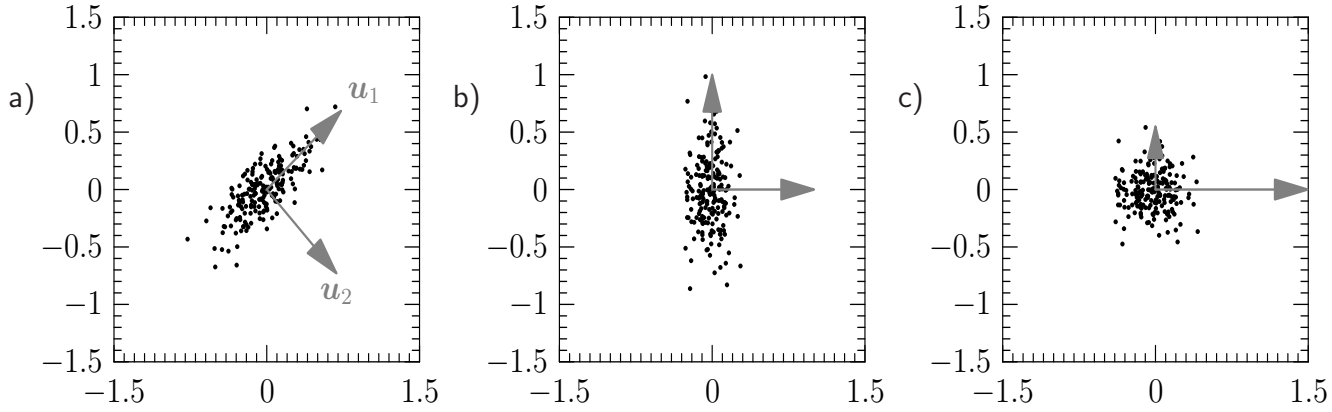

Figure 4.3: Illustration of PCA and whitening. a) The original data "cloud". The arrows show the principal components. The first one points in the direction of the largest variance in the data, and the second in the remaining orthogonal direction. b) When the data is transformed to the principal components, i.e. the principal components are taken as the new coordinates, the variation in the data is aligned with those new axes, which is because the principal components are uncorrelated. c) When the principal components are further normalized to unit variance, the data cloud has equal variance in all directions, which means it has been whitened. The change in the lengths of the arrows reflects this normalization; the larger the variance, the shorter the arrow. (Note that there is a small error in this plot: The scales on the axes are wrong because the variances seem to be smaller than one.)

the principal components by dividing them by their standard deviations. Denoting the principal components by y_i, this means we compute

$$s_i = \frac{y_i}{\sqrt{\text{var}(y_i)}} \tag{4.34}$$

to get whitened components s_i. Whitening is a useful preprocessing method that is often used before other learning. Whitening by PCA is illustrated in Figure 4.3.

4.5.2 The family of whitening transformations

It must be noted that there are many whitening transformations. In fact, if the random variables $s_i, i = 1 \ldots, n$ are white, then any *orthogonal transformation* of those variables is also white (the proof is left as an exercise). Often, whitening is based on PCA because PCA is a well-known method that can be computed very fast, but it must be kept in mind that PCA is just one among the many whitening transformations.

Later, we will often use the fact that the connection between orthogonality and uncorrelatedness is even stronger for whitened data. In fact, if we compute two linear components $\sum_i v_i s_i$ and $\sum_i w_i s_i$ from white data, they are uncorrelated *only* if the two vectors $\mathbf{v}$ and $\mathbf{w}$ (which contain the entries v_i and w_i, respectively) are orthogonal.

In general, we have the following theoretical result. For white data, multiplication by a square matrix gives white components if *and only if* the matrix is orthogonal. Thus, when we have computed one particular whitening transformation, we also know that *only* orthogonal transformations of the transformed data can be white.

Note here the tricky point in terminology: a matrix if called orthogonal if its columns, or equivalently its rows, are orthogonal, *and* the norms of its columns are all equal to one. To emphasize this, some authors call an orthogonal matrix ortho*normal*. We stick to the word "orthogonal" here.

4.5.3 Whitening exhausts second-order information

Information contained in the covariance matrix is called "second-order" information. It is often said that whitening exhausts second-order information. This is natural because it makes the covariance matrix "trivial", i.e. equal to identity. Another viewpoint is to consider what happens if we do PCA on whitened data. Basically, there is no point in doing PCA because the variance is the same in all directions!

After whitening, the only information that is left in the data is of "higher order" (i.e. higher than second). Later, when talking about independent component analysis, we will see what such higher-order information is.

Chapter 5

Factor analysis

5.1 Formulation as a generative model

Factor analysis (FA) can be seen as a more principled version of PCA in the sense that we do not just arbitrarily decide to look at direction of maximum variance, but we formulate a statistical model from which such maximization comes out naturally. It can also be seen as the theoretical bridge between PCA and independent component analysis, which will be the topic later.

In FA, we model the observed data using a model which "generates" it from latent (hidden, hypothetical) variables, as follows

$$x_i = \sum_{j=1}^{m} a_{ij} s_j + n_i \text{ for all } i = 1 \dots n \tag{5.1}$$

or in matrix form

$$\mathbf{x} = \sum_{i=1}^{m} \mathbf{a}_i s_i + \mathbf{n} \quad \text{or} \quad \mathbf{x} = \mathbf{A}\mathbf{s} + \mathbf{n} \tag{5.2}$$

Here, we are still assuming that the x_i have zero mean, which means we can assume the same of s_i. In the model,

- the s_j are latent random variables, called factors. They are *uncorrelated*, and have unit variance by definition; in other words, $\mathbf{s}$ is white. It is essential that the number of factors m is smaller than the number of observed variables.
- the a_{ij} are parameters which tell how "strongly" each factor contributes to each observed variable. These are traditionally called "factor loadings".
- the n_i are variables which explain the part of the data not explained by the factors. They are uncorrelated of $\mathbf{s}$, and uncorrelated with each other. We denote the diagonal covariance matrix of $\mathbf{n}$ by $\mathbf{D}$. They can be thought of as noise in many cases. Traditionally, they are called "specific factors" whereas the s_j are called "common factors".

An interesting property of factor loadings is that they give the covariances between the observed variables and the factors: $\text{cov}(x_i, s_j) = a_{ij}$.

5.2 Gaussianity assumption

To have a properly defined probability distribution, the assumptions on the correlations given above are not sufficient. We have to also assume the explicit probability densities for the quantities involved. The classic assumption is that the $\mathbf{s}$ and $\mathbf{n}$ follow a joint Gaussian (i.e. normal) distribution. This multivariate Gaussian distribution has a density function equal to

$$p(x_1, \dots, x_n) = \frac{1}{(2\pi)^{n/2}(\det \mathbf{C})^{1/2}} \exp(-\frac{1}{2} \sum_{i,j} x_i x_j [\mathbf{C}^{-1}]_{ij}) = \frac{1}{(2\pi)^{n/2}(\det \mathbf{C})^{1/2}} \exp(-\frac{1}{2} \mathbf{x}^T \mathbf{C}^{-1} \mathbf{x}) \tag{5.3}$$

where $\mathbf{C}$ is a positive-definite parameter matrix (equal to the covariance matrix of the data), $\mathbf{C}^{-1}$ is its inverse, and $[\mathbf{C}^{-1}]_{ij}$ is the i, j-th element of the inverse. Thus, the probability distribution is completely characterized by the covariances (as always, the means are assumed zero here).

In classic factor analysis, it is usually assumed that the interesting structure of the data is in the covariances. In fact, such methods typically use only information contained in the covariance matrix of the sample. The theoretical justification for

considering only the sample covariance matrix is that if the data is jointly Gaussian, the covariance matrix of the data contains all the information on the distribution.[1]

Covariance-based methods are thus perfectly sufficient if the distribution of the data is gaussian. However, methods based on the gaussian distribution may neglect some of the most important aspects of the data, as will be seen in the next chapter.

5.3 Factor rotation problem

In fact, under the gaussianity assumption, the estimation of the factor analysis model cannot be unique: there is a rotation ambiguity, Consider any orthogonal matrix $\mathbf{U}$. We can transform the factors and factor loading matrix by this rotation, and we get

$$\mathbf{x} = (\mathbf{AU})(\mathbf{U}^T\mathbf{s}) + \mathbf{n} \tag{5.4}$$

Now, $\mathbf{s}$ is white, and an orthogonal rotation of a white random vector is still white. On the other hand, the distribution of a zero-mean Gaussian random vector is completely defined by its covariance. Thus, $\mathbf{U}^T\mathbf{s}$ has exactly the same distribution as $\mathbf{s}$. This means that there is no way we could distinguish between the models (5.2) and (5.4) (unless we make some more assumptions, for example on $\mathbf{A}$, which will be considered below).

So, just like in the case of PCA, especially when PCA is defined based on dimension reduction (Section 4.3), we see that FA really only defines a *subspace* in the data space, and the actual values in the matrix $\mathbf{A}$ are not meaningful.

5.4 Least-squares estimation

The parameters to estimate in the model are basically $\mathbf{A}$ and noise covariance $\mathbf{D}$. We know that $\mathbf{A}$ can be estimated only up to a rotation.

A simple estimation method for FA is based on a least-squares (LS) criterion. The fundamental idea is to express the covariance matrix of $\mathbf{x}$ as

$$\text{cov}(\mathbf{x}) = \mathbf{A}\mathbf{A}^T + \mathbf{D} \tag{5.5}$$

where $\mathbf{D}$ is the covariance matrix of the n_i. Thus, FA is seen as approximating the sample covariance. This obviously leads to the proposal of a LS criterion

$$J_{LS} = \|\text{cov}(\mathbf{x}) - \mathbf{A}\mathbf{A}^T - \mathbf{D}\|^2 \tag{5.6}$$

(In this norm, the matrices are considered as vectors, so the squared norm is just the sum of squares.) Note that this approximation does not make any sense unless the number of factors is small: If the number of factors is the same as the number of observed variables, the approximation can always be made exact even with a zero $\mathbf{D}$.

5.4.1 Connection between PCA and FA

The LS estimation criterion J_{LS} shows a direct connection to PCA. If we know the covariance $\mathbf{D}$, the objective can be written simply as

$$J_{LS} = \|(\text{cov}(\mathbf{x}) - \mathbf{D}) - \mathbf{A}\mathbf{A}^T\|^2 \tag{5.7}$$

Based on the optimal dimension reduction property of PCA, we can prove that this is essentially like PCA but we just apply the eigen-value decomposition on the matrix $\text{cov}(\mathbf{x}) - \mathbf{D}$ instead of the original covariance matrix. Denote $\text{cov}(\mathbf{x}) - \mathbf{D} = \mathbf{C}$. A variant of the PCA theory says that the minimum of

$$J_{LS} = \|\mathbf{C} - \mathbf{A}\mathbf{A}^T\|^2 \tag{5.8}$$

for any given symmetric positive semi-definite matrix $\mathbf{C}$ is obtained when the columns of $\mathbf{A}$ (which is constrained to have fewer columns than rows) are the first eigenvectors of $\mathbf{C}$ multiplied by some scaling constants. Again, the general proof is very complicated, but the basic intuition can be obtained by considering the special case of a single factor, $m = 1$, where we denote $\mathbf{A} = \mathbf{a}$. Consider the norm in (5.8) as a dot-product:

$$J_{LS} = \|\mathbf{C} - \mathbf{a}\mathbf{a}^T\|^2 = \langle \mathbf{C} - \mathbf{a}\mathbf{a}^T\,,\, \mathbf{C} - \mathbf{a}\mathbf{a}^T \rangle \tag{5.9}$$

To manipulate this, we need the fundamental formula for the dot-product between two matrices (Prove it!):

$$\langle \mathbf{M}_1, \mathbf{M}_2 \rangle = \text{tr}(\mathbf{M}_1^T\mathbf{M}_2) = \text{tr}(\mathbf{M}_1\mathbf{M}_2^T) \tag{5.10}$$

Thus, we obtain

$$J_{LS} = \text{tr}\left(\mathbf{C}\mathbf{C}^T + \mathbf{a}\mathbf{a}^T\mathbf{a}\mathbf{a}^T - 2\mathbf{C}\mathbf{a}\mathbf{a}^T\right) = \|\mathbf{a}\|^4 - 2\mathbf{a}^T\mathbf{C}\mathbf{a} + \text{tr}(\mathbf{C}\mathbf{C}) \tag{5.11}$$

[1] It is a sufficient statistics in terms of estimation theory

We recognize the same quadratic form $\mathbf{a}^T\mathbf{C}\mathbf{a}$ which we wanted to *maximize* in PCA (in PCA, $\mathbf{C}$ was specifically the covariance matrix). Here, we minimize its negative which is the same. The difference to the PCA case is that in PCA, we constrain $\|\mathbf{a}\| = 1$ whereas here, we have instead a penalty term $\|\mathbf{a}\|^4$. We will not go into details, but it can be proven that the penalty has the same effect as the norm constraint, and thus minimization of J_{LS} is equivalent to PCA, i.e. taking as $\mathbf{a}$ the eigenvector of $\mathbf{C} = \text{cov}(\mathbf{x}) - \mathbf{D}$ with the largest eigenvalue.

In fact, the connection does not require that we know the noise covariance $\mathbf{D}$. If the noise is white, i.e. $\mathbf{D} = \sigma^2\mathbf{I}$, the eigenvectors of $\text{cov}(\mathbf{x})$ and $\text{cov}(\mathbf{x}) - \mathbf{D}$ are equal (only the eigenvalues are different). Thus, if the noise (or "specific factors") is white, the factors and principal components are the same.

The main difference between PCA and FA is thus in the possibility to have different "noise levels" for different observed variables, i.e. the matrix $\mathbf{D}$ might have different values in its diagonal entries. This is useful in some applications, in which, for example, one sensor can be faulty and thus give a x_i which has erroneous values, and even with a very high variance. Moreover, FA gives a properly defined probabilistic model (a probability density function) of the data. However, PCA is often preferred because of its simplicity.[2]

5.4.2 Likelihood of the model

It is also possible to formulate the likelihood of the factor analysis model. However, the formulae get quite complicated and do not bring anything which is really interesting for us in this context. It can be proven that maximization of likelihood reduces to PCA under some constraints similar to those we had above but the derivations are more complicated.

5.5 Classic factor rotation methods

Since the factor analysis does not have a unique solution, many methods have been developed to choose the "best" one among them. Basically, the methods can be divided into two groups:

- heuristic methods based on simplifying the structure of the factor loading matrix $\mathbf{A}$,
- methods based on statistical properties of the factors. In particular, if the factors are non-Gaussian, the factor rotation is uniquely defined. This is the basis of independent component analysis, treated later.

Here, we will consider heuristic methods based on the structure of $\mathbf{A}$, typically called "factor rotations" in the literature.

The methods are based on *sparsity* of $\mathbf{A}$. A more thorough treatment of sparsity will be given in the ICA context, Here, the basic idea is that *as many of the factor loadings a_{ij} as possible should be essentially zero* . "Essentially" is here in a purely intuitive sense and doesn't have a rigorous definition.

The reason for seeking a rotation which makes many elements in $\mathbf{A}$ essentially zero is to make interpretation simpler. Ideally, after rotation we would have a matrix which shows clear and simple connections between the observed variables and the factors. For example, assume that the initially estimated matrix $\mathbf{A}$ is

$$\mathbf{A} = \begin{pmatrix} -0.9511 & 0.9511 \\ -1.6435 & -1.6435 \\ 2.3655 & 2.3655 \\ -2.9154 & -2.9154 \\ -3.7010 & 3.7010 \end{pmatrix} \tag{5.12}$$

Now, after judiciously chosen rotation, given by $\mathbf{A}\mathbf{U}$ for some orthogonal matrix $\mathbf{U}$, this matrix in fact becomes

$$\mathbf{A}^* = \begin{pmatrix} 0 & 1.3450 \\ -2.3243 & 0 \\ 3.3453 & 0 \\ -4.1230 & 0 \\ 0 & 5.2340 \end{pmatrix} \tag{5.13}$$

(of course, the values can become exactly zero only in an artificial example, not with real data.) We see that with the matrix $\mathbf{A}^*$, the interpretation becomes much simpler. In fact, we now can classify the observed variables as related to only one of the factors: variables x_1 and x_5 are only related to s_2, and the rest are only related to s_1. Obviously, the interpretation of the factors is easier.

[2] A minor difference between the two methods is in scaling of the components: in FA we define their variance to be equal to one, while in PCA the norms of $\mathbf{w}_i$ are defined to be one.

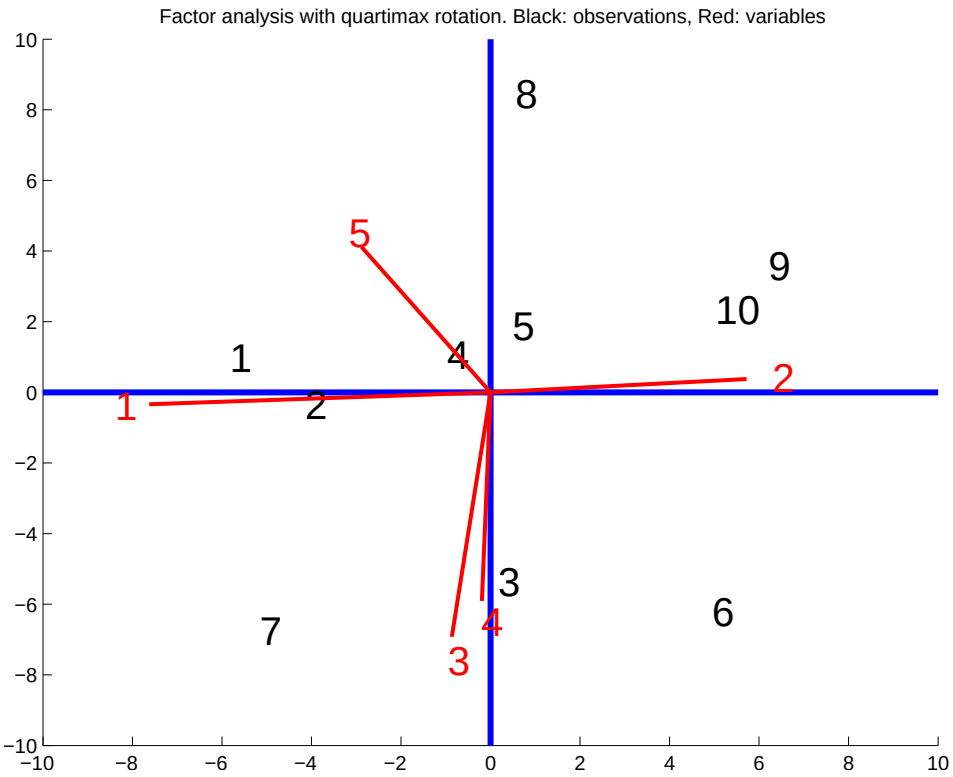

Figure 5.1: Illustration of a classic factor rotation.

How do we make a factor rotation? Typically, we maximize a sum of some function of the a_{ij}

$$J_{rot}(\mathbf{U}) = \sum_{ij} G([\mathbf{AU}]_{ij}) \tag{5.14}$$

where G is a function to be chosen, and $\mathbf{A}$ is the initially estimated factor loading matrix, and $\mathbf{U}$ is an orthogonal $m \times m$ matrix.

The first thing to point out is that we cannot take a quadratic function $G(y) = y^2$, because then J_{rot} is constant (Prove it!). A classic solution is to take the fourth power:

$$G(y) = y^4 \tag{5.15}$$

This is called the *quartimax* rotation. The motivation is that the fourth power encourages large absolute values for factor loadings. Because the sum of squares is constant, such large values imply that some other loadings must be very small, hopefully very close to zero.

Many other rotations exist as well, but the basic idea is usually the same: making the structure of $\mathbf{A}$ as simple as possible.[3]

Let's go back to the illustration in Chapter 4. For simplicity, let us assume that the factor matrix $\mathbf{A}$ is the same as the principal components, do $\mathbf{A} = (\mathbf{u}_1, \mathbf{u}_2)$. (So, note that we normalize the $\mathbf{A}$ as in PCA and not as typically in factor analysis.) We can rotate the two eigenvectors $\mathbf{u}_1$ and $\mathbf{u}_2$ using quartimax to obtain

$$\mathbf{u}_1^* = \begin{pmatrix} -0.7625 & 0.5724 & -0.0865 & -0.0191 & -0.2882 \end{pmatrix}^T \tag{5.16}$$

and

$$\mathbf{u}_2^* = \begin{pmatrix} -0.0338 & 0.0377 & -0.6922 & -0.5909 & 0.4113 \end{pmatrix}^T \tag{5.17}$$

Again, we can plot each observation by projecting it on the coordinate axes given by $\mathbf{u}_1^*$ and $\mathbf{u}_2^*$. Moreover, we can plot the original variables, simply by using the values in $\mathbf{u}_1^*$ and $\mathbf{u}_2^*$. See Figure 5.1 for the results.

The difference to the original PCA case is that now the factors are more aligned with the observed variables. In particular, factor #1 (horizontal axis) is mainly describing variables #1 and #2, and factor #2 is mainly describing variables #3 and #4. Variable #5 is, however, partly represented in both factors.

[3] I'm not completely sure if the rotation given here is exactly what is called the quartimax rotation. One possible difference is in the way $\mathbf{A}$ is normalized: as pointed out above, it can be normalized so that its columns are orthogonal, or so that the s_i are white, and this will change the maxima of the objective function.

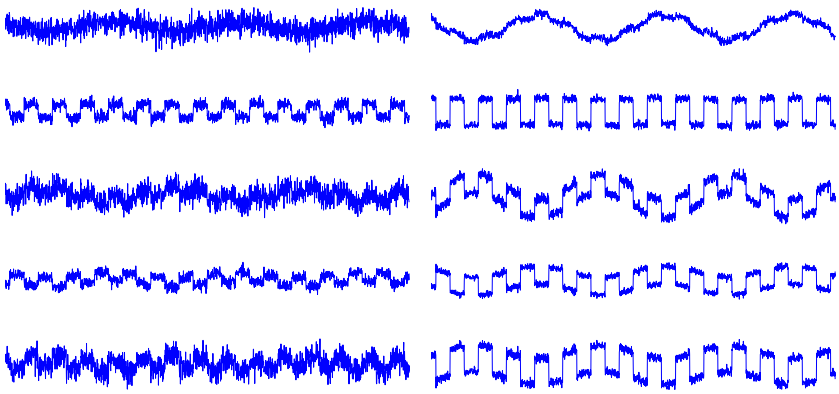

Figure 5.2: Illustration of denoising. Left: some of the original signals. Right: denoised versions of those signals.

5.6 PCA/FA and noise reduction

Often in signal processing applications it is assumed that PCA or FA can reduce noise (e.g. coming from physical measurement devices). This is based on the assumption that when we compute the projection on the columns of $\mathbf{A}$ (or, on the principal component axes), the noise $\mathbf{n}$ is largely projected out. The factors part of the model $\sum_i \mathbf{a}_i s_i$ is then interpreted as corresponding to the "real" signal in the system, and the $\mathbf{n}$ are just noise. Thus, we compute

$$\tilde{\mathbf{x}} = \mathbf{A}\mathbf{A}^T\mathbf{x} \tag{5.18}$$

which is a denoised version of $\mathbf{x}$, with $\mathbf{A}$ being estimated by FA or PCA. (We assume here for simplicity that $\mathbf{A}$ has orthogonal columns as typical in PCA, so we can "invert" it by taking the transpose. In FA the normalization is different.)

The point is that we are ignoring the part of the data which is orthogonal to the "signal subspace", which is spanned by the columns of $\mathbf{A}$, and estimated by PCA. If the dimension of the signal subspace is small, the total amount of noise in the system can be reduced quite a lot.

Another interpretation of this operation is that we estimate the factors as $\hat{\mathbf{s}} = \mathbf{A}^T\mathbf{x}$, and then reconstruct the noise-free data as $\tilde{\mathbf{x}} = \mathbf{A}\mathbf{s}$.

Consider, for example that there are two "real" factors in a 100-dimensional space. Assume the noise variance is constant for all i (i.e. noise covariance is identity times a constant). We cannot get rid of the noise which is inside the signal subspace, i.e. the projection of the noise on the that subspace. But the amount of noise in the signal subspace is only 2% of the total noise variance, so we get rid of 98% of the noise by doing (5.18).

The reason why this works is that projection is essentially setting most of the coordinates to zero. Consider a coordinate system $z_1, \ldots, z_{100}$, in which the two first coordinates are in the signal subspace and the rest in the noise subspace. Assume further that the noise covariance is of the form $\sigma^2\mathbf{I}$, in which case the covariance of the noise in this new coordinate system (denote it by $\tilde{n}_i$ has the same covariance. This means that we have

$$z_1 = s_1 + \tilde{n}_1 \tag{5.19}$$
$$z_2 = s_2 + \tilde{n}_2 \tag{5.20}$$
$$z_3 = n_3 \tag{5.21}$$
$$z_4 = n_4 \tag{5.22}$$
$$\vdots \tag{5.23}$$
$$z_{100} = n_{100} \tag{5.24}$$

Now, the projection onto the signal subspace means simply setting the z_i to zero for all $i > 2$. This obviously reduces the noise quite a lot.

This is illustrated in Fig. 5.2. We generated 50 artificial signals according to the factor analysis model (only 5 are plotted to save space). They really only contain two interesting signals (factors), and the rest is noise. After denoising, the signals underlying the observed data are more apparent.

Why is it that you cannot just take the mean of all the signals? In some cases this is possible. However, the signals may have different signs in different signals (the a_{ij} can be negative), and they have different strengths in different signals, so PCA/FA can find the optimal weighting of the signals.

5.7 Effect of scaling the variables

An important question in practice is scaling of the variables. It is important to understand that the results of PCA and FA can change a lot with scaling.

In particular, it is often useful to rescale all the variables to unit norm, i.e. to standardize them. Then, the covariance matrix is really the matrix of the *correlation coefficients*.

If you do PCA on the original covariance matrix (i.e. without scaling the variables) or on the correlation matrix (i.e. standardizing the variables), are the results same, or is there a simple connection between the components? The answer is "no".

Consider a diagonal matrix $\mathbf{D}$ which rescales the variables to obtain the scaled variables as

$$\mathbf{y} = \mathbf{D}\mathbf{x} \tag{5.25}$$

Let's do PCA on $\mathbf{y}$. We have

$$\text{cov}(\mathbf{y}) = \mathbf{D}\text{cov}(\mathbf{x})\mathbf{D} \tag{5.26}$$

and thus the optimization problem

$$\max_{\|\mathbf{w}\|=1} \mathbf{w}^T \text{cov}(\mathbf{y})\mathbf{w} = \max_{\|\mathbf{w}\|=1} \mathbf{w}^T [\mathbf{D}\text{cov}(\mathbf{x})\mathbf{D}]\mathbf{w} \tag{5.27}$$

So, the solution is to do the eigenvalue decompostion on the matrix $\mathbf{D}\text{cov}(\mathbf{x})\mathbf{D}$ instead of $\text{cov}(\mathbf{x})$. Is there a simple connection between the two eigen-value decompositions? No: The eigenvectors can be quite different. As a simple counterexample, assume $\text{cov}(\mathbf{x})$ is diagonal, with no two diagonal elements equal to each other, and in descending order. Then, the eigenvalue decomposition is obtained as the vectors

$$\mathbf{e}_i = (0, 0, 0, 1, 0, 0, 0)^T \tag{5.28}$$

where the 1 is in the i-th place. Now, a general $\mathbf{D}$ can change them so that the largest elements become the smallest, and vice versa. Thus, the last principal component can become the first. In particular, consider what happens if we normalize the variances to unity. Then $\text{cov}(\mathbf{y})$ equals identity. Thus, the eigenvectors are not defined at all.

One interpretation is that scaling of the variables changes the geometry of the space. On the other hand, the constraint of unit norm is still there, and not changed. So, in the new scaling, the meaning of the constraint of unit norm is no longer the same, since the constraint is not rescaled with the variables.

So, we see that scaling of the variables can change the principal components quite a lot. In factor analysis, the results can change or not depending on the particular factor analysis method used.

5.8 Choosing the number of principal components or factors

Another important question in practice is how to choose the number of principal components or factors. This is a very difficult question to which it is difficult to give any clear answers. In some cases, you know what the right dimension is: for example in the case of visualization, you usually use two dimensions.

A heuristic solution is to take enough principal components to explain a certain proportion of the variance, for example 90%.

There have been attempts to provide some principled ways of choosing the number of principal components based on (mainly) Bayesian methods of model selection. However, they are not very often used in practice because they often seem to give bad answers.

Chapter 6

Independent Component Analysis: Definition

6.1 Motivation

A classic (if somewhat unrealistic) motivation of independent component analysis (ICA) is the following. Imagine that you are in a room where three people are speaking simultaneously. (The number three is completely arbitrary, it could be anything larger than one.) You also have three microphones, which you hold in different locations. The microphones give you three recorded time signals, which we could denote by $x_1(t), x_2(t)$ and $x_3(t)$, with x_1, x_2 and x_3 the amplitudes, and t the time index. Each of these recorded signals is a weighted sum of the speech signals emitted by the three speakers, which we denote by $s_1(t), s_2(t)$, and $s_3(t)$. We could express this as a linear equation:

$$x_1(t) = a_{11}s_1(t) + a_{12}s_2(t) + a_{13}s_3(t) \tag{6.1}$$

$$x_2(t) = a_{21}s_1(t) + a_{22}s_2(t) + a_{23}s_3(t) \tag{6.2}$$

$$x_3(t) = a_{31}s_1(t) + a_{32}s_2(t) + a_{33}s_3(t) \tag{6.3}$$

where the a_{ij} with $i, j = 1, ..., 3$ are some parameters that depend on the distances of the microphones from the speakers. It would be very useful if you could now estimate the original speech signals $s_1(t), s_2(t)$, and $s_3(t)$, using only the recorded signals $x_i(t)$. This is called the *cocktail-party problem*. For the time being, we omit any time delays or other extra factors from our simplified mixing model.

As an illustration, consider the waveforms in Fig. 6.1. The original speech signals could look something like those on the left, and the mixed signals could look like those in the middle. The problem is to recover the "source" signals on the left using only the data in the middle column.

