Medical Image Analysis

Koen Van Leemput

September 7, 2023

Contents

Chapter 1

Smoothing and Interpolation

A fundamental prerequisite for solving many medical image analysis tasks is the ability to smooth and interpolate signals in two or three dimensions. In this chapter the basic principles of these techniques are reviewed.

1.1 Linear regression

Let $\mathbf{x} = (x_1, \ldots, x_D)^{\mathrm{T}}$ denote the spatial position in a D-dimensional space. In medical imaging, D is typically 2 or 3. Given N measurements $\{t_n\}_{n=1}^N$ at locations $\{\mathbf{x}_n\}_{n=1}^N$, a frequent task is to predict the value t at a new location $\mathbf{x}$. A simple model, known as *linear regression*, uses the function value

$$y(\mathbf{x}, \mathbf{w}) = w_0 + w_1 x_1 + \ldots + w_D x_D$$

as its prediction, where $w_0, \ldots, w_D$ are tunable weights that need to be estimatated from the available measurements. A more general form uses nonlinear functions of the input locations instead:

$$y(\mathbf{x}, \mathbf{w}) = w_0 + \sum_{m=1}^{M-1} w_m \phi_m(\mathbf{x}),$$

which greatly increases the flexibility of the model. Here the functions $\phi_m(\mathbf{x})$ are known as *basis functions*, and it is often convenient to define an additional "dummy" basis function $\phi_0(\mathbf{x}) = 1$, so that the model can be written as

$$y(\mathbf{x}, \mathbf{w}) = \sum_{m=0}^{M-1} w_m \phi_m(\mathbf{x}), \tag{1.1}$$

where $\mathbf{w} = (w_0, \ldots, w_{M-1})^{\mathrm{T}}$ are M tunable parameters.

In order to find suitable values of the parameters of the model, the following energy can be minimzed with respect to $\mathbf{w}$:

$$E(\mathbf{w}) = \sum_{n=1}^{N} \left(t_n - \sum_{m=0}^{M-1} w_m \phi_m(\mathbf{x}_n) \right)^2,$$

which simply sums of the squared distances between the measurements t_n and the model's predictions $y(\mathbf{x}_n, \mathbf{w})$. Taking the partial derivative with respect to parameter w_m yields

$$\frac{\partial E(\mathbf{w})}{\partial w_m} = -2 \sum_{n=1}^{N} \left(t_n - \sum_{m=0}^{M-1} w_m \phi_m(\mathbf{x}_n) \right) \phi_m(\mathbf{x}_n),$$

so that the gradient is given by

$$\nabla E(\mathbf{w}) = \begin{pmatrix} \frac{\partial E(\mathbf{w})}{\partial w_0} \\ \vdots \\ \frac{\partial E(\mathbf{w})}{\partial w_{M-1}} \end{pmatrix} = -2 \mathbf{\Phi}^{\mathrm{T}} \left(\mathbf{t} - \mathbf{\Phi} \mathbf{w} \right),$$

where $\mathbf{t} = (t_1, \ldots, t_N)^{\mathrm{T}}$ is a vector stacking all measurements, and

$$\mathbf{\Phi} = \begin{pmatrix} \phi_0(\mathbf{x}_1) & \phi_1(\mathbf{x}_1) & \ldots & \phi_{M-1}(\mathbf{x}_1) \\ \phi_0(\mathbf{x}_2) & \phi_1(\mathbf{x}_2) & \ldots & \phi_{M-1}(\mathbf{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_0(\mathbf{x}_N) & \phi_1(\mathbf{x}_N) & \ldots & \phi_{M-1}(\mathbf{x}_N) \end{pmatrix} \tag{1.2}$$

is a $N \times M$ matrix containing the value of all the basis functions in each of the measurement locations. Settings this gradient to zero gives

$$\mathbf{\Phi}^{\mathrm{T}} \left(\mathbf{t} - \mathbf{\Phi} \mathbf{w} \right) = \mathbf{0}$$

with solution

$$\mathbf{w} = \left(\mathbf{\Phi}^{\mathrm{T}} \mathbf{\Phi} \right)^{-1} \mathbf{\Phi}^{\mathrm{T}} \mathbf{t}. \tag{1.3}$$

With these parameters, a prediction at a new location $\mathbf{x}$ is obtained by evaluating (1.1).

1.1.1 Regularization

In some cases prior knowledge is available about the parameters $\mathbf{w}$ (for instance, that their values are typically small), or about the expected behavior of $y(\mathbf{x}, \mathbf{w})$ (for instance that it is a smooth function). This can be taken into account by adding a regularization term to the energy, optimizing

$$E(\mathbf{w}) + \lambda \|\mathbf{\Omega} \mathbf{w}\|^2 \tag{1.4}$$

instead. Here λ is a parameter controlling the strength of the regularization, and $\mathbf{\Omega}$ is a matrix chosen so that it penalizes undesirable values of $\mathbf{w}$. An example of $\mathbf{\Omega}$ could simply be the identify matrix, so that the regularization energy becomes

$$\|\mathbf{w}\|^2 = w_0^2 + \ldots + w_{M-1}^2,$$

which penalizes large values of $\mathbf{w}$. Other examples include $\mathbf{\Omega} = \mathbf{\Phi}$, so that large values of the *predictions* $\hat{\mathbf{t}} = \mathbf{\Phi}\mathbf{w}$ at the measurement locations are penalized; or even $\mathbf{\Omega} = \mathbf{\Gamma}\mathbf{\Phi}$, with $\mathbf{\Gamma}$ detecting rapid spatial fluctuations in $\hat{\mathbf{t}}$ (e.g., first- or second-order finite differences).

Optimizing (1.4) with respect to $\mathbf{w}$, by setting its gradient to zero and solving for $\mathbf{w}$ as before, yields

$$\mathbf{w} = \left(\mathbf{\Phi}^{\mathrm{T}}\mathbf{\Phi} + \lambda\mathbf{\Omega}^{\mathrm{T}}\mathbf{\Omega}\right)^{-1}\mathbf{\Phi}^{\mathrm{T}}\mathbf{t}, \tag{1.5}$$

which is a straightforward extension of (1.3)

1.2 Smoothing and interpolation of 1D signals

Before addressing applications in two- or three-dimensional imaging problems, it is instructive to first study the one-dimensional (1D) setting. In this scenario, we are seeking functions of the form

$$y(x, \mathbf{w}) = \sum_{m=0}^{M-1} w_m \phi_m(x), \tag{1.6}$$

where x is a scalar (a location in 1D space). Furthermore, the N measurement locations lie on a regular grid with unit interval:

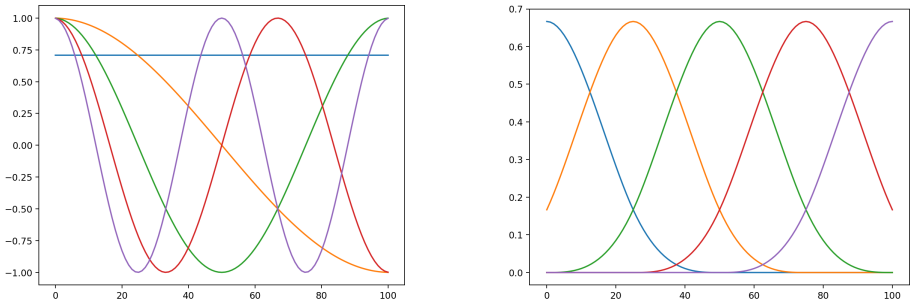
$$x_n = n - 1, \quad \forall n = 1, \ldots, N,$$

i.e., the signals $\mathbf{t}$ are 1D "images".

In signal processing applications, the basis functions associated with the "type II" discrete cosine transform (DCT) often have useful analytical and numerical properties. In particular, they often allow to compress signals efficiently by discarding the highest frequencies; $\mathbf{\Phi}\mathbf{w}$ and $\mathbf{\Phi}^{\mathrm{T}}\mathbf{t}$ can be computed very quickly using the fast Fourier transform (FTT); and $\mathbf{\Phi}^{\mathrm{T}}\mathbf{\Phi} = N\mathbf{I}$. They are defined as follows:

$$\phi_m(x) = \left(\frac{1}{\sqrt{2}}\right)^{[m=0]} \cos\left(\pi m(x + \frac{1}{2})/N\right), \tag{1.7}$$

for frequencies $m = 0, \ldots, M-1$, where $M \leq N$. An example is shown in Fig. 1.1a for $M = 5$.



(a) DCT (b) B-spline

Figure 1.1: Two sets of often-used basis functions ($M = 5$).

Another set of useful basis functions are derived from B-splines, which are symmetrical functions constructed from the repeated convolution of a rectangular pulse β^0:

$$\begin{aligned} \beta^0(x) &= \begin{cases} 1, & -\frac{1}{2} < x < \frac{1}{2} \\ \frac{1}{2}, & |x| = \frac{1}{2} \\ 0, & \text{otherwise,} \end{cases} \\ \beta^p(x) &= \underbrace{\left(\beta^0 * \beta^0 * \cdots * \beta^0\right)}_{(p+1) \text{ times}}(x). \end{aligned}$$

Here p is called the *order* of the B-spline, and $*$ denotes a convolution:

$$(f * g)(x) = \int_{\tau=-\infty}^{\infty} f(\tau) g(x-\tau) \mathrm{d}\tau.$$

Often-used B-spline orders (e.g, for interpolation) are $p = 1$:

$$\beta^1(x) = \begin{cases} 1 - |x|, & |x| < 1 \\ 0, & \text{otherwise,} \end{cases}$$

and $p = 3$ ("cubic" B-spline):

$$\beta^3(x) = \begin{cases} \frac{2}{3} - |x|^2 + \frac{|x|^3}{2}, & |x| < 1 \\ \frac{(2-|x|)^3}{6}, & 1 \le |x| < 2 \\ 0, & \text{otherwise.} \end{cases}$$

B-splines of order zero, one and three are illustrated in Fig. 1.2

B-splines are typically used to construct basis functions by scaling them with some factor h, and shifting them to be h unites apart:

$$\phi_m(x) = \beta^p\left(\frac{x - mh}{h}\right). \tag{1.8}$$

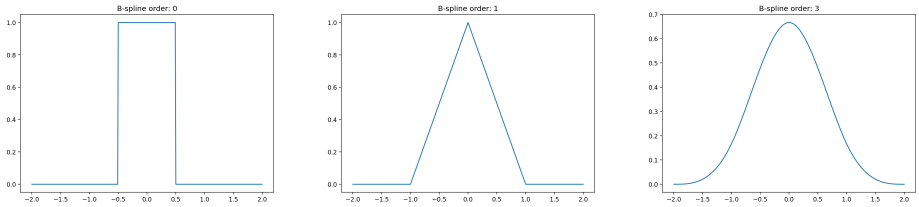


Figure 1.2: B-splines $\beta^0(x)$, $\beta^1(x)$ and $\beta^3(x)$.

This is illustrated in Fig. 1.1b for the case $h = 25$. The usefulness of these type of basis functions stems from their limited support (they are only non-zero within a small range), which can be used to dramatically reduce the number of terms that need to be summed over in practical implementations (e.g., in (1.6)). Furthermore, in interpolation applications B-splines enjoy a number of theoretical and numerical advantages (e.g., the curvature of $y(x, \mathbf{w})$ is minimized, and the required matrix inversions can be performed with very fast filter-based methods) [1].