Actually, if we knew the mixing parameters a_{ij}, we could solve the linear equation in (6.1) simply by inverting the linear system.

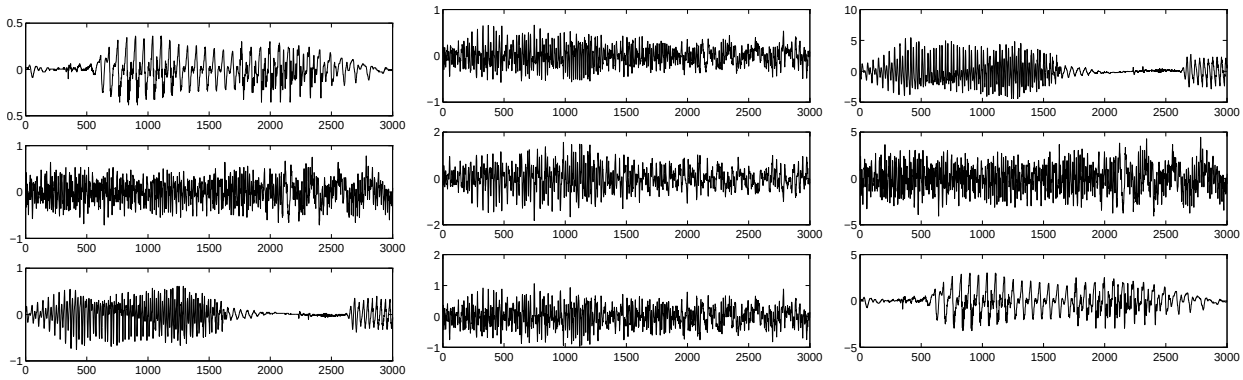

Figure 6.1: Left: The original audio signals. Middle: The observed mixtures of the original signals. Right: The estimates of the original signals, obtained by ICA using only the observed signals in the middle. The original signals were very accurately estimated, up to multiplicative signs.

The point is, however, that here we know *neither* the a_{ij} *nor* the $s_i(t)$, so the problem is considerably more difficult.

One approach to solving this problem would be to use some information on the statistical properties of the signals $s_i(t)$ to estimate both the a_{ij} and the $s_i(t)$. Actually, and perhaps surprisingly, it turns out that it is enough to assume that $s_1(t), s_2(t)$, and $s_3(t)$ are, at each time instant t, *statistically independent*. This is not an unrealistic assumption in many cases, and it need not be exactly true in practice. Independent component analysis can be used to estimate the a_{ij} based on the information of their independence, and this allows us to separate the three original signals, $s_1(t)$, $s_2(t)$, and $s_3(t)$, from their mixtures, $x_1(t)$, $x_2(t)$, and $x_2(t)$.

Figure 6.1, on the right gives the three signals estimated by the ICA methods discussed in the next chapters. As can be seen, these are very close to the original source signals (the signs of some of the signals are reversed, but this has no significance.) These signals were estimated using only the mixtures in the middle column, together with the very weak assumption of the independence of the source signals.

Independent component analysis was originally developed to deal with problems that are closely related to the cocktail-party problem. It has been found out that the cocktail-party problem with real audio signals is actually much more difficult due to time delays, echos and reverberation in the room. At the same time, it has become clear that this principle has a lot of other interesting applications as well, some of which are considered in Chapter 10.

6.1.1 ICA as estimation of a generative model

To rigorously define ICA, we can use a statistical latent variables model, which is very closely related to the factor analysis model. We observe n random variables $x_1, ..., x_n$, which are modeled as linear combinations of n random variables $s_1, ..., s_n$:

$$x_i = a_{i1}s_1 + a_{i2}s_2 + ... + a_{in}s_n, \quad \text{for all } i = 1, ..., n \tag{6.4}$$

where the $a_{ij}, i, j = 1, ..., n$ are some real coefficients. Note the basic differences to the factor analysis model

1. The number of components (factors) is equal to the number of observed variables. This is basically assumed for the sake of mathematical simplicity: the matrix $\mathbf{A}$ is then invertible. It is related to the idea that we are estimating a factor rotation after PCA/FA, so we don't want to reduce the dimension of the data anymore.
2. There is no noise term. This is partly justified by the large number of components: some of them can be noise whereas others are more interesting components

In addition, the statistical assumptions on the components are quite different, as will be discussed in the next subsection.

Note that we have here dropped the time index t that was used in the previous section. This is because in the basic ICA model, we assume that each mixture x_i as well as each independent component s_j is a random variable, instead of a proper time signal or time series.

ICA is very closely related to the method called *blind source separation* (BSS) or blind signal separation. A "source" means here an original signal, i.e., independent component, like the speaker in the cocktail-party problem. "Blind" means that we know very little, if anything, of the mixing matrix, and make very weak assumptions on the source signals. ICA is one method, perhaps the most widely used, for performing blind source separation.

Using the same vector notation as in factor analysis, the ICA model is written as

$$\mathbf{x} = \sum_{i=1}^{n} \mathbf{a}_i s_i \tag{6.5}$$

or

$$\mathbf{x} = \mathbf{A}\mathbf{s} \tag{6.6}$$

where the matrix $\mathbf{A}$, often called the "mixing" matrix, is square.

6.1.2 Model assumptions in ICA

To make sure that the basic ICA model just given can be estimated, we have to make certain assumptions and restrictions.

1. The independent components are assumed statistically *independent*.

This is of course where the name of the method comes from. Surprisingly, not much more than this assumption is needed to ascertain that the model can be estimated. This is why ICA is such a powerful method with applications in many different areas.

Basically, random variables $y_1, y_2, ..., y_n$ are said to be independent if information on the value of y_i does not give any information on the value of y_j for $i \neq j$. Technically, independence can be defined by the probability densities. Let us denote by

$p(y_1, y_2, ..., y_n)$ the joint probability density function (pdf) of the y_i, and by $p_i(y_i)$ the marginal pdf of y_i, i.e., the pdf of y_i when it is considered alone. Then we say that the y_i are *independent* if and only if the joint pdf is factorizable in the following way:

$$p(y_1, y_2, ..., y_n) = p_1(y_1)p_2(y_2)...p_n(y_n). \tag{6.7}$$

2. The independent components must have *nongaussian* distributions.

This is what distinguishes ICA from (classical) factor analysis. Intuitively, one can say that the gaussian distributions are "too simple". The distribution is completely characterized by the covariance matrix which is not enough to estimate $\mathbf{A}$ as we saw already in Chapter 5, in the form of the factor rotation problem. Thus, ICA is essentially impossible if the observed variables have gaussian distributions. The reasons will be treated in more detail in Section 6.4 below. Note that in the basic model we do *not* assume that we know what the nongaussian distributions of the ICs look like; if they are known, the problem will be considerably simplified.

3. For simplicity, we assume that the unknown mixing matrix is *square* and invertible.

In other words, the number of independent components is equal to the number of observed mixtures. This assumption can sometimes be relaxed. We make it here because it simplifies the estimation very much. Then, after estimating the matrix $\mathbf{A}$, we can compute its inverse, say $\mathbf{B}$, and obtain the independent components simply by

$$\mathbf{s} = \mathbf{B}\mathbf{x} \tag{6.8}$$

It is also assumed here that the mixing matrix is *invertible*. If this is not the case, there are redundant mixtures that could be omitted, in which case the matrix would not be square; then we find again the case where the number of mixtures is not equal to the number of ICs.

Thus, under the preceding three assumptions (or at the minimum, the two first ones), the ICA model is identifiable, meaning that the mixing matrix and the ICs can be estimated up to some trivial indeterminacies that will be discussed next. We will not prove the identifiability of the ICA model here, since the proof is quite complicated. On the other hand, in the next chapter we develop estimation methods, and the developments there give a kind of a nonrigorous, constructive proof of the identifiability.

6.2 Illustration of ICA

To illustrate the ICA model in statistical terms, consider two independent components that have the following uniform distributions:

$$p(s_i) = \begin{cases} \frac{1}{2\sqrt{3}}, & \text{if } |s_i| \leq \sqrt{3} \\ 0, & \text{otherwise} \end{cases} \tag{6.9}$$

The range of values for this uniform distribution were chosen so as to make the mean zero and the variance equal to one, as was agreed in the previous section. The joint density of s_1 and s_2 is then uniform on a square. This follows from the basic definition that the joint density of two independent variables is just the product of their marginal densities (see Eq. (6.7)): we simply need to compute the product. The joint density is illustrated in Fig. 6.2 (left) by showing data points randomly drawn from this distribution.

Now let us mix these two independent components. Let us take the following mixing matrix:

$$\mathbf{A}_0 = \begin{pmatrix} 5 & 10 \\ 10 & 2 \end{pmatrix} \tag{6.10}$$

This gives us two mixed variables, x_1 and x_2. It is easily computed that the mixed data has a uniform distribution on a parallelogram, as shown in Fig. 6.2 on the right. Note that the random variables x_1 and x_2 are not independent anymore; an easy way to see this is to consider whether it is possible to predict the value of one of them, say x_2, from the value of the other. Clearly, if x_1 attains one of its maximum or minimum values, then this completely determines the value of x_2. They are therefore not independent. (For variables s_1 and s_2 the situation is different: from Fig. 6.2 on the left it can be seen that knowing the value of s_1 does not in any way help in guessing the value of s_2.)

The problem of estimating the data model of ICA is now to estimate the mixing matrix $\mathbf{A}$ using only information contained in the mixtures x_1 and x_2. Actually, from Fig. 6.2 you can see an intuitive way of estimating $\mathbf{A}$: The *edges* of the parallelogram are in the directions of the columns of $\mathbf{A}$. This means that we could, in principle, estimate the ICA model by first estimating the joint density of x_1 and x_2, and then locating the edges. So, the problem seems to have a solution.

On the other hand, consider a mixture of ICs with a different type of distribution, called supergaussian. Supergaussian random variables typically have a pdf with a peak at zero. The marginal distribution of such an IC is given in Fig. 6.3. The joint distribution of the original independent components is given in Fig. 6.4 on the left, and the mixtures are shown in Fig. 6.4 on the right. Here, we see some kind of edges, but in very different places this time.

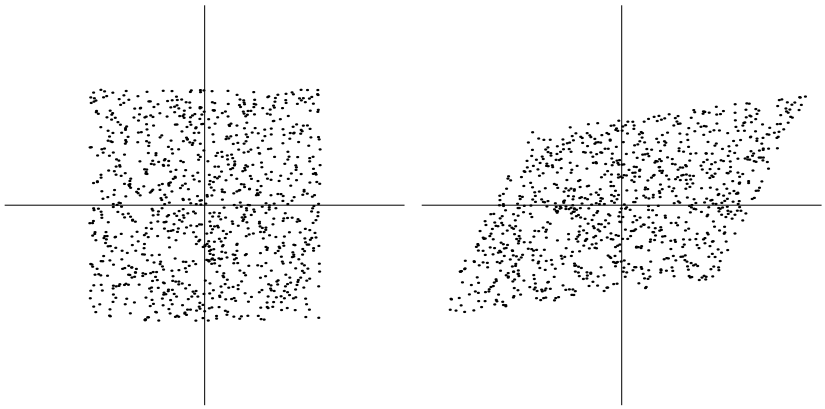

Figure 6.2: *Left*: The joint distribution of the independent components s_1 and s_2 with uniform distributions. Horizontal axis: s_1, vertical axis: s_2. *Right*: The joint distribution of the observed mixtures x_1 and x_2. Horizontal axis: x_1, vertical axis: x_2. (Not in the same scale)

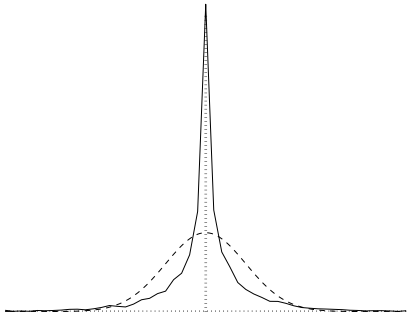

Figure 6.3: The density of one supergaussian independent component. The gaussian density if give by the dashed line for comparison.

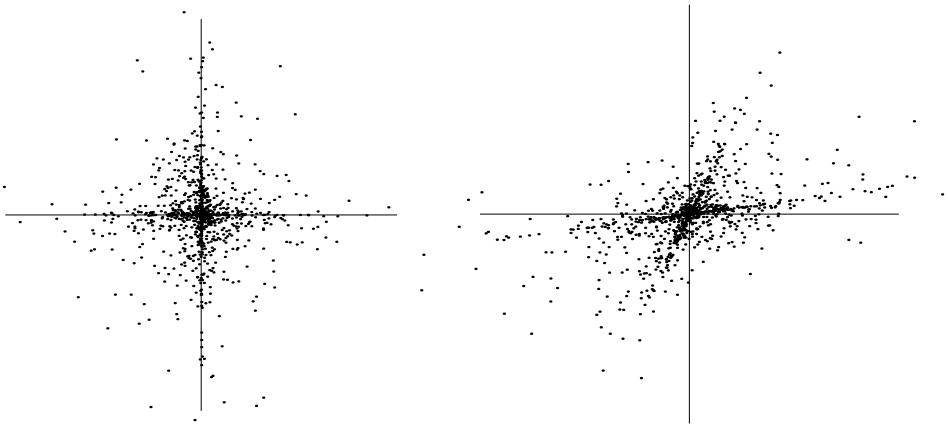

Figure 6.4: *Left*: The joint distribution of the independent components s_1 and s_2 with supergaussian distributions. Horizontal axis: s_1, vertical axis: s_2. *Right*: The joint distribution of the observed mixtures x_1 and x_2, obtained from supergaussian independent components. Horizontal axis: x_1, vertical axis: x_2.

In practice, however, locating the edges would be a very poor method because it only works with variables that have very special distributions. For most distributions, such edges cannot be found; we use only for illustration purposes distributions that visually show edges. Moreover, methods based on finding edges, or other similar heuristic methods, tend to be computationally quite complicated, and unreliable.

What we need is a method that works for any distributions of the independent components, and works fast and reliably. Such methods are treated later. In the rest of this chapter, however, we discuss the connection between ICA and whitening.

6.3 ICA is stronger than whitening

Given some random variables, it is straightforward to linearly transform them into uncorrelated variables, as shown in Section 4.5. Therefore, it would be tempting to try to estimate the independent components by such a method, which is typically a form of whitening, and often implemented by principal component analysis. In this section, we show that this is not possible, and discuss the relation between ICA and decorrelation methods. It will be seen that whitening is, nevertheless, a useful preprocessing technique for ICA.

6.3.1 Uncorrelatedness and independence

Two random variables y_1 and y_2 are said to be *uncorrelated*, if their covariance is zero:

$$\mathrm{cov}(y_1, y_2) = E\{y_1 y_2\} - E\{y_1\}E\{y_2\} = 0 \tag{6.11}$$

In these lecture notes, all random variables are assumed to have zero mean, unless otherwise mentioned. Thus, covariance is equal to $= E\{y_1 y_2\}$.

If random variables are independent, they are uncorrelated. This is because if the y_1 and y_2 are independent, then for any two functions, h_1 and h_2, we have

$$E\{h_1(y_1)h_2(y_2)\} = E\{h_1(y_1)\}E\{h_2(y_2)\} \tag{6.12}$$

Taking $h_1(y_1) = y_1$ and $h_2(y_2) = y_2$, we see that this implies uncorrelatedness.

On the other hand, uncorrelatedness does *not* imply independence. For example, assume that (y_1, y_2) are discrete valued and follow such a distribution that the pair are with probability 1/4 equal to any of the following values: $(0, 1), (0, -1), (1, 0)$, and $(-1, 0)$. Then y_1 and y_2 are uncorrelated, as can be simply calculated. On the other hand,

$$E\{y_1^2 y_2^2\} = 0 \neq \frac{1}{4} = E\{y_1^2\}E\{y_2^2\} \tag{6.13}$$

so the condition in Eq. (6.12) is violated, and the variables cannot be independent.

6.3.2 Whitening is only half ICA

Now, suppose that the data in the ICA model is whitened. Whitening transforms the mixing matrix into a new one, $\tilde{\mathbf{A}}$. We have from (6.6)

$$\mathbf{z} = \mathbf{VAs} = \tilde{\mathbf{A}}\mathbf{s} \tag{6.14}$$

One could hope that whitening solves the ICA problem, since whiteness or uncorrelatedness is related to independence. This is, however, not so. Uncorrelatedness is weaker than independence, and is not in itself sufficient for estimation of the ICA model. To see this, consider an *orthogonal* transformation $\mathbf{U}$ of $\mathbf{z}$:

$$\mathbf{y} = \mathbf{Uz} \tag{6.15}$$

Due to the orthogonality of $\mathbf{U}$, we have

$$E\{\mathbf{y}\mathbf{y}^T\} = E\{\mathbf{U}\mathbf{z}\mathbf{z}^T\mathbf{U}^T\} = \mathbf{UIU}^T = \mathbf{I} \tag{6.16}$$

In other words, $\mathbf{y}$ is white as well. Thus, we cannot tell if the independent components are given by $\mathbf{z}$ or $\mathbf{y}$ using the whiteness property alone. Since $\mathbf{y}$ could be any orthogonal transformation of $\mathbf{z}$, *whitening gives the ICs only up to an orthogonal transformation*. This is not sufficient in most applications. Note that this is just a restatement of the factor rotation problem, or the non-uniqueness of whitening, which were discussed earlier.

On the other hand, whitening is useful as a preprocessing step in ICA. The utility of whitening resides in the fact that the *new mixing matrix* $\tilde{\mathbf{A}} = \mathbf{VA}$ *is orthogonal*. This can be seen from

$$E\{\mathbf{z}\mathbf{z}^T\} = \tilde{\mathbf{A}}E\{\mathbf{s}\mathbf{s}^T\}\tilde{\mathbf{A}}^T = \tilde{\mathbf{A}}\tilde{\mathbf{A}}^T = \mathbf{I} \tag{6.17}$$

This means that we can restrict our search for the mixing matrix to the space of orthogonal matrices. Instead of having to estimate the n^2 parameters that are the elements of the original matrix $\mathbf{A}$, we only need to estimate an orthogonal mixing

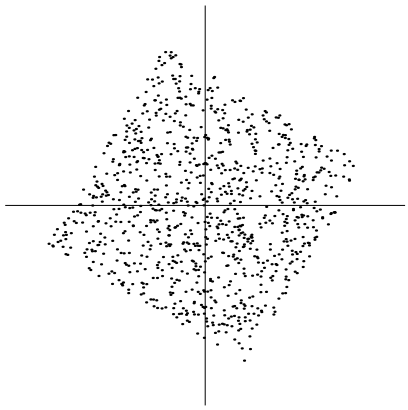

Figure 6.5: The joint distribution of the whitened mixtures of uniformly distributed independent components.

matrix $\tilde{\mathbf{A}}$. An orthogonal matrix contains $n(n-1)/2$ degrees of freedom. For example, in two dimensions, an orthogonal transformation is determined by a single angle parameter. In larger dimensions, an orthogonal matrix contains only about half of the number of parameters of an arbitrary matrix.

Thus one can say that whitening solves half of the problem of ICA. Because whitening is a very simple and standard procedure, much simpler than any ICA algorithms, it is a good idea to reduce the complexity of the problem this way. The remaining half of the parameters has to be estimated by some other method; several will be introduced in the next chapters.

From the viewpoint of factor analysis, we can thus say that ICA is a method for solving the *factor rotation problem*. In ICA, we solve it using the statistical properties of the data, instead of assuming a simple structure for $\mathbf{A}$ as in classic factor rotation methods.

A graphical illustration of the effect of whitening can be seen in Fig. 6.5, in which the data in Fig. 6.2 has been whitened. The square defining the distribution is now clearly a rotated version of the original square in Fig. 6.2. All that is left is the estimation of a single angle that gives the rotation.

In most ICA estimation methods treated below, we will assume that the data has been preprocessed by whitening.

6.3.3 Ambiguities of ICA

Although ICA solves the factor rotation problem, it is easy to see that the following ambiguities or indeterminacies will necessarily hold:

1. We cannot determine the variances (energies) of the independent components.

The reason is that, both $\mathbf{s}$ and $\mathbf{A}$ being unknown, any scalar multiplier in one of the sources s_i could always be canceled by dividing the corresponding column $\mathbf{a}_i$ of $\mathbf{A}$ by the same scalar, say α_i:

$$\mathbf{x} = \sum_i (\frac{1}{\alpha_i}\mathbf{a}_i)(s_i\alpha_i) \tag{6.18}$$

As a consequence, we may quite as well fix the magnitudes of the independent components. Since they are random variables, the most natural way to do this is to assume that each has unit variance: $E\{s_i^2\} = 1$. Then the matrix $\mathbf{A}$ will be adapted in the ICA solution methods to take into account this restriction. Note that this still leaves the *ambiguity of the sign*: we could multiply an independent component by -1 without affecting the model. This ambiguity is, fortunately, insignificant in most applications.

2. We cannot determine the order of the independent components.

The reason is that, again both $\mathbf{s}$ and $\mathbf{A}$ being unknown, we can freely change the order of the terms in the sum in (6.5), and call any of the independent components the first one. Formally, a permutation matrix $\mathbf{P}$ and its inverse can be substituted in the model to give $\mathbf{x} = \mathbf{A}\mathbf{P}^{-1}\mathbf{P}\mathbf{s}$. The elements of $\mathbf{P}\mathbf{s}$ are the original independent variables s_j, but in another order. The matrix $\mathbf{A}\mathbf{P}^{-1}$ is just a new unknown mixing matrix, to be solved by the ICA algorithms.

Without loss of generality, we can assume that both the mixture variables and the independent components have zero mean. If the assumption of zero mean is not true, we can do some preprocessing to make it hold. This is possible by *centering* the observable variables, i.e., subtracting their sample mean.

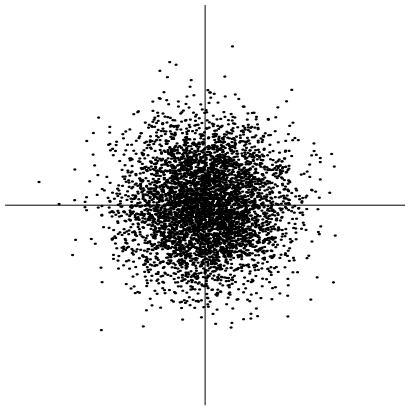

Figure 6.6: The multivariate distribution of two independent gaussian variables.

6.4 Why gaussian variables are forbidden

Whitening also helps us understand why gaussian variables are forbidden in ICA. Assume that the joint distribution of two ICs, s_1 and s_2, is gaussian. This means that their joint pdf is given by

$$p(s_1, s_2) = \frac{1}{2\pi} \exp(-\frac{s_1^2 + s_2^2}{2}) = \frac{1}{2\pi} \exp(-\frac{\|\mathbf{s}\|^2}{2}) \tag{6.19}$$

Now, assume that the mixing matrix $\mathbf{A}$ is orthogonal. For example, we could assume that this is so because the data has been whitened. Then, also $\mathbf{x}$ is gaussian, and white, as shown above. This means that its covariance matrix is equal to identity. Thus, by definition, we get the joint density of the mixtures x_1 and x_2 as density is given by

$$p(x_1, x_2) = \frac{1}{2\pi} \exp(-\frac{\|\mathbf{x}\|^2}{2}) \tag{6.20}$$

and we see that the orthogonal mixing matrix does not change the pdf, since it does not appear in this pdf at all. The original and mixed distributions are identical. Therefore, there is no way how we could infer the mixing matrix from the mixtures.

The phenomenon that the orthogonal mixing matrix cannot be estimated for gaussian variables is related to the property that uncorrelated jointly gaussian variables are necessarily independent. Thus, the information on the independence of the components does not get us any further than whitening.

Graphically, we can see this phenomenon by plotting the distribution of the orthogonal mixtures, which is in fact the same as the distribution of the ICs. This distribution is illustrated in Fig. 6.6. The figure shows that the density is rotationally symmetric. Therefore, it does not contain any information on the directions of the columns of the mixing matrix $\mathbf{A}$. This is why $\mathbf{A}$ cannot be estimated.

Thus, we are back to the factor rotation problem: in the case of *gaussian* independent components, we can only estimate the ICA model up to an orthogonal transformation. In other words, the matrix $\mathbf{A}$ is not identifiable for gaussian independent components.

What happens if we try to estimate the ICA model and *some of the components are gaussian, some nongaussian*? In this case, we *can* estimate all the nongaussian components, but the gaussian components cannot be separated from each other. In other words, some of the estimated components will be arbitrary linear combinations of the gaussian components. Actually, this means that in the case of just one gaussian component, we can estimate the model, because the single gaussian component does not have any other gaussian components that it could be mixed with.

Chapter 7

ICA by maximization of non-gaussianity

7.1 Introduction

Here, we introduce a simple and intuitive principle for estimating the model of independent component analysis (ICA). This is based on maximization of nongaussianity.

Nongaussianity is actually of paramount importance in ICA estimation. Without nongaussianity the estimation is not possible at all, as shown in Chapter 6. Therefore, it is not surprising that nongaussianity could be used as a leading principle in ICA estimation. This is at the same time probably the main reason for the rather late resurgence of ICA research: In most of classic statistical theory, random variables are assumed to have gaussian distributions, thus precluding methods related to ICA.

We start by intuitively motivating the maximization of nongaussianity by the central limit theorem. As a first practical measure of nongaussianity, we introduce the fourth-order cumulant, or kurtosis. Using kurtosis, we derive practical algorithms by gradient and fixed-point methods. Finally, we discuss the connection between these methods and the technique called projection pursuit.

7.2 "Nongaussian is independent"

The central limit theorem is a classic result in probability theory. It says that the distribution of a sum of independent random variables tends toward a gaussian distribution, under certain conditions. Loosely speaking, a sum of two independent random variables usually has a distribution that is closer to gaussian than any of the two original random variables.

Let us now assume that the data vector $\mathbf{x}$ is distributed according to the ICA data model:

$$\mathbf{x} = \mathbf{A}\mathbf{s} \tag{7.1}$$

i.e., it is a mixture of independent components. For pedagogical purposes, let us assume in this motivating section that all the independent components have identical distributions. Estimating the independent components can be accomplished by finding the right linear combinations of the mixture variables, since we can invert the mixing as

$$\mathbf{s} = \mathbf{A}^{-1}\mathbf{x} \tag{7.2}$$

Thus, to estimate one of the independent components, we can consider a linear combination of the x_i. Let us denote this by $y = \mathbf{b}^T\mathbf{x} = \sum_i b_i x_i$, where $\mathbf{b}$ is a vector to be determined. Note that we also have $y = \mathbf{b}^T\mathbf{A}\mathbf{s}$. Thus, y is a certain linear combination of the s_i, with coefficients given by $\mathbf{b}^T\mathbf{A}$. Let us denote this vector by $\mathbf{q}$. Then we have

$$y = \mathbf{b}^T\mathbf{x} = \mathbf{q}^T\mathbf{s} = \sum_i q_i s_i \tag{7.3}$$

If $\mathbf{b}$ were one of the rows of the inverse of $\mathbf{A}$, this linear combination $\mathbf{b}^T\mathbf{x}$ would actually equal one of the independent components. In that case, the corresponding $\mathbf{q}$ would be such that just one of its elements is 1 and all the others are zero.

The question is now: *How could we use the central limit theorem to determine* $\mathbf{b}$ *so that it would equal one of the rows of the inverse of* $\mathbf{A}$*?* In practice, we cannot determine such a $\mathbf{b}$ exactly, because we have no knowledge of matrix $\mathbf{A}$, but we can find an estimator that gives a good approximation.

Let us vary the coefficients in $\mathbf{q}$, and see how the distribution of $y = \mathbf{q}^T\mathbf{s}$ changes. The fundamental idea here is that since a sum of even two independent random variables is more gaussian than the original variables, $y = \mathbf{q}^T\mathbf{s}$ is usually more gaussian than any of the s_i and becomes least gaussian when it in fact equals one of the s_i. (Note that this is strictly true only if the s_i have identical distributions, as we assumed here.) In this case, obviously only one of the elements q_i of $\mathbf{q}$ is nonzero.

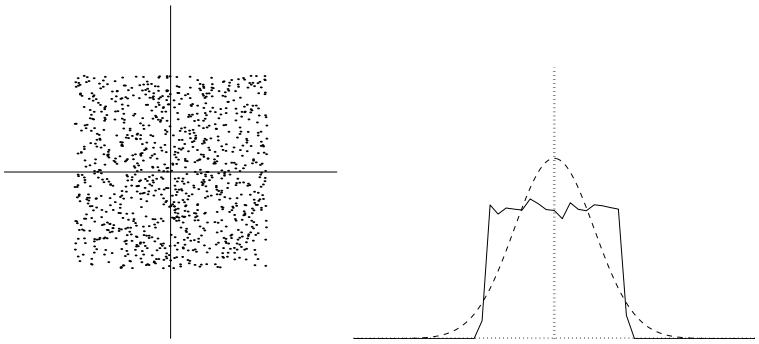

Figure 7.1: *Left*: The joint distribution of two independent components with uniform densities. *Right*: The estimated density of one uniform independent component, with the gaussian density (dashed curve) given for comparison.

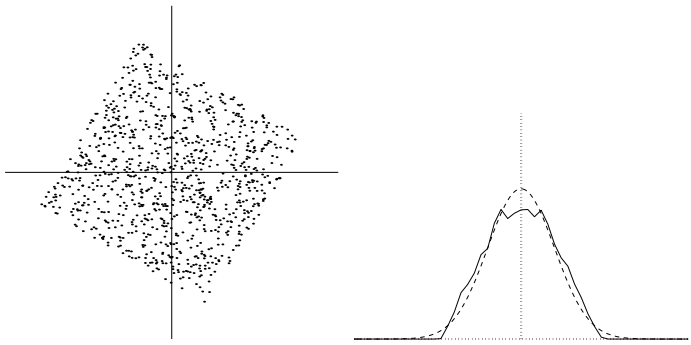

Figure 7.2: *Left:* The joint density of two whitened mixtures of independent components with uniform densities. *Right:* The marginal density of one of the whitened mixtures. It is closer to the gaussian density (given by the dashed curve) than the density of an independent component shown in Fig. 7.1.