1.2.1 Smoothing

In some applications, the measurements $t_n, n = 1, \ldots, N$ are only noisy observations, and the aim is to recover the "denoised" underlying signal $\hat{t}_n = y(x_n, \mathbf{w})$ at the locations x_n. Collecting these denoised estimates into a vector $\hat{\mathbf{t}} = (\hat{t}_1, \ldots, \hat{t}_N)^{\mathrm{T}}$, we obtain (cf. (1.6)):

$$\hat{\mathbf{t}} = \mathbf{\Phi}\mathbf{w}. \tag{1.9}$$

Plugging in the solution (1.5) then yields

$$\hat{\mathbf{t}} = \mathbf{S}\mathbf{t},$$

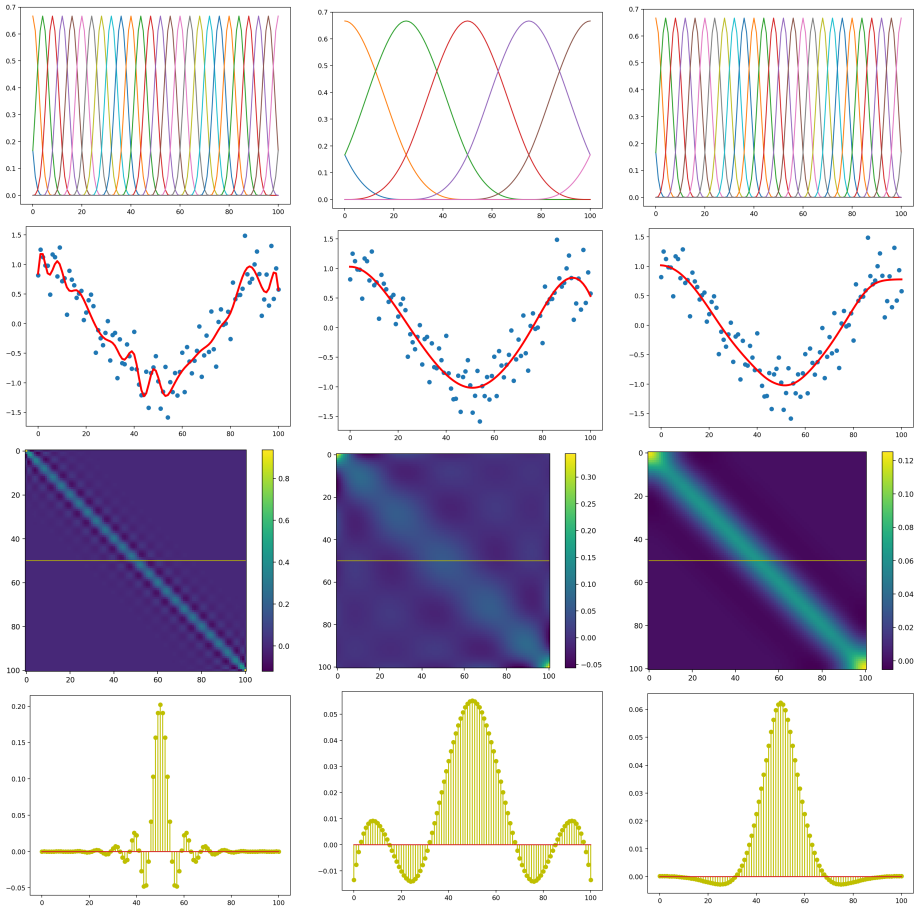
where

$$\mathbf{S} = \mathbf{\Phi}\left(\mathbf{\Phi}^{\mathrm{T}}\mathbf{\Phi} + \lambda\mathbf{\Omega}^{\mathrm{T}}\mathbf{\Omega}\right)^{-1}\mathbf{\Phi}^{\mathrm{T}}$$

is a $N \times N$ "smoothing" matrix that transforms a noisy signal $\mathbf{t}$ into a "denoised" signal $\hat{\mathbf{t}}$ by making linear combinations of the elements in $\mathbf{t}$. The n-th row of $\mathbf{S}$ contains the weights assigned to the various elements of $\mathbf{t}$ in the computation of $\hat{t}_n$, as illustrated in Fig. 1.3 (bottom row).

The amount of smoothing that is applied can be controlled by both the number of basis functions M and the regularization parameter λ: chosing higher M or lower λ will produce less smoothing. This effect is illustrated in Fig. 1.3. In the limit, when $M = N$ and $\lambda = 0$, $\mathbf{S}$ becomes the identify matrix[1] and no smoothing is applied at all.

[1] Because $\mathbf{\Phi}$ is square when $M = N$, so that $(\mathbf{\Phi}^{\mathrm{T}}\mathbf{\Phi})^{-1} = \mathbf{\Phi}^{-1}(\mathbf{\Phi}^{\mathrm{T}})^{-1}$.



(a) $M = 28$ and $\gamma = 0$ (b) $M = 7$ and $\gamma = 0$ (c) $M = 28$ and $\gamma = 1000$

Figure 1.3: Smoothing of a 1D signal when the number of basis functions (M) and the regularization strength (γ) is varied. The regularizer is of the form $\mathbf{\Omega} = \mathbf{\Gamma\Phi}$, where $\mathbf{\Gamma}$ computes second-order finite differences (the n-th row has all zeros except for columns $n-1$, n and $n+1$, who have elements -1, 2 and -1, respectively). From top to bottom: basis functions that were used; noisy signal $\mathbf{t}$ and denoised signal $\hat{\mathbf{t}}$ in blue and red, respectively; smooting matrix $\mathbf{S}$; and the middle row of $\mathbf{S}$.

1.2.2 Interpolation

When several medical images need to be compared, a common problem is that the intensities of an image need to be evaluated at locations x different from the integer locations $x_1 = 0, x_2 = 1, \ldots x_N = N-1$ where the intensities $t_1, t_2, \ldots, t_N$ are defined. A standard solution is to set $\lambda = 0$, and use the shifted B-spline basis functions of (1.8) with scaling factor $h = 1$, i.e., a B-spline β^p is centered around each of the N integer coordinates x_n. Because $M = N$, the solution is given by[1]

$$\mathbf{w} = \mathbf{\Phi}^{-1}\mathbf{t},$$

for which fast, dedicated numerical solvers are available [1].

As already analyzed above, the resulting function $y(x, \mathbf{w})$ will pass exactly through the original intensities t_n at the integer locations x_n. However, the model will also fill in "interpolating" intensity values everywhere else, with the exact behavior depending on the order of the B-spline. For order $p = 0$, so-called *nearest neighbor* interpolation is obtained, in which the predicted value at location x is simply the value t_n of the measurement location x_n that is nearest to x[2] . For $p = 1$, *linear* interpolation is obtained, in which $y(x, \mathbf{w})$ is piece-wise linear between the integer locations $\{x_n\}_{n=1}^{N}$. Finally, $p = 3$ results in so-called *cubic* interpolation, which is the method of choice in most practical applications. The effect of the B-spline order on the interpolation behavior of the model is illustrated in Fig. 1.4.

1.3 Smoothing and interpolation in higher dimensions

When going to $D = 2$ dimensions, the spatial locations $\mathbf{x} = (x_1, x_2)$ consist of two coordinates x_1 and x_2, and the measurement signal becomes a two-dimensional image, represented by a matrix with N_1 rows and N_2 columns:

$$\mathbf{T} = \begin{pmatrix} t_{1,1} & t_{1,2} & \cdots & t_{1,N_2} \\ t_{2,1} & t_{2,2} & \cdots & t_{2,N_2} \\ \vdots & \vdots & \ddots & \vdots \\ t_{N_1,1} & t_{N_1,2} & \cdots & t_{N_1,N_2} \end{pmatrix}.$$

It will be convenient to define a *vectorization* operation – denoted by vec$(\cdot)$ – that re-arranges this $N_1 \times N_2$ image into a 1D signal $\mathbf{t}$ of length $N = N_1 N_2$, by

[2]Strictly speaking an exception is made at locations x that fall exactly half-way between two integer locations, where the average value is returned.

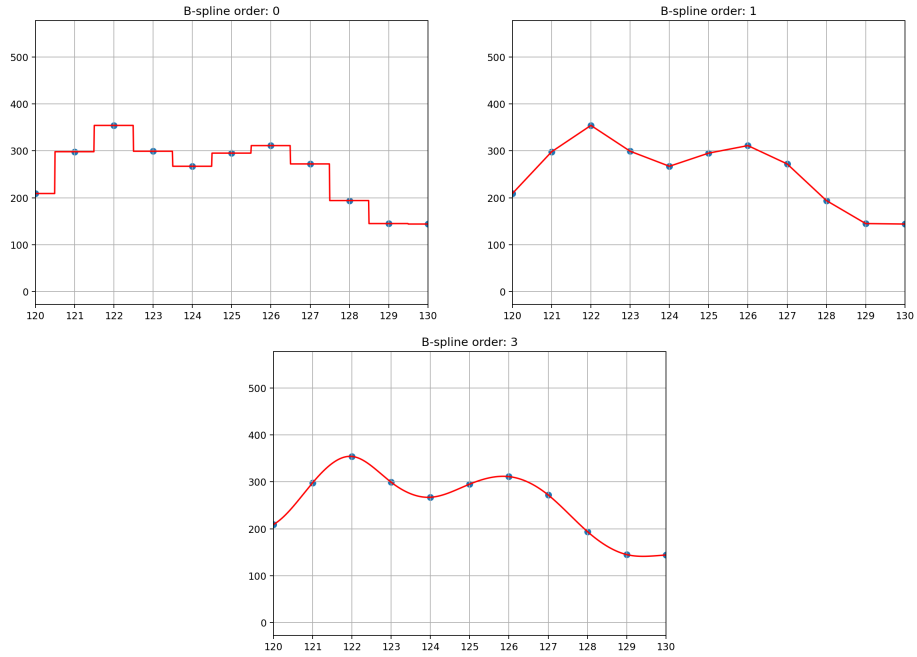


Figure 1.4: Nearest-neighbor, linear and cubic interpolation.

stacking the columns of $\mathbf{T}$ under each other:

$$\mathbf{t} = \text{vec}(\mathbf{T}) = \begin{pmatrix} t_{1,1} \\ \vdots \\ t_{N_1,1} \\ t_{1,2} \\ \vdots \\ t_{N_1,2} \\ \vdots \\ t_{N_1,N_2} \end{pmatrix}. \tag{1.10}$$

In this re-arrangment, element t_{n_1,n_2} in $\mathbf{T}$ corresponds to element t_n in $\mathbf{t}$, where $n = n_1 + (n_2 - 1)N_1$.

In order to proceed, we also need to choose appropriate basis functions $\phi_m(\mathbf{x})$ that work in 2D. A convenient choice is often *separable* basis functions, which are simply the product of two 1D basis funcions (one taking as input x_1, and the other one x_2). If there are M_1 basis functions in the first (row) direction, and M_2 in the second (column) direction, taking all the combinations yields a total of $M = M_1 M_2$ basis functions in 2D, given by

$$\phi_m(\mathbf{x}) = \phi_{m_1}(x_1)\phi_{m_2}(x_2) \tag{1.11}$$

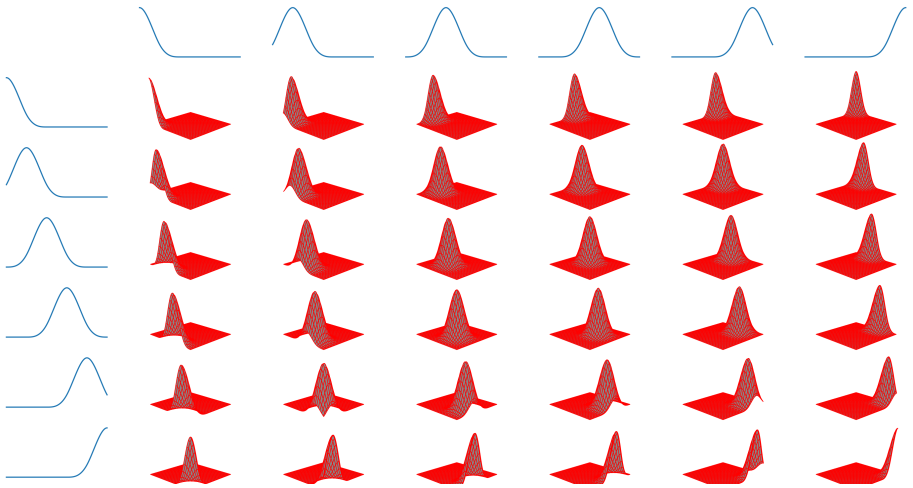


Figure 1.5: The 36 separable 2D basis functions produced from two sets of 6 1D basis functions.

for $m = m_1 + m_2 M_1$ (recall that m_1 and m_2 run from 0 to $M_1 - 1$ and $M_2 - 1$ respectively). Fig. 1.5 illustrates the 2D basis functions generated from 1D B-spline basis functions this way.

To simplify notation, it will be convenient to use the Kronecker product of two matrices:

$$\mathbf{A} \otimes \mathbf{B} = \left(\begin{array}{c|c|c} a_{1,1}\mathbf{B} & a_{2,1}\mathbf{B} & \dots \\ \hline a_{2,1}\mathbf{B} & a_{2,2}\mathbf{B} & \dots \\ \hline \vdots & \vdots & \ddots \end{array} \right).$$

If $\mathbf{\Phi}_1$ denotes the $N_1 \times M_1$ matrix containing the output of the M_1 basis functions evaluated at the integer locations $x_1 = 0, \dots, N_1 - 1$, and $\mathbf{\Phi}_2$ is similarly defined for the second direction, then

$$\mathbf{\Phi} = \mathbf{\Phi}_2 \otimes \mathbf{\Phi}_1 \tag{1.12}$$

is a $N \times M$ matrix that contains vectorized versions of all M basis functions in 2D, where the m-th column contains (1.11) evaluated at all the pixel locations in $\mathbf{T}$.

Equipped with (1.10), (1.11) and (1.12), any smoothing or interpolation problem in 2D can be directly mapped into a 1D problem, so that the solutions described in Sec. 1.2 can (in principle) be directly applied. The extension to 3D is also straightforward; for example, vectorized versions of the output of $M = M_1 M_2 M_3$ basis functions, evaluated at all $N = N_1 N_2 N_3$ locations, are given by

$$\mathbf{\Phi} = \mathbf{\Phi}_3 \otimes \mathbf{\Phi}_2 \otimes \mathbf{\Phi}_1.$$

1.3.1 Exploiting separability

Although the problem is in theory solved, in practice the computations involve storing and inverting a $M \times M$ matrix, which can be problematic when M is very large (i.e., when the model involves many basis functions). As an example, consider the interpolation of a 3D volume of size $256 \times 256 \times 256$, an application in which there are $M = 256^3$ basis functions. At this size, naively storing a matrix with M^2 elements at 64 bits per element (double-precision floating point) would take 2048 TB; subsequently inverting it would take $\mathcal{O}(M^3) = \mathcal{O}(256^9)$ operations, making it entirely infeasible. In contrast, re-arranging the computations as outlined below only requires storing and inverting three 256×256 matrices, taking 512 KB and $\mathcal{O}(256^3)$ operations each. For a 2D image of size 256×256, the savings are more modest but still very substantial: storing a matrix of 32 GB vs. two matrices of 512 KB, and $\mathcal{O}(256^6)$ operations vs. two times $\mathcal{O}(256^3)$ operations for matrix inversions, which is almost 10 million times faster.

When no regularization is used ($\lambda = 0$), the separability of the basis functions can be exploited as follows. In 2D, element $m = m_1 + m_2 M_1$ in the length-M vector $\mathbf{c} = \mathbf{\Phi}^{\mathrm{T}}\mathbf{t}$ is given by

$$\begin{aligned} c_m &= \sum_{n=1}^{N} \phi_{n,m} t_n \\ &= \sum_{n_1=1}^{N_1} \sum_{n_2=1}^{N_2} \phi^1_{n_1,m_1} \phi^2_{n_2,m_2} t_{n_1,n_2} \\ &= \sum_{n_1=1}^{N_1} \phi^1_{n_1,m_1} \left(\sum_{n_2=1}^{N_2} \phi^2_{n_2,m_2} t_{n_1,n_2} \right), \end{aligned}$$

where $\phi^1_{:,:}$ and $\phi^2_{:,:}$ denote elements in $\mathbf{\Phi}_1$ and $\mathbf{\Phi}_2$, respectively. The key insight is that the same summation over the second direction (in parentheses) is needed for each new m_1, and therefore needs to be computed only once. In matrix form, this can be expressed as

$$\mathbf{C} = \mathbf{\Phi}_1^{\mathrm{T}} \mathbf{T} \mathbf{\Phi}_2,$$

where $\mathbf{C}$ is a $M_1 \times M_2$ matrix such that $\mathrm{vec}(\mathbf{C}) = \mathbf{c}$. Similarly, it can be shown that $\mathbf{\Phi}\mathbf{w}$ can be more efficiently computed as $\mathbf{\Phi}_x \mathbf{W} \mathbf{\Phi}_y^{\mathrm{T}}$, with $\mathrm{vec}(\mathbf{W}) = \mathbf{w}$. Therefore, the solution (1.3), which can be written as

$$\mathbf{\Phi}^{\mathrm{T}} (\mathbf{\Phi}\mathbf{w}) = \mathbf{\Phi}^{\mathrm{T}}\mathbf{t},$$

can also be expressed as

$$\mathbf{\Phi}_1^{\mathrm{T}} \left(\mathbf{\Phi}_1 \mathbf{W} \mathbf{\Phi}_2^{\mathrm{T}} \right) \mathbf{\Phi}_2 = \mathbf{\Phi}_1^{\mathrm{T}} \mathbf{T} \mathbf{\Phi}_2,$$

so that finally

$$\mathbf{W} = \left(\mathbf{\Phi}_1^{\mathrm{T}} \mathbf{\Phi}_1 \right)^{-1} \mathbf{\Phi}_1^{\mathrm{T}} \mathbf{T} \mathbf{\Phi}_2 \left(\mathbf{\Phi}_2^{\mathrm{T}} \mathbf{\Phi}_2 \right)^{-1} \tag{1.13}$$

can be used to compute the elements of $\mathbf{w}$ very efficiently.

1.3.2 Smoothing in 2D

Plugging the solution (1.13) in (1.9), and using the same approach as above, it is easy to see that a smoothed image can be obtained as

$$\hat{\mathbf{T}} = \mathbf{S}_1 \mathbf{T} \mathbf{S}_2^{\mathrm{T}},$$

where

$$\mathbf{S}_1 = \mathbf{\Phi}_1 \left(\mathbf{\Phi}_1^{\mathrm{T}} \mathbf{\Phi}_1 \right)^{-1} \mathbf{\Phi}_1^{\mathrm{T}}$$

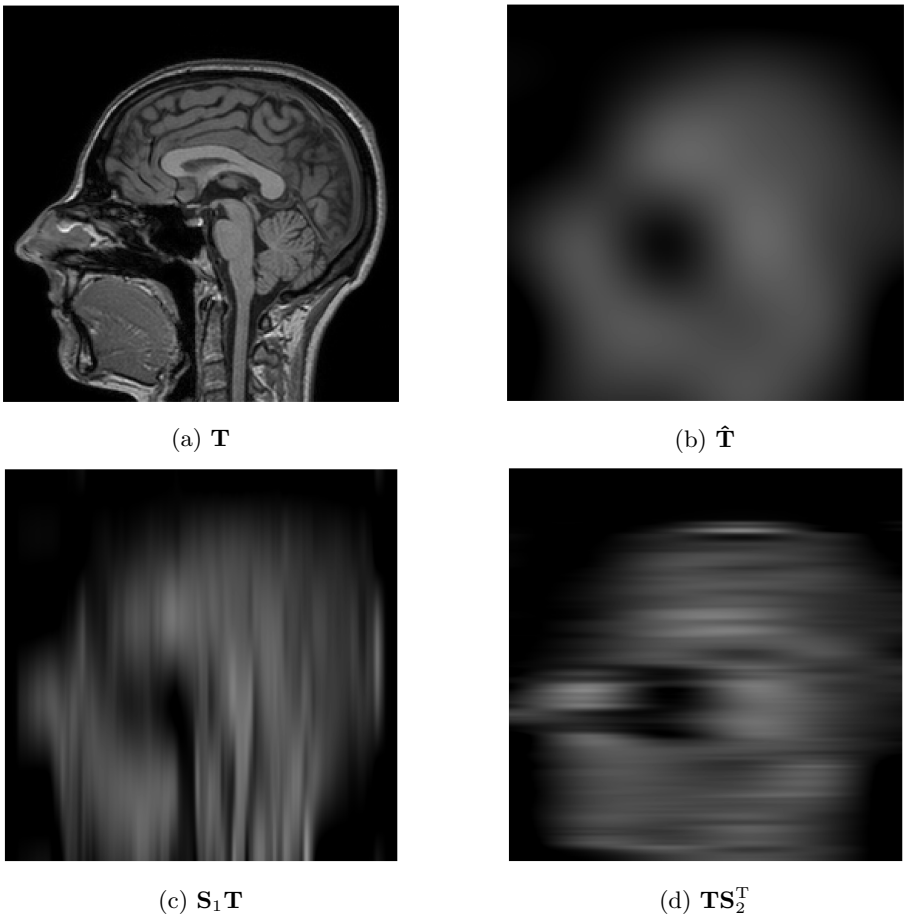
is a $N_1 \times N_1$ smoothing matrix that smoothes each column in $\mathbf{T}$ independently in the row-direction only, and the corresponding $\mathbf{S}_2$ subsequently smoothes each row in the column direction (or vice versa). This is illustrated in Fig. 1.6.

1.3.3 Interpolation in 2D

For interpolation, the parameters are simply obtained as

$$\mathbf{W} = \mathbf{\Phi}_1^{-1} \mathbf{T} \left(\mathbf{\Phi}_2^{-1} \right)^{\mathrm{T}}.$$

Interpolated values at new locations $\mathbf{x}$ are then computed using (1.1), exploiting the separability of the basis functions (1.11). Nearest-neighbor, linear and cubic interpolation in 2D are illustrated in Fig. 1.7.



(a) $\mathbf{T}$ (b) $\hat{\mathbf{T}}$

(c) $\mathbf{S}_1\mathbf{T}$ (d) $\mathbf{T}\mathbf{S}_2^{\mathrm{T}}$

Figure 1.6: Smoothing of a 2D image $\mathbf{T}$ by fitting the 2D basis functions of Fig. 1.5 to it. Computationally the smoothed result $\hat{\mathbf{T}}$ can be obtained by smoothing across the rows and then the columns (or vice versa), using the 1D basis functions shown in Fig. 1.5.

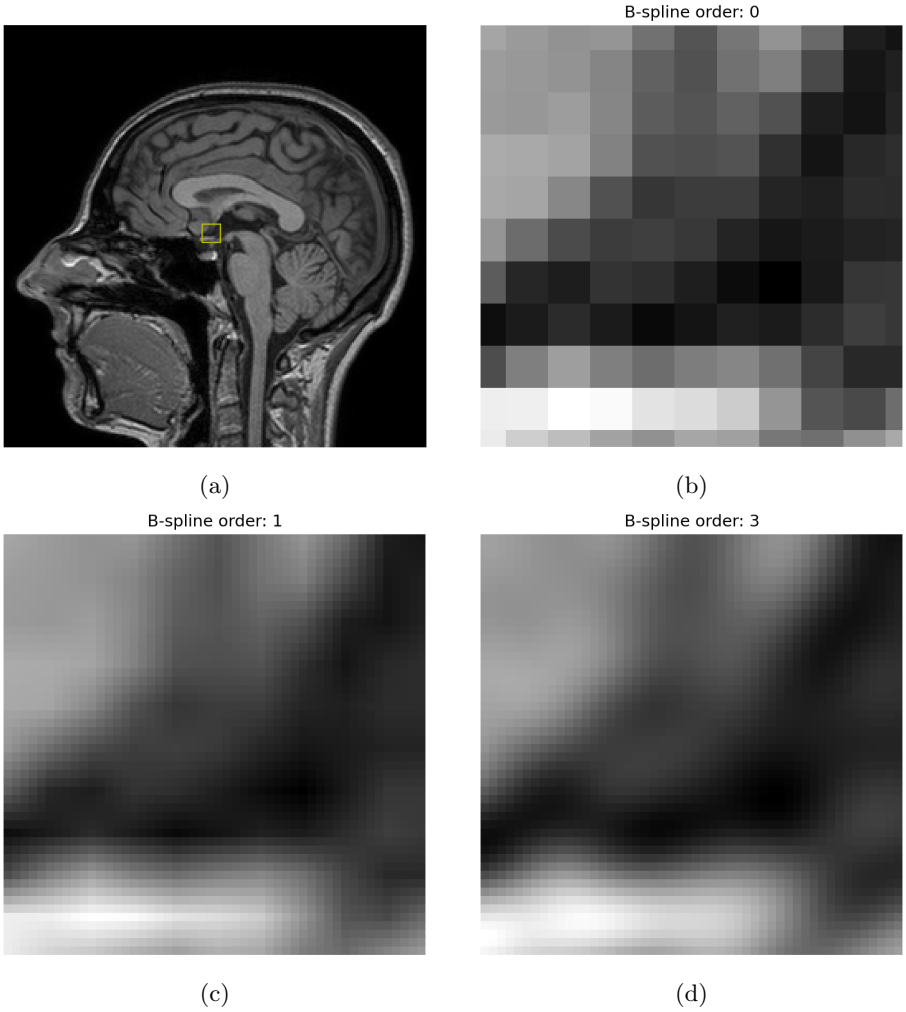


Figure 1.7: 2D nearest-neighbor (b), linear (c) and cubic (d) interpolation within the small image area indicated in (a).

Chapter 2

Image Registration

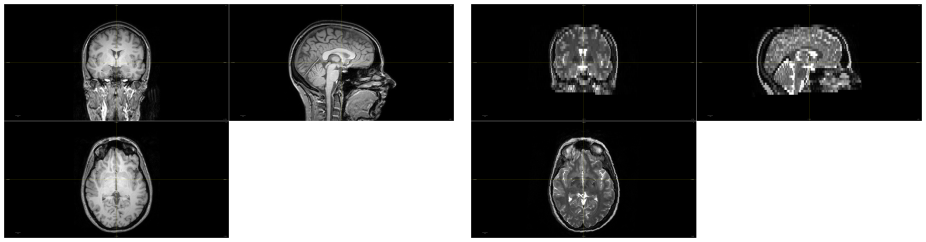
In many situations, the information contained in two or more images needs to be combined. Examples of such situations include interpreting images of the same patient acquired at different time points, or with different imaging modalities, as well as comparing the anatomy or function between various subject groups (e.g., to study how patients differ from controls in a clinical research study). In order for images to be used this way, they need to be *spatially aligned* so that corresponding structures appear in corresponding locations. The process of aligning images is called *image registration*. This chapter introduces some of its basic concepts in a medical imaging context.