We do not in practice know the values of $\mathbf{q}$, but we do not need to, because $\mathbf{q}^T\mathbf{s} = \mathbf{b}^T\mathbf{x}$ by the definition of $\mathbf{q}$. We can just let $\mathbf{b}$ vary and look at the distribution of $\mathbf{b}^T\mathbf{x}$.

Therefore, we could take as $\mathbf{b}$ a vector that *maximizes the nongaussianity* of $\mathbf{b}^T\mathbf{x}$. Such a vector would necessarily correspond to a $\mathbf{q} = \mathbf{A}^T\mathbf{b}$, which has only one nonzero component. This means that $y = \mathbf{b}^T\mathbf{x} = \mathbf{q}^T\mathbf{s}$ equals one of the independent components! Maximizing the nongaussianity of $\mathbf{b}^T\mathbf{x}$ thus gives us one of the independent components.

In fact, the optimization landscape for nongaussianity in the n-dimensional space of vectors $\mathbf{b}$ has $2n$ local maxima, two for each independent component, corresponding to s_i and $-s_i$ (recall that the independent components can be estimated only up to a multiplicative sign).

We can illustrate the principle of maximizing nongaussianity by simple examples. Let us consider two independent components that have uniform densities. Their joint distribution is illustrated in Fig. 7.1 (left), in which a sample of the independent components is plotted on the two-dimensional (2-D) plane. Figure 7.1 (right) also shows a histogram estimate of the uniform densities. These variables are then linearly mixed, and the mixtures are whitened as a preprocessing step. Whitening was explained earlier; let us recall briefly that it means that $\mathbf{x}$ is linearly transformed into a random vector

$$\mathbf{z} = \mathbf{V}\mathbf{x} = \mathbf{V}\mathbf{A}\mathbf{s} \tag{7.4}$$

whose correlation matrix equals unity: $E\{\mathbf{z}\mathbf{z}^T\} = \mathbf{I}$. Thus the ICA model still holds, though with a different mixing matrix. (Even without whitening, the situation would be similar.) The joint density of the whitened mixtures is given in Fig. 7.2. It is a rotation of the original joint density.

Now, let us look at the densities of the two linear mixtures z_1 and z_2. One of them is estimated in Fig. 7.2, on the right. One can clearly see that the densities of the mixtures are closer to a gaussian density than the densities of the independent components shown in Fig. 7.1. Thus we see that the mixing makes the variables closer to gaussian. Finding the rotation that rotates the square in Fig. 7.2 back to the original ICs in Fig. 7.1 would give us the two maximally nongaussian linear combinations with uniform distributions.

A second example with very different densities shows the same result. In Fig. 7.3, the joint distribution of very supergaussian independent components is shown. The marginal density of a component is estimated in Fig. 7.3. The density has a large peak

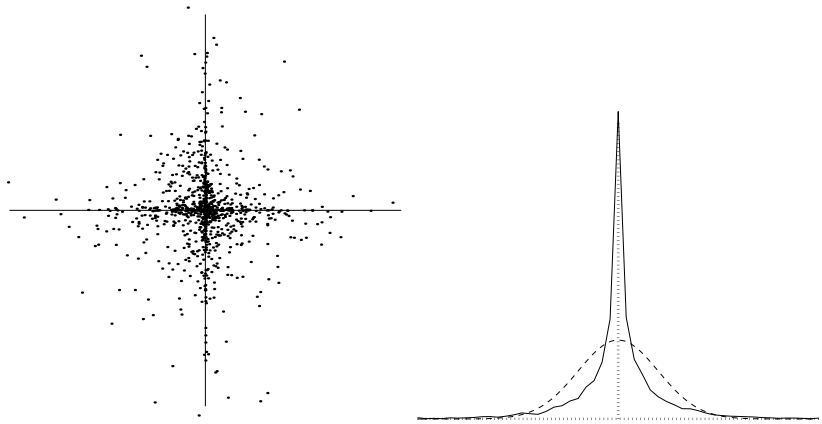

Figure 7.3: *Left*: The joint distribution of the two independent components with supergaussian densities. *Right*: The estimated density of one supergaussian independent component.

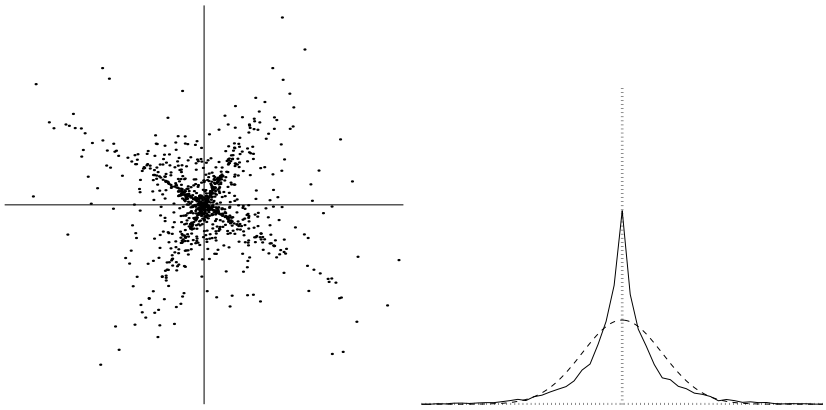

Figure 7.4: *Left:* The joint distribution of two whitened mixtures of independent components with supergaussian densities. *Right:* The marginal density of one of the whitened mixtures in Fig. 7.4. It is closer to the gaussian density (given by dashed curve) than the densities of the independent components.

at zero, as is typical of supergaussian densities (see below). Whitened mixtures of the independent components are shown in Fig. 7.4 on the left. The densities of two linear mixtures are given in Fig. 7.4 on the right. They are clearly more gaussian than the original densities, as can be seen from the fact that the peak is much lower. Again, we see that mixing makes the distributions more gaussian.

To recapitulate, we have formulated ICA estimation as the search for directions that are maximally nongaussian: Each local maximum gives one independent component. Our approach here is somewhat heuristic, but it will be seen in the next section that it has a perfectly rigorous justification. From a practical point of view, we now have to answer the following questions: How can the nongaussianity of $\mathbf{b}^T\mathbf{x}$ be measured? And how can we compute the values of $\mathbf{b}$ that maximize (locally) such a measure of nongaussianity? The rest of this chapter is devoted to answering these questions.

7.3 Measuring nongaussianity by kurtosis

7.3.1 Extrema of kurtosis give independent components

Kurtosis and its properties

To use nongaussianity in ICA estimation, we must have a quantitative measure of nongaussianity of a random variable, say y. In this section, we show how to use kurtosis, a classic measure of nongaussianity, for ICA estimation. Kurtosis is the name given to the fourth-order cumulant of a random variable.

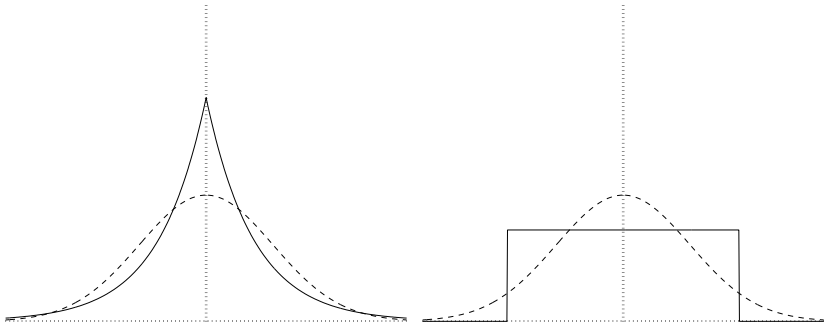

Figure 7.5: *Left*: The density function of the Laplacian distribution, which is a typical supergaussian distribution. For comparison, the gaussian density is given by a dashed curve. Both densities are normalized to unit variance. *Right*: The density function of the uniform distribution, which is a typical *sub*gaussian distribution.

The kurtosis of y, denoted by $\mathrm{kurt}(y)$, is defined by

$$\mathrm{kurt}(y) = E\{y^4\} - 3(E\{y^2\})^2 \tag{7.5}$$

Remember that all the random variables here have zero mean; in the general case, the definition of kurtosis is slightly more complicated. To simplify things, we can further assume that y has been normalized so that its variance is equal to one: $E\{y^2\} = 1$. Then the right-hand side simplifies to $E\{y^4\} - 3$. This shows that kurtosis is simply a normalized version of the fourth moment $E\{y^4\}$.

For a gaussian y, the fourth moment equals $3(E\{y^2\})^2$, which can be shown by integration by parts:

$$(E\{y^2\})^2 = \sigma^2 \int \frac{1}{\sqrt{2\pi}} y^2 \exp(-\frac{1}{2\sigma^2}y^2) = 0 - \sigma^2 \int \frac{1}{\sqrt{2\pi}} \frac{1}{3} y^3 \frac{-y}{\sigma^2} \exp(-\frac{1}{2\sigma}y^2) = \frac{1}{3}E\{y^4\} \tag{7.6}$$

Thus, kurtosis is zero for a gaussian random variable. For most (but not quite all) nongaussian random variables, kurtosis is nonzero.

Kurtosis can be both positive or negative. Random variables that have a negative kurtosis are called subgaussian, and those with positive kurtosis are called supergaussian. In statistical literature, the corresponding expressions platykurtic and leptokurtic are also used. Supergaussian random variables have typically a "spiky" probability density function (pdf) with heavy tails, i.e., the pdf is relatively large at zero and at large values of the variable, while being small for intermediate values. A typical example is the Laplacian distribution, whose pdf is given by

$$p(y) = \frac{1}{\sqrt{2}} \exp(-\sqrt{2}|y|) \tag{7.7}$$

Here we have normalized the variance to unity; this pdf is illustrated in Fig. 7.5. Subgaussian random variables, on the other hand, have typically a "flat" pdf, which is rather constant near zero, and very small for larger values of the variable. A typical example is the uniform distribution, whose density is given by

$$p(y) = \begin{cases} \frac{1}{2\sqrt{3}}, & \text{if } |y| \leq \sqrt{3} \\ 0, & \text{otherwise} \end{cases} \tag{7.8}$$

which is normalized to unit variance as well; it is illustrated in Fig. 7.5.

Typically nongaussianity is measured by the absolute value of kurtosis. The square of kurtosis can also be used. These measures are zero for a gaussian variable, and greater than zero for most nongaussian random variables. There are nongaussian random variables that have zero kurtosis, but they can be considered to be very rare.

Kurtosis, or rather its absolute value, has been widely used as a measure of nongaussianity in ICA and related fields. The main reason is its simplicity, both computational and theoretical. Computationally, kurtosis can be estimated simply by using the fourth moment of the sample data (if the variance is kept constant). Theoretical analysis is simplified because of the following linearity property: If x_1 and x_2 are two independent random variables, it holds

$$\mathrm{kurt}(x_1 + x_2) = \mathrm{kurt}(x_1) + \mathrm{kurt}(x_2) \tag{7.9}$$

and

$$\mathrm{kurt}(\alpha x_1) = \alpha^4 \mathrm{kurt}(x_1) \tag{7.10}$$

where α is a constant. These properties can be easily proven using the general definition of cumulants (which we do not go through in this course), but they can also be directly shown using the definition. These properties are to be compared with the corresponding properties of variance; the additivity is then true for any uncorrelated variables.

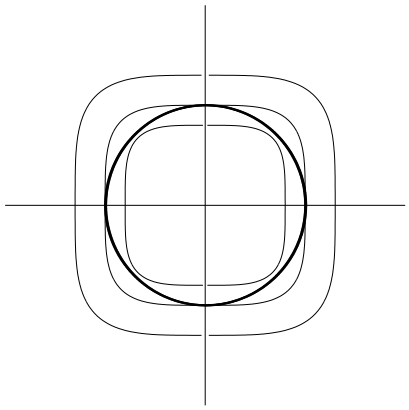

Figure 7.6: The optimization landscape of kurtosis. The thick curve is the unit sphere, and the thin curves are the contours where F in (7.12) is constant.

Optimization landscape in ICA

To illustrate in a simple example what the optimization landscape for kurtosis looks like, and how independent components could be found by kurtosis minimization or maximization, let us look at a 2-D model $\mathbf{x} = \mathbf{A}\mathbf{s}$. Assume that the independent components s_1, s_2 have kurtosis values $\text{kurt}(s_1), \text{kurt}(s_2)$, respectively, both different from zero. Recall that they have unit variances by definition. We look for one of the independent components as $y = \mathbf{b}^T\mathbf{x}$.

Let us again consider the transformed vector $\mathbf{q} = \mathbf{A}^T\mathbf{b}$. Then we have $y = \mathbf{b}^T\mathbf{x} = \mathbf{b}^T\mathbf{A}\mathbf{s} = \mathbf{q}^T\mathbf{s} = q_1 s_1 + q_2 s_2$. Now, based on the additive property of kurtosis, we have

$$\text{kurt}(y) = \text{kurt}(q_1 s_1) + \text{kurt}(q_2 s_2) = q_1^4 \text{kurt}(s_1) + q_2^4 \text{kurt}(s_2) \tag{7.11}$$

On the other hand, we made the constraint that the variance of y is equal to 1, based on the same assumption concerning s_1, s_2. This implies a constraint on $\mathbf{q}$: $E\{y^2\} = q_1^2 + q_2^2 = 1$. Geometrically, this means that vector $\mathbf{q}$ is constrained to the unit circle on the 2-D plane.

The optimization problem is now: What are the maxima of the function $|\text{kurt}(y)| = |q_1^4 \text{kurt}(s_1) + q_2^4 \text{kurt}(s_2)|$ on the unit circle? To begin with, we may assume for simplicity that the kurtoses are equal to 1. In this case, we are simply considering the function

$$F(\mathbf{q}) = q_1^4 + q_2^4 \tag{7.12}$$

Some contours of this function, i.e., curves in which this function is constant, are shown in Fig. 7.6. The unit sphere, i.e., the set where $q_1^2 + q_2^2 = 1$, is shown as well. This gives the "optimization landscape" for the problem.

It is not hard to see that the maxima are at those points where exactly one of the elements of vector $\mathbf{q}$ is zero and the other nonzero; because of the unit circle constraint, the nonzero element must be equal to 1 or -1. But these points are exactly the ones when y equals one of the independent components $\pm s_i$, and the problem has been solved.

If the kurtoses are both equal to -1, the situation is similar, because taking the absolute values, we get exactly the same function to maximize. Finally, if the kurtoses are completely arbitrary, as long as they are nonzero, more involved algebraic manipulations show that the absolute value of kurtosis is still maximized when $y = \mathbf{b}^T\mathbf{x}$ equals one of the independent components.

Sketch of the proof We provide a sketch of a geometric proof of the fundamental maximality property of kurtosis. More precisely, we prove that the maxima of the function

$$F(\mathbf{q}) = |\text{kurt}(\mathbf{q}^T\mathbf{s})| = |q_1^4 \text{kurt}(s_1) + q_2^4 \text{kurt}(s_2)| \tag{7.13}$$

in the constraint set $\|\mathbf{q}\|^2 = 1$ are obtained when only one of the components of $\mathbf{q}$ is nonzero. For simplicity, we consider here the 2-D case, and assume that both of the kurtoses are positive. We make the change of variables $t_i = q_i^2$. The geometrical form of the constraint set of $\mathbf{t} = (t_1, t_2)$ is then the line segment connecting the points $(0, 1)$ and $(1, 0)$. Note that the objective function is now quadratic. The geometrical shape of the set $F(\mathbf{t}) = \text{const.}$ is an ellipse. If you draw the constraint set and some of the ellipses, it is not very difficult to see that the maximum of $F(\mathbf{t})$ is obtained when one of the t_i is one and the other one is zero. This proves the maximality property if the kurtoses are both positive.

Now we see the *utility of preprocessing by whitening.* For whitened data $\mathbf{z}$, we seek for a linear combination $\mathbf{w}^T\mathbf{z}$ that maximizes

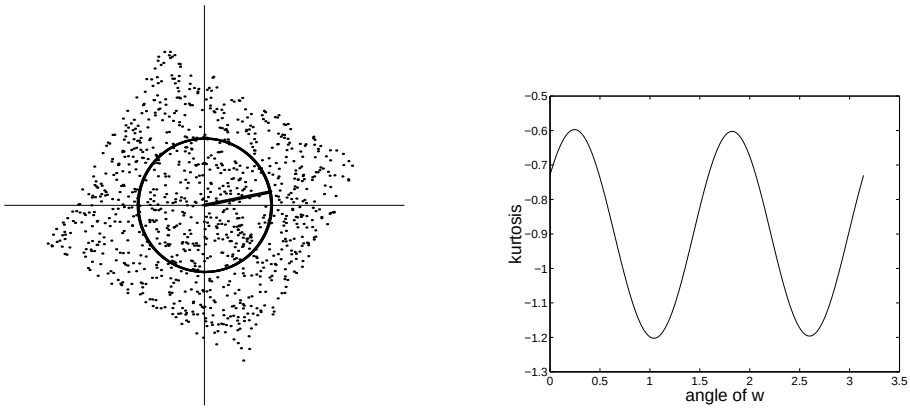

Figure 7.7: We search for projections (which correspond to points on the unit circle) that maximize nongaussianity, using whitened mixtures of uniformly distributed independent components. *Left*: The projections can be parameterized by the angle. (The particular direction shown is arbitrary and not the optimum.) Right: The values of kurtosis for projections as a function of the angle. Kurtosis is minimized, and its absolute value maximized, in the directions of the independent components.

nongaussianity. This simplifies the situation here, since we have $\mathbf{q} = (\mathbf{VA})^T\mathbf{w}$ and therefore

$$\|\mathbf{q}\|^2 = (\mathbf{w}^T\mathbf{VA})(\mathbf{A}^T\mathbf{V}^T\mathbf{w}) = \|\mathbf{w}\|^2 \tag{7.14}$$

This means that constraining $\mathbf{q}$ to lie on the unit sphere is equivalent to constraining $\mathbf{w}$ to be on the unit sphere. Thus we maximize the absolute value of kurtosis of $\mathbf{w}^T\mathbf{z}$ under the simpler constraint that $\|\mathbf{w}\| = 1$. Also, after whitening, the linear combinations $\mathbf{w}^T\mathbf{z}$ can be interpreted as *projections* on the line (that is, a 1-D subspace) spanned by the vector $\mathbf{w}$. Each point on the unit sphere corresponds to one projection.

As an example, let us consider the whitened mixtures of uniformly distributed independent components in Fig. 7.2. We search for a vector $\mathbf{w}$ such that the linear combination or projection $\mathbf{w}^T\mathbf{z}$ has maximum nongaussianity, as illustrated in Fig. 7.7. In this two-dimensional case, we can parameterize the points on the unit sphere by the *angle* that the corresponding vector $\mathbf{w}$ makes with the horizontal axis. Then, we can plot the kurtosis of $\mathbf{w}^T\mathbf{z}$ as a function of this angle, which is given in Fig. 7.7. The plot shows kurtosis is always negative, and is minimized at approximately 1 and 2.6 radians. These directions are thus such that the absolute value of kurtosis is maximized. They can be seen in Fig. 7.7 to correspond to the directions given by the edges of the square, and thus they do give the independent components.

In the second example, we see the same phenomenon for whitened mixtures of supergaussian independent components. Again, we search for a vector $\mathbf{w}$ such that the projection in that direction has maximum nongaussianity, as illustrated in Fig. 7.8. We can plot the kurtosis of $\mathbf{w}^T\mathbf{z}$ as a function of the angle in which $\mathbf{w}$ points, as given in Fig. 7.8. The plot shows kurtosis is always positive, and is maximized in the directions of the independent components. These angles are the same as in the preceding example because we used the same mixing matrix. Again, they correspond to the directions in which the absolute value of kurtosis is maximized.

7.3.2 Gradient algorithm using kurtosis

In practice, to maximize the absolute value of kurtosis, we would start from some vector $\mathbf{w}$, compute the direction in which the absolute value of the kurtosis of $y = \mathbf{w}^T\mathbf{z}$ is growing most strongly, based on the available sample $\mathbf{z}(1), ..., \mathbf{z}(T)$ of mixture vector $\mathbf{z}$, and then move the vector $\mathbf{w}$ in that direction. This idea is implemented in gradient methods and their extensions.

The gradient of the absolute value of kurtosis of $\mathbf{w}^T\mathbf{z}$ can be simply computed as

$$\frac{\partial|\text{kurt}(\mathbf{w}^T\mathbf{z})|}{\partial\mathbf{w}} = 4\,\text{sign}(\text{kurt}(\mathbf{w}^T\mathbf{z}))[E\{\mathbf{z}(\mathbf{w}^T\mathbf{z})^3\} - 3\mathbf{w}\|\mathbf{w}\|^2] \tag{7.15}$$

since for whitened data we have $E\{(\mathbf{w}^T\mathbf{z})^2\} = \|\mathbf{w}\|^2$. Since we are optimizing this function on the unit sphere $\|\mathbf{w}\|^2 = 1$, the gradient method must be complemented by projecting $\mathbf{w}$ on the unit sphere after every step. This can be done simply by dividing $\mathbf{w}$ by its norm.

Actually, in many applications we know in advance the nature of the distributions of the independent components, i.e., whether they are subgaussian or supergaussian. Then we can simply plug the correct sign of kurtosis in the algorithm, and avoid its estimation.

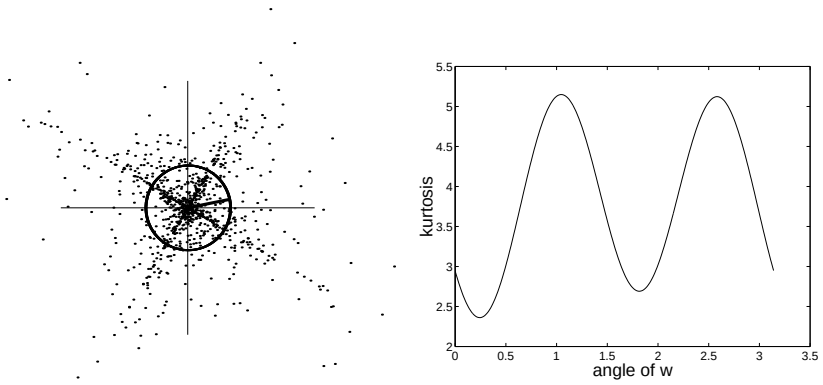

Figure 7.8: Again, we search for projections that maximize nongaussianity, this time with whitened mixtures of supergaussian independent components. *Left*: The projections can be parameterized by the angle. *Right*: The values of kurtosis for projections in different angles as in Fig. 7.8. Kurtosis, as well as its absolute value, is maximized in the directions of the independent components.

7.3.3 A fast fixed-point algorithm (FastICA) using kurtosis

A common problem with gradient algorithms is that the convergence is slow, and depends on a good choice of the learning rate sequence. A bad choice of the learning rate can, in practice, destroy convergence. Therefore, some ways to make the learning radically faster and more reliable would be useful. The fixed-point iteration algorithms offer such an alternative.

To derive a more efficient fixed-point iteration, we note that at a stable point of the gradient algorithm, the gradient must point in the direction of $\mathbf{w}$, that is, the gradient must be equal to $\mathbf{w}$ multiplied by some scalar constant. Only in such a case, adding the gradient to $\mathbf{w}$ does not change its direction, and we can have convergence (this means that after normalization to unit norm, the value of $\mathbf{w}$ is not changed except perhaps by changing its sign). (This can be proven more rigorously using the technique of Lagrange multipliers.)

Equating the gradient of kurtosis in (7.15) with $\mathbf{w}$ times a scalar multiplier, this means that we should have

$$\mathbf{w} \propto [E\{\mathbf{z}(\mathbf{w}^T\mathbf{z})^3\} - 3\|\mathbf{w}\|^2\mathbf{w}] \tag{7.16}$$

where $\propto$ means "proportional to", i.e. equal up to a scalar multiplier. So, this immediately suggests a fixed-point algorithm where we first compute the right-hand side, and give this as the new value for $\mathbf{w}$:

$$\mathbf{w} \leftarrow E\{\mathbf{z}(\mathbf{w}^T\mathbf{z})^3\} - 3\mathbf{w} \tag{7.17}$$

This is effectively like taking a gradient step with an infinite step size (cf. exercice with power method).

After every fixed-point iteration, $\mathbf{w}$ is divided by its norm to remain on the constraint set. (Thus $\|\mathbf{w}\| = 1$ always, which is why it can be omitted from (7.16).) The final vector $\mathbf{w}$ gives one of the independent components as the linear combination $\mathbf{w}^T\mathbf{z}$. In practice, the expectations in (7.17) must be replaced by their estimates.

Note that convergence of this fixed-point iteration means that the old and new values of $\mathbf{w}$ point in the same direction, i.e., their dot-product is (almost) equal to 1. It is not necessary that the vector converges to a single point, since $\mathbf{w}$ and $-\mathbf{w}$ define the same direction. This is again because the independent components can be defined only up to a multiplicative sign.

Actually, it turns out that such an algorithm works very well, converging very fast. Note that in general, there is no guarantee that such a fixed-point algorithm would work at all: Here we are basically quite lucky and find a very good fixed-point algorithm easily.

7.3.4 Examples

Here we show what happens when we run the FastICA algorithm that maximizes the absolute value of kurtosis, using the two example data sets used in this chapter. First we take a mixture of two uniformly distributed independent components. The mixtures are whitened, as always in this chapter. The goal is now to find a direction in the data that maximizes the absolute value of kurtosis, as illustrated in Fig. 7.7.

We initialize, for purposes of the illustration, the vector $\mathbf{w}$ as $\mathbf{w} = (1, 0)^T$. Running the FastICA iteration just two times, we obtain convergence. In Fig. 7.9 (left), the obtained vectors $\mathbf{w}$ are shown. The dashed line gives the direction of $\mathbf{w}$ after the first iteration, and the solid line gives the direction of $\mathbf{w}$ after the second iteration. The third iteration did not significantly change

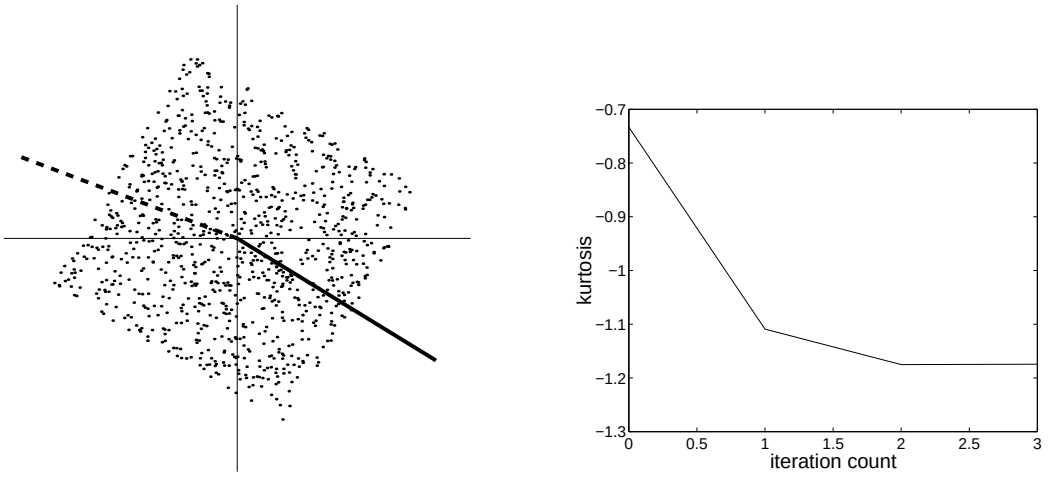

Figure 7.9: Result of FastICA using kurtosis, for ICs with uniform distributions. *Left*: Dashed line: $\mathbf{w}$ after the first iteration (plotted longer than actual size). Solid line: $\mathbf{w}$ after the second iteration. *Right*: The value of kurtosis shown as function of iteration count.

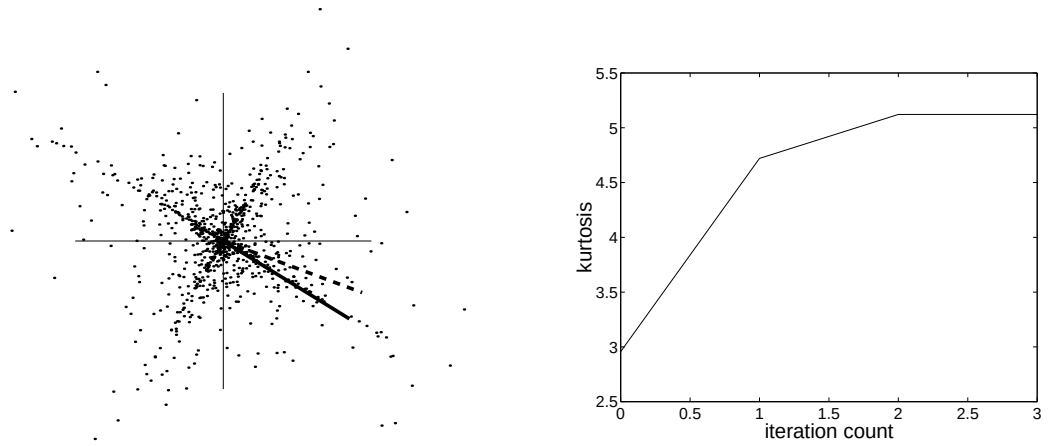

Figure 7.10: Result of FastICA with kurtosis, this time for supergaussian ICs. Dash-dotted line: $\mathbf{w}$ after the first iteration (plotted longer than actual size). Solid line: $\mathbf{w}$ after the second iteration. *Right*: The value of kurtosis shown as a function of iteration count.

the direction of $\mathbf{w}$, which means that the algorithm converged. (The corresponding vector is not plotted.) The figure shows that the value of $\mathbf{w}$ may change drastically during the iteration, because the values $\mathbf{w}$ and $-\mathbf{w}$ are considered as equivalent. This is because the sign of the vector cannot be determined in the ICA model.

The kurtoses of the projections $\mathbf{w}^T\mathbf{z}$ obtained in the iterations are also plotted in Fig. 7.9 (right), as a function of iteration count. The plot shows that the algorithm steadily increased the *absolute value* of the kurtosis of the projection, until it reached convergence at the third iteration.