2.1 Coordinate systems

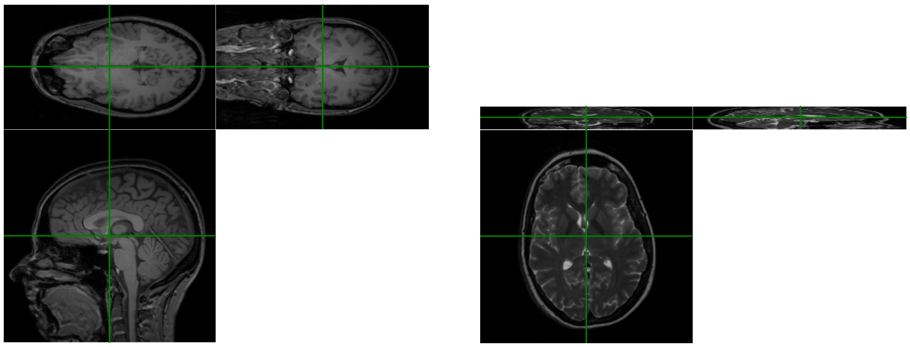
So far we have taken a rather cavalier attitude regarding the spatial locations of voxels (pixels in 3D): for 3D images we have simply assumed that we can index individual voxels by their location $\mathbf{x} = (x_1, x_2, x_3)^{\mathrm{T}}$, where x_1, x_2 and x_3 are integers. In reality, however, scanners can generate images with almost any voxel size (spacing between the voxels in each of the three dimensions) and in any orientation. In magnetic resonance imaging (MRI), for instance, it is not uncommon to acquire multiple images when a patient is inside the scanner, each defined on its own image grid. An example is shown in Fig. 2.1

In order to relate multiple images to each other, we'll need to differentiate the concept of "voxel coordinates" (which are defined in integers units, and index indivual voxels in a 3D image grid) from that of "world coordinates" (indicating the spatial positions of voxels in the real world, and measured in mm). Specifically, let $\mathbf{v} = (v_1, v_2, v_3)^{\mathrm{T}}$ denote the voxel coordinate of a voxel in an image of size $N_1 \times N_2 \times N_3$, where $v_d = 0, \ldots N_d - 1$ for all dimensions $d = 1, \ldots, 3$. In addition to a $N_1 \times N_2 \times N_3$ array of intensities, the scanner will also encode a 3×3 matrix $\mathbf{A}$ and a 3×1 vector $\mathbf{t}$ to map voxel coordinates into world coordinates as follows:

$$\mathbf{x} = \mathbf{A}\mathbf{v} + \mathbf{t}, \tag{2.1}$$



(a) Displayed in world coordinates



(b) Displayed in voxel coordinates

Figure 2.1: Two MR images acquired within the same scan session: T1-weighted (left) and T2-weighted (right). Although the two images show corresponding structures in corresponding locations in *world coordinates* (a), the 3D array of image intensities acquird by the scanner is actually very different between the two scans: The T1-weighted scan is a $256 \times 256 \times 150$ volume with voxel size $0.94 \times 0.94 \times 1.2\ mm^3$ acquired in the sagittal direction, whereas T2-weighted scan is a $256 \times 256 \times 28$ volume with voxel size $0.90 \times 0.90 \times 4.98\ mm^3$ acquired in the axial direction. This is clearly visible when displaying the two images in *voxel coordinates* instead (b).

where

$$\mathbf{A} = \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{3,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix} \quad \text{and} \quad \mathbf{t} = \begin{pmatrix} t_1 \\ t_2 \\ t_3 \end{pmatrix}. \tag{2.2}$$

For instance, the combination

$$\mathbf{A} = \begin{pmatrix} 0.95 & 0 & 0 \\ 0 & 0.95 & 0 \\ 0 & 0 & 4.5 \end{pmatrix} \quad \text{and} \quad \mathbf{t} = \begin{pmatrix} -120.0 \\ -120.0 \\ -65.0 \end{pmatrix}$$

for an image of size $256 \times 256 \times 30$ would indicate that the size of a voxel is $0.95\text{mm} \times 0.95\text{mm} \times 4.5$ mm, and that the origin of the world coordinate system (the location where $\mathbf{x} = \mathbf{0}$) lies somewhere in the middle of the image array. Very often, $\mathbf{A}$ will also include more general spatial transformations, such as 3D rotations or "flipping" of axes (negative voxel spacings, so that an increase in some voxel coordinate(s) will results in a *decrease* in world coordinate(s)).

Who decides what directions $\mathbf{A}$ should encode, or where the origin of the world coordinate system should be located? This is a matter of convention, and several such conventions exist. The origin is often considered to be approximately at the center of the anatomical structure being scanned; a well-known world coordinate system is the RAS convention, where x_1 increases towards the **R**ight of the patient, x_2 towards the **A**nterior (front), and x_3 towards the **S**uperior (top) of the patient. Another often-used convention is LPS (**L**eft, **P**osterior, **S**uperior), which is similar to the RAS system but with the direction of the first two axes swapped. When working with medical images, it is critically important to know what convention is used for each image, lest a patient be operated on the wrong side of their body!

Consider again the situation in Fig. 2.1, where two different images of the same patient were acquired within the same scanning session: one T1-weighted scan with voxel-to-world mapping $\{\mathbf{A}_{T1}, \mathbf{t}_{T1}\}$, and one T2-weighted scan with $\{\mathbf{A}_{T2}, \mathbf{t}_{T2}\}$. In order to compute the voxel coordinate $\mathbf{v}_{T2}$ in the T2-weighted scan corresponding to a s voxel coordinate $\mathbf{v}_{T1}$ in the T1-weighed one, it is often convenient to re-write (2.1) as follows:

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ 1 \end{pmatrix} = \underbrace{\begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & t_1 \\ a_{2,1} & a_{2,2} & a_{2,3} & t_2 \\ a_{3,1} & a_{3,2} & a_{3,3} & t_3 \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{\mathbf{M}} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ 1 \end{pmatrix}, \tag{2.3}$$

where the 4×4 matrix $\mathbf{M}$ is called an *affine* matrix. This technique, which uses so-called homogeneous coordinates (vectors are augmented with a 1 at the end), absorbes the addtion of the vector $\mathbf{t}$ into a single matrix multiplication, which makes concatenating several affine matrices very easy: the mapping of $\mathbf{v}_{T1}$ to $\mathbf{v}_{T2}$ is given by:

$$\begin{pmatrix} \mathbf{v}_{T2} \\ 1 \end{pmatrix} = \mathbf{M}_{T2}^{-1} \cdot \mathbf{M}_{T1} \cdot \begin{pmatrix} \mathbf{v}_{T1} \\ 1 \end{pmatrix}.$$

We will soon see examples where more than two affine matrices are combined this way.

2.2 Transformation models

In many situations, images will not be perfectly aligned as in Fig. 2.1. This could be because the patient has moved between the two acquisitions, for instance because the scanning was performed on a different day or on a different scanner (e.g., MRI vs. CT scan). It could also be because of organ deformation, caused by breathing in a dynamic MRI, for instance, or because of tumor shrinkage or patient weight loss during the course of a radiation therapy treatment. In such situations, an additional geometrical transformation exists that a registration method should try to recover so that the images can still be compared to each other.

Let $\mathbf{x} = (x_1, \ldots, x_D)^{\mathrm{T}}$ denote a spatial location (in world coordinates) in an image that we will call the *fixed* image in the remainder, where $D = 2$ in 2D and $D = 3$ in 3D images. Similarly, we will use $\mathbf{y} = (y_1, \ldots, y_D)^{\mathrm{T}}$ to denote spatial locations (again in world coordinates) in another image, which we will refer to as the *moving* image. Our task is to find a transformation model

$$\mathbf{y}(\mathbf{x}, \mathbf{w}) = \begin{pmatrix} y_1(\mathbf{x}, \mathbf{w}) \\ \vdots \\ y_D(\mathbf{x}, \mathbf{w}) \end{pmatrix} \tag{2.4}$$

so that points in the fixed image are mapped to the corresponding anatomical locations in the moving image. Here $y_d(\mathbf{x}, \mathbf{w})$ is a function that governs how points in the fixed image move along the d-th direction in the moving image as the parameter vector $\mathbf{w}$ is varied.

2.2.1 Linear transformations

Affine transformation: In the most general form of linear transformations, lines that are straight in the fixed image will still be straight when mapped into the moving image, but angles and areas/volumes are typically not preserved. An illustration is provided in Fig. 2.2a. Such transformations are useful to roughly align images of different individuals or time points (for instance as a first step in a subsequent nonlinear registration algorithm), so that the major differences in size and orientation are removed. The transformation model is governed by:

$$\mathbf{y}(\mathbf{x}, \mathbf{w}) = \mathbf{A}\mathbf{x} + \mathbf{t},$$

where $\mathbf{A}$ and $\mathbf{t}$ are given by

$$\mathbf{A} = \begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{pmatrix} \quad \text{and} \quad \mathbf{t} = \begin{pmatrix} t_1 \\ t_2 \end{pmatrix}$$

in 2D, and by (2.2) in 3D. Here $\mathbf{t}$ implements a translation, whereas rotation, scaling and skewing are encoded in $\mathbf{A}$. This transformation model is called the *affine* transformation model. Referrring to (2.4), the d-th coordinate of points mapped into the moving image is therefore given by the linear function

$$y_d(\mathbf{x}, \mathbf{w}_d) = t_d + a_{d,1}x_1 + \ldots + a_{d,D}x_D, \tag{2.5}$$

with parameters $\mathbf{w}_d = (t_d, a_{d,1}, \ldots, a_{d,D})^{\mathrm{T}}$. Note that these parameters only contain elements of the d-th row of $\mathbf{A}$ and $\mathbf{t}$; the vector $\mathbf{w} = (\mathbf{w}_1^{\mathrm{T}}, \ldots, \mathbf{w}_D^{\mathrm{T}})^{\mathrm{T}}$ collects all the transformation parameters of the entire D-dimensional model.

In 3D, the affine matrix notation of (2.3) can be used to compute the mapping of a voxel coordinate $\mathbf{v}_F$ in the fixed image into the corresponding voxel coordinate $\mathbf{v}_M$ in the moving image as follows:

$$\mathbf{v}_M = \mathbf{M}_M^{-1} \cdot \mathbf{M} \cdot \mathbf{M}_F \cdot \mathbf{v}_F, \tag{2.6}$$

where $\mathbf{M}_F$ and $\mathbf{M}_M$ denote the voxel-to-world affine matrix of the fixed and the moving image, respectively.

Rigid transformation: In situations where the moving and the fixed image were both taken from the same rigid structure (e.g, the head) in the same subject, the ability of the affine transformation model to arbitrarily scale and skew images becomes a liabilty. In those cases, a model that can only account for a translation and a rotation is more appropriate:

$$\mathbf{y}(\mathbf{x}, \mathbf{w}) = \mathbf{R}\mathbf{x} + \mathbf{t},$$

where $\mathbf{R}^{\mathrm{T}}\mathbf{R} = \mathbf{I}$ and $\det(\mathbf{R}) = 1$. This transformation model is called the *rigid* transformation model, and is illustrated in Fig. 2.2b. In 2D, rotations can be enforced by parameterizing the rotation matrix $\mathbf{R}$ as follows:

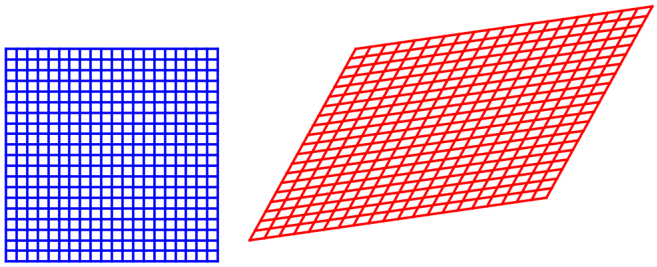
$$\mathbf{R} = \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{pmatrix},$$

where α is a rotation angle. The parameters of this model are therefore $\mathbf{w} = (\alpha, t_1, t_2)^{\mathrm{T}}$. In 3D there are many possible parametrizations of the rotation matrix, but one possibility is as follows:

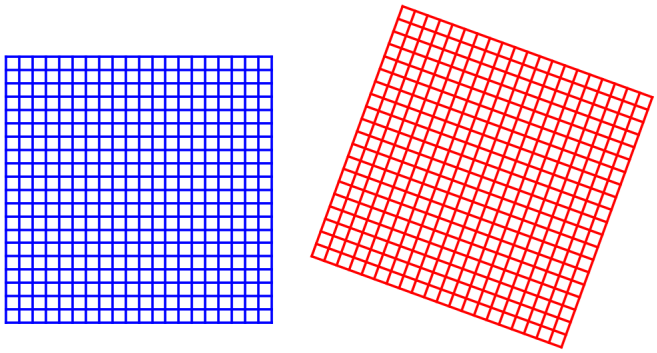
$$\mathbf{R} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\gamma) & -\sin(\gamma) \\ 0 & \sin(\gamma) & \cos(\gamma) \end{pmatrix} \cdot \begin{pmatrix} \cos(\beta) & 0 & -\sin(\beta) \\ 0 & 1 & 0 \\ \sin(\beta) & 0 & \cos(\beta) \end{pmatrix} \cdot \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{pmatrix},$$

where α, β and γ are three rotation angles; the parameters of the model are then given by $\mathbf{w} = (\alpha, \beta, \gamma, t_1, t_2, t_3)^{\mathrm{T}}$.

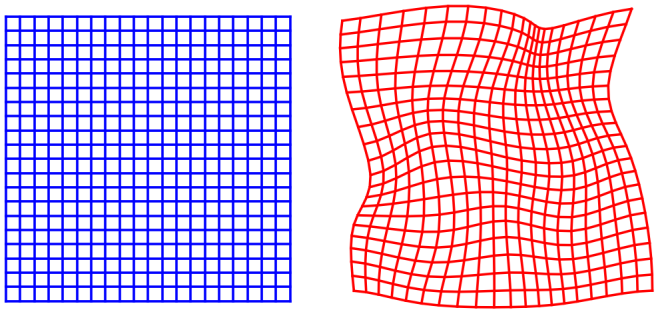
Provided the affine matrix $\mathbf{M}$ is constructed by using $\mathbf{A} = \mathbf{R}$, (2.6) remains valid to map voxel coordinates in the fixed image into voxel coordinates in the moving image.



(a) Affine transformation



(b) Rigid transformation



(c) Nonlinear transformation

Figure 2.2: Illustration of affine, rigid and nonlinear transformation models. The red grid shows how the regular grid shown in blue is characteristically mapped under each transformation model.

2.2.2 Nonlinear transformations

In situations where tissue deformation needs to be modeled, the linear function (2.5) (one for each dimension d) in affine registration is generalized to a nonlinear one. Since global differences in overall size and orientation have typically already been removed using a preceding affine registration[1], only the *residual deformation* δ is modeled:

$$y_d(\mathbf{x}, \mathbf{w}_d) = x_d + \delta(\mathbf{x}, \mathbf{w}_d), \quad \text{where} \quad \delta(\mathbf{x}, \mathbf{w}_d) = \sum_{m=0}^{M-1} w_{d,m}\phi_m(\mathbf{x}). \tag{2.7}$$

Here $\phi_m(\mathbf{x})$ are M basis functions, which are typically taken to be *seperable* (cf. (1.11) and Fig. 1.5), with weights $\mathbf{w}_d = (w_{d,0}, \dots w_{d,M-1})^{\mathrm{T}}$. The interpretation of these weights is that, when they are all set to zero (i.e., when $\mathbf{w}_d = \mathbf{0}$), the deformation δ is also zero and no deformation is applied. An illustration of a nonlinear transformation encoded this way is provided in Fig. 2.2c.

It is worth reiterating that the motion of points along each individual dimension d in the moving image is governed by its own set of parameters $\mathbf{w}_d$. In 2D, there will therefore by two sets of parameters: $\mathbf{w} = (\mathbf{w}_1, \mathbf{w}_2)^{\mathrm{T}}$, whereas in 3D there will be three: $\mathbf{w} = (\mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3)^{\mathrm{T}}$.

2.3 Landmark-based registration

One way to register two images is by manually annotating corresponding points in both images, and then finding a spatial transformation that brings matching point pairs close to each other. Letting $\{\mathbf{x}_n\}_{n=1}^N$ denote a set of N point locations annotated in the fixed image, and $\{\mathbf{y}_n\}_{n=1}^N$ the corresponding locations in the moving image, registration can be obtained by minimizing the energy

$$E(\mathbf{w}) = \sum_{n=1}^{N} \|\mathbf{y}_n - \mathbf{y}(\mathbf{x}_n, \mathbf{w})\|^2$$

with respect to the transformation parameters $\mathbf{w}$. Here $\|\mathbf{a} - \mathbf{b}\|^2 = \sum_{d=1}^{D}(a_1 - b_1)^2$ measures the squared Eucledian distance between two points $\mathbf{a}$ and $\mathbf{b}$.

Affine transformation: Optimizing $E(\mathbf{w})$ with respect to $\mathbf{w}$ will be particularly straightforward when the mapping in each dimension d is governed by its own set of parameters $\mathbf{w}_d$, as is the case in both the affine and nonlinear formulation of (2.5) and (2.7). This is because the energy can then be split into

[1] For instance by replacing the affine voxel-to-world mapping of the fixed image $\mathbf{M}_F$ by $\mathbf{M} \cdot \mathbf{M}_F$

a sum of D independent energies, one for each dimension:

$$\begin{aligned} E(\mathbf{w}) &= \sum_{n=1}^{N}\sum_{d=1}^{D}\left[y_{n,d} - y_d(\mathbf{x}_n, \mathbf{w_d})\right]^2 \\ &= \sum_{d=1}^{D} E_d(\mathbf{w}_d) \quad \text{with} \quad E_d(\mathbf{w}_d) = \sum_{n=1}^{N}\left[y_{n,d} - y_d(\mathbf{x}_n, \mathbf{w_d})\right]^2, \end{aligned}$$

where $y_{n,d}$ denotes the d-th element of $\mathbf{y}_n$. Optimizing each $E_d(\mathbf{w}_d)$ with respect to $\mathbf{w}_d$ is easy: in both the affine and the nonlinear case, $\mathbf{y}_d(\mathbf{x}_n, \mathbf{w_d})$ is only *linearly* dependent on $\mathbf{w}_d$, reducing the problem to the form of linear regression analyzed in Sec. 1.1. Taking affine registration as an example, the energy for the d-th dimension is given by (cf. (2.5))

$$E_d(\mathbf{w}_d) = \sum_{n=1}^{N}\left[y_{n,d} - t_d - a_{d,1}x_{n,1} - \ldots - a_{d,D}x_{n,D}\right]^2,$$

which is minimized at solution

$$\begin{pmatrix} t_d \\ a_{d,1} \\ \vdots \\ a_{d,D} \end{pmatrix} = \left(\mathbf{X}^{\mathrm{T}}\mathbf{X}\right)^{-1}\mathbf{X}^{\mathrm{T}}\begin{pmatrix} y_{1,d} \\ \vdots \\ y_{N,d} \end{pmatrix}, \tag{2.8}$$

where

$$\mathbf{X} = \begin{pmatrix} 1 & x_{1,1} & \cdots & x_{1,D} \\ 1 & x_{2,1} & \cdots & x_{2,D} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{N,1} & \cdots & x_{N,D} \end{pmatrix}.$$

The d-th row of the affine transformation parameters $\mathbf{A}$ and $\mathbf{t}$ is therefore given by (2.8). Doing this for all D dimensions yields all the required parameter values.

Rigid transformation: Optimizing $E(\mathbf{w})$ with respect to $\mathbf{w}$ is more involved when a rigid transformation model is used, since the constraints that $\mathbf{R}^{\mathrm{T}}\mathbf{R} = \mathbf{I}$ and $\det(\mathbf{R}) = 1$ prevent us from decoupling the problem across dimensions. For the translation vector $\mathbf{t}$ this is not yet an issue, and we can therefore use the same approach as before to deduce that, for a given rotation $\mathbf{R}$, the energy

$$E(\mathbf{w}) = \sum_{n=1}^{N}\|\mathbf{y}_n - \mathbf{R}\mathbf{x}_n - \mathbf{t}\|^2 \tag{2.9}$$

is minimized when

$$\mathbf{t} = \bar{\mathbf{y}} - \mathbf{R}\bar{\mathbf{x}}, \tag{2.10}$$

where $\bar{\mathbf{y}} = \frac{1}{N}\sum_{n=1}^{N}\mathbf{y}_n$ and $\bar{\mathbf{x}} = \frac{1}{N}\sum_{n=1}^{N}\mathbf{x}_n$. Plugging this result into (2.9), we can reformulate the energy as

$$E(\mathbf{w}) = \sum_{n=1}^{N} \|\tilde{\mathbf{y}}_n - \mathbf{R}\tilde{\mathbf{x}}_n\|^2 \quad \text{where} \quad \tilde{\mathbf{y}}_n = \mathbf{y}_n - \bar{\mathbf{y}} \quad \text{and} \quad \tilde{\mathbf{x}}_n = \mathbf{x}_n - \bar{\mathbf{x}}.$$

Under the constraint $\mathbf{R}^{\mathrm{T}}\mathbf{R} = \mathbf{I}$, this is minimized when [2]

$$\mathbf{R} = \mathbf{V}\mathbf{U}^{\mathrm{T}},$$

where $\mathbf{U}$ and $\mathbf{V}$ are $D \times D$ matrices such that

$$\mathbf{U}^{\mathrm{T}}\mathbf{U} = \mathbf{I}, \quad \mathbf{V}^{\mathrm{T}}\mathbf{V} = \mathbf{I} \quad \text{and} \quad \sum_{n=1}^{N}\mathbf{x}_n\mathbf{y}_n^{\mathrm{T}} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\mathrm{T}},$$

where $\mathbf{\Sigma}$ is diagonal. One such a solution is obtained by computing the singular value decomposition (SVD) of the matrix $\sum_{n=1}^{N}\mathbf{x}_n\mathbf{y}_n^{\mathrm{T}}$. However, this solution does not necessarily satisfy the second constraint of rotational matrices that $\det(\mathbf{R}) = 1$. It is also possible[2] that $\det(\mathbf{R}) = -1$, in which case a valid rotation can be obtained by "flipping" one of the columns of $\mathbf{R}$ by reversing the sign of all the elements in it.

Once $\mathbf{R}$ has been found, (2.10) can be used to find the remaining parameters $\mathbf{t}$.

2.4 Intensity-based registration

Landmark-based registration suffers from a number of shortcomings, including the need for manually annotating images and the fact that registrations are computed from only a handful of points. This limits both the efficiency and the accuracy with which images can be aligned.

Another class of algorithms can perform registrations fully automatically, by minimizing energies that are computed directly from raw image intensities. Below we review two well-known methods in this family.

2.4.1 Sum of squared differences

When the fixed and the moving image both depict the same anatomical structures with the same (or very similar) intensity characteristics, a successful registration will be characterized by small differences in absolute intensity values at corresponding locations. Scenarios where this idea can be used include co-registering two CT scans, where intensity values have a direct physical interpretation, or two MR images acquired with similar pulse sequences on similar hardware (e.g., T1-weighted images acquired at different time points or from different subjects on the same scanner). In the latter case, images will often still

[2] Since $\det(\mathbf{AB}) = \det(\mathbf{A})\det(\mathbf{B})$, we have that $\det(R) = \det(U)\det(V)$. Furthermore, $\det(\mathbf{U}) \pm 1$ and $\det(\mathbf{V}) \pm 1$ since $\mathbf{U}^{\mathrm{T}}\mathbf{U} = \mathbf{I}$ and $\mathbf{V}^{\mathrm{T}}\mathbf{V} = \mathbf{I}$.

need to be pre-processed to make their intensities comparable (for instance by intensity rescaling), since MR scanners do not typically provide measurements in quantitative physical units.

On a technical level, registration can be obtained by minimizing the sum of squared differences in intensities at corresponding locations:

$$E(\mathbf{w}) = \sum_{n=1}^{N} \left[\mathcal{F}(\mathbf{x}_n) - \mathcal{M}(\mathbf{y}(\mathbf{x}_n, \mathbf{w}))\right]^2 . \tag{2.11}$$

Here, $\mathbf{x}_n$ are the world coordinates of the voxels in the *fixed* image, and $\mathcal{F}(\mathbf{x}_n)$ denotes the image intensity of those voxels. Similary, $\mathcal{M}(\mathbf{y}(\mathbf{x}_n, \mathbf{w}))$ denotes the image intensities in the *moving* image, evaluated at the mapped locations $\mathbf{y}(\mathbf{x}_n, \mathbf{w})$. Since these will generally fall in between original voxel locations in the moving image, image interpolation will be required (cf. Sec. 1.3.3). When the mapped location falls *outside* of the image area of the moving image, a constant intensity will typically be assigned (for instance zero).

Gauss-Newton optimization

Unlike in landmark-based registration, finding parameter values $\mathbf{w}$ that minimize the energy (2.11) is no longer given by closed-form solutions, and numerical optimization methods need to be used. When the number of parameters of the transformation model is low, for instance in rigid or affine registration, generic optimization algorithms that only evaluate $E(\mathbf{w})$ (and perhaps its gradient $\nabla E(\mathbf{w})$) will work quite well. For cases with many degrees of freedom, such as flexible nonlinear deformations, a dedicated optimization "trick" can be used that exploits the specific structure of the energy (2.11) to obtain faster solutions.