Similar experiments were performed for the whitened mixtures of two *supergaussian* independent components, which were illustrated in Fig. 7.8. The obtained vectors are shown in Fig. 7.10. Again, convergence was obtained after two iterations. As in the preceding experiment, the absolute value of the kurtosis of the projection steadily increased, until it reached convergence at the third iteration.

In these examples, we only estimated one independent component. Of course, one often needs more than one component. Figures 7.7 and 7.8 indicate how this can be done: The directions of the independent components are orthogonal in the whitened space, so the second independent component can be found as the direction orthogonal to the $\mathbf{w}$ corresponding to the estimated independent component. For more dimensions, we need to rerun the algorithm, always constraining the current $\mathbf{w}$ to be orthogonal to the previously estimated vectors $\mathbf{w}$. This will be explained in more detail in Section 7.4.

7.4 Estimating several independent components

7.4.1 Constraint of uncorrelatedness

In this chapter, we have so far estimated only one independent component. This is why these algorithms are sometimes called "one-unit" algorithms. In principle, we could find more independent components by running the algorithm many times and using different initial points. This would not be a reliable method of estimating many independent components, however.

The key to extending the method of maximum nongaussianity to estimate more independent component is based on the following property: The vectors $\mathbf{w}_i$ corresponding to different independent components are orthogonal in the whitened space, as shown earlier. To recapitulate, the independence of the components requires that they are uncorrelated, and in the whitened space we have $E\{(\mathbf{w}_i^T\mathbf{z})(\mathbf{w}_j^T\mathbf{z})\} = \mathbf{w}_i^T\mathbf{w}_j$, and therefore uncorrelatedness in equivalent to orthogonality. This property is a direct consequence of the fact that after whitening, the mixing matrix can be taken to be orthogonal. The $\mathbf{w}_i$ are in fact by definition the rows of the inverse of the mixing matrix, and these are equal to the columns of the mixing matrix, because by orthogonality $\mathbf{A}^{-1} = \mathbf{A}^T$.

Thus, to estimate several independent components, we need to run any of the one-unit algorithms several times (possibly using several units) with vectors $\mathbf{w}_1, ..., \mathbf{w}_n$, and to prevent different vectors from converging to the same maxima we must *orthogonalize* the vectors $\mathbf{w}_1, ..., \mathbf{w}_n$ after every iteration. We present in the following different methods for achieving decorrelation.

7.4.2 Deflationary orthogonalization

A simple way of orthogonalization is deflationary orthogonalization using the Gram-Schmidt method. This means that we estimate the independent components one by one. When we have estimated p independent components, or p vectors $\mathbf{w}_1, ..., \mathbf{w}_p$, we run any one-unit algorithm for $\mathbf{w}_{p+1}$, and after every iteration step subtract from $\mathbf{w}_{p+1}$ the projections $(\mathbf{w}_{p+1}^T\mathbf{w}_j)\mathbf{w}_j, j = 1, ..., p$ of the previously estimated p vectors, and then renormalize $\mathbf{w}_{p+1}$. More precisely, we alternate the following steps:

1. Choose m, the number of ICs to estimate. Set $p \leftarrow 1$.
2. Initialize $\mathbf{w}_p$ (e.g. randomly)
3. Do an iteration of a one-unit algorithm on $\mathbf{w}_p$.
4. Do the following orthogonalization:
$$\mathbf{w}_p \leftarrow \mathbf{w}_p - \sum_{j=1}^{p-1}(\mathbf{w}_p^T\mathbf{w}_j)\mathbf{w}_j \tag{7.18}$$
5. Normalize $\mathbf{w}_p$ by dividing it by its norm.
6. If $\mathbf{w}_p$ has not converged, go back to step 3.
7. Set $p \leftarrow p + 1$. If p is not greater than the desired number of ICs, go back to step 2.

7.4.3 Symmetric orthogonalization

In certain applications, it may be desirable to use a symmetric decorrelation, in which no vectors are "privileged" over others. This means that the vectors $\mathbf{w}_i$ are not estimated one by one; instead, they are estimated in parallel. One motivation for this is that the deflationary method has the drawback that estimation errors in the first vectors are cumulated in the subsequent ones by the orthogonalization. Another one is that the symmetric orthogonalization methods enable parallel computation of ICs.

Symmetric orthogonalization is done by first doing the iterative step of the one-unit algorithm on every vector $\mathbf{w}_i$ in parallel, and afterwards orthogonalizing all the $\mathbf{w}_i$ by special symmetric methods. In other words:

1. Choose the number of independent components to estimate, say m.
2. Initialize the $\mathbf{w}_i, i = 1, ..., m$ (e.g., randomly).
3. Do an iteration of a one-unit algorithm on every $\mathbf{w}_i$ in parallel.
4. Do a symmetric orthogonalization of the matrix $\mathbf{W} = (\mathbf{w}_1, ..., \mathbf{w}_m)^T$.
5. If not converged, go back to step 3.

The symmetric orthogonalization of $\mathbf{W}$ can be accomplished, e.g., by the classic method involving matrix square roots,
$$\mathbf{W} \leftarrow (\mathbf{W}\mathbf{W}^T)^{-1/2}\mathbf{W} \tag{7.19}$$
The inverse square root $(\mathbf{W}\mathbf{W}^T)^{-1/2}$ is obtained from the eigenvalue decomposition of $\mathbf{W}\mathbf{W}^T = \mathbf{E}\,\mathrm{diag}(d_1, ..., d_m)\,\mathbf{E}^T$ as
$$(\mathbf{W}\mathbf{W}^T)^{-1/2} = \mathbf{E}\,\mathrm{diag}(d_1^{-1/2}, ..., d_m^{-1/2})\,\mathbf{E}^T \tag{7.20}$$

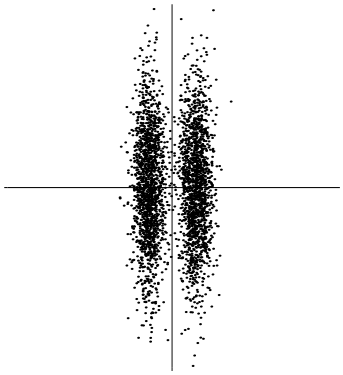

Figure 7.11: An illustration of projection pursuit and the "interesting" directions. The data in this figure is clearly divided into two clusters. The goal in projection pursuit is to find the projection (here, on the horizontal axis) that reveals the clustering or other structure of the data.

7.5 ICA and projection pursuit

It is interesting to note how the approach to ICA described in this Chapter makes explicit the connection between ICA and another technique: projection pursuit.

7.5.1 Searching for interesting directions

Projection pursuit is a technique developed in statistics for finding "interesting" projections of multidimensional data. Such projections can then be used for optimal visualization of the data, and for such purposes as density estimation and regression.

When projection pursuit is used for exploratory data analysis, we usually compute a couple of the most interesting 1-D projections. (The definition of interestingness will be treated in the next section.) Some structure of the data can then be visualized by showing the distribution of the data in the 1-D subspaces, or on 2-D planes spanned by two of the projection pursuit directions. This method is en extension of the classic method of using principal component analysis (PCA) for visualization, in which the distribution of the data is shown on the plane spanned by the two first principal components.

An example of the problem can be seen in Fig. 7.11. In reality, projection pursuit is of course used in situations where the number of dimensions is very large, but for purposes of illustration, we use here a trivial 2-D example. In the figure, the interesting projection of the data would be on the horizontal axis. This is because that projection shows the clustering structure of the data. In contrast, projections in very different directions (here, projection on the vertical axis) would show only an ordinary gaussian distribution. It would thus be useful to have a method that automatically finds the horizontal projection in this example.

7.5.2 Nongaussian is interesting

The basic question in projection pursuit is thus to define what kind of projections are interesting.

It is usually argued that the gaussian distribution is the least interesting one, and that the most interesting directions are those that show the least gaussian distribution. One motivation for this is that distributions that are multimodal, i.e., show some clustering structure, are far from gaussian.

The usefulness of using the most nongaussian projections for visualization can be seen in Fig. 7.11. Here the most nongaussian projection is on the horizontal axis; this is also the projection that most clearly shows the clustered structure of the data. On the other hand, the projection on the vertical direction, which is also the direction of the first principal component, fails to show this structure. This also shows that PCA does not use the clustering structure. In fact, clustering structure is not visible in the covariance or correlation matrix on which PCA is based.

Thus projection pursuit is usually performed by finding the most nongaussian projections of the data. This is the same thing that we did in this chapter to estimate the ICA model. This means that all the nongaussianity measures and the corresponding ICA algorithms presented in this chapter could also be called projection pursuit "indices" and algorithms.

It should be noted that in the formulation of projection pursuit, no data model or assumption about independent components is made. If the ICA model holds, optimizing the ICA nongaussianity measures produce independent components; if the model does not hold, then what we get are the projection pursuit directions.

Chapter 8

ICA by maximum likelihood estimation

A very popular approach for estimating the independent component analysis (ICA) model is maximum likelihood (ML) estimation. One interpretation of ML estimation is that we take those parameter values as estimates that give the highest probability for the observations. Here, we show how to apply ML estimation to ICA estimation.

8.1 The likelihood of the ICA model

8.1.1 Deriving the likelihood

It is not difficult to derive the likelihood in the ICA model. This is based on using the well-known result on the density of a linear transform, which says that the density p_x of the mixture vector

$$\mathbf{x} = \mathbf{A}\mathbf{s} \tag{8.1}$$

can be formulated as

$$p_x(\mathbf{x}) = |\det \mathbf{B}| p_s(\mathbf{s}) \tag{8.2}$$

where $\mathbf{B} = \mathbf{A}^{-1}$. This is in fact true for a linear transform of any vector and has nothing to do with ICA. Now, when we use the definition of ICA, we have further

$$p_x(\mathbf{x}) = |\det \mathbf{B}| \prod_i p_i(s_i) \tag{8.3}$$

where the p_i denote the densities of the independent components. This can be expressed as a function of $\mathbf{B} = (\mathbf{b}_1, ..., \mathbf{b}_n)^T$ and $\mathbf{x}$, giving

$$p_x(\mathbf{x}) = |\det \mathbf{B}| \prod_i p_i(\mathbf{b}_i^T \mathbf{x}) \tag{8.4}$$

Assume that we have T observations of $\mathbf{x}$, denoted by $\mathbf{x}(1), \mathbf{x}(2), ..., \mathbf{x}(T)$. Then the likelihood can be obtained as the product of this density evaluated at the T points. This is denoted by L and considered as a function of $\mathbf{B}$:

$$L(\mathbf{B}) = \prod_{t=1}^{T} \prod_{i=1}^{n} p_i(\mathbf{b}_i^T \mathbf{x}(t)) |\det \mathbf{B}| \tag{8.5}$$

Very often it is more practical to use the logarithm of the likelihood, since it is algebraically simpler. This does not make any difference here since the maximum of the logarithm is obtained at the same point as the maximum of the likelihood. The log-likelihood is given by

$$\log L(\mathbf{B}) = \sum_{t=1}^{T} \sum_{i=1}^{n} \log p_i(\mathbf{b}_i^T \mathbf{x}(t)) + T \log |\det \mathbf{B}| \tag{8.6}$$

The basis of the logarithm makes no difference, though in the following the natural logarithm is used.

To simplify notation, we can denote the sum over the sample index t by an expectation operator, and divide the likelihood by T to obtain

$$\frac{1}{T} \log L(\mathbf{B}) = E\{\sum_{i=1}^{n} \log p_i(\mathbf{b}_i^T \mathbf{x})\} + \log |\det \mathbf{B}| \tag{8.7}$$

The expectation here is not the theoretical expectation, but an average computed from the observed sample. Of course, in the algorithms the expectations are eventually replaced by sample averages, so the distinction is purely theoretical.

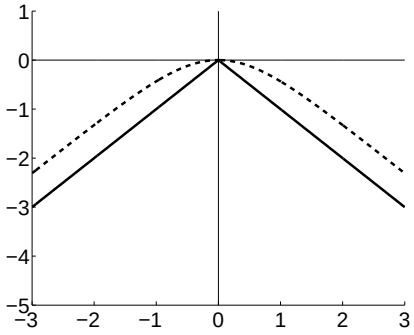

Figure 8.1: Illustration of the (negative of) log cosh function (dashed line) and its comparison with the absolute value function (solid line).

8.1.2 Estimation of the densities

Problem of semiparametric estimation

Above, we expressed the likelihood as a function of the parameters of the model, which are the elements of the mixing matrix. For simplicity, we used the elements of the inverse $\mathbf{B}$ of the mixing matrix. This is allowed since the mixing matrix can be directly computed from its inverse.

There is another thing to estimate in the ICA model, though. This is the densities of the independent components. Actually, the likelihood is a function of these densities as well. This makes the problem much more complicated, because the estimation of densities is, in general, a nonparametric problem. Nonparametric means that it cannot be reduced to the estimation of a finite parameter set. In fact the number of parameters to be estimated is infinite, or in practice, very large. Thus the estimation of the ICA model has also a nonparametric part, which is why the estimation is sometimes called "semiparametric".

Nonparametric estimation of densities is known to be a difficult problem. Many parameters are always more difficult to estimate than just a few; since nonparametric problems have an infinite number of parameters, they are the most difficult to estimate. This is why we would like to avoid the nonparametric density estimation in the ICA. There are two ways to avoid it.

First, in some cases we might know the densities of the independent components in advance, using some prior knowledge on the data at hand. In this case, we could simply use these prior densities in the likelihood. Then the likelihood would really be a function of $\mathbf{B}$ only. If reasonably small errors in the specification of these prior densities have little influence on the estimator, this procedure will give reasonable results. In fact, it will be shown below that this is the case.

For example, if we know that the independent component are supergaussian, we could use the Laplacian density, in which case

$$\log p_i(s) = -\sqrt{2}|s| - \frac{1}{2}\log 2 \tag{8.8}$$

However, optimization methods may not not work well with such a peaked density, so in practice, it is often preferred to use

$$\log p_i(s) = -\log\cosh(s) \tag{8.9}$$

This exotic function is actually just a smoother version of the absolute value function. It should be properly normalized so that the integral of p_i is equal to one, and the variance equals one as well, but such normalization has little importance in practice. The function is illustrated in Fig. 8.1.

A second way to solve the problem of density estimation is to approximate the densities of the independent components by a family of densities that are specified by a limited number of parameters. If the number of parameters in the density family needs to be very large, we do not gain much from this approach, since the goal was to reduce the number of parameters to be estimated. However, if it is possible to use a very simple family of densities to estimate the ICA model for any densities p_i, we will get a simple solution. Fortunately, this turns out to be the case. We can use an extremely simple parameterization of the p_i, consisting of the choice between two densities, i.e., a single binary parameter.

A simple density family

It turns out that in maximum likelihood estimation, it is enough to use just *two* approximations of the density of an independent component. For each independent component, we just need to determine which one of the two approximations is better. This shows that, first, we can make small errors when we fix the densities of the independent components, since it is enough that we use a density that is in the same half of the space of probability densities. Second, it shows that we can estimate the independent components using very simple models of their densities, in particular, using models consisting of only two densities.

The validity of these approaches is shown in the following theorem.

Theorem 1 *Denote by $\tilde{p}_i$ the* assumed *densities of the independent components, and*

$$g_i(s_i) = \frac{\partial}{\partial s_i} \log \tilde{p}_i(s_i) = \frac{\tilde{p}_i'(s_i)}{\tilde{p}_i(s_i)} \tag{8.10}$$

Constrain the estimates of the independent components $y_i = \mathbf{b}_i^T \mathbf{x}$ to be uncorrelated and to have unit variance. Assume that the data follows the ICA model. Then, local maxima of the likelihood of $\mathbf{B}$ are obtained at the points where $\pm y_i$ equal the independent components when sample size T goes to infinity, if the assumed densities $\tilde{p}_i$ fulfill

$$E\{s_i g_i(s_i) - g_i'(s_i)\} > 0 \tag{8.11}$$

for all i, where the expectations are taken with respect to the real densities p_i.[1].

This theorem shows rigorously that small misspecifications in the densities p_i do not affect the local consistency of the ML estimator, since sufficiently small changes do not change the sign in (8.11).

Moreover, the theorem shows how to construct families consisting of only two densities, so that the condition in (8.11) is true for one of these densities. In principle, if you consider taking $-g$ instead of some original g in the theorem, you see that the sign changes. Thus, we could in principle just use g and $-g$ depending on the distribution to always get a positive sign. However, this is not directly possible because we have to make sure that g corresponds to the derivative of some log-pdf, so it cannot be any function. In the following, we consider thus an approach which is slightly modified.

For example, consider the following log-densities:

$$\log \tilde{p}_i^+(s) = \alpha_1 - 2 \log \cosh(s) \tag{8.12}$$

$$\log \tilde{p}_i^-(s) = \alpha_2 - [s^2/2 - \log \cosh(s)] \tag{8.13}$$

where α_1, α_2 are positive parameters that are fixed so as to make these two functions logarithms of probability densities. Actually, these constants can be ignored in the following. The factor 2 in (8.12) is not important, but it is usually used here; also, the factor $1/2$ in (8.13) could be changed.

The motivation for these functions is that $\tilde{p}_i^+$ is a *supergaussian* density, because the $\log \cosh$ function is close to the absolute value that would give the Laplacian density. The density given by $\tilde{p}_i^-$ is *subgaussian*, because it is like a gaussian log-density, $-s^2/2$ plus a constant, that has been somewhat "flattened" by the $\log \cosh$ function.

Simple computations show that the value of the nonpolynomial moment in (8.11) is for $\tilde{p}_i^+$

$$2E\{-\tanh(s_i)s_i + (1 - \tanh(s_i)^2)\} \tag{8.14}$$

and for $\tilde{p}_i^-$ it is

$$E\{\tanh(s_i)s_i - (1 - \tanh(s_i)^2)\} \tag{8.15}$$

since the derivative of $\log \cosh$ equals the tanh function, and the derivative of $\tanh(s)$ equals $1 - \tanh(s)^2$, and $E\{s_i^2\} = 1$ by definition. We see that the signs of these expressions are always opposite. Thus, for practically any distributions of the s_i, one of these functions fulfills the condition, i.e., has the desired sign, and estimation is possible. Of course, for some distribution of the s_i the nonpolynomial moment in the condition could be zero, which corresponds to the case of zero kurtosis in cumulant-based estimation; such cases can be considered to be very rare.

Thus we can just compute the nonpolynomial moments for the two prior distributions in (8.12) and (8.13), and choose the one that fulfills the stability condition in (8.11). This can be done on-line during the maximization of the likelihood. This always provides an objective which has the maxima at the independent components, and solves the problem of semiparametric estimation.

In fact, the nonpolynomial moment in question measures the shape of the density function in much the same way as kurtosis. For $g(s) = -s^3$, we would actually obtain kurtosis. Thus, the choice of nonlinearity could be compared with the choice whether to minimize or maximize kurtosis, as encountered earlier. That choice was based on the value of the sign of kurtosis; here we use the sign of a nonpolynomial moment.

Indeed, the nonpolynomial moment of this chapter is the same as the one encountered in the preceding one when using more general measures of nongaussianity. However, it must be noted that the set of nonlinearities that we can use here is more restricted than those used earlier. This is because the nonlinearities g_i used must correspond to the derivative of the logarithm of a probability density function (pdf). For example, we cannot use the function $g(s) = s^3$ because the corresponding pdf would be of the form $\exp(s^4/4)$, and this is not integrable, i.e., it is not a pdf at all.

[1] Thus, in the terminology of estimation theory, the estimator is consistent.

8.2 Algorithms for maximum likelihood estimation

To perform maximum likelihood estimation in practice, we need an algorithm to perform the numerical maximization of likelihood. In this section, we discuss different methods to this end. We show how to derive simple gradient algorithms, of which especially the natural gradient algorithm has been widely used. A version of FastICA is also possible.

8.2.1 Gradient algorithm

The simplest algorithms for maximizing likelihood are obtained by gradient methods. One can easily derive the gradient of the log-likelihood in (8.7) as:

$$\frac{1}{T}\frac{\partial \log L}{\partial \mathbf{B}} = [\mathbf{B}^T]^{-1} + E\{\mathbf{g}(\mathbf{B}\mathbf{x})\mathbf{x}^T\} \tag{8.16}$$

Here, $\mathbf{g}(\mathbf{y}) = (g_i(y_i), ..., g_n(y_n))$ is a component-wise vector function that consists of the so-called (negative) score functions g_i of the distributions of s_i, defined as

$$g_i = (\log p_i)' = \frac{p_i'}{p_i}. \tag{8.17}$$

This immediately gives the following gradient algorithm for ML estimation:

$$\mathbf{B} \leftarrow \mathbf{B} + \mu \left([\mathbf{B}^T]^{-1} + E\{\mathbf{g}(\mathbf{B}\mathbf{x})\mathbf{x}^T\} \right) \tag{8.18}$$

for some step size parameter μ. A stochastic version of this algorithm could be used as well. This means that the expectation is omitted, and in each step of the algorithm, only one data point is used:

$$\mathbf{B} \leftarrow \mathbf{B} + \mu \left([\mathbf{B}^T]^{-1} + \mathbf{g}(\mathbf{B}\mathbf{x})\mathbf{x}^T \right) \tag{8.19}$$

This algorithm is often called the Bell-Sejnowski algorithm.

The algorithm in Eq. (8.18) converges very slowly, however, especially due to the inversion of the matrix $\mathbf{B}$ that is needed in every step. Note that here the data does not have to be whitened, so $\mathbf{B}$ cannot be constrained to be orthogonal, and the inverse of $\mathbf{B}$ is not trivial to compute. In fact, the convergence can be improved by whitening the data, and especially by using the natural gradient.

8.2.2 The natural gradient algorithm

The natural (or relative) gradient method simplifies the maximization of the likelihood considerably, and makes it better conditioned. The principle of the natural gradient is based on the geometrical structure of the parameter space, and is related to the principle of relative gradient, which uses the Lie group structure of the ICA problem. In the case of basic ICA, both of these principles amount to multiplying the right-hand side of (8.18) by $\mathbf{B}^T\mathbf{B}$. Thus, defining $\mathbf{y} = \mathbf{B}\mathbf{x}$, we obtain

$$\mathbf{B} \leftarrow \mathbf{B} + \mu(\mathbf{I} + E\{\mathbf{g}(\mathbf{y})\mathbf{y}^T\})\mathbf{B} \tag{8.20}$$

This algorithm is also called the infomax algorithm.[2]

Interestingly, this algorithm can be interpreted as *nonlinear decorrelation*. The idea is that the algorithm converges when $E\{\mathbf{g}(\mathbf{y})\mathbf{y}^T\} = \mathbf{I}$, which means that the y_i and $g_j(y_j)$ are uncorrelated for $i \neq j$. This is a nonlinear extension of the ordinary requirement of uncorrelatedness.

In practice, one can use, for example, the two densities described in Section 8.1.2. For supergaussian independent components, the pdf defined by (8.12) is usually used. This means that the component-wise nonlinearity g is the tanh function:

$$g^+(y) = -2\tanh(y) \tag{8.21}$$

For subgaussian independent components, other functions must be used. For example, one could use the pdf in (8.13), which leads to

$$g^-(y) = \tanh(y) - y \tag{8.22}$$

(Another possibility is to use $g(y) = -y^3$ for subgaussian components.) These nonlinearities are illustrated in Fig. 8.2.

The choice between the two nonlinearities in (8.21) and (8.22) can be made by computing the nonpolynomial moment:

$$E\{-\tanh(s_i)s_i + (1 - \tanh(s_i)^2)\} \tag{8.23}$$

[2]Here, "infomax" actually refers to an alternative derivation of the objective function, and not to any particular algorithm, so the terminology is not very logical.

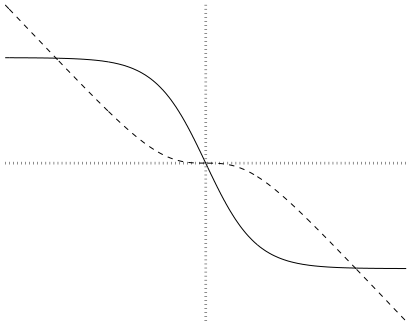

Figure 8.2: The functions g^+ in Eq. (8.21) and g^- in Eq. (8.22), given by the solid line and the dashed line, respectively.

1. Center the data to make its mean zero
2. Choose an initial (e.g., random) separating matrix $\mathbf{B}$. Choose initial values of $\gamma_i, i = 1, ..., n$, either randomly or using prior information. Choose the learning rates μ and μ_γ.
3. Compute $\mathbf{y} = \mathbf{Bx}$. Compute y_i normalized to unit variane, denote them by $\tilde{y}_i$.
4. If the nonlinearities are not fixed a priori:
   (a) update $\gamma_i = (1 - \mu_\gamma)\gamma_i + \mu_\gamma E\{-\tanh(\tilde{y}_i)y_i + (1 - \tanh(\tilde{y}_i)^2)\}$.
   (b) if $\gamma_i > 0$, define g_i as in (8.21), otherwise define it as in (8.22).
5. Update the separating matrix by
$$\mathbf{B} \leftarrow \mathbf{B} + \mu[\mathbf{I} + \mathbf{g}(\mathbf{y})\mathbf{y}^T]\mathbf{B} \tag{8.24}$$
   where $\mathbf{g}(y) = (g_1(y_1), ..., g_n(y_n))^T$.
6. If not converged, go back to step 3.

Table 8.1: The on-line stochastic natural gradient algorithm for maximum likelihood estimation. Preliminary whitening is not shown here, but in practice it is highly recommended. Note that the components may not be properly normalized to unit variance after the convergence of the algorithm, and such normalization should be done afterwards if needed.

using some estimates of the independent components. If this nonpolynomial moment is positive, the nonlinearity in (8.21) should be used, otherwise the nonlinearity in (8.22) should be used. This is because of the condition in Theorem 1.

The choice of nonlinearity can be made while running the gradient algorithm, using the running estimates of the independent components to estimate the nature of the independent components (that is, the sign of the nonpolynomial moment). Note that the use of the polynomial moment requires that the estimates of the independent components are first scaled properly, constraining them to unit variance, as in the theorem.

The resulting algorithm is recapitulated in Table 8.1. In this version, whitening and the above-mentioned normalization in the estimation of the nonpolynomial moments are omitted; in practice, these may be very useful.

8.2.3 FastICA

Likelihood can be maximized by a fixed-point algorithm (FastICA) as well. The derivation of the algorithm is rather complicated, so we will not go into details. It has the nice feature that it contains the adaptation of the sign considered in Section 8.1.2, so it can estimate both sub- and supergaussian components. The simplest way of describing the algorithm is by modifying the iteration in (7.17) so that it takes the form

$$\mathbf{w} \leftarrow E\{\mathbf{z}g(\mathbf{w}^T\mathbf{z})\} - E\{g'(\mathbf{w}^T\mathbf{z})\}\mathbf{w} \tag{8.25}$$

where g is the nonlinear function in the ML estimation framework. If all the components are estimated using the symmetric orthogonalization approach, this algorithm can be shown to approximately maximize likelihood.

8.3 Examples

Here we show the results of applying maximum likelihood estimation to the two mixtures introduced in the preceding Chapter. Here, we use whitened data. This is not strictly necessary, but the algorithms converge much better with whitened data. The algorithms were always initialized so that $\mathbf{B}$ was the identity matrix.

First, we used the natural gradient ML algorithm in Table 8.1. In the first example, we used the data consisting of two mixtures of two subgaussian (uniformly distributed) independent components, and took the nonlinearity to be the one in (8.21), corresponding to the density in (8.12). The algorithm did *not* converge properly, as shown in Fig. 8.3. This is because the nonlinearity was not correctly chosen. Indeed, computing the nonpolynomial moment (8.23), we saw that it was negative, which means that the nonlinearity in (8.22) should have been used. Using the correct nonlinearity, we obtained correct convergence, as in Fig. 8.4. In both cases, several hundred iterations were performed.

Next we did the corresponding estimation for two mixtures of two *supergaussian* independent components. This time, the nonlinearity in (8.21) was the correct one, and gave the estimates in Fig. 8.5. This could be checked by computing the nonpolynomial moment in (8.23): It was positive. In contrast, using the nonlinearity in (8.22) gave completely wrong estimates, as seen in Fig. 8.6.

In contrast to the gradient algorithm, FastICA effortlessly finds the independent components in both cases. In Fig. 8.7, the results are shown for the subgaussian data, and in Fig. 8.8, the results are shown for the supergaussian data. In both cases the algorithm converged correctly, in a couple of iterations.

8.4 Likelihood vs. kurtosis

Since now we have two objectives of estimating ICA (likelihood and kurtosis), it is interesting to compare the advantages of the two approaches.

On a conceptual level, maximization of non-Gaussianity using kurtosis is simple, and gives an intuitive understanding of the process. The resulting algorithms are easy to derive as well. However, from the viewpoint of the statistical properties of the resulting estimators, likelihood tends to be much better.

First, likelihood has smaller asymptotic variance for typical densities of the independent components. Asymptotic variance is an approximation of the estimation error which is due to the finite sample size. Thus, the estimation errors are likely to be smaller for likelihood.

Another problem is that kurtosis can be very sensitive to outliers, i.e. observations with very large values, which are likely to be measurement errors. Assume, for example, that a sample of 1000 values of a random variable (with zero mean and unit variance, say) contains one value equal to 10. Then the kurtosis equals at least $10^4/1000 - 3 = 7$, which means that the single value makes kurtosis large. Thus we see that the value of kurtosis may depend on only a few observations in the tails of the distribution, which may be erroneous or irrelevant observations. In other words, kurtosis is not a robust measure of nongaussianity. Likelihood is usually much more robust because it does not typically use fast-growing nonlinear functions such as the fourth power in kurtosis (although this depends on the model you use for non-Gaussian densities).

It is important to notice that actually likelihood can be interpreted as a measure of non-Gaussianity as well. For lack of time, this important connection is not treated in detail on this course; see next Chapter for supplementary material. So, the question is not whether non-Gaussianity or likelihood is a better estimation principle for ICA, but rather what kind of a non-Gaussianity measure is good. The discussion above points out that the non-Gaussianity measure given by likelihood is statistically better than the one given by kurtosis.

Note that the choice of the the optimization algorithm (gradient vs. fixed-point) is independent of the choice of the objective function.