The method is known as *Gauss-Newton*, and involves linearizing $\mathcal{M}(\mathbf{y}(\mathbf{x}_n, \mathbf{w}))$ with respect to $\mathbf{w}$ around the current parameter values. Specifically, for the nonlinear transformation model of (2.7), where deformations are encoded in each dimension d separately, the partial derivative of $\mathcal{M}(\mathbf{y}(\mathbf{x}_n, \mathbf{w}))$ with respect to $w_{d,m}$ is given by (chain rule):

$$\frac{\partial \mathcal{M}(\mathbf{y}(\mathbf{x}_n, \mathbf{w}))}{\partial w_{d,m}} = \frac{\partial \mathcal{M}(\mathbf{y}(\mathbf{x}_n, \mathbf{w}))}{\partial y_d} \; \frac{\partial y_d(\mathbf{x}_n, \mathbf{w}_d)}{\partial w_{d,m}} = g_{d,n} \; \phi_m(\mathbf{x}_n). \tag{2.12}$$

Here the fact that $\frac{\partial y_d(\mathbf{x}_n, \mathbf{w}_d)}{\partial w_{d,m}} = \phi_m(\mathbf{x}_n)$ was used, and the notation $g_{d,n} = \frac{\partial \mathcal{M}(\mathbf{y}(\mathbf{x}_n, \mathbf{w}))}{\partial y_d}$ was introduced as shorthand for the partial spatial derivative of the moving image in the d-th dimension, evaluated at position $\mathbf{y}(\mathbf{x}_n, \mathbf{w})$. By using a cubic B-spline interpolation model, such spatial derivatives can be conveniently computed everywhere [3].

For a small deviation $\boldsymbol{\epsilon}$ from some parameter values $\mathbf{w}$, (2.12) can now be used in a first-order Taylor expansion to obtain the approximation

$$\mathcal{M}(\mathbf{y}(\mathbf{x}_n, \mathbf{w} + \boldsymbol{\epsilon})) \simeq \mathcal{M}(\mathbf{y}(\mathbf{x}_n, \mathbf{w})) + \sum_{d=1}^{D} \sum_{m=0}^{M} \big(g_{d,n} \phi_m(\mathbf{x}_n)\big) \epsilon_{d,m}. \tag{2.13}$$

Plugging this into the energy (2.11), and defining

$$\tau_n = \mathcal{F}(\mathbf{x}_n) - \mathcal{M}(\mathbf{y}(\mathbf{x}_n, \mathbf{w}))$$

we obtain

$$E(\mathbf{w} + \boldsymbol{\epsilon}) = \sum_{n=1}^{N} \left[\tau_n - \sum_{d=1}^{D} \sum_{m=0}^{M} \big(g_{d,n}\phi_m(\mathbf{x}_n)\big)\epsilon_{d,m} \right]^2,$$

which can be recognized has having the form of a linear regression problem with parameters $\boldsymbol{\epsilon}$. The value of $\boldsymbol{\epsilon}$ minimizing $E(\mathbf{w} + \boldsymbol{\epsilon})$ is therefore given by (cf. Sec. 1.1)

$$\boldsymbol{\epsilon} = \left(\boldsymbol{\Psi}^{\mathrm{T}}\boldsymbol{\Psi}\right)^{-1} \boldsymbol{\Psi}^{\mathrm{T}}\boldsymbol{\tau} \tag{2.14}$$

where $\boldsymbol{\tau} = (\tau_1, \ldots, \tau_N)^{\mathrm{T}}$ and

$$\boldsymbol{\Psi} = \left(\begin{array}{c|c|c} \mathbf{G}_1\boldsymbol{\Phi} & \cdots & \mathbf{G}_D\boldsymbol{\Phi} \end{array} \right),$$

where $\mathbf{G}_d = \mathrm{diag}(g_{d,n}, \ldots, g_{d,N})$ and $\boldsymbol{\Phi}$ is given by (1.2).

This result suggests the following simple iterative algorithm for optimizing $E(\mathbf{w})$:

1. Select a starting value for the transformation parameters, e.g., $\mathbf{w} = \mathbf{0}$ (no deformation);
2. Use (2.14) to compute a small update $\boldsymbol{\epsilon}$ to the current parameters $\mathbf{w}$;
3. Update the parameters: $\mathbf{w} \leftarrow \mathbf{w} + \boldsymbol{\epsilon}$;
4. Repeat steps 2. and 3. until convergence is detected.

There is one catch, though: In the derivation above, we have assumed that $\boldsymbol{\epsilon}$ is "small". What happens if we compute $\boldsymbol{\epsilon}$ and one or more of its components are actually quite large? Then our approximation (2.13) will have been a poor one, and we might find that the energy $E(\mathbf{w})$ *increases* (instead of *decreasing*) after we use $\boldsymbol{\epsilon}$ to update $\mathbf{w}$. To avoid such situations, several methods to modify $\boldsymbol{\epsilon}$ exist. One such a modification is the *Levenberg-Marquardt* algorithm, which replaces (2.14) by

$$\boldsymbol{\epsilon} = \left(\boldsymbol{\Psi}^{\mathrm{T}}\boldsymbol{\Psi} + \lambda\mathbf{I}\right)^{-1} \boldsymbol{\Psi}^{\mathrm{T}}\boldsymbol{\tau}, \tag{2.15}$$

where $\lambda \geq 0$ is a tunable parameter that is typically updated after each iteration. If the parameter update $\boldsymbol{\epsilon}$ does not decrease $E(\mathbf{w})$, the update is rejected and λ is increased until a decrease in $E(\mathbf{w})$ is obtained; this will happen eventually because for very large λ the algorithm devolves into a gradient-descent algorithm with a small step size. Conversely, if $E(\mathbf{w})$ decreases a smaller λ is used in the next iteration of the algorithm to improve efficiency.

2.4.2 Mutual Information

When two images need to be aligned that were acquired with two different imaging modalities (e.g., CT vs. MR, or MR vs. PET), the intensity characteristics of most anatomical structures will generally be quite diferent between the two images. This precludes the use of the sum-of-squared-differences energy as a registration criterion.

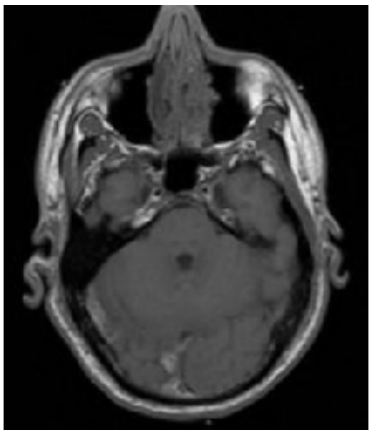
In such situations, fully automatic registrations can still be obtained by optimizing the *Mutual Information* (MI) between the two images [4, 5]. For given transformation parameter values $\mathbf{w}$, the frequency with which specific *discretized intensity pairs* (f, m) occur is analyzed. Here, $f \in \{1, \ldots, B\}$ and $m \in \{1, \ldots, B\}$ denote intensities in the fixed and moving image, respectively, after both images have been preprocessed to only contain B discrete intensity levels[3]. More specifically, a *joint histogram*

$$\mathbf{H} = \begin{pmatrix} h_{1,1} & \ldots & h_{1,B} \\ \vdots & \ddots & \vdots \\ h_{B,1} & \ldots & h_{B,B} \end{pmatrix}$$

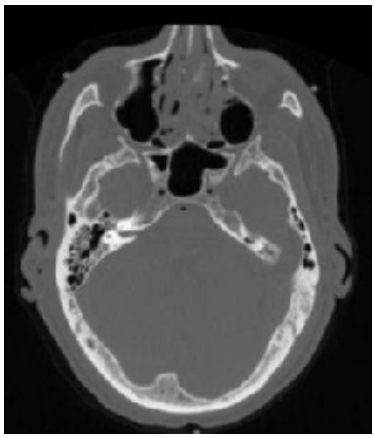
is computed, where entry $h_{f,m}$ contains the number of times the intensity in the fixed image was f, while the corresponding intensity in the moving image was m. In practice, this is done by starting with an empty joint histogram ($\mathbf{H} = \mathbf{0}$), and filling it up by looping over all N voxels in the fixed image. For each such voxel n, its intensity $\mathcal{F}(\mathbf{x}_n)$ as well as the corresponding intensity in the moving image $\mathcal{M}(\mathbf{y}(\mathbf{x}_n, \mathbf{w}))$ is determined. The former will directly give us a value for f, but deciding on m is more involved since $\mathbf{y}(\mathbf{x}_n, \mathbf{w})$ will typically fall somewhere between the voxels in the moving image. An easy solution is to use nearest-neighbor interpolation, so that m is simply the intensity of the voxel that is closest to $\mathbf{y}(\mathbf{x}_n, \mathbf{w})$; however in practice more involved schemes are typically used. A related issue is what to do if $\mathbf{y}(\mathbf{x}_n, \mathbf{w})$ maps to a location outside of the image area of the moving image. In many implementations such voxels are simply skipped (i.e., they do not contribute to the joint histogram). For most voxels, though, a valid intensity pair (f, m) is obtained; the corresponding joint histogram count $h_{f,m}$ is then increased by one, and the next voxel in the fixed image is visited. An example of a joint histogram computed this way is shown in Fig. 2.3e.

A key insight for registration purposes is that the joint histogram will typically have many entries with small counts when the two images are well aligned: most of the encountered intensity combinations (f, m) will be concentrated in just a few histogram bins. In constrast, when the images are moved out of alignment, intensity pairs will become more variable, "smearing out" the counts from high-count histogram entries into (what were previously) low-count bins. This process is illustrated in Fig. 2.4. This phenomenon can be exploited to

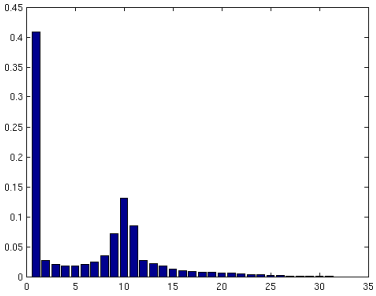
[3] Typically this is done by dividing the intensity range in each image into B contiguous intervals, and recording the interval ("bin") number of each voxel's intensity. Often-used values for B are 32, 64 or 128.



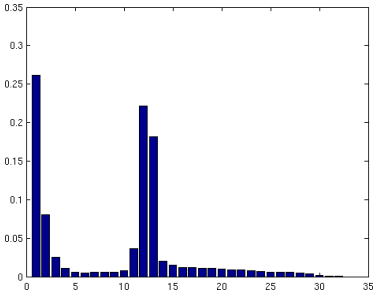
(a) MR



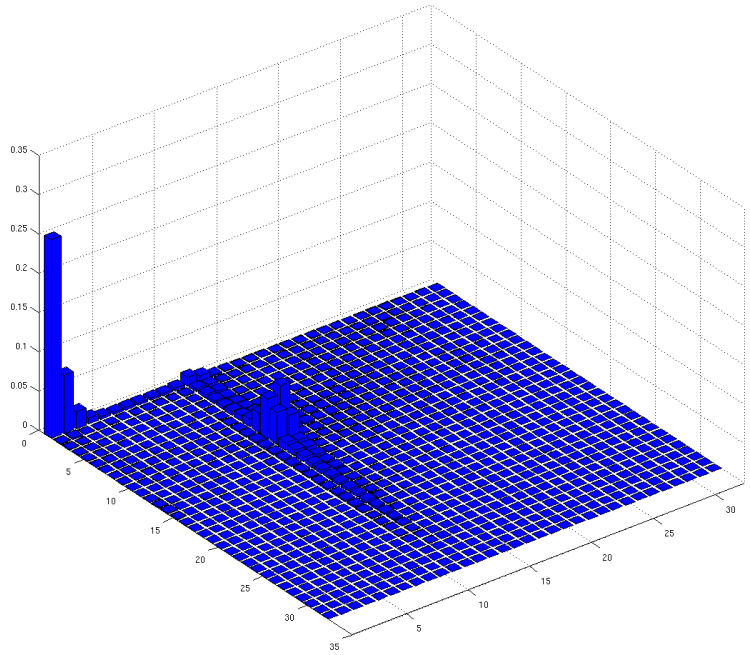
(b) CT



(c) Marginal histogram (MR)



(d) Marginal histogram (CT)



(e) Joint histogram

Figure 2.3: An MR and CT image in perfect alignment, along with their normalized (i.e., divided by N) joint and marginal histograms. The number of histogram bins was $B = 32$.

define the following registration energy:

$$E(\mathbf{w}) = H_{F,M} \quad \text{with} \quad H_{F,M} = -\sum_{f=1}^{B}\sum_{m=1}^{B} p_{f,m}\log(p_{f,m}),$$

where $p_{f,m} = h_{f,m}/N$ are normalized histogram counts. These can be interpreted as *probabilities* of seeing specific intensity combinations (f, m) when the images are aligned with parameter value $\mathbf{w}$. The quantity $H_{F,M}$ is known as the *joint entropy* in information theory. It measures how predictable intensity combinations (f, m) are, and is directly related to data compression: The more predictable the intensity combinations, the lower the joint entropy and the fewer bits will theoretically be needed to store or communicate the fixed-moving image pair. Applied to our registration setting, the joint entropy will be higher when the joint histogram is "smeared out", which will happen when the images are not well aligned. This explains why $H_{F,M}$ can be used as an energy function to drive an automatic registration process.

One problem with using the joint entropy is that it attains low values not just when registrations are good, but also when some non-desirable trivial solutions are found. As an example, consider the scenario where $\mathbf{w}$ is so that the fixed image and the moving image only overlap in a region in the background. In most implementations, the joint histogram is only computed from this overlapping region, as explained before, and therefore the only intensity pair with a non-zero count will be the one where both f and m are zero. The joint entropy for this solution will be zero, which is its lowest possible value.

In order to avoid such pathologoical solutions, it is customary to alter the energy as follows:

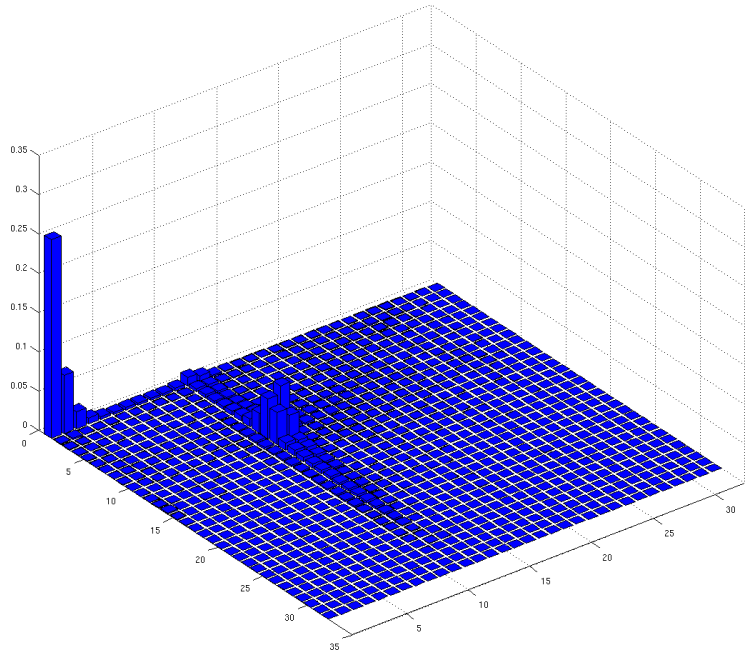
$$E(\mathbf{w}) = H_{F,M} - H_F - H_M, \tag{2.16}$$

where H_F and H_M are the *marginal entropies* of the fixed and moving image, respectively. They are defined as follows:

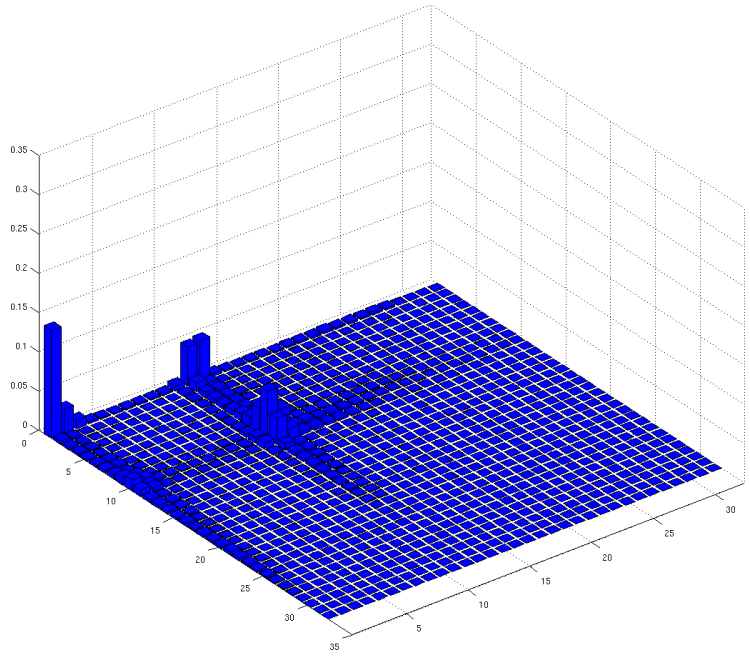
$$H_F = -\sum_{f=1}^{B} p_f \log(p_f) \quad \text{and} \quad H_F = -\sum_{f=1}^{B} p_m \log(p_m),$$

where $p_f = \sum_{m=1}^{B} p_{f,m}$ denotes the probability of encountering a voxel with intensity f in the fixed image, and $p_m = \sum_{f=1}^{B} p_{f,m}$ the corresponding probability of encountering intensity m in the moving image. Adding these two terms to the energy function will steer the registration towards overlapping in areas that are "interesting" in both images, i.e., areas with actual content (with high entropy).

Since the quantity $H_F + H_M - H_{F,M}$ is known in information theory as *mutual information*, using its negative as energy function in (2.16) for registration purposes is known as *mutual information-based registration*. In a practical implementation, numerical optimization will need to be used to find values $\mathbf{w}$ with low energy. Currently these are mostly general-purpose optimizers that only need to be able to evaluate $E(\mathbf{w})$ and its gradient $\nabla E(\mathbf{w})$.



(a)



(b)

Figure 2.4: Normalized joint histogram of the MR and CT scans shown in Fig. 2.3, both when the images are in perfect alignment (a) and when the CT image is translated with respect to the MR image (b).

Chapter 3

Model-based Segmentation

The ability to efficiently delineate anatomical structures from medical images is important in many applications. Examples include measuring the number and volume of lesions and how they change over time, planning radiation therapy treatments or surgical interventions, and characterizing shape changes that occur in specific patient groups. The process of delineating structures is called *image segmentation*, and automating it has traditionally been approached using model-based techniques. This chapter reviews some well-established techniques in this category.

3.1 Generative models

Image segmentation methods are often based on so-called *generative* models, i.e., models that describe how images can be generated synthetically by random sampling from some probability distribution. Generative models for medical image segmentation generally consist of two parts:

- A *prior* distribution that makes predictions about where anatomical structures typically occur throughout the image. We will refer to this component of the model as the *labeling model*. Let $\mathbf{l} = (l_1, \ldots, l_N)^{\mathrm{T}}$ be a (vectorized) label image with a total of N voxels, with $l_n \in \{1, \ldots, K\}$ denoting the one of K possible labels assigned to voxel n, indicating which of the K anatomical structures the voxel belongs to. The labeling model is then specified by some probability distribution $p(\mathbf{l}|\boldsymbol{\theta}_l)$ that typically depends on a set of parameters $\boldsymbol{\theta}_l$.

- A *likelihood* function that predicts how any given label image, where each voxel is assigned a unique anatomical label, translates into an image where each voxel has an intensity. Because this really is a (often very simplistic) model of how a medical imaging device generates images from known anatomy, we will refer to this component of the model as the *imaging*

model. Given a label image $\mathbf{l}$, the imaging model generates a corresponding intensity image $\mathbf{d} = (d_1, \ldots, d_N)^{\mathrm{T}}$ by randomly sampling from some probability distribution $p(\mathbf{d}|\mathbf{l}, \boldsymbol{\theta}_d)$ with parameters $\boldsymbol{\theta}_d$.

In summary, the generative model is fully specified by two parametric distributions $p(\mathbf{l}|\boldsymbol{\theta}_l)$ and $p(\mathbf{d}|\mathbf{l}, \boldsymbol{\theta}_d)$, which depend on parameters $\boldsymbol{\theta} = (\boldsymbol{\theta}_l^{\mathrm{T}}, \boldsymbol{\theta}_d^{\mathrm{T}})^{\mathrm{T}}$ that are either assumed to be known in advance, or more frequently, need to be estimated from the image data itself. The exact form of the used distributions depends on the segmentation problem at hand. In general, the more realistic the models, the better the segmentations that can be obtained with them.

Once the exact generative model has been chosen and appropriate values $\hat{\boldsymbol{\theta}}$ for its parameters are known, properties of the underlying segmentation of an image can be inferred by inspecting the posterior probability distribution $p(\mathbf{l}|\mathbf{d}, \hat{\boldsymbol{\theta}})$. Using Bayes' rule, this distribution is given by

$$p(\mathbf{l}|\mathbf{d}, \hat{\boldsymbol{\theta}}) = \frac{p(\mathbf{d}|\mathbf{l}, \hat{\boldsymbol{\theta}}_d) p(\mathbf{l}|\hat{\boldsymbol{\theta}}_l)}{p(\mathbf{d}|\hat{\boldsymbol{\theta}})}, \tag{3.1}$$

with $p(\mathbf{d}|\hat{\boldsymbol{\theta}}) = \sum_{\mathbf{l}} p(\mathbf{d}|\mathbf{l}, \hat{\boldsymbol{\theta}}_d) p(\mathbf{l}|\hat{\boldsymbol{\theta}}_l)$[1]. For instance, one might look for the segmentation $\hat{\mathbf{l}}$ that has the maximum a posteriori (MAP) probability:

$$\hat{\mathbf{l}} = \arg\max_{\mathbf{l}} p(\mathbf{l}|\mathbf{d}, \hat{\boldsymbol{\theta}}), \tag{3.2}$$

or estimate the volume of the anatomical structure corresponding to label k by assessing it's expected value

$$\sum_{\mathbf{l}} V_k(\mathbf{l}) p(\mathbf{l}|\mathbf{d}, \hat{\boldsymbol{\theta}}) \tag{3.3}$$

where $V_k(\mathbf{l})$ counts the number of voxels that have label k in $\mathbf{l}$.

3.2 Gaussian mixture model

A very simple generative model that is nevertheless quite useful in practice, is the so-called *Gaussian mixture model*. In this model, the segmentation prior is of the form

$$p(\mathbf{l}|\boldsymbol{\theta}_l) = \prod_{n=1}^{N} p(l_n|\boldsymbol{\theta}_l) \tag{3.4}$$

$$= \prod_{n=1}^{N} \pi_{l_n} \tag{3.5}$$

[1] In practice, one seldom needs to explicitly calculate the denominator $p(\mathbf{d}|\hat{\boldsymbol{\theta}})$ because it doesn't involve $\mathbf{l}$, and one simply compares alternate segmentations by evaluating $p(\mathbf{d}|\mathbf{l}, \hat{\boldsymbol{\theta}}_d) p(\mathbf{l}|\hat{\boldsymbol{\theta}}_l)$ instead.

where the parameters $\boldsymbol{\theta}_l = (\pi_1, \ldots, \pi_K)^{\mathrm{T}}$ consist of a set of probabilities π_k satisfying $\pi_k \geq 0, \forall k$ and $\sum_{k=1}^{K} \pi_k = 1$. In other words, this model assumes that the labels are assigned to the voxels independently from one another, i.e., the probability that a certain label occurs in a particular voxel is unaffected by the labels assigned to other voxels ((3.4)), and each label occurs, on average, with a relative frequency of π_k ((3.5)).

For the likelihood function, it is assumed that the intensity in each voxel only depends on the label in that voxel and not on that in other voxels:

$$p(\mathbf{d}|\mathbf{l}, \boldsymbol{\theta}_d) = \prod_{n=1}^{N} p(d_n|l_n, \boldsymbol{\theta}_d), \tag{3.6}$$

and that the intensity distribution associated with each label k is Gaussian with mean μ_k and variance σ_k^2:

$$p(d_n|l_n, \boldsymbol{\theta}_d) = \mathcal{N}(d_n|\mu_{l_n}, \sigma_{l_n}^2), \tag{3.7}$$

where

$$\mathcal{N}(d|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\big[-\frac{(d-\mu)^2}{2\sigma^2}\big] \tag{3.8}$$

and $\boldsymbol{\theta}_d = (\mu_1, \ldots, \mu_K, \sigma_1^2, \ldots \sigma_K^2)^{\mathrm{T}}$.

It is instructive to write down the probability with which this model generates a given image $\mathbf{d}$:

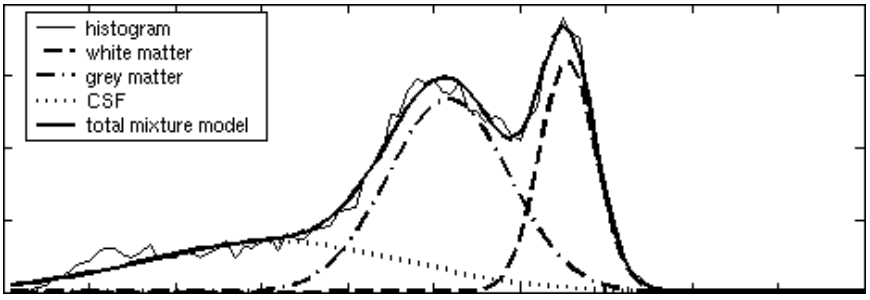
$$\begin{aligned} p(\mathbf{d}|\boldsymbol{\theta}) &= \sum_{\mathbf{l}} p(\mathbf{d}|\mathbf{l}, \boldsymbol{\theta}_d) p(\mathbf{l}|\boldsymbol{\theta}_l) \\ &= \sum_{\mathbf{l}} \left[\prod_{n=1}^{N} \mathcal{N}(d_n|\mu_{l_n}, \sigma_{l_n}^2) \prod_{n=1}^{N} \pi_{l_n} \right] \qquad (3.9) \\ &= \prod_{n=1}^{N} p(d_n|\boldsymbol{\theta}) \qquad (3.10) \end{aligned}$$

with

$$p(d|\boldsymbol{\theta}) = \sum_{k=1}^{K} \mathcal{N}(d|\mu_k, \sigma_k^2)\pi_k. \tag{3.11}$$

(Although the transition from (3.9) to (3.10) may appear non-trivial, it is merely algebra and can easily be understood by considering that first the label and then the intensity is drawn *independently* in each individual voxel, hence the *product* over all voxels in (3.10).) (3.11) explains why this model is called the Gaussian mixture model: the intensity distribution in any voxel, independent of its spatial location, is given by the same linear superposition of Gaussians. Since no spatial information is encoded in the model, it can directly be visualized as a way to approximate the histogram, as shown in Fig. 3.1.