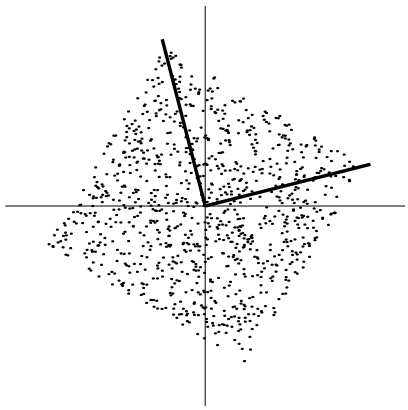

Figure 8.3: Problems of convergence with the (natural) gradient method for maximum likelihood estimation. The data was two whitened mixtures of subgaussian independent components. The nonlinearity was the one in (8.21), which was not correct in this case. The resulting estimates of the columns of the whitened mixing matrix are shown in the figure: they are not aligned with the edges of the square, as they should be.

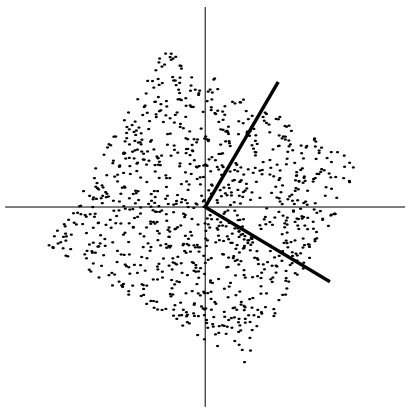

Figure 8.4: The same as in Fig. 8.3, but with the correct nonlinearity, given by (8.22). This time, the natural gradient algorithm gave the right result. The estimated vectors *are* aligned with the edges of the square.

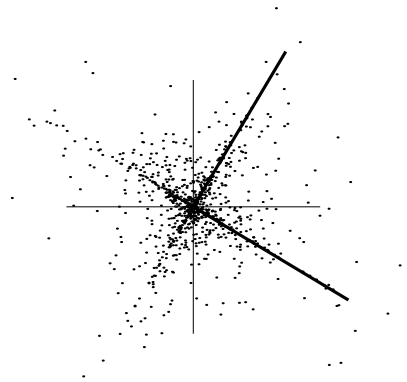

Figure 8.5: In this experiment, data was two whitened mixtures of supergaussian independent components. The nonlinearity was the one in (8.21). The natural gradient algorithm converged correctly.

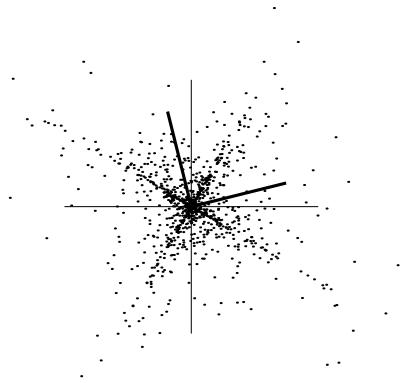

Figure 8.6: Again, problem of convergence with the natural gradient method for maximum likelihood estimation. The nonlinearity was the one in (8.22), which was not correct in this case.

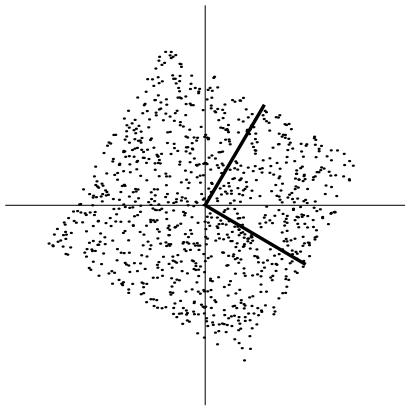

Figure 8.7: FastICA automatically estimates the nature of the independent components, and converges fast to the maximum likelihood solution. Here, the solution was found in 2 iterations for subgaussian independent components.

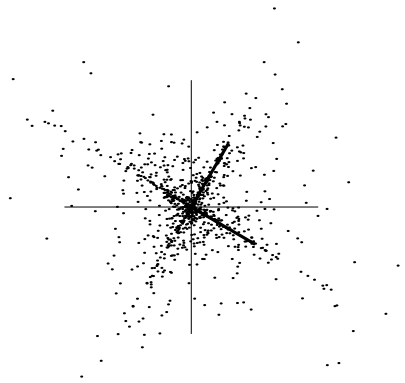

Figure 8.8: FastICA this time applied on supergaussian mixtures. Again, the solution was found in 2 iterations.

Chapter 9

ICA and unifying information-theoretic approach

An important approach for independent component analysis (ICA) estimation is based on information theory. One of the main utilities of this framework is that it serves as a unifying framework for many estimation principles, in particular maximum likelihood (ML) estimation and maximization of nongaussianity. In particular, this approach gives a rigorous justification for the heuristic principle of nongaussianity.

9.1 Definition of entropy

Entropy is the basic concept of information theory. The entropy of a random variable is related to the information that the observation of the variable gives. The more "random", i.e., unpredictable and unstructured the variable is, the larger its entropy. In fact, entropy is closely related to coding length: any regularity, i.e. lack of randomness can be used to reduce coding length. This is the central idea in information theory, but we will not go in depth to that theory, and we will just use entropy as an interesting measure of "structure", loosely defined, in a probability distribution.

For continuous-valued random variables, the entropy is sometimes called differential entropy. The differential entropy H of a random vector $\mathbf{y}$ with density $p_y(\boldsymbol{\eta})$ is defined as

$$H(\mathbf{y}) = -\int p_y(\boldsymbol{\eta}) \log p_y(\boldsymbol{\eta}) \mathrm{d}\boldsymbol{\eta} \tag{9.1}$$

A fundamental result of information theory is that *a gaussian variable has the largest entropy* among all random variables of equal variance. This means that entropy could be used as a measure of nongaussianity. In fact, this shows that the gaussian distribution is the "most random" or the least structured of all distributions.

It is not difficult to see what kind of random variables have small differential entropies. They are the ones whose probability densities take large values, since these give strong negative contributions to the integral in Eq. (9.1). This means that certain small intervals are quite probable. Entropy is thus small for distributions that are clearly concentrated on certain values, i.e., when the variable is clearly clustered, or has a pdf that is very "spiky". We see that entropy is small when the variable is not very random, that is, it is typically contained in some limited intervals with high probabilities.

9.2 Entropy as a robust non-Gaussianity measure

In Chapter 7, we showed how to measure nongaussianity by kurtosis, thus obtaining a simple ICA estimation method. However, kurtosis is not robust against outliers. Thus, other measures of nongaussianity might be better than kurtosis in practice.

Here we shall consider a variant of entropy called *negentropy*, which is the second important measure of nongaussianity. Its properties are in many ways opposite to those of kurtosis: It is robust but computationally complicated.

To obtain a measure of nongaussianity that is zero for a gaussian variable and always nonnegative, one often uses a normalized version of differential entropy, called negentropy. Negentropy J is defined as follows

$$J(\mathbf{y}) = H(\mathbf{y}_{gauss}) - H(\mathbf{y}) \tag{9.2}$$

where $\mathbf{y}_{gauss}$ is a gaussian random variable of the same correlation (and covariance) matrix as $\mathbf{y}$. Due to the above-mentioned maximality property of the gaussian distribution, negentropy is always nonnegative, and it is zero if and only if $\mathbf{y}$ has a gaussian distribution. Thus, it is a well-defined measure of non-Gaussianity.

The advantages of using negentropy, or equivalently, differential entropy, as a measure of nongaussianity is that it is usually much more robust against outliers than kurtosis, and it is well justified by statistical theory. In fact, using negentropy is closely related to likelihood, as we will show next.

9.3 Entropy and likelihood

To see the connection between likelihood and (neg)entropy, consider the expectation of the log-likelihood in (8.6):

$$\frac{1}{T}E\{\log L(\mathbf{B})\} = \sum_{i=1}^{n} E\{\log p_i(\mathbf{b}_i^T\mathbf{x})\} + \log|\det \mathbf{B}| \tag{9.3}$$

If the p_i were equal to the actual pdf's of $\mathbf{b}_i^T\mathbf{x}$, the first term would be equal to $-\sum_i H(\mathbf{b}_i^T\mathbf{x})$. In practice, the connection may be just as strong. This is because in practice we do not know the distributions of the independent components that are needed in ML estimation. A reasonable approach would be to estimate the density of $\mathbf{b}_i^T\mathbf{x}$ as part of the ML estimation method, and use this as an approximation of the density of s_i. Then, the p_i in this approximation of likelihood are indeed approximately equal to the actual pdf's of $\mathbf{b}_i^T\mathbf{x}$.

To develop the connection further, let us further constrain the y_i to be *uncorrelated* and of unit variance, as we did in the non-gaussianity-based ICA methods. This means $E\{\mathbf{y}\mathbf{y}^T\} = \mathbf{B}E\{\mathbf{x}\mathbf{x}^T\}\mathbf{B}^T = \mathbf{I}$, which implies

$$\det \mathbf{I} = 1 = \det(\mathbf{B}E\{\mathbf{x}\mathbf{x}^T\}\mathbf{B}^T) = (\det \mathbf{B})(\det E\{\mathbf{x}\mathbf{x}^T\})(\det \mathbf{B}^T) \tag{9.4}$$

and this implies that $\det \mathbf{B}$ must be constant since $\det E\{\mathbf{x}\mathbf{x}^T\}$ does not depend on $\mathbf{B}$. Moreover, for y_i of unit variance, entropy and negentropy differ only by a constant and the sign, as can be seen in (9.2). Thus we obtain,

$$\frac{1}{T}E\{\log L(\mathbf{B})\} = \text{const.} - \sum_i J(y_i) \tag{9.5}$$

where the constant term does not depend on $\mathbf{B}$. This shows the fundamental relation between negentropy and likelihood.

We see in (9.5) that finding an invertible linear transformation $\mathbf{B}$ that maximizes likelihood is roughly equivalent to finding directions in which the negentropy is maximized. We have seen previously that negentropy is a measure of nongaussianity. Thus, (9.5) shows that *ICA estimation by maximization of likelihood is equivalent to maximizing the sum of nongaussianities of the estimates of the independent components*, when the estimates are constrained to be uncorrelated.

Thus, we see that maximum likelihood estimation gives another rigorous justification of our more heuristically introduced idea of finding maximally nongaussian directions.

In practice, however, there are also some differences between these two criteria.

1. Negentropy, and other measures of nongaussianity, enable the deflationary, i.e., one-by-one, estimation of the independent components, since we can look for the maxima of nongaussianity of a single projection $\mathbf{b}^T\mathbf{x}$. This is not possible with likelihood.
2. A smaller difference is that in using nongaussianity, we force the estimates of the independent components to be uncorrelated. This is not necessary when using likelihood. Thus the optimization space is slightly reduced. This may slightly deteriorate the quality of the estimator, but in most cases, this deterioration is not significant.

9.4 Mutual information as measure of dependence

9.4.1 Definition

Another motivation for the information-theoretic approach is that it may not be very realistic in many cases to assume that the data follows the ICA model. Therefore, we would like to develop an approach that does not assume anything about the data. What we want to have is a general-purpose measure of the dependence of the components of a random vector. Using such a measure, we could define ICA as a linear decomposition that minimizes that dependence measure. Such an approach can be developed using mutual information, which is a well-motivated information-theoretic measure of statistical dependence.

Mutual information I between m (scalar) random variables, $y_i, i = 1...m$ is defined as follows

$$I(y_1, y_2, ..., y_m) = \sum_{i=1}^{m} H(y_i) - H(\mathbf{y}) \tag{9.6}$$

Mutual information is a natural measure of the dependence between random variables. It is always nonnegative, and zero if and only if the variables are statistically independent.

An intuitive interpretation is that just looking at the $H(y_i)$ means considering the structure in the marginal distributions alone, and neglecting any structure in the dependencies (e.g. correlations) of the variables. In fact, if you assume that the y_i are independent, it is rather easy to show that the $H(\mathbf{y})$ equals the sum of $H(y_i)$. The idea is that any correlations can only reduce $H(\mathbf{y})$, and therefore it cannot be larger than the sum of $H(y_i)$.[1]

Mutual information takes into account the whole dependence structure of the variables, and not just the covariance like whitening and related methods. Therefore, we can use mutual information as the criterion for finding the ICA representation. This approach is an alternative to the model estimation approach. We define the ICA of a random vector $\mathbf{x}$ as an invertible transformation:

$$\mathbf{s} = \mathbf{B}\mathbf{x} \tag{9.7}$$

where the matrix $\mathbf{B}$ is determined so that the mutual information of the transformed components s_i is minimized. If the data follows the ICA model, this allows estimation of the data model. On the other hand, in this definition, we do not need to assume that the data follows the model. In any case, minimization of mutual information can be interpreted as giving the maximally independent components.

9.4.2 Transformation formula for entropy

Consider an *invertible* transformation of the random vector $\mathbf{x}$, say

$$\mathbf{y} = \mathbf{f}(\mathbf{x}) \tag{9.8}$$

In this section, we show the connection between the entropy of $\mathbf{y}$ and that of $\mathbf{x}$.

A short, if somewhat sloppy derivation is as follows. Denote by $J\mathbf{f}(\boldsymbol{\xi})$ the Jacobian matrix of the function $\mathbf{f}$, i.e., the matrix of the partial derivatives of $\mathbf{f}$ at point $\boldsymbol{\xi}$. The classic relation between the density p_y of $\mathbf{y}$ and the density p_x of $\mathbf{x}$, can then be formulated as

$$p_y(\boldsymbol{\eta}) = p_x(\mathbf{f}^{-1}(\boldsymbol{\eta}))|\det J\mathbf{f}(\mathbf{f}^{-1}(\boldsymbol{\eta}))|^{-1} \tag{9.9}$$

Now, expressing the entropy as an expectation

$$H(\mathbf{y}) = -E\{\log p_y(\mathbf{y})\} \tag{9.10}$$

we get

$$\begin{aligned} E\{\log p_y(\mathbf{y})\} &= E\{\log[p_x(\mathbf{f}^{-1}(\mathbf{y}))|\det J\mathbf{f}(\mathbf{f}^{-1}(\mathbf{y}))|^{-1}]\} \\ &= E\{\log[p_x(\mathbf{x})|\det J\mathbf{f}(\mathbf{x})|^{-1}]\} = E\{\log p_x(\mathbf{x})\} - E\{\log|\det J\mathbf{f}(\mathbf{x})|\} \end{aligned} \tag{9.11}$$

Thus we obtain the relation between the entropies as

$$H(\mathbf{y}) = H(\mathbf{x}) + E\{\log|\det J\mathbf{f}(\mathbf{x})|\} \tag{9.12}$$

In other words, the entropy is increased in the transformation by $E\{\log|\det J\mathbf{f}(\mathbf{x})|\}$.

An important special case is *the linear transformation*

$$\mathbf{y} = \mathbf{M}\mathbf{x} \tag{9.13}$$

in which case we obtain

$$H(\mathbf{y}) = H(\mathbf{x}) + \log|\det\mathbf{M}| \tag{9.14}$$

This also shows that differential entropy is *not scale-invariant*. Consider a random variable x. If we multiply it by a scalar constant, α, differential entropy changes as

$$H(\alpha x) = H(x) + \log|\alpha| \tag{9.15}$$

Thus, just by changing the scale, we can change the differential entropy. This is why the scale of x often is fixed before measuring its differential entropy.

[1] This can be understood from the coding length interpretation of entropy: $H(y_i)$ is the coding length of coding the variable y_i separately, where as $H(\mathbf{y})$ is the length of coding the whole vector. Using the whole vector is always better for coding because then the whole structure of the data is taken into account, and therefore the $H(\mathbf{y})$ is not larger than the sum of the $H(y_i)$. If the variables are independent, then no improvement in coding can be obtained by coding them together, and therefore $H(\mathbf{y})$ is equal to $\sum_{i=1}^{m} H(y_i)$, and mutual information is zero.

9.4.3 Mutual information and likelihood

Using the formula for the differential entropy of a transformation we obtain a corresponding result for mutual information. We have for an invertible linear transformation $\mathbf{y} = \mathbf{B}\mathbf{x}$:

$$I(y_1, y_2, ..., y_n) = \sum_{i=1}^{n} H(\mathbf{b}_i^T \mathbf{x}) - H(\mathbf{x}) - \log |\det \mathbf{B}| \tag{9.16}$$

Note that $H(\mathbf{x})$ does not depend on $\mathbf{B}$.

Mutual information and likelihood are intimately connected. To approximate mutual information, we could take a fixed approximation of the densities y_i, and plug this in the definition of entropy. Denote the pdf's by $G_i(y_i) = \log p_i(y_i)$. Then we could approximate (9.16) as

$$I(y_1, y_2, ..., y_n) = -\sum_{i} E\{G_i(y_i)\} - \log |\det \mathbf{B}| - H(\mathbf{x}) \tag{9.17}$$

Now we see that this approximation is equal to the approximation of the likelihood used earlier (except, again, for the global sign and the additive constant given by $H(\mathbf{x})$).

Thus, using mutual information does not really lead to new methods but simply shows that using likelihood is approximately equivalent to minimizing mutual information, which is arguably the most principled measure of dependence of components.

Chapter 10

Applications of component analyses

10.1 Denoising and compression by PCA and FA

These will be treated in computer assignments.

10.2 Blind separation of brain sources in MEG data

The classic application of ICA is in blind source separation. The idea is that each of the independent components corresponds to a "signal" (i.e. random variable) emitted by an independent source. If such sources are physically independent, i.e. they "work" independently of each other, the signals emitted are likely to be statistically independent as well.

This is often called "blind source separation", where "blind" refers to the idea that we don't know much about the signals. In fact, in ICA we only assume that they are independent and non-Gaussian, which means we don't really assume anything on the actual form of the signals (as opposed to, for example, the assumption that a signal has activity only at the frequency of 10Hz, which is typical with the data we analyze in this section).

As an example, consider a set of measurement of brain activity by magnetoencephalography (MEG).[1] It has 122 observed variables, and 17,730 observations (time points) of each of them. The variables correspond to the measurements of different sensors of the scalp, and the values of the variables are related to brain activity, or some artifacts produced by muscles, technical devices or other sources which are not in the brain.

Fig. 10.1 shows a random collection of 10 of such measured variables. What we can see is that there seem to be some underlying signals which are really mixed in these measurements.

Fig. 10.2 shows the first 10 principal components of the data. The principal components look similar to the original variables. They don't seem to "separate" any interesting sources. This is understandable since PCA is essentially a method for factor analysis for gaussian data, and therefore, the factor rotation remains undetermined.

Fig. 10.3 shows the independent components obtained by applying ICA on the 10 principal components. The components have been ordered based on their kurtoses, the largest first. We can see clearly that some of the components do separate some signals which were mixed in many observed signals.

The kurtoses of the components are shown in Fig. 10.4. We can see that they are mainly positive, which is typically the case in real-world signals. The histograms of the ten first independent components are shown in Fig. 10.5. The histograms show that in addition to supergaussianity, many of the distributions are asymmetric (also called "skewed"), which is another form of non-Gaussianity in addition to the sub- and supergaussianity characteristic.

Fig. 10.6 shows some of the independent components obtained by applying ICA on the 30 principal components, which is more than in above. The components have been ordered based on their kurtoses, the largest first. First, we see that the number of principal components chosen does have a large effect on ICA. It seems also that there are more components which are clearly separated, which indicates that 10 may have been too small a dimension to use.

[1] available as a matlab file at
`http://www.cis.hut.fi/projects/ica/eegmeg/MEG_art.gz`

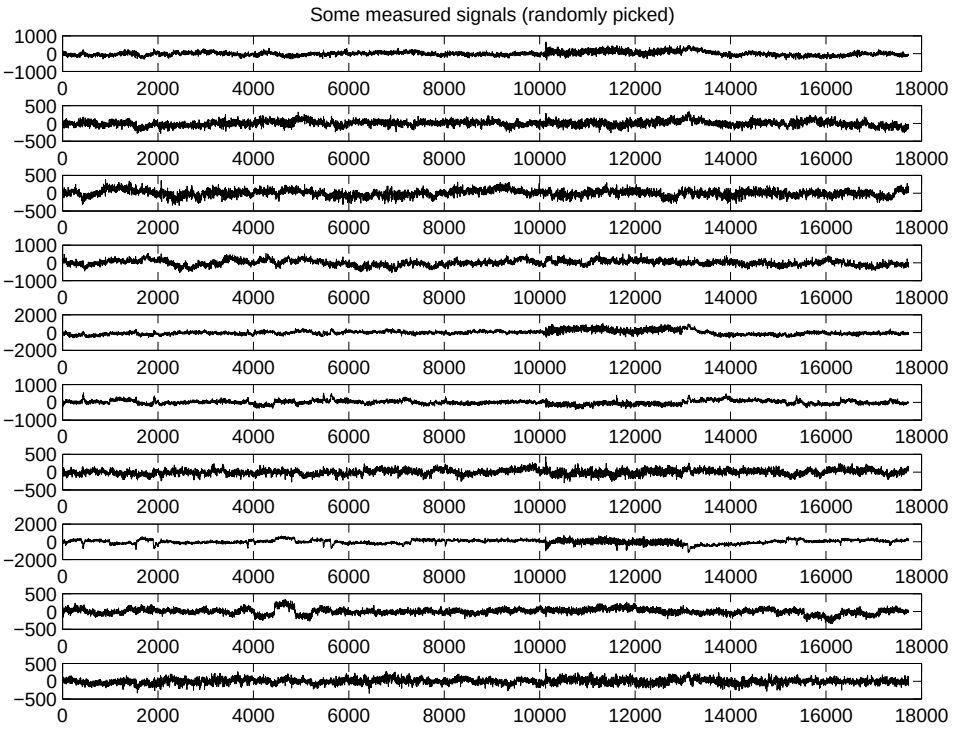

Figure 10.1: Some of the original MEG signals.

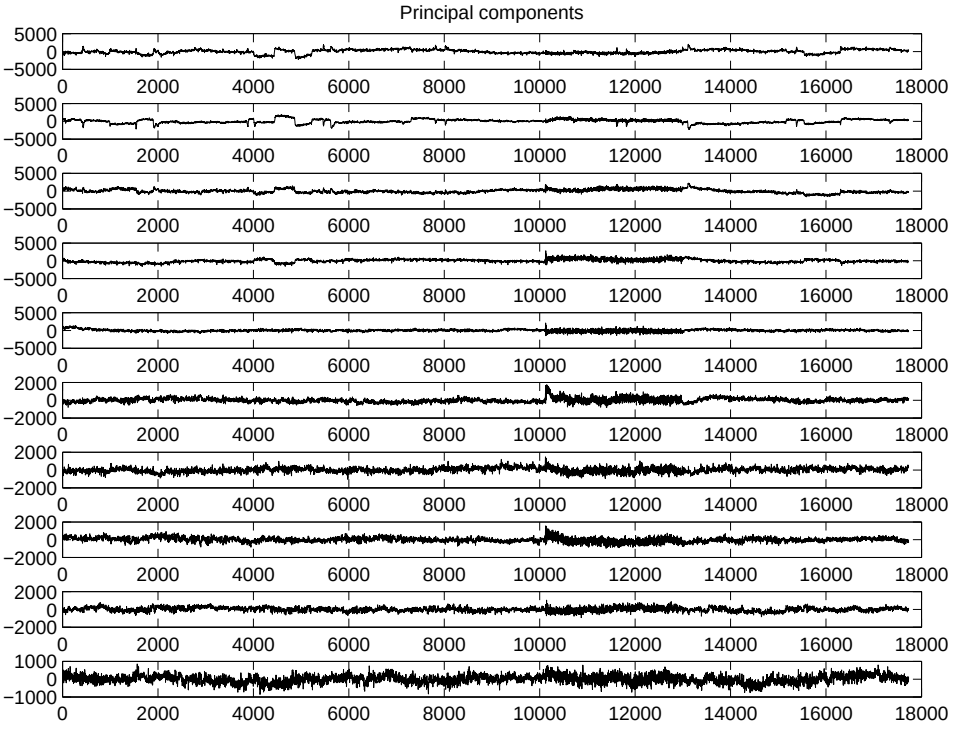

Figure 10.2: The 10 first principal components of the MEG signals.

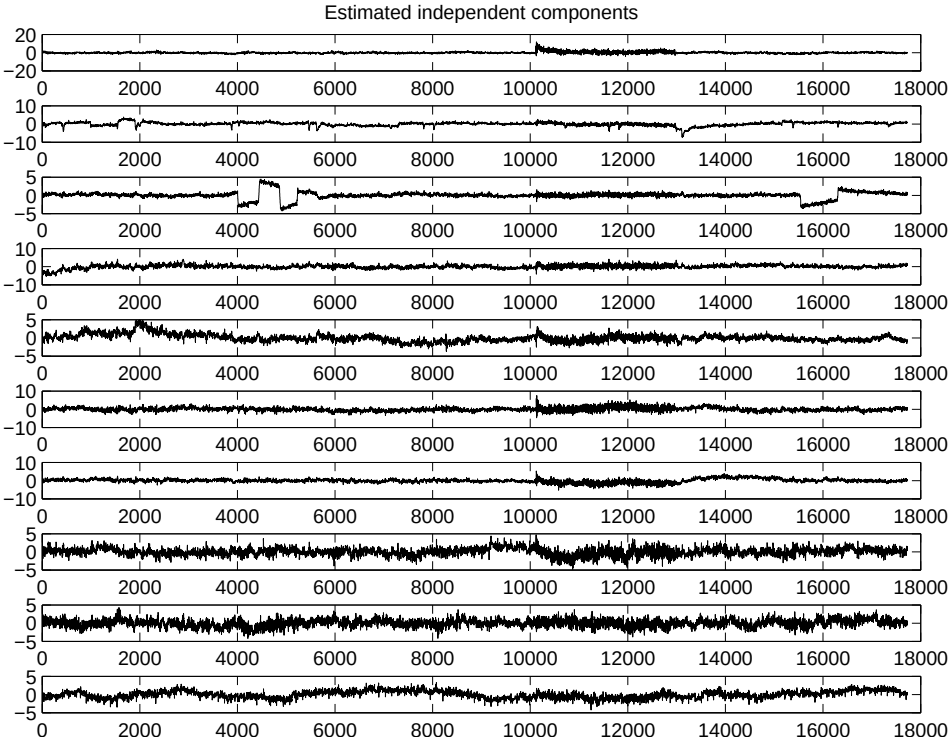

Figure 10.3: The independent components computed from 10 principal components.

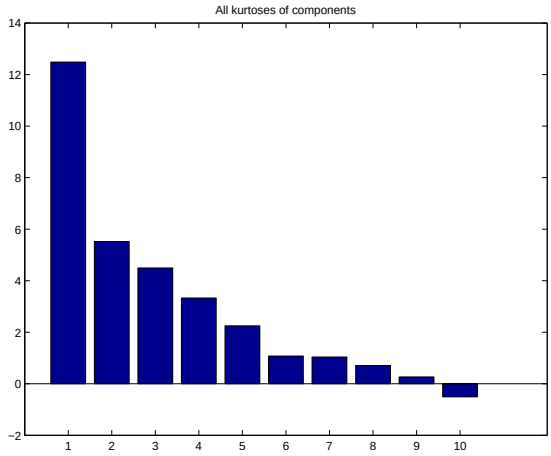

Figure 10.4: The kurtoses of the independent components computed from 10 principal components.

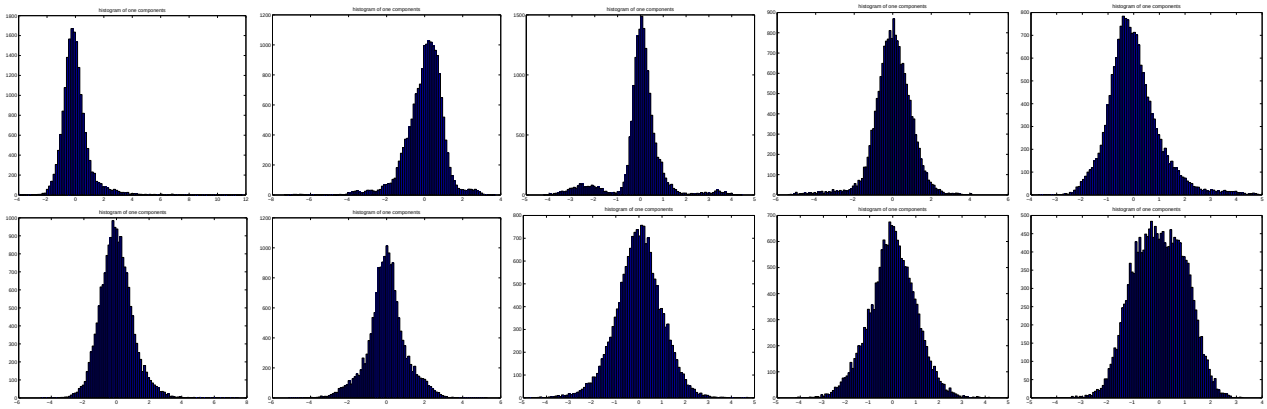

Figure 10.5: The histogram of the independent components computed from 10 principal components.

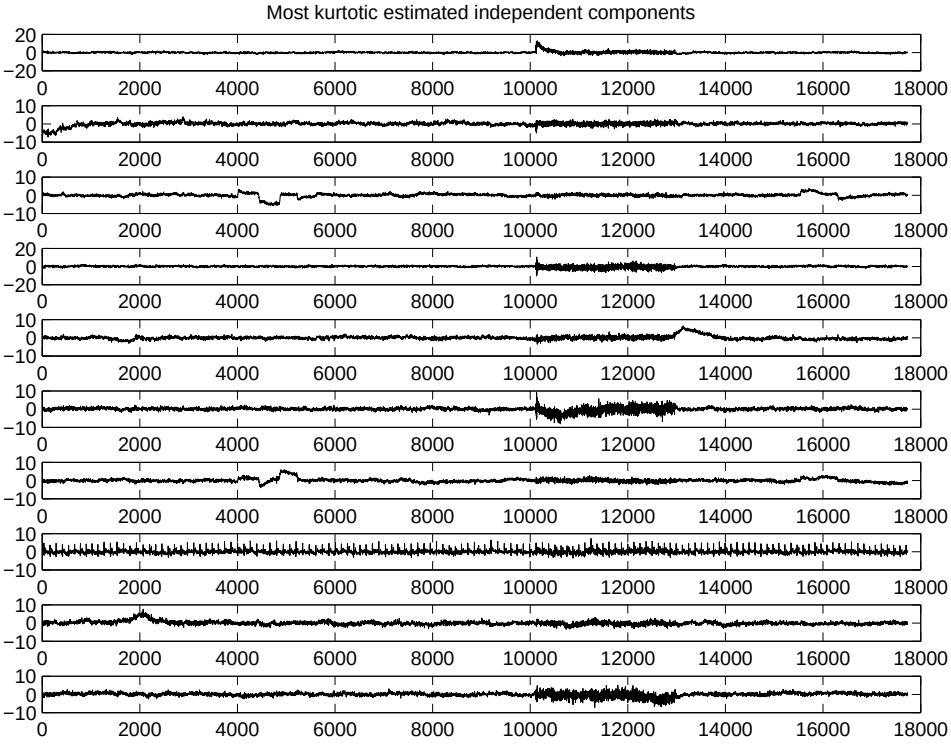

Figure 10.6: The independent components computed from 30 principal components.

10.3 Image feature extraction

10.3.1 Motivation

A fundamental approach in signal processing is to design a statistical generative model of the observed signals. The components in the generative model then give a representation of the data. Such a representation can then be used in such tasks as compression, denoising, and pattern recognition. This approach is also useful from a neuroscientific viewpoint, for modeling the properties of neurons in primary sensory areas.

Here, we consider a certain class of widely used signals, which we call natural images. This means images that we encounter in our lives all the time; images that depict wild-life scenes, human living environments, etc. The working hypothesis here is that this class is sufficiently homogeneous so that we can build a statistical model using observations of those signals, and then later use this model for processing the signals, for example, to compress or denoise them.

We will see that ICA does provide a model that is very similar to the most sophisticated low-level image representations used in image processing and vision research. ICA gives a statistical justification for using those methods that have often been more heuristically justified.

10.3.2 Linear representations

Definition

Image representations are often based on discrete linear transformations of the observed data. Consider a black-and-white image whose gray-scale value at the pixel indexed by x and y is denoted by $I(x, y)$. Many basic models in image processing express the image $I(x, y)$ as a linear superposition of some features or basis functions $a_i(x, y)$:

$$I(x, y) = \sum_{i=1}^{n} a_i(x, y)s_i \tag{10.1}$$

where the s_i are stochastic coefficients, different for each image $I(x, y)$. Alternatively, we can just collect all the pixel values in a single vector $\mathbf{x} = (x_1, x_2, ..., x_m)^T$, in which case we can express the representation as

$$\mathbf{x} = \mathbf{As} \tag{10.2}$$

just like in basic ICA. We assume here that the number of transformed components equals the number of observed variables, although this need not be the case in general. This kind of a linear superposition model gives a useful description on a low level where we can ignore such higher-level nonlinear phenomena as occlusion.

In practice, we may not model a whole image using the model in (10.1). Rather, we apply it on image patches or windows. Thus we partition the image into patches of, for example, 8×8 pixels and model the patches with the model in (10.1). Care must then be taken to avoid border effects.

Standard linear transformations widely used in image processing are, for example, the Fourier, Haar, Gabor, and cosine transforms. Each of them has its own favorable properties. Recently, a lot of interest has been aroused by methods that attempt to combine the good qualities of frequency-based methods (Fourier and cosine transforms) with the basic pixel-by-pixel representation.

Gabor analysis

Gabor functions or Gabor filters are functions that are extensively used in image processing. These functions are localized with respect to three parameters: spatial location, orientation, and frequency. This is in contrast to Fourier basis function that are not localized in space, and the basic pixel-by-pixel representation that is not localized in frequency or orientation.

Let us first consider, for simplicity, one-dimensional (1-D) Gabor functions instead of the two-dimensional (2-D) functions used for images. The Gabor functions are then of the form

$$g_{1d}(x) = \exp(-\alpha^2(x - x_0)^2)[\cos(2\pi\beta(x - x_0) + \gamma) + i\sin(2\pi\beta(x - x_0) + \gamma)] \tag{10.3}$$

where

- α is the constant in the gaussian modulation function, which determines the width of the function in space.
- x_0 defines the center of the gaussian function, i.e., the location of the function.
- β is the frequency of oscillation, i.e., the location of the function in Fourier space.
- γ is the phase of the harmonic oscillation.

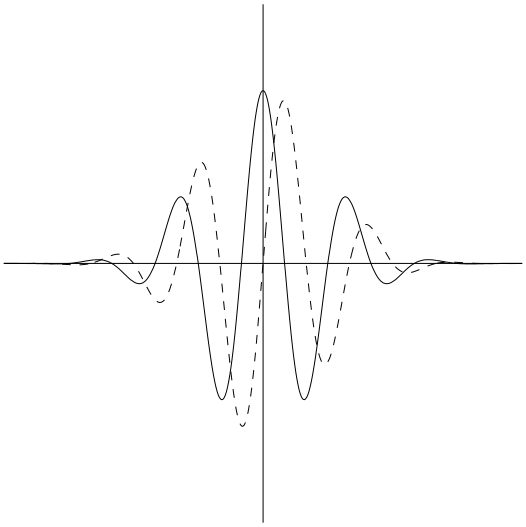

Figure 10.7: A pair of 1-D Gabor functions. These functions are localized in space as well as in frequency. The real part is given by the solid line and the imaginary part by the dashed line.

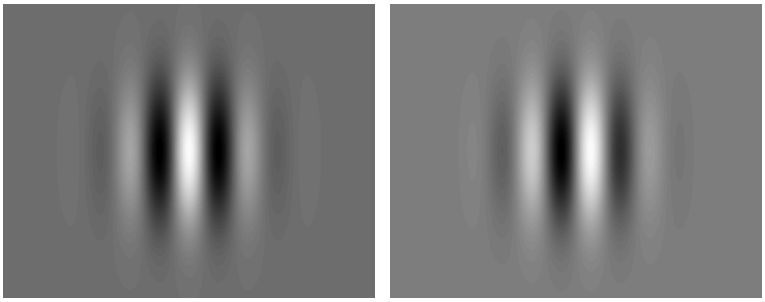

Figure 10.8: A pair of 2-D Gabor functions. These functions are localized in space, frequency, and orientation. The real part is on the left, and the imaginary part on the right. These functions have not been rotated.

Actually, one Gabor function as in (10.3) defines two scalar functions: One as its real part and the other one as its imaginary part. Both of these are equally important, and the representation as a complex function is done mainly for algebraic convenience. A typical pair of 1-D Gabor functions is plotted in Fig. 10.7.

Two-dimensional Gabor functions are created by first taking a 1-D Gabor function along one of the dimensions and multiplying it by a gaussian envelope in the other dimension:

$$g_{2d}(x, y) = \exp(-\alpha^2 (y - y_0)^2) g_{1d}(x) \tag{10.4}$$

where the parameter α in the gaussian envelope need not be the same in both directions. Second, this function is rotated by an orthogonal transformation of (x, y) to a given angle. A typical pair of the real and imaginary parts of a Gabor functions are shown in Fig. 10.8.

Gabor analysis is an example of multi-resolution analysis, which means that the image is analyzed separately at different resolutions, or frequencies. This is because Gabor functions can be generated at different sizes by varying the parameter α, and at different frequencies by varying β.

An open question is what set of values should one choose for the parameters to obtain a useful representation of the data.

10.3.3 ICA and Sparse Coding

The transforms just considered are fixed transforms, meaning that the basis vectors are fixed once and for all, independent of any data. In many cases, however, it would be interesting to estimate the transform from data. Estimation of the representation in Eq. (10.1) consists of determining the values of s_i and $a_i(x, y)$ for all i and (x, y), given a sufficient number of observations of images, or in practice, image patches $I(x, y)$.

The question is then: What principles should be used to estimate a transform from the data? Our starting point here is a representation principle called *sparse coding* that has recently attracted interest both in signal processing and in theories on the visual system. In sparse coding, the data vector is represented using a set of basis vectors so that *only a small number of basis vectors are activated at the same time.*

In a neural network interpretation, each basis vector corresponds to one neuron, and the coefficients s_i are given by their activations. Thus, only a small number of neurons is activated for a given image patch. Thus, the principle of sparse coding could be expressed by the property that *a given neuron is activated only rarely.* This means that the coefficients s_i have sparse distributions. The distribution of s_i is called sparse when s_i has a probability density with a peak at zero, and heavy tails, which is the case, for example, with the Laplacian (or double exponential) density. In general, sparseness can be equated with supergaussianity.

In the simplest case, we can assume that the sparse coding is linear, in which case sparse coding fits into the framework used in this chapter. One could then estimate a linear sparse coding transformation of the data by formulating a measure of sparseness of the components, and maximizing the measure in the set of linear transformations. In fact, since sparsity is closely related to supergaussianity, ordinary measures of nongaussianity, such as kurtosis and the approximations of negentropy, could be interpreted as measures of sparsity as well. Maximizing sparsity is thus one method of maximizing nongaussianity, and we saw in earlier that maximizing nongaussianity of the components is one method of estimating the ICs. Thus, sparse coding can be considered as one method for ICA. At the same time, sparse coding gives a different interpretation of the goal of the transform.

10.3.4 Estimating ICA bases from images

Thus, ICA and sparse coding give essentially equivalent methods for estimating features from natural images, or other kinds of data sets. Here we show the results of such an estimation.

First, we must note that ICA applied to image data usually gives one component representing the local mean image intensity, often called the DC component. This component normally has a distribution that is not sparse; often it is even subgaussian. Thus, it must be treated separately from the other, supergaussian components, at least if the sparse coding interpretation is to be used. Therefore, in all experiments we first subtract the local mean, and then estimate a suitable sparse coding basis for the rest of the components. Because the data then has lost one linear dimension, the dimension of the data must be reduced, for example, using principal component analysis (PCA).

Each image was first linearly normalized so that the pixels had zero mean and unit variance. A set of 50,000 image patches (windows) of 32×32 pixels were taken at random locations from the images. From each patch the local mean was subtracted as just explained. To remove noise, the dimension of the data was reduced to 256. The preprocessed dataset was used as the input to the FastICA algorithm, using the tanh nonlinearity (i.e. a robust measure of non-gaussianity).

The principal component weights are shown in Fig. 10.9. These are a bit like Fourier basis vectors, i.e. sinusoids. However, the latter ones look more like noise.

Figure 10.10 shows the ICA basis vectors A_i. Alternatively, we can characterize the results by plotting the rows of the inverse of the mixing matrix: Figure 10.11 shows these "feature detectors". The basis vectors are clearly localized in space, as well as in frequency and orientation. Thus the features are *closely related to Gabor functions.* In fact, one can approximate these basis functions by Gabor functions, so that for each basis vector one minimizes the squared error between the basis vector and a Gabor function. This gives very good fits, and shows that Gabor functions are a good approximation. Alternatively, one could characterize the ICA basis functions by noting that many of them could be interpreted as edges or bars.

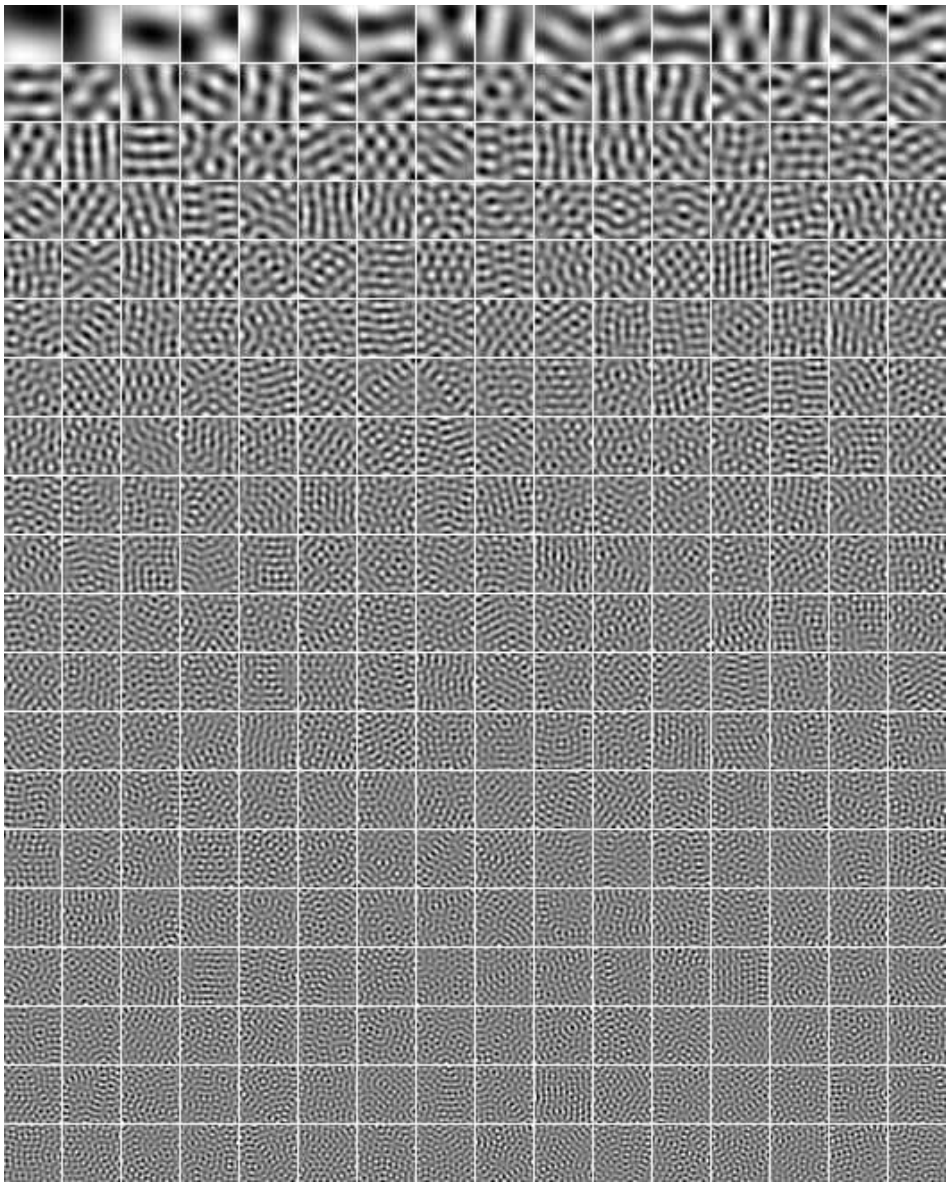

Figure 10.9: The 320 first principal components weights W_i of image patches of size 32×32. The order of decreasing variances is left to right on each row, and top to bottom. For each principal component, the weights are plotted as an image patch.

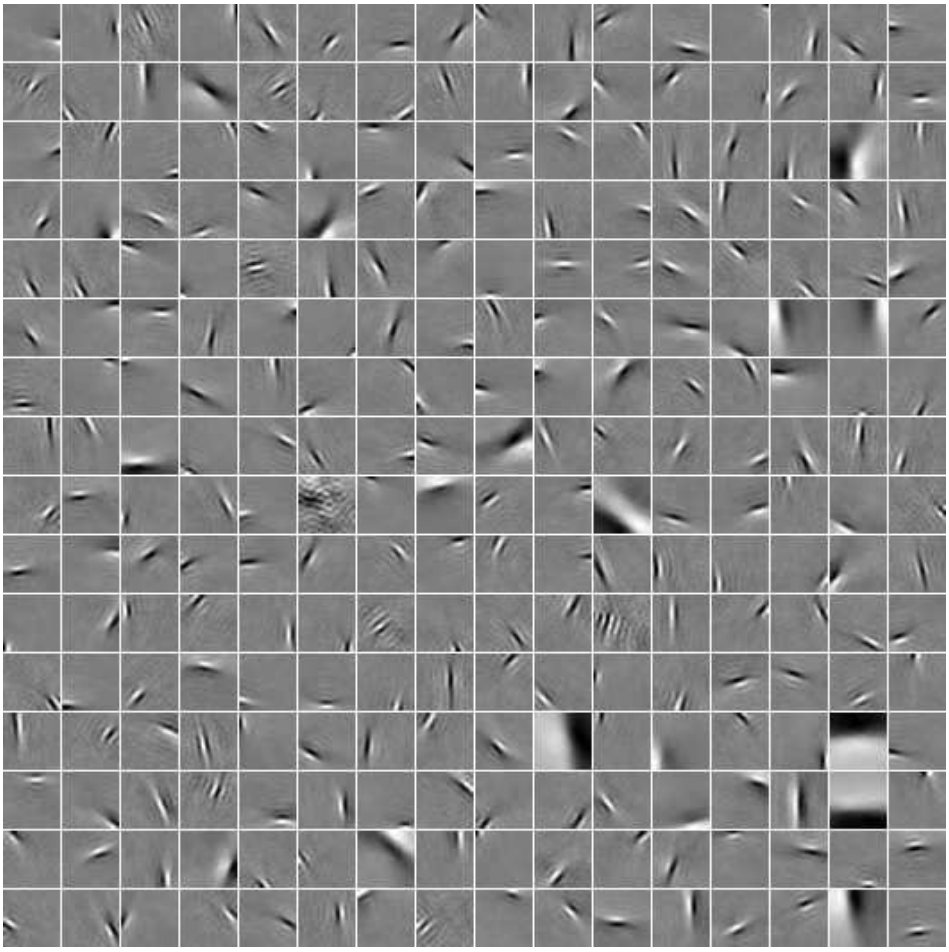

Figure 10.10: The whole set of features A_i obtained by ICA. This was accomplished by FastICA with a symmetric orthogonalization and the tanh nonlinearity

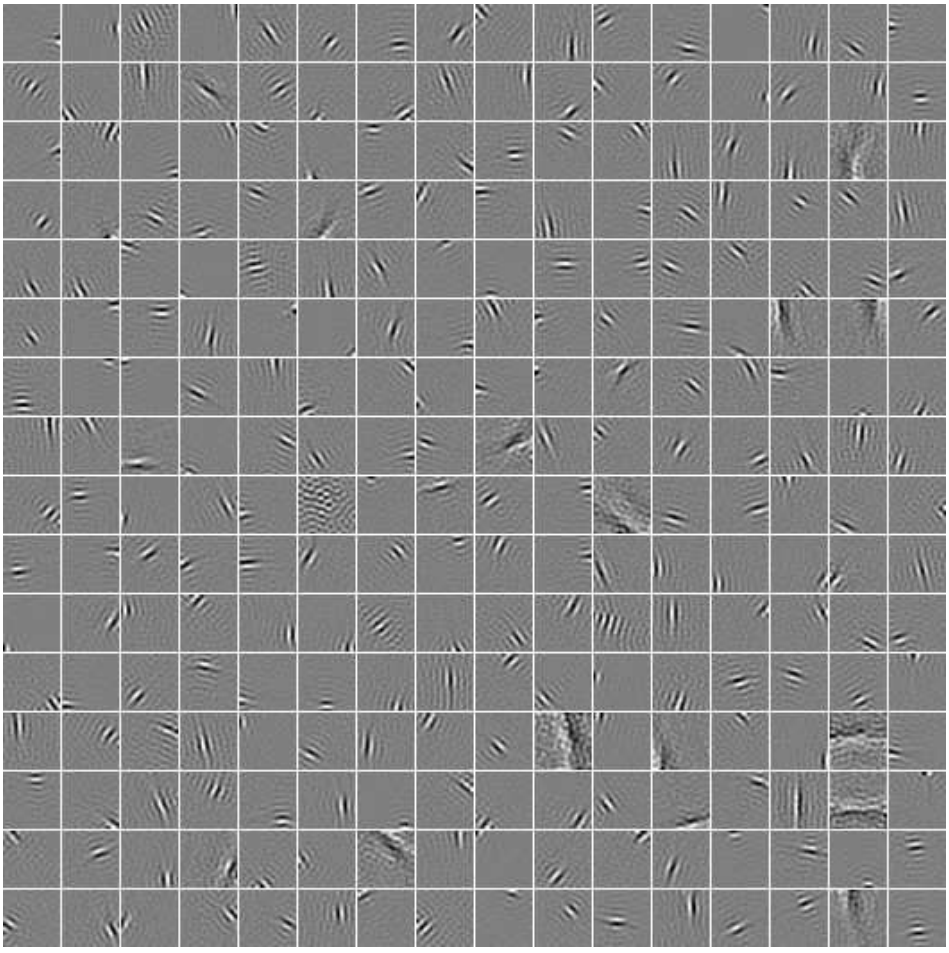

Figure 10.11: The whole set of feature detectors W_i obtained by ICA. Note that due to the lack of orthogonality of the mixing matrix, the rows of its inverse are not equal to its columns. For an orthogonal matrix, like the one given by PCA, there is no such difference.

Chapter 11

Sparse coding with overcomplete basis

A variant of ICA which has recently become popular in machine learning is sparse coding with overcomplete basis, also called (overcomplete) dictionary learning.

11.1 Motivation

An important restriction basic ICA is that the number of components cannot be larger than the dimension of the data. In the generative models such as ICA, we had to assume that the matrix $\mathbf{A}$ is invertible. A matrix can be invertible only if it is square: thus the number of components cannot be larger than the number of data variables. However, it can be argued that the number of components or features should be larger than the dimension for certain kinds of data.

11.2 Definition of generative model

Now we define a generative model which has more components than the data has dimensions. In this context, we call the columns $\mathbf{a}_i$of $\mathbf{A}$ *basis vectors* or *dictionary atoms/elements*. A set of basis vectors which contains more vectors than the space has dimensions is called an *overcomplete basis* or *overcomplete dictionary*.

The definition of a generative model with an overcomplete basis is rather straightforward. We just need to express the image as a linear superposition

$$\mathbf{x} = \sum_{i=1}^{m} \mathbf{a}_i s_i \tag{11.1}$$

where the only difference to previous models is that the number of components m is arbitrarily large. We also need to specify the statistical properties of the components s_i. In the basic case, we assume that they are sparse and statistically independent.

For technical reasons, another modification is also usually introduced at this point: we assume that the image is not exactly a linear sum of the components, but there is noise as well. That is, gaussian noise $\boldsymbol{\nu}$ is added to each pixel:

$$\mathbf{x} = \sum_{i=1}^{m} \mathbf{a}_i s_i + \boldsymbol{\nu} \tag{11.2}$$

This does not change the behaviour of the model very much, especially if the noise level is small, but it simplifies the computations in this case. (In the case of basic ICA, introduction of noise in the model just complicates things, so it is usually neglected.)

Note that the meaning of overcompleteness changes when the dimension is reduced by PCA. From the viewpoint of statistical modelling, the dimension of the data is then the dimension given by PCA. So, even a basis which has the same number of vectors as there are pixels can be called overcomplete, because the number of pixels is larger than the PCA-reduced dimension.

Despite the simplicity of the definition of the model, the overcomplete basis model is much more complicated to estimate. What is interesting is that it has a richer behaviour than the basic sparse coding and ICA models because it leads to some nonlinearities in the computation of the components. We will treat this point first.

11.3 Nonlinear computation of the basis coefficients

Consider first the case where the basis vectors $\mathbf{a}_i$ are given, and we want to compute the coefficients s_i for an input data point (e.g. image) $\mathbf{x}$. The fundamental problem is that the linear system given by the basis vectors $\mathbf{a}_i$ is not invertible: If one tries to solve for the s_i given an $\mathbf{x}$, there are more unknowns s_i than there are equations. So, computation of the s_i seems impossible. Indeed, it is impossible in the sense that even if the data (image) were created as a linear sum of the $\mathbf{a}_i$ for some coefficient values s_i, we cannot recover those original coefficients from the input data alone, without some further information.

As an illustration, consider an image with two pixels with values $(1, 1)$. Assume we use a basis with three vectors: $(0, 1)$, $(1, 0)$, and $(1, 1)$. Thus, we have

$$(1, 1) = (0, 1)s_1 + (1, 0)s_2 + (1, 1)s_3 \tag{11.3}$$

Obviously, we could represent the image by setting $s_1 = 0$, $s_2 = 0$, and $s_3 = 1$. But, equally well, we could set $s_1 = 1$, $s_2 = 1$, and $s_3 = 0$. Even if the image was exactly generated using one of these choices for s_i, we cannot tell which one it was by using information in the image alone. Actually, there is an infinite number of different solutions: you could take any weighted average with of the two solution just given, and it would be a solution as well.

However, there is a partial solution to this problem. The key is to use sparseness. Since we know that the s_i are sparse, we can try decide to find the *sparsest solution*. In the illustration above, we would choose the solution $s_1 = 0$, $s_2 = 0$, and $s_3 = 1$ because it is the sparsest possible in the sense that only one coefficient is different from zero.[1]

There is a clear probabilistic justification for such a procedure. Basically, we can find the most probable values for the coefficients s_i, under the assumption that the s_i have sparse distributions. This is possible by using conditional probabilities in a manner similar to Bayes' rule. Now we will derive the procedure based on probabilistic reasoning. By the definition of conditional pdf's, we have

$$p(\mathbf{s}|\mathbf{x}) = \frac{p(\mathbf{s}, \mathbf{x})}{p(\mathbf{x})} = \frac{p(\mathbf{x}|\mathbf{s})p(\mathbf{s})}{p(\mathbf{x})} \tag{11.4}$$

which is the basis for Bayes' rule. The formula can be simplified because $p(\mathbf{x})$ does not depend on $\mathbf{s}$. Since our goal is to find the $\mathbf{s}$ which maximizes $p(\mathbf{s}|\mathbf{x})$, we can just ignore this constant. We can also maximize its logarithm instead because it is often simpler, and equivalent because logarithm is a strictly increasing function. This gives us the following objective function to maximize:

$$\log p(\mathbf{x}|\mathbf{s}) + \log p(\mathbf{s}) \tag{11.5}$$

Such estimation of the $\mathbf{s}$ is called maximum a posteriori (MAP) estimation.

Now, we have to compute the probabilities $\log p(\mathbf{x}|\mathbf{s})$ and $\log p(\mathbf{s})$ needed. The first thing we consider is the *prior* distribution $p(\mathbf{s})$ of the s_i. In Bayesian inference, the prior distribution (or prior for short) incorporates the knowledge we have before making any observations. What prior knowledge do we have here? First, we know that the components are sparse. Second, we assume that they are independent, which is a simple approximation although it is not terribly precise. Thus, $\log p(\mathbf{s})$ is similar to what was used in ordinary ICA estimation and linear sparse coding. It can be expressed as

$$\log p(\mathbf{s}) = \sum_{i=1}^{m} G(s_i) \tag{11.6}$$

where the function G is the same kind of function we used in ICA estimation.

To compute $p(\mathbf{x}|\mathbf{s})$, we will use the noisy version of the model in Equation (11.2). Assume that we know the variance of the gaussian noise, and denote it by by σ^2. Then, the conditional probability of $\mathbf{x}$ given all the s_i is the gaussian pdf of $\boldsymbol{\nu} = \sum_{i=1}^{m} \mathbf{a}_i s_i - \mathbf{x}$. By definition of the gaussian pdf, the pdf of a single noise variable is thus

$$p(\nu_i) = \frac{1}{\sqrt{2\pi}} \exp(-\frac{1}{2\sigma^2}\nu_i^2) \tag{11.7}$$

So, the conditional log-pdf for one variable (e.g. pixel) is

$$\log p(x_i|\mathbf{s}) = -\frac{1}{2\sigma^2}\nu_i^2 - \frac{1}{2}\log 2\pi = -\frac{1}{2\sigma^2}[x_i - \sum_{j=1}^{m} \mathbf{a}_{ij} s_j]^2 - \frac{1}{2}\log 2\pi \tag{11.8}$$

We assume that the noise is independent in all variables, so the conditional pdf of the whole data vector $\mathbf{x}$ is the sum of these log-pdf's:

$$\log p(\mathbf{x}|\mathbf{s}) = -\frac{1}{2\sigma^2} \sum_{x,y} \|\mathbf{x} - \sum_{i=1}^{m} \mathbf{a}_i s_i\|^2 - \frac{n}{2}\log 2\pi \tag{11.9}$$

The constant $\frac{1}{2}\log 2\pi$ can be omitted for simplicity.

[1] Another solution would be to use the Moore-Penrose pseudo-inverse. However, that method is less justified by statistical principles, and less useful in practice.

Putting all this together: To find the most probable $s_1, \dots, s_m$ that generated the data, we maximize

$$\log p(\mathbf{s}|\mathbf{x}) = \log p(\mathbf{x}|\mathbf{s}) + \log p(\mathbf{s}) + \text{const.} = -\frac{1}{2\sigma^2} \sum_{x,y} \|\mathbf{x} - \sum_{i=1}^{m} \mathbf{a}_i s_i\|^2 + \sum_{i=1}^{m} G(s_i) + \text{const.} \tag{11.10}$$

where the "const" means terms which do not depend on $\mathbf{s}$. Maximization of this objective function is usually not possible in closed form, and numerical optimization methods have to be used. We have here assumed that the $\mathbf{a}_i$ are known; their estimation will be considered below.

Maximization of such an objective function leads to a *nonlinear* computation of the components s_i. This is in stark contrast to ordinary (non-overcomplete) models, in which the s_i are a linear function of the $\mathbf{x}$.

11.4 Estimation of the basis

Estimation of the basis vectors $\mathbf{a}_i$ can be performed using the same principle as estimation of the s_i. Basically, the solution is hidden in Equation (11.10). First, note that the pdf in Equation (11.9) depends on the $\mathbf{a}_i$ as well. So, that equation actually describes $p(\mathbf{x}|\mathbf{s}, \mathbf{a}_1, \dots, \mathbf{a}_m)$ instead of just $p(\mathbf{x}|\mathbf{s})$. Further, if we backtrack in the logic that lead us to Equation (11.10), we see that the conditional probability in Equation (11.10), when considered as a function of both $\mathbf{s}$ and the $\mathbf{a}_i$, is equal to $p(\mathbf{s}, \mathbf{a}_1, \dots, \mathbf{a}_m|I)$, if we assume a flat (constant) prior for the $\mathbf{a}_i$. This is the conditional probability of *both* $\mathbf{s}$ and the $\mathbf{a}_i$, given the data $\mathbf{x}$. Thus, the conditional log-pdf can be interpreted as essentially the likelihood of the $\mathbf{a}_i$.

Estimation of the $\mathbf{a}_i$ can now be performed by maximizing the conditional pdf in Equation (11.10) for a *sample* of images $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_T$. (Obviously, we cannot estimate a basis from a single image.) As usual, we assume that the images in the sample have been collected independently from each other, in which case the log-pdf for the sample is simply the sum of the log-pdf. So, we obtain the final objective function

$$\sum_{t=1}^{T} \log p(\mathbf{s}(t), \mathbf{a}_1, \dots, \mathbf{a}_m|\mathbf{x}_t) = -\frac{1}{2\sigma^2} \sum_{t=1}^{T} \sum_{x,y} \|\mathbf{x}_t - \sum_{i=1}^{m} \mathbf{a}_i s_i(t)\|^2 + \sum_{t=1}^{T} \sum_{i=1}^{m} G(s_i(t)) + \text{const.} \tag{11.11}$$

When we maximize this objective function with respect to all the basis vectors $\mathbf{a}_i$ and cell outputs $s_i(t)$ (the latter are different for each image), we obtain, at the same time, the estimates of the components and the basis vectors.[2] In other words, we compute both the nonlinear component values and the columns $\mathbf{a}_i$.

11.5 Approach using energy-based models

An alternative approach for estimating an overcomplete representation is the following: We give up a generative model and concentrate on generalizing the sparseness criteria. Basically, we take the log-likelihood of the basic ICA model, and relax the constraint that there cannot be too many components in the sense of linear dotproducts we compute. This approach is computationally more efficient because we do not need to compute the nonlinear estimates of the components s_i which requires another optimization.

Consider the log-likelihood of the basic ICA model, which we reproduce here with slightly different notation

$$\log L(\mathbf{v}_1, \dots, \mathbf{v}_n; \mathbf{z}_1, \dots, \mathbf{z}_T) = T \log |\det(\mathbf{V})| + \sum_{i=1}^{m} \sum_{t=1}^{T} G_i(\mathbf{v}_i^T \mathbf{z}_t) \tag{11.12}$$

where $\mathbf{z}_t$ is the whitened preprocessed data sample, and the $\mathbf{v}_i$ are the parameter vectors in the preprocessed space.

Now, could we just use the formula in Equation (11.12) with more components than dimensions? Let us denote the dimension of the data by n. Then, this means that we just take $m > n$ to achieve an overcomplete representation.

Unfortunately, this is not possible. The problem is the term $\log |\det(\mathbf{V})|$. The simple reason is that if $m > n$, the matrix $\mathbf{V}$, which collects the $\mathbf{v}_i$ as its rows, would not be square, and the determinant is only defined for a square matrix.

On the other hand, the second term on the right-hand side in Equation (11.12) is just a sum of measures of sparseness of the components, so this term need not be changed if we want to have an overcomplete representation.

So, we have to understand the real meaning of the term $\log |\det(\mathbf{V})|$ to obtain a model with an overcomplete representation. This term is actually the logarithm of what is called the *normalization constant* or a *partition function*. It is a function of the model parameters which makes the pdf of the data fulfill the fundamental constraint that the integral of the pdf is equal to

[2] One technical problem with this procedure is that the scales of the independent components are not fixed, which leads to serious problems. This problem can be solved simply by normalizing the variances of the independent components to be equal to unity at every optimization step. Alternatively, one can normalize the basis vector $\mathbf{a}_i$ to unit norm at every step.

one—a constraint that every pdf must fulfill. A likelihood is nothing else than a pdf interpreted as a function of the parameters, and computed for the whole sample instead of one observation. So, the likelihood must fulfill this constraint as well.

The normalization constant is, in theory, obtained in a straightforward manner. Let us define the pdf (for one observation) by replacing the first term in Equation (11.12) by the proper normalization constant, which we denote by Z:

$$\log L(\mathbf{z}; \mathbf{v}_1, \dots, \mathbf{v}_n) = -\log Z(\mathbf{V}) + \sum_{i=1}^{n} G_i(\mathbf{v}_i^T \mathbf{z}) \tag{11.13}$$

Normalization of the pdf means that we should have

$$\int L(\mathbf{z}; \mathbf{v}_1, \dots, \mathbf{v}_n) d\mathbf{z} = 1 \tag{11.14}$$

In the present case, this means

$$\int L(\mathbf{z}; \mathbf{v}_1, \dots, \mathbf{v}_n) d\mathbf{z} = \frac{1}{Z(\mathbf{V})} \int \prod_{i=1}^{n} \exp(G_i(\mathbf{v}_i^T \mathbf{z})) d\mathbf{z} = 1 \tag{11.15}$$

So, in principle, we just need to take

$$Z(\mathbf{V}) = \int \prod_{i=1}^{n} \exp(G_i(\mathbf{v}_i^T \mathbf{z})) d\mathbf{z} \tag{11.16}$$

because this makes the integral in Equation (11.15) equal to one.

However, in practice, evaluation of the integral in Equation (11.16) is extremely difficult even with the best numerical integration methods. So, the real problem when we take more vectors $\mathbf{v}_i$ than there are dimensions in the data, is the computation of the normalization constant.

Estimation of the model by maximization of likelihood requires that we know Z. If we omit Z and maximize only the first term in Equation (11.13), the estimation goes completely wrong: If the G_i have a single peak at zero (like the negative log cosh function), as we have assumed in earlier chapters, the maximum of such a truncated likelihood is obtained when the $\mathbf{v}_i$ are all zero, which is quite absurd!

So, the model becomes much more complicated to estimate since we don't know how to normalize the pdf as a function of the vectors $\mathbf{v}_i$. This is in stark contrast to the basic case where the number of components equals the number of input variables: the function Z is simply obtained from the determinant of the matrix collecting all the vectors $\mathbf{v}_i$.

Fortunately, there are methods for estimating models in the case where Z cannot be easily computed. First of all, there is a number of methods for computing Z approximately, so that the maximum likelihood estimation is computationally possible. However, in our case, it is probably more useful to look at methods which estimate the model directly, avoiding the computation of the normalization constant. Score matching and contrastive divergence are two methods for estimating such "non-normalized" models. We will not go into details on those methods here, but just mention that such methods exist.

11.6 Overcomplete basis from natural images

We estimated an overcomplete representation from natural images using the method in Section 11.5. Thus, we defined the model using the non-normalized log-likelihood in Equation 11.13. We basically used the classic (negative) log cosh function as G, but we allowed a bit more flexibility by allowing rescaling of the G_i by defining $G_i(u) = -\alpha_i \log\cosh(u)$, where α_i are parameters that are estimated at the same time as the $\mathbf{v}_i$. We also constrained the norms of the $\mathbf{v}_i$ to be equal to one. We used the score matching approach to estimate the parameters without computation of the normalization constant.

To reduce the computational load, we took patches of 16×16 pixels. We preprocessed the data just like with ICA, but the dimension reduction was less strong: we retained 128 principal components, i.e. one half of the dimensions. Then, we estimated a representation with 512 receptive fields. The representation is thus 4 times overcomplete when compared to the PCA dimension, and two times overcomplete when compared with the number of pixels.

The resulting receptive fields are shown in Figure 11.1. To save space, only a random sample of 192 receptive fields is shown. The receptive fields are quite similar to those estimated by basic ICA or sparse coding. Some are more oscillatory, though.

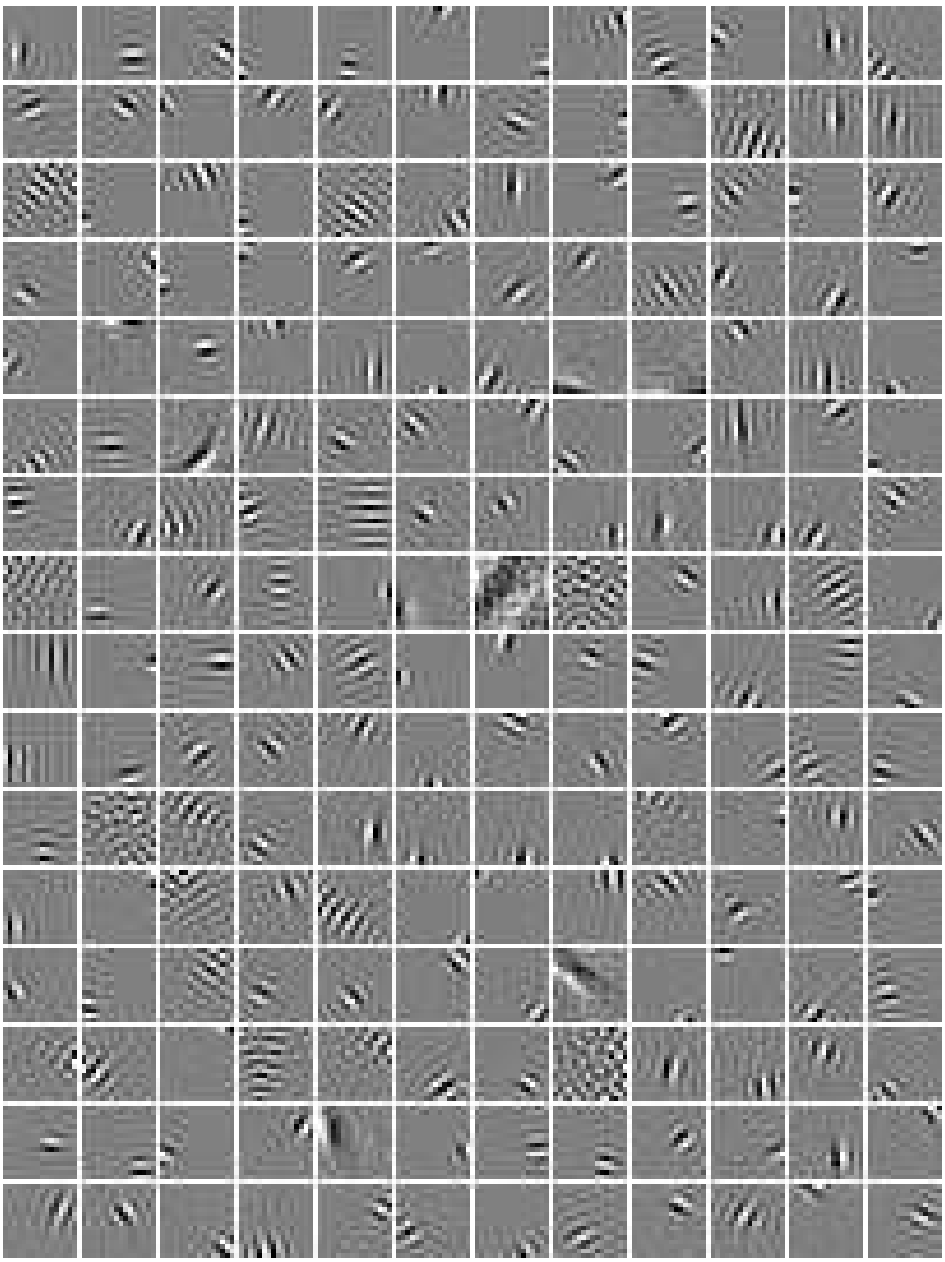

Figure 11.1: Image features ("receptive fields") in a four times overcomplete basis for canonically preprocessed data, estimated using the model in Equation (11.13) and score matching estimation. Only a random sample of the features is shown to save space.

Chapter 12

Clustering

12.1 Definition and motivating applications

The idea in clustering is to divide the data points into a number of groups, so that points in the same group are "similar". Similarity usually means that they are close to each other in the data space.

For a basic example, see Fig. 12.1. Here, the data points marked with x's and those with o's clearly form two distinct groups. With clustering methods, we want to find such structure automatically, and in high-dimensional spaces where we cannot just plot the data and have a look.

Clustering is also useful even if the data does not easily divide into clusters. Assume you want to represent your continuous-valued data by integer numbers, for example, to store them in binary format. One approach is to divide it into "clusters", and represent each data point by the index of the cluster is "belongs" to. In this case, the idea is often called "quantization", but the methods for finding the clustering are usually exactly the same. Some kind of quantization is used by any image compression algorithm[1]. But if you have some new kind of data in an n-dimensional space, and existing compression algorithms don't seem to be suitable, learning a quantization with the following method is one possibility.

12.2 K-means algorithm

12.2.1 The algorithm

A classic algorithm for clustering is k-means. It goes as follows:

1. Choose the number of clusters to model, k.
2. Choose (usually randomly) the initial values of the cluster centerpoints, $\mathbf{w}_i, i = 1, \ldots, k$.
3. Repeat until convergence
   (a) For each data point $\mathbf{x}(t), t = 1, \ldots, T$, find the closest centerpoint,
   $$\min_i \|\mathbf{w}_i - \mathbf{x}(t)\|^2 \tag{12.1}$$
   Denote the index of the closest centerpoint for $\mathbf{x}(t)$ by $I(t)$.
   (b) Replace each centerpoint $\mathbf{w}_i$ by the *mean* of the data points for which the (old) $\mathbf{w}_i$ was closest, i.e.
   $$\mathbf{w}_i \leftarrow \frac{\sum_{t:I(t)=i} \mathbf{x}(t)}{\text{number of data points in sum above}} \tag{12.2}$$

The algorithm is very intuitive and often very fast. It is illustrated in Figs. 12.2-12.4.

[1] Although the clusters in such algorithms are often fixed instead of being learned, and the quantization is usually applied on components vaguely related to PCA and ICA, often separately compressing each component

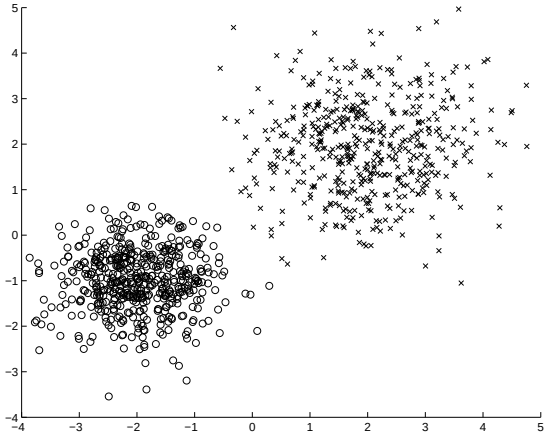

Figure 12.1: Illustration of clustering. The data points marked with x's and o's form two distinct groups, i.e. clusters.

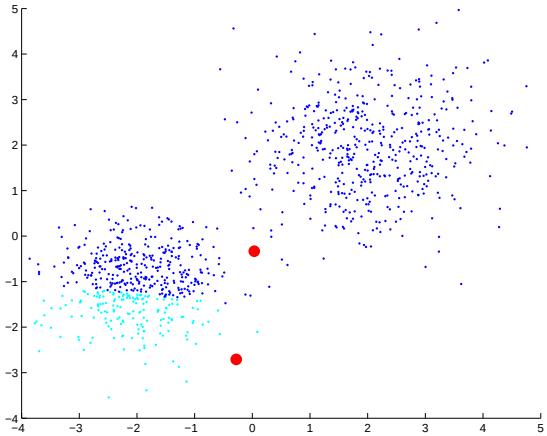

Figure 12.2: Illustration of k-means 1. The center points (red balls) are initialized at random initial points. The closest centerpoints are shown by colouring of the data points.

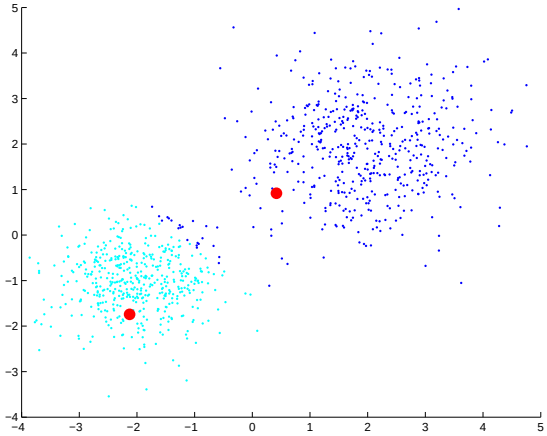

Figure 12.3: Illustration of k-means 2. The centerpoints after one iteration.

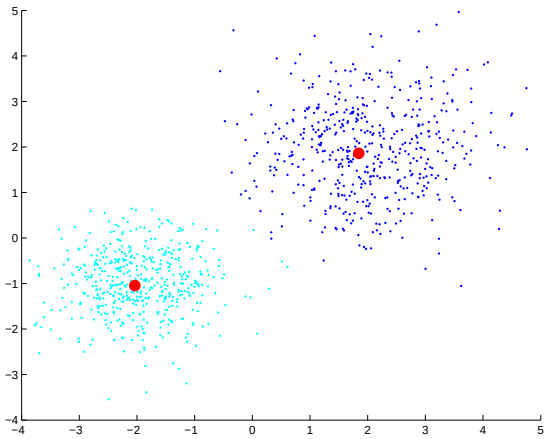

Figure 12.4: Illustration of k-means 3. After two iterations, the algorithm has almost converged.

12.2.2 Definition of objective function for k-means

To understand the k-means algorithm, we show next that it is minimizing a reconstruction error, also called quantization error. Imagine that we replace each data point by some cluster centerpoint. The sum of squared errors incurred in this approximation is the quantization error. Such an operation is actually widely used in practice when you want to represent continuous-valued data in binary format.

Let us define the objective function as

$$J(\mathbf{w}_1, \ldots, \mathbf{w}_k, i(1), \ldots, i(T)) = \sum_{t=1}^{T} \|\mathbf{x}(t) - \mathbf{w}_{i(t)}\|^2 \tag{12.3}$$

where $i(t)$ is the cluster centerpoint which replaces the data point $\mathbf{x}(t)$. Note that we introduced here the $i(t)$ as additional arguments of the objective function: their values are free and not defined in any way at this stage.

Now, we prove that J is decreased at every step of the k-means algorithm, unless the algorithm has converged (or there are points which could be equally well clustered in two different points, which is a rare coincidence). In fact, the algorithm proceeds by minimizing J alternatively w.r.t. the $\mathbf{w}$ and the $i(t)$.

Consider the step in k-means where the $I(t)$ are computed. It is rather obvious that given the $\mathbf{w}_i$, the best assignment of the data points to the clusters is to take the cluster whose centerpoint is closest, as in the algorithm. Thus, the definition of $I(t)$ is the one that minimizes J *given* the $\mathbf{w}_i$.

Next, we need to show that the update of the $\mathbf{w}_i$ decreases (or does not change) J as well. Again, in fact, it is the optimal update which finds $\mathbf{w}_i$ obtaining the smallest J possible for the given $I(t)$. To show this, we consider the set of points in the same cluster separately, since the objective function separates to a sum, each depending of one $\mathbf{w}_i$. For the points with $I(t) = 1$, for example, we need to minimize

$$J_1(\mathbf{w}_1) = \sum_{t:I(t)=1} \|\mathbf{x}(t) - \mathbf{w}_1\|^2 \tag{12.4}$$

where we have made the decomposition

$$J = \sum_j J_j(\mathbf{w}_j) = \sum_j \sum_{t:I(t)=j} \|\mathbf{x}(t) - \mathbf{w}_j\|^2 \tag{12.5}$$

We compute the gradient and obtain

$$\nabla_{\mathbf{w}_1} J_1(\mathbf{w}_1) = 2 \sum_{t:I(t)=1} \mathbf{x}(t) - \mathbf{w}_1 = 0 \tag{12.6}$$

$$\Leftrightarrow \tag{12.7}$$

$$\{\text{Number of points for which } I(t) = 1\} \times \mathbf{w}_1 = \sum_{t:I(t)=1} \mathbf{x}(t) \tag{12.8}$$

which is zero at the point where $\mathbf{w}_i$ is the average of the $\mathbf{x}(t)$ with $I(t) = 1$, which is the second update step.

Thus, we see that in fact, that the algorithm first chooses the optimal $i(t)$, given the $\mathbf{w}_i$, and then the optimal $\mathbf{w}_i$, given the $i(t)$. If the algorithm is not in a local minimum, one of these steps must decrease the objective function.[2]

One utility of such analysis in terms of an objective function is that it indicates that the algorithm is likely to converge to some point. It cannot, for example, oscillate between two solutions with different values of the objective function because then the objective function would also oscillate up and down, and we just proved that this is not possible. (If the solutions have the same value of the objective function, they can be considered equally good.)

Some things are worth mentioning:

- The convergence is local, and the global minimum is not achieved in general. This is similar to any gradient algorithm.
- Determining the correct number of clusters, k, is a very complicated issue. A practical approach would be to try out different values, and determine for example the one which gives acceptable reconstruction error in the context of vector quantization, or an intuitive interpretation in the case of exploratory data analysis.

12.3 Gaussian mixture model

The gaussian mixture model is a probabilistic formulation of the clustering problem. It also generalizes the k-means algorithm.

Assume we have observed a number of vectors $\mathbf{x}(1), \ldots, \mathbf{x}(T)$ as usual, and denote by n the dimension of the data space. We observe data which comes from a number of clusters. Denote by k the number of clusters. The probabilities that the data come from each cluster are denoted by $\pi_c, c = 1, \ldots k$. Obviously, we must have $\sum_c \pi_c = 1$. Inside each cluster, the data has a gaussian distribution with mean $\boldsymbol{\mu}_c$ and covariance $\mathbf{C}_c$. Thus, we have a generative model; the data generating mechanism is as follows:

1. We choose randomly the cluster index, so that the probability of choosing cluster c is π_c. Denote the result by r.
2. Generate a data point by drawing a point from a multivariate gaussian distribution with mean $\boldsymbol{\mu}_r$ and covariance $\mathbf{C}_r$

Here, r is a latent variable. For each observed data point, we have one realization of the latent variable.

This is a relatively rich clustering model. In addition to the means (centerpoints) of the clusters $\boldsymbol{\mu}_c$, the model allows each cluster to have its own covariance and prior probability π_c. In particular:

1. Some clusters can have larger probability than others.
2. Some clusters can have larger variance than others: so they are "bigger" in terms of the area in data space which they cover.
3. The covariance of a cluster need not be diagonal, so a cluster might be elongated in some direction, which is quite different from the underlying assumptions in the k-means algorithm.

This flexibility is illustrated in Fig. 12.5. The model definition also means that the number of parameters is quite large, so the model can be difficult to estimate if we don't have many data points.

The model definition also implies that the most likely cluster for a point need not be the one with the closest centerpoint. This will be seen in detail in Equation (12.14) below.

12.4 Likelihood of gaussian mixture model

To begin with, we can formulate the likelihood of the model by assuming we observe the latent variables (this is called the "complete likelihood"). For a single data point, we have the joint distribution

$$p(\mathbf{x}, r|\boldsymbol{\mu}_c, \mathbf{C}_c, \pi_c, c = 1 \ldots k) = \frac{1}{(2\pi)^{n/2}(\det \mathbf{C}_r)^{\frac{1}{2}}} \exp(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_r)^T \mathbf{C}_r^{-1}(\mathbf{x} - \boldsymbol{\mu}_r))\pi_r \tag{12.9}$$

Note that here we have a joint "pdf" of a discrete and a continuous-valued variable. Intuitively, this should not be difficult to understand, and a rigorous treatment is possible by noting that for any given r, this is a pdf for $\mathbf{x}$. This gives the likelihood for the whole observed sample:

$$p(\mathbf{x}(t), r(t), t = 1 \ldots T|\boldsymbol{\mu}_c, \mathbf{C}_c, \pi_c, c = 1 \ldots k) = \prod_t \frac{1}{(2\pi)^{n/2}(\det \mathbf{C}_{r(t)})^{\frac{1}{2}}} \exp(-\frac{1}{2}(\mathbf{x}(t) - \boldsymbol{\mu}_{r(t)})^T \mathbf{C}_{r(t)}^{-1}(\mathbf{x}(t) - \boldsymbol{\mu}_{r(t)}))\pi_{r(t)} \tag{12.10}$$

[2] It is perhaps not completely clear what "local minimum" exactly means when we have discrete-valued variables, but maybe the concept is intuitively clear.

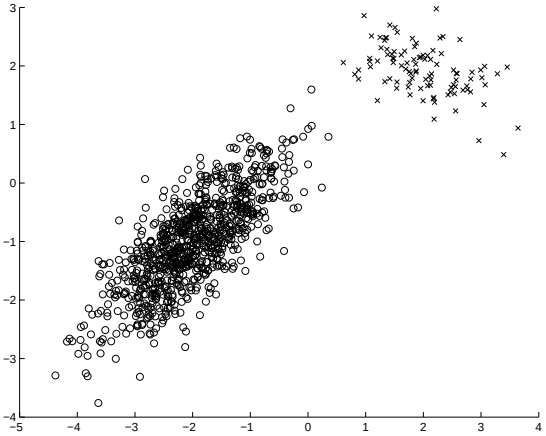

Figure 12.5: Illustration of a case where the flexibility of the Gaussian mixture model is needed. These clusters have different probabilities, different numbers of data points, different general "spreads" and different covariance structures.

This is a classic example where we would like to "integrate out" the latent variables. This means that since we don't observe the r, we would like to obtain a likelihood which is not a function of the r. It can be obtained by computing the marginal density of the $\mathbf{x}$ according to the general formula of marginal probability

$$p(\mathbf{x}) = \int p(\mathbf{x}, r) dr \tag{12.11}$$

In this case, since the latent variables are discrete-valued, the integration means in fact a simple summation. So, the likelihood of the sample is

$$\begin{aligned}\prod_t p(\mathbf{x}(t)|\boldsymbol{\mu}_c, \mathbf{C}_c, \pi_c, c = 1 \dots k) &= \prod_t \sum_{r=1}^{k} p(\mathbf{x}(t), r|\boldsymbol{\mu}_c, \mathbf{C}_c, c = 1 \dots k) \\ &= \prod_t \sum_{r=1}^{k} \frac{1}{(2\pi)^{n/2}(\det \mathbf{C}_r)^{\frac{1}{2}}} \exp(-\frac{1}{2}(\mathbf{x}(t) - \boldsymbol{\mu}_r)^T \mathbf{C}_r^{-1}(\mathbf{x}(t) - \boldsymbol{\mu}_r))\pi_r \end{aligned} \tag{12.12}$$

So, we can write the log-likelihood for the whole sample as

$$\log p(\mathbf{x}(1), \dots, \mathbf{x}(T)|\boldsymbol{\mu}_c, \mathbf{C}_c, \pi_c, c = 1 \dots k) = \sum_{t=1}^{T} \log \sum_{r=1}^{k} \frac{1}{(2\pi)^{n/2}(\det \mathbf{C})^{\frac{1}{2}}} \exp(-\frac{1}{2}(\mathbf{x}(t) - \boldsymbol{\mu}_r)^T \mathbf{C}_r^{-1}(\mathbf{x}(t) - \boldsymbol{\mu}_r))\pi_r \tag{12.13}$$

Basically, we could thus approach the estimation of the model in three ways:

1. We maximize the log-likelihood in (12.13) by a general-purpose optimization method such as the gradient method. This poses no particular problems, except for the one we always encounter with the gradient method: choice of step size.
2. In fact, it is not completely necessary to integrate out the latent variables as we did above. We could also maximize the joint likelihood in Eq. (12.10) with respect to both the parameters and the latent variables $r(t)$. For this, we could use an alternating variables approach, see Section 3.8. This is not very difficult, and would lead to an algorithm which is rather similar to k-means, but we don't go into details.
3. A third approach, which is by far the most widely-used in the case of the Gaussian mixture model, is the Expectation-Maximization (EM) algorithm, to be discussed next.

12.5 EM algorithm: a heuristic approach

The EM algorithm is basically a more sophisticated version of the alternating variables method in Section 3.8. In fact, it can be shown to (locally) maximize the likelihood in (12.13) instead of the complete likelihood (but the proof is quite complicated), and the likelihood can be considered to be a more principled objective.

The basic idea is that instead of computing just the closest centerpoint like in k-means, we compute the posterior probabilities that the data point belongs to each cluster, given the observed data. They actually come quite easily from (12.9)

$$p(r(t) = c|\mathbf{x}(t), \boldsymbol{\mu}_c, \mathbf{C}_c, \pi_c, c = 1 \dots k) = \frac{1}{(2\pi)^{n/2}(\det \mathbf{C}_c)^{\frac{1}{2}}} \exp(-\frac{1}{2}(\mathbf{x}(t) - \boldsymbol{\mu}_c)^T \mathbf{C}_c^{-1}(\mathbf{x}(t) - \boldsymbol{\mu}_c))\pi_c / p(\mathbf{x}(t)) \tag{12.14}$$

where we also see the fact mentioned earlier: this posterior probability is not just a function of the distance between the data points and the centerpoints, but takes into account the covariance structure and the prior probabilities of the clusters.

Here, the division by $p(\mathbf{x}(t))$ is needed to normalize the probabilities. However, it is a bit difficult to compute. Fortunately, we can normalize these probabilities quite easily after we have computed them. Let us compute

$$q_{t,c} = (\det \mathbf{C}_c)^{-\frac{1}{2}} \exp(-\frac{1}{2}(\mathbf{x}(t) - \boldsymbol{\mu}_c)^T \mathbf{C}_c^{-1}(\mathbf{x}(t) - \boldsymbol{\mu}_c))\pi_c \tag{12.15}$$

for all t and c. Here, we have omitted some constants because they will disappear in the normalization anyway. Now, we normalize these quantities by

$$q^*_{t,c} = \frac{q_{t,c}}{\sum_c q_{t,c}} \tag{12.16}$$

These q^* are now properly normalized and give the posterior probabilities in (12.14).

The intuitive idea in EM is that we can use these $q^*_{t,c}$ as "soft" cluster memberships to estimate the Gaussian models for each cluster. This means that given the q^*, we estimate

$$\hat{\boldsymbol{\mu}}_c = \frac{1}{\sum_t q^*_{t,c}} \sum_t q^*_{t,c}\mathbf{x}(t) \tag{12.17}$$

$$\hat{\mathbf{C}}_c = \frac{1}{\sum_t q^*_{t,c}} \sum_t q^*_{t,c}(\mathbf{x}(t) - \hat{\boldsymbol{\mu}}_c)(\mathbf{x}(t) - \hat{\boldsymbol{\mu}}_c)^T \tag{12.18}$$

$$\hat{\pi}_c = \frac{1}{T} \sum_t q^*_{t,c} \tag{12.19}$$

where the first update is quite similar to k-means: for each new centerpoint (here: mean of gaussian cluster), we take a weighted mean of the data points, where the weighting comes from the probability that the datapoint belongs to this cluster. If the probabilities were zero or one, this would be like k-means. The other two updates generalize k-means to the case where the clusters have different covariance structures and prior probabilities.

We have now constructed the EM algorithm, which is obtained by repeating (12.15-12.19) until convergence. (To be precise: in (12.15), the parameters are replaced by their estimates.)

12.6 EM algorithm: general theory

Why is the EM algorithm given above justified? In fact, the EM algorithm is a general principle for designing algorithms for estimation of latent variable models. For most models, the algorithm is not very useful, but for models involving gaussian variables, it may provide an interesting algorithm.

In general, consider a latent variables model $p(\mathbf{x}, \mathbf{s}, \boldsymbol{\theta})$ where $\mathbf{x}$ is the observed variable vector, $\mathbf{s}$ is the vector of latent variables, and $\boldsymbol{\theta}$ is the vector of parameters. For simplicity of exposition, denote by $\mathbf{X}$ the matrix which contains all the observations of $\mathbf{x}$, and likewise for $\mathbf{S}$. The general form of the i-th step of the EM algorithm is:

Expectation step Consider the posterior distribution of the latent variables: $p(\mathbf{S}|\mathbf{X}, \boldsymbol{\theta}_{i-1})$ where we have used the estimate of $\boldsymbol{\theta}$ from the previous step. Compute the expectation of the log-posterior of $\boldsymbol{\theta}$ with respect to this distribution:

$$\int p(\mathbf{S}|\mathbf{X}, \boldsymbol{\theta}_{i-1}) \log p(\mathbf{X}, \mathbf{S}, \boldsymbol{\theta}) d\mathbf{S} \tag{12.20}$$

Maximization step Maximize (12.20) with respect to $\boldsymbol{\theta}$. Take this as the new estimate $\boldsymbol{\theta}_i$.

The expectation and maximization steps are iterated until convergence.

If the posterior probabilities of the latent variables were all zero or one (assume for simplicity that they are discrete-valued), the expectation step would simply mean computing the complete likelihood with the latent variables fixed to their most likely values. Thus, we would be maximizing the complete likelihood by alternating between the maximization with respect to the latent variables and the maximization with respect to the parameters. The idea in EM is that it is better to consider the whole posterior distribution of the latent variables, and integrate over it, instead of just using the most likely values of the latent variables.

12.7 Deriving the gaussian mixture case of EM

Now we derive the EM algorithm for Gaussian mixtures based on this general theory. In the gaussian mixture model case, the latent variables $\mathbf{s}$ are r, and the parameter vector $\boldsymbol{\theta}$ consists of the $\boldsymbol{\mu}_c, \mathbf{C}_c$.

Computing the $q^*_{t,c}$ amounts to computing the posterior of the latent variables $r(t)$ given the parameters in the previous step. In fact, the r are independent for different t, so its posterior can really be represented, for each t, as the set of probabilities $q^*_{t,c}$. Above, these q^* were indeed derived as the posterior probabilities of $p(r|\mathbf{x}, \boldsymbol{\theta})$. So, the integral in (12.20) can be computed as the sum over the different values of r, with these weights. Using the complete likelihood in (12.10), the integral in (12.20) becomes in this case

$$\sum_{t,c} \left[-\frac{1}{2}(\mathbf{x}(t) - \boldsymbol{\mu}_c)^T \mathbf{C}_c^{-1}(\mathbf{x}(t) - \boldsymbol{\mu}_c) + \log \pi_c - \frac{1}{2} \log \det \mathbf{C} - \log(2\pi)^{n/2} \right] q^*_{t,c} \tag{12.21}$$

To maximize this, for example, with respect to $\boldsymbol{\mu}_c$, we compute the gradient and set it to zero:

$$\sum_t -\mathbf{C}_c^{-1}(\mathbf{x}(t) - \boldsymbol{\mu}_c) q^*_{t,c} = 0 \tag{12.22}$$

which gives as the solution the EM update already given in (12.17). Likewise, maximization with respect to $\mathbf{C}_c$ gives (12.18); the calculations are essentially the same as in finding the maximum likelihood estimator for the covariance matrix of the multivariate Gaussian distribution. For π_c, it is more tricky. The derivative of (12.21) with respect to π_c is given by

$$\sum_t \frac{1}{\pi_c} q^*_{t,c} = \frac{1}{\pi_c} \sum_t q^*_{t,c} \tag{12.23}$$

However, we are here maximizing under the constraint $\sum_c \pi_c = 1$, so we cannot just set this gradient to zero. In the optimization section, we saw that at the optimum, the gradient should point at a direction which is orthogonal to the constraint surface (this can be rigorously shown using Lagrangian multipliers). Here, the vector of all ones is orthogonal to the constraint surface. So, we should have

$$\frac{1}{\pi_c} \sum_t q^*_{t,c} = \lambda \tag{12.24}$$

for all c. This obviously requires

$$\pi_c = \frac{\sum_t q^*_{t,c}}{\lambda} \tag{12.25}$$

The value of λ has to be chosen so that the constraint $\sum_c \pi_c = 1$ is fulfilled: it is easy to verify that $\lambda = T$ fulfills the constraint because $\sum_c q^*_{t,c} = 1$. Thus, we have proven that (12.19) is the correct EM update.

Chapter 13

Nonlinear projection methods

Here we consider methods which try to find some kind of principal components which can have an arbitrary nonlinear shape, thus generalizing the theory of (linear) PCA we had in the beginning of this course.

13.1 Metric (linear) multi-dimensional scaling

13.1.1 Basic idea

The basis for many nonlinear methods is multi-dimensional scaling (MDS). This is basically a variant of PCA where the main difference is that we work on a matrix of distances instead of a covariance matrix or the original data points. That is, the data we input to the method is a matrix of distances of the data points:

$$d_{ij} = \text{distance between data point } i \text{ and } j \tag{13.1}$$

where we use the indices i and j for the observations, instead of t which was used in previous chapters. Now, the goal is to represent this matrix as a low-rank approximation, i.e. using only a few "components".

Often, the distances are normalized so that the column and row sums are zero. This can be accomplished by

$$\tilde{d}_{ij} = d_{ij} - \frac{1}{N}\sum_i d_{ij} - \frac{1}{N}\sum_j d_{ij} + \frac{1}{N^2}\sum_{ij} d_{ij} \tag{13.2}$$

where N denotes the number of data points, or in general objects whose distances are given (previously, T was the number of data points). It is simple to prove that the row and column sums of $\tilde{d}$ are zero.

A simple approach would now be to compute the eigenvalue decomposition of the matrix of the $\tilde{d}$. Since the eigen-value decomposition allows us to compute the best low-rank approximation to any symmetric matrix (it doesn't have to be a covariance matrix), we would obtain a low-rank approximation to the distance matrix which is in some way optimal.

13.1.2 Case of Eulidean distances

If the distances given are the ordinary (squared) Euclidean distances in an n-dimensional space, and the data variables has zero means, it can be shown that in fact, the solution is essentially the same as in PCA. Denote each data point by $\mathbf{x}_i$ (unlike in previous chapters!). Then each element is given by

$$d_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\|^2 = \|\mathbf{x}_i\|^2 + \|\mathbf{x}_j\|^2 - 2\mathbf{x}_i^T\mathbf{x}_j \tag{13.3}$$

and, in fact, the normalized matrix is

$$\tilde{d}_{ij} = -2\mathbf{x}_i^T\mathbf{x}_j \tag{13.4}$$

which is left as an exercise. Thus, if we normalize these distances so that each row and column sum is zero, the obtain nothing else than the matrix $\mathbf{X}^T\mathbf{X}$ multiplied by some constant. Note, however, that this is *not* proportional to the covariance matrix, which is $\frac{1}{N}\mathbf{X}\mathbf{X}^T$ (in our notation, where each row is one variable and each column an observation).

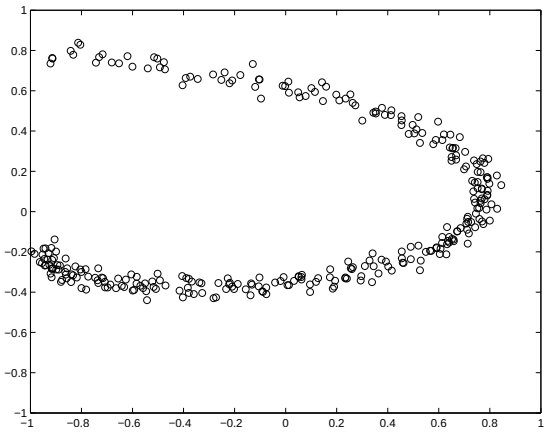

Figure 13.1: A data cloud whose dependencies cannot be represented using a linear PCA. However, a nonlinear version might be able to reproduce the shape of the data set quite exactly using a single nonlinear principal component which follows the U-shape.

Nevertheless, the eigenvalue decomposition of this matrix gives essentially the same as the EVD of the covariance matrix $\frac{1}{N}\mathbf{X}\mathbf{X}^T$. In fact, from the definition of eigenvectors of $\frac{1}{N}\mathbf{X}\mathbf{X}^T$, which give the principal components, we obtain

$$\frac{1}{N}\mathbf{X}\mathbf{X}^T\mathbf{w} = \lambda\mathbf{w} \tag{13.5}$$

$$\Rightarrow$$

$$(\frac{1}{N}\mathbf{X}^T\mathbf{X})(\mathbf{X}^T\mathbf{w}) = \lambda(\mathbf{X}^T\mathbf{w}) \tag{13.6}$$

simply by multilying the equation by $\mathbf{X}^T$ from the left. Thus, each eigenvalue of the covariance matrix is also an eigenvalue of $\frac{1}{N}\mathbf{X}^T\mathbf{X}$. The corresponding eigenvector is $\mathbf{X}^T\mathbf{w}$ (or some rescaled version) which is actually the values of the principal component $\mathbf{w}^T\mathbf{x}$ collected into a vector. Note that if we normalize the norm of $\mathbf{X}^T\mathbf{w}$ to one, as is typical in eigenvalue computations, we are normalizing the principal components so that their variances are equal to one, which is different from when we did in the PCA chapter.

Since $\frac{1}{N}\mathbf{X}^T\mathbf{X}$ has a dimension of $N \times N$, it has N eigenvalues. However, it has only rank n (the dimension of the data space). Thus, $N-n$ eigenvalues are zero. This means that in fact, the non-zero eigenvalues of $\frac{1}{N}\mathbf{X}^T\mathbf{X}$ are exactly those of the covariance matrix, and the eigenvectors are transformed as described above. However, note the change of sign in (13.4) which means that the non-zero eigenvalues of $\tilde{d}$ are negative (since the eigenvalues of a covariance matrix are non-negative)! So, we will actually take the eigenvectors corresponding to the *smallest* eigenvalues (but largest in absolute value) of $\frac{1}{N}\mathbf{X}^T\mathbf{X}$.

How do we then use these eigenvectors? We actually let the eigenvectors of $\frac{1}{N}\mathbf{X}^T\mathbf{X}$ define the values of the components for each data point. That is, the value of the k-th component for the i-th data point is given by the i-th entry in the eigenvector corresponding to the k-th smallest eigenvalue. As we saw before, the eigenvectors are given by $\mathbf{X}^T\mathbf{w}_k$ where $\mathbf{w}_k$ is the vector defining the k-th principal component. So, the i-th entry of this eigenvector is $\mathbf{x}_i^T\mathbf{w}_k$, which is nothing else than the value of the k-th principal component for the i-th data point (when the principal components are rescaled as described above). This may be a bit confusing, but it is important to understand that in MDS, the entries in the eigenvectors directly give the values of the components for all data points. This is in contrast to basic PCA the eigenvectors give weights with which we multiply the data variables in order to get the values of the components. In the general case which we consider below, this distinction is important.

This is called the "metric" version of MDS in the statistical literature. In machine learning it would more often be called "linear" in the sense that the components now are linear functions of the data points, because they are given by just linear projections on the eigenvectors like in PCA.

13.2 Nonlinear MDS

The real interest of MDS for us is in the nonlinear case. Consider the data cloud in Figure 13.1. It contains structure which cannot be meaningfully represented using linear components. However, using a nonlinear transformation, a single component should be able to represent the data quite well.

Application of linear (metric) MDS, which is the same as PCA, on this data is shown in Fig. 13.2. We computed just one principal component and colour-coded it so that the colour of each data point shows the value of the principal component at

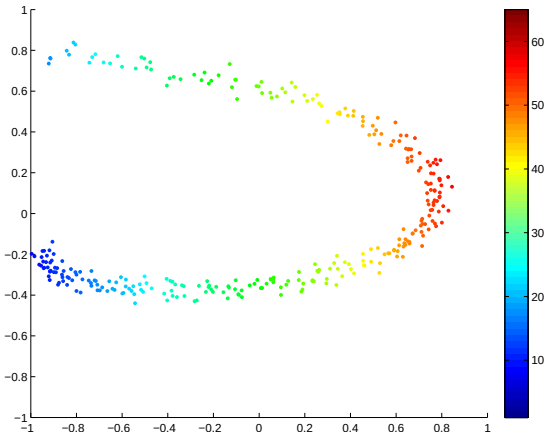

Figure 13.2: Application of metric MDS (=PCA) on the data set in Fig. 13.1. The value of the principal components is given by the colour code going from blue to red. (Please ignore the numbers next to the color bar, they are completely arbitrary.)

that data point. We can see that metric MDS is not able to learn the nonlinear structure of the data (which was obvious anyway), and it is mainly representing the left-right axis.

A simple approach to the nonlinear case would be to take some nonlinear functions of the Euclidean distances. This leads to different methods: Kernel PCA, Laplacian Eigenmaps, and IsoMap.

13.2.1 Kernel PCA

In kernel PCA, we simply take a nonlinear function of the Euclidean distances, defining

$$d_{ij} = f(\|\mathbf{x}_i - \mathbf{x}_j\|^2) \tag{13.7}$$

Alternatively we could take $d_{ij} = f(\mathbf{x}_i^T \mathbf{x}_j)$, which leads to yet another class of methods.

The nonlinear function f could be the Gaussian kernel:

$$d_{ij} = -\exp(-\frac{1}{2\sigma^2}\|\mathbf{x}_i - \mathbf{x}_j\|^2) \tag{13.8}$$

in which case we have to define the value of the parameter σ^2. We take here the minus sign to have an increasing function of the Euclidean distances (we could also add 1 to make them non-negative).

The idea here is to look at the local distances only, in the sense that large distances are less important since they are all set to (almost) a constant, zero. This is in stark contrast to PCA, in which far-away point can have a strong influence since the squared Euclidean distance grow quadratically when going away from the origin.

So, to do kernel PCA, we compute the eigenvectors corresponding to the largest eigenvalues of the matrix in (13.8). Possibly we also want to normalize the distance matrix so that the row and column sums are both zero, as proposed above.

The projections of the data points to the components, i.e. the representation of each data point, are then given by the entries in each eigenvector. That is, when we have computed an eigenvector $\mathbf{e}_k$ of the matrix of the d_{ij}, we define the nonlinear projection of the data as

$$\text{proj}_k(\mathbf{x}_i) = e_{ki} \tag{13.9}$$

where e_{ki} is the i-th entry of the eigenvector $\mathbf{e}_k$. Thus, the nonlinear function is basically defined at the location of each data point only.

Application of Kernel PCA on the data in Fig 13.1, including normalization of column and row sums, is shown in Fig. 13.3. The method does not seem to be able to find the structure with this data.

13.2.2 Laplacian eigenmaps

A very simple modification of kernel PCA often works much better. It is known under the name "Laplacian eigenmaps".[1]

[1]The way I present Laplacian eigenmaps here is rather different from how they are usually presented, but I think that the method I present here is a special case of the general framework. If you read the original paper, note that here a) the selection of which connections

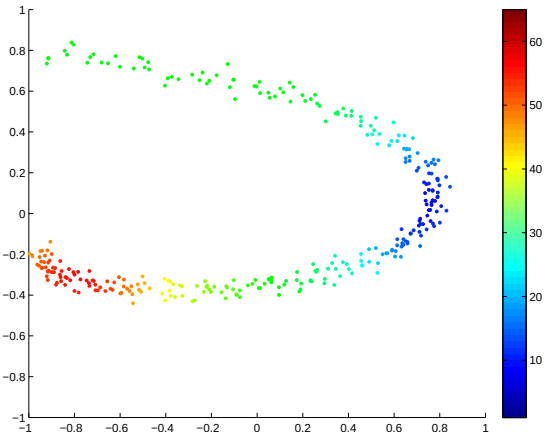

Figure 13.3: Application of kernel PCA on the data set in Fig. 13.1. the color code shows the values the nonlinear principal component for each data point. (Please ignore the numbers next to the color bar, they are completely arbitrary.) Does not seem to work very well.

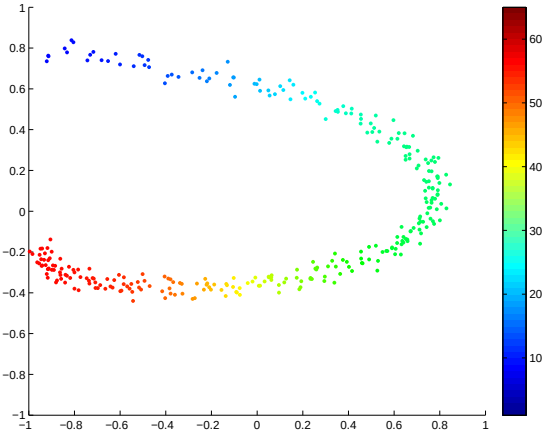

Figure 13.4: Application of the modified version of kernel PCA ("Laplacian eigenmaps") on the data set in Fig. 13.1. the color code shows the values of the nonlinear principal component for each data point. Now, the method seems to find the "true" nonlinear component of the data.

Let us normalize the rows of the distance matrix d_{ij} by dividing them by the sums of their entries $\sum_j d_{ij}$, and let us subtract 1 from each entry in the matrix. Then, we compute the eigenvalue decomposition as with kernel PCA, and it works for this data, see Figure 13.4.

13.2.3 IsoMap

In Isomap, the idea is to compute the "geodesic" distances instead of the Euclidean ones. The idea is that if you are, for example, on the surface of a ball (actually, we all are!) the "true" distance from one point to another is not the simple Euclidean distance but the distance that you have to travel of the surface of the ball. When given a set of data points, we assume they are on some nonlinear manifold which you can estimate, and then we compute the distances along this manifold. See Fig. 13.5 which explains the concept better.

The geodesic distances can be computed using shortest-path algorithms. First, you compute the Euclidean distances between all the data points. Then, you find the shortest path from $\mathbf{x}_i$ to $\mathbf{x}_j$ which is obtained so that you only are allowed to move between the data points, *and* you are not allowed to move to a data point which is further away from your current data point than some set threshold. This means you will have to move within the data "cloud". If there are many data points, it may

are considered is based on the Euclidean distances, b) the generalized eigenvalue computation is transformed to a conventional one by simple algebraic manipulations, c) the eigenvalues are shifted by subtracting an identity matrix, and d) the uninteresting eigenvalue is removed by subtracting 1 from all the entries in the matrix.

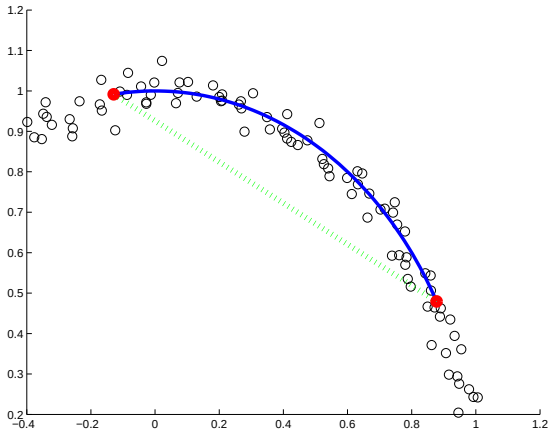

Figure 13.5: Illustration of geodesic distances. The data set defines implicitly a kind of a manifold, i.e. a nonlinear structure. The distance between the two red points is more naturally given by the length of the arc (blue) instead of the length of the line connecting them (green dashed line).

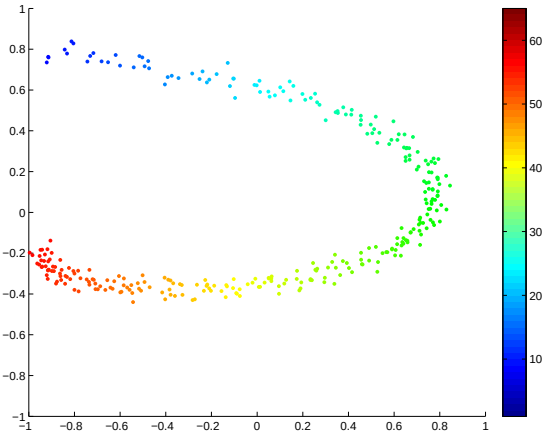

Figure 13.6: Application of Isomap on the data set in Fig. 13.1.

become too difficult to compute the geodesic distances between all of them. Then, it may be useful to choose some data points as "landmarks", and compute the distances of all data points to those landmarks only, or use similar computational shortcuts.

After computing the geodesic distances like this, we just apply MDS on these distances: we center the distances as in (13.2) and do the eigen-value decomposition of that matrix.

Application of Isomap on the data se Fig 13.1 is shown in Fig. 13.6. It finds correctly the underlying structure in this case.

13.3 Kohonen's self-organizing map

The self-organizing map (SOM) is another method for nonlinear projection and visualization of high-dimensional data. In addition it clusters the data.

The parameters in the SOM consists of a number of model vectors, which are like cluster centerpoints. The central idea is that the model vectors are organized into a meaningful two-dimensional order (grid as in Fig. 13.7). The SOM algorithm learns the model vectors so that they optimally describe the observed data in two ways: The model vectors provide a meaningful quantization of the data, and similar model vectors are closer to each other in the grid than the more dissimilar ones. The latter property is called the preservation of "topography".

SOM uses a neighborhood function h to express the topography mathematically. It is a decreasing function of the distance

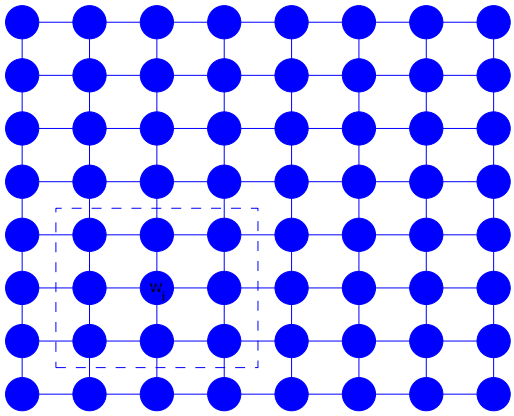

Figure 13.7: In SOM, the model vectors $\mathbf{w}_i$, which are like cluster centerpoints, are arranged on a grid, for example this rectangular two-dimensional grid. The grid defines which centerpoints are close to each other (the "neighbourhood" function), and this affects the learning so that vectors which are close to each other end up representing similar data points. If the neighbourhood function is binary, it basically says which model vectors are close to each other. The dashed square in the figure shows the set of model vectors which are close to $\mathbf{w}_i$. The lines connecting the adjacent nodes are another way of showing the grid structure.

between the i-th and j-th model vectors on the topographic grid. In the simplest case, we use a binary h by setting

$$h_{i,j} = \begin{cases} 1, \text{ if } \|\mathbf{r}_i - \mathbf{r}_j\|^2 \leq \alpha \\ 0 \text{ otherwise} \end{cases} \tag{13.10}$$

for some distance α, which could be just 1 or 2. The $\mathbf{r}_i$ and $\mathbf{r}_j$ are the locations of the model vectors on the grid (not in the data space). For example, in Fig. 13.7, the square shows the neighbours of the model vector $\mathbf{w}_i$, which are the model vectors for which the neighbourhood function is equal to one, which are simply called the "neighbours".

One variant of the SOM algorithm is then obtained as a simple modification of k-means:

1. Initialize the model vectors $\mathbf{w}_i$ (e.g. randomly).
2. For each data point, find the closest model vector $\mathbf{w}_i$.
3. For each model vector $\mathbf{w}_i$, collect the set of those data points whose closest model vector was a neighbour of $\mathbf{w}_i$ or $\mathbf{w}_i$ itself.
4. Take for each new model vector the mean over the set collected in step 3.
5. Repeat from step 2 until convergence.

The difference to k-means is in step 3, in which we collect the indices not only of those data points for which $\mathbf{w}_i$ is closest, but also of those for which one of the neighbours is the closest. This adds a smoothing effect in the sense that model vectors which are close to each other on the grid tend to move close to each other in the data space as well.

The output of the SOM is dependent on the initial value (unlike MDS). In Figs 13.8-13.9 we show the results of SOM for two different initializations. In the first one, it found the right solution, but in the second, it got stuck something like a local minimum (strictly speaking, we cannot talk about local minima since we are not minimizing a well-defined objective function, but the same general idea applies here). One way of alleviating the problem of "local minima" is to use a more intelligent initialization, for example, by PCA.

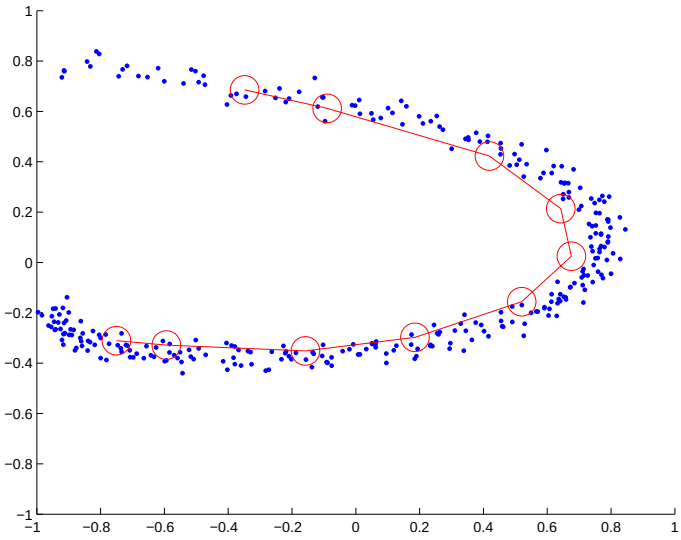

Figure 13.8: Application of SOM to the same data as above. Here, we arranged the model vectors on a one-dimensional grid, i.e. in a single "row", instead of a two-dimensional grid. The connecting lines in the plot show the order of the model vectors in the row. Starting from a good initial point, the SOM found a good solution.

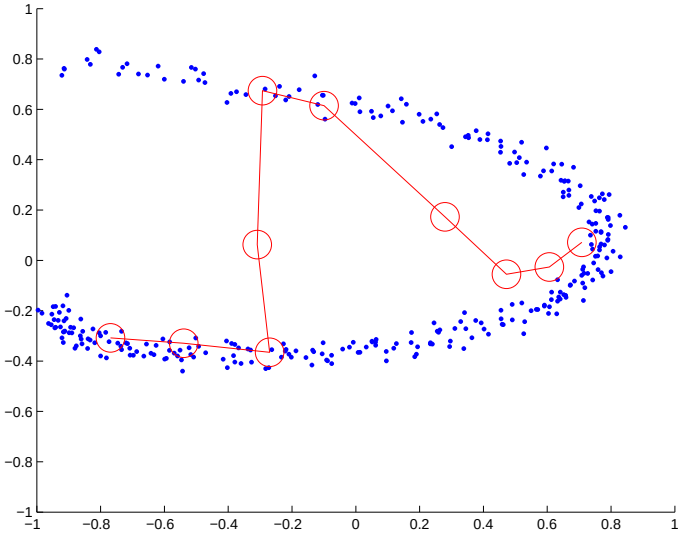

Figure 13.9: Application of SOM to the same data as above. Starting from a bad initial point, the SOM got stuck in a "local minimum".

Chapter 14

Short comparison of methods

To end the story, let's have a short comparison of the methods we learned.

	PCA/FA, nonlinear projections	ICA	clustering (k-means, GMM)
Applicable to what dimensions?	medium / high	low / medium (≤ 1000)	low/medium/high
Applicable to what sample sizes?	small/medium/high (nonliner: not high)	rather high	small/medium/high
Immediate goal	visualization, reduce computation & noise	finding underlying components / sources	intuitive interpretation, compression
Utility as preprocessing	generally useful, e.g. for ICA, regression & classification	sometimes useful, particularly if combined with variable selection	Apply e.g. regression separately in each cluster
Optimization problem	quadratic, except for SOM (thus EVD)	complicated, general (Gradient method, FastICA)	very special form (k-means, EM)
Problems with local maxima	no, except for SOM	yes	yes
Probabilistic	only FA	yes	only gaussian mixture model

The point on sample sizes may not be obvious based on the course material. But if you tried out ICA with rather small data sizes, you would see that the errors are too large. Basically, measuring non-Gaussianity needs a lot of data points, much more that measuring, e.g. the mean or (co)variance. Also, the amount of data needed grows fast as a function of dimension, so ICA cannot be used in very high dimensions.

Regarding local maxima, we see that methods based on maximization of quadratic forms (PCA, some variants of FA, MDS and some of its variants) are superior, since they have no local maxima and can be solved by eigen-value methods.

ICA is not really a preprocessing method in itself since it does not reduce the dimension and often the orthogonal rotation that it makes on the whitened data does not make any difference in further processing. Many classification methods, for example, would give identical results for ICA and data whitened by PCA. However, this need not be so, because we might be able to select the most useful independent components for further processing. Furthermore, many fashionable classification methods include such selection of components implicitly in the form of an L1 penalty (e.g. the LASSO methods). For those methods, preprocessing by ICA might be very useful because selecting a subset of independent components is likely to be more meaningful that selecting a subset of the original variables.

At this point, it is a useful exercice to go back to the four real-life examples in the introductory chapter (Figs. 1.1–1.4). Which methods would enable us to make the analyses reported in those publications? The authors did not always use exactly the same methods as described in this course, but the methods in this course are able to produce similar results.