(a)



(b)

Figure 3.1: In the Gaussian mixture model, the histogram is described as a linear superposition of Gaussian distributions: (a) MR scan of the head, after removing all non-brain tissue and other pre-processing steps; and (b) corresponding histogram and its representation as a sum of Gaussians.

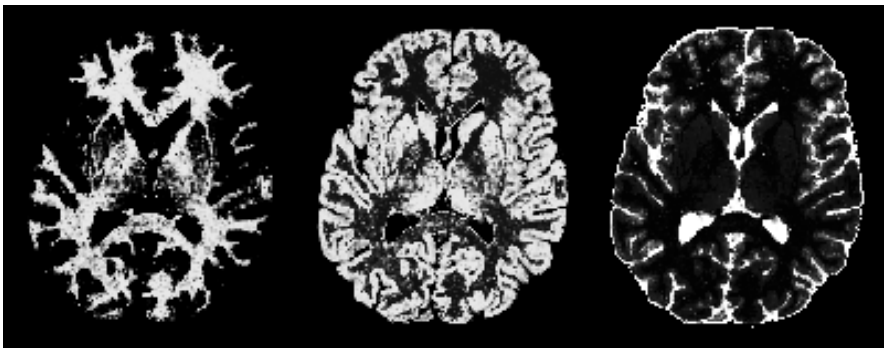


Figure 3.2: Visualization of the segmentation posterior corresponding to the data and model of Fig. 3.1. High intensities correspond to high probabilities and vice versa.

Because of the assumption of statistical independence between voxels, the segmentation posterior (3.1) reduces to a simple form that is factorized (i.e., appears as a product) over the voxels:

$$\begin{aligned} p(\mathbf{l}|\mathbf{d},\hat{\boldsymbol{\theta}}) &= \frac{p(\mathbf{d}|\mathbf{l},\hat{\boldsymbol{\theta}}_d)p(\mathbf{l}|\hat{\boldsymbol{\theta}}_l)}{p(\mathbf{d}|\hat{\boldsymbol{\theta}})} \\ &= \frac{\prod_{n=1}^{N}\mathcal{N}(d_n|\hat{\mu}_{l_n},\hat{\sigma}^2_{l_n})\prod_{n=1}^{N}\hat{\pi}_{l_n}}{\prod_{n=1}^{N}\sum_{k=1}^{K}\mathcal{N}(d_n|\hat{\mu}_k,\hat{\sigma}^2_k)\hat{\pi}_k} \\ &= \prod_{n=1}^{N} p(l_n|d_n,\hat{\boldsymbol{\theta}}), \end{aligned} \tag{3.12}$$

where

$$p(l_n|d_n,\hat{\boldsymbol{\theta}}) = \frac{\mathcal{N}(d_n|\hat{\mu}_{l_n},\hat{\sigma}^2_{l_n})\hat{\pi}_{l_n}}{\sum_{k=1}^{K}\mathcal{N}(d_n|\hat{\mu}_k,\hat{\sigma}^2_k)\hat{\pi}_k}. \tag{3.13}$$

Therefore, the segmentation posterior is fully specified by each voxel's K posterior probabilities of belonging to each structure; such segmentation posteriors can be visualized as images where high and low intensities correspond to high and low probabilities, respectively. The segmentation corresponding to the image and Gaussian mixture model of Fig. 3.1 is visualized in 3.2 this way. Evidently, the sum of all the structures' posterior probabilities add to one in each voxel: $\sum_{k=1}^{K} p(k|d_n,\hat{\boldsymbol{\theta}}) = 1, \forall n$.

Because of the factorized form of the segmentation posterior, the MAP segmentation (3.2) is simply given by

$$\hat{\mathbf{l}} = \arg\max_{\mathbf{l}} p(\mathbf{l}|\mathbf{d},\hat{\boldsymbol{\theta}}) = \arg\max_{l_1,\ldots,l_N} \prod_{n=1}^{N} p(l_n|d_n,\hat{\boldsymbol{\theta}}), \tag{3.14}$$

i.e., each voxel is assigned exclusively to the label with the highest posterior

probability. Similarly, the expected volume of the anatomical structure corresponding to label k is given by (3.3)

$$\sum_{\mathbf{l}} V_k(\mathbf{l}) p(\mathbf{l}|\mathbf{d}, \hat{\boldsymbol{\theta}}) = \sum_{n=1}^{N} p(k|d_n, \hat{\boldsymbol{\theta}}), \tag{3.15}$$

i.e., a "soft" count of voxels belonging to the structure, where voxels contribute according to their posterior probability of belonging to that structure.

3.3 Markov random field priors

It is worth emphasizing that in the Gaussian mixture model, a voxel's posterior probability of belonging to each of the K structures is computed using only the local intensity of the voxel itself ((3.13)). Although this works quite well in some applications, there is often an intensity overlap between the to-be-segmented structures, causing severe segmentation errors in such a purely intensity-driven strategy. An example of this is shown in Fig. 3.3, where a simple Gaussian mixture model with $K = 2$ classes was used to segment a brain MR scan into tumor tissue and other structures. The parameters $\hat{\boldsymbol{\theta}}$ were obtained by manually clicking on a set of representative voxels within the tumor, recording their intensities, and computing their mean and variance as estimates of $\hat{\mu}_1$ and $\hat{\sigma}_1^2$, respectively; the mean and variance $\{\hat{\mu}_2\, \hat{\sigma}_2^2\}$ for the "other" class was simply the mean and variance of all the image's voxels' intensities; and the tumor was estimated to cover approximately one tenth of the image area, so that we used $\hat{\pi}_1 = 0.1$ and $\hat{\pi}_2 = 0.9$. It can be seen from the figure that while this model generally captures the tumor quite well, many small image areas outside of the tumor also have a high posterior probability of belonging to the tumor class, limiting the usefulness of the results.

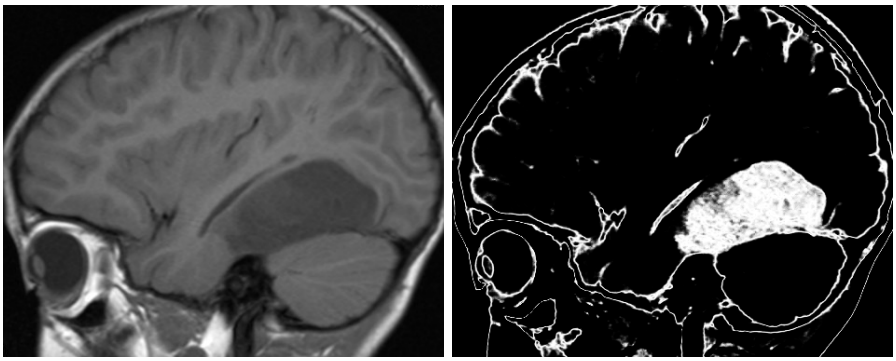
In order to avoid this type of segmentation errors, we need to use more advanced models for the prior distribution $p(\mathbf{l}|\boldsymbol{\theta}_l)$ that more realistically reflect the shape of the structures we are looking for.

3.3.1 Markov random field model

An often-used improvement to the simplistic prior of (3.5) is to use a prior that explicitly prefers voxels with the same label to be clustered spatially rather than scattered randomly throughout the image area. A computationally attractive way of achieving this is to formulate the prior as

$$p(\mathbf{l}|\boldsymbol{\theta}_l) = \frac{1}{Z(\boldsymbol{\theta}_l)} \exp(-U(\mathbf{l}|\boldsymbol{\theta}_l)), \tag{3.16}$$

where $U(\mathbf{l}|\boldsymbol{\theta}_l)$ is an "energy" functional that is high for undesired configurations of $\mathbf{l}$ and low otherwise, resulting in low prior probabilities of undesired configurations and high probabilities otherwise. $Z(\boldsymbol{\theta}_l) = \sum_{\mathbf{l}} \exp(-U(\mathbf{l}|\boldsymbol{\theta}_l))$ is



(a) (b)

Figure 3.3: The Gaussian mixture model does not encode any spatial information and is therefore susceptible to segmentation errors caused by intensity overlap between the structures of interest: (a) a brain MR scan of a person with a tumor; and (b) the voxels' posterior probability of belonging to the tumor for a 2-component Gaussian mixture model.

a normalizing constant that ensures that $\sum_{\mathbf{l}} p(\mathbf{l}|\boldsymbol{\theta}_l) = 1$ but typically does not need to be computed explicitly in practical situations.

For reasons that will soon become clear, the energy functional is often chosen to be of the form

$$U(\mathbf{l}|\boldsymbol{\theta}_l) = \beta \sum_{(n,n')} \delta(l_n \neq l_{n'}), \tag{3.17}$$

where the sum is running over all the voxel pairs (n, n') that are neighbors in the image grid (for instance, each voxel has six direct neighbors in a 3-D image grid, or 26 if also the diagonal directions are allowed), $\delta(k \neq l)$ equals zero if $k = l$ and one otherwise, and β is a parameter that controls how strongly undesired configurations are penalized. Stated differently, the energy functional is proportional to the number of times two neighboring voxels have a different class assignment in $\mathbf{l}$, thereby encouraging configurations in which the labels are spatially clustered. Prior knowledge that some classes tend to occur more frequently than others can also be incorporated by adding an extra term:

$$U(\mathbf{l}|\boldsymbol{\theta}_l) = \beta \sum_{(n,n')} \delta(l_n \neq l_{n'}) - \sum_{n=1}^{N} \log(\pi_{l_n}). \tag{3.18}$$

The parameters of this model are $\boldsymbol{\theta}_l = (\beta, \pi_1, \ldots, \pi_K)^{\mathrm{T}}$; for $\beta = 0$ (no undesired pair-wise combination penalized) this model reduces to the standard Gaussian mixture prior of (3.5).

The computational attractiveness of the model lies in the fact that, although it defines a global prior that induces statistical dependencies between labels in voxels that are far apart, calculating the conditional distribution in a single voxel

requires only looking up the labels assigned to its neighboring voxels (so-called Markov property). Indeed, using the notation $\mathbf{l}_{\setminus n} = (l_1, \ldots, l_{n-1}, l_{n+1}, \ldots, l_n)^{\mathrm{T}}$ to denote the vector of labels in all voxels except voxel n, and $\mathfrak{N}_n$ the set of voxels that form neighboring pairs with voxel n, we have (Bayes' rule)

$$\begin{aligned} p(l_n|\mathbf{l}_{\setminus n}) &= \frac{p(\mathbf{l})}{p(\mathbf{l}_{\setminus n})} \\ &= \frac{p(\mathbf{l})}{\sum_{l_n} p(\mathbf{l})} \\ &= \frac{\exp(-U(\mathbf{l}|\boldsymbol{\theta}_l))}{\sum_{l_n} \exp(-U(\mathbf{l}|\boldsymbol{\theta}_l))} \\ &= \frac{\exp\big(-\beta \sum_{n' \in \mathfrak{N}_n} \delta(l_n \neq l_{n'}) + \log \pi_{l_n}\big)}{\sum_{k=1}^{K} \exp\big(-\beta \sum_{n' \in \mathfrak{N}_n} \delta(l_{n'} \neq k) + \log \pi_k\big)} \\ &= \frac{\pi_{l_n} \cdot \exp\big(-\beta \sum_{n' \in \mathfrak{N}_n} \delta(l_n \neq l_{n'})\big)}{\sum_{k=1}^{K} \pi_k \cdot \exp\big(-\beta \sum_{n' \in \mathfrak{N}_n} \delta(l_{n'} \neq k)\big)}. \end{aligned} \tag{3.19}$$

The second to last step is explained by the fact that all the remaining terms of (3.18) cancel out in the numerator and the denominator.

Comparing (3.19) to the standard Gaussian mixture prior, the probability of having label k in voxel n is no longer simply π_k, but changes according to the labels assigned to its neighboring voxels. If the majority of neighbors has label k, for instance, the probability of having k in n will be higher than π_k.

3.3.2 Inference using the mean-field approximation

Combining the more advanced Markov random field prior with the same likelihood as before, in which the intensity in each voxel is distributed according to a Gaussian associated with its label ((3.6) and (3.7)), we can in principle evaluate possible segmentations by comparing their posterior $p(\mathbf{l}|\mathbf{d}, \hat{\boldsymbol{\theta}})$. However, handling this posterior explicitly is difficult in practice because it can no longer be written as a simple product of voxel-wise contributions in the same way as before ((3.12)).

There exist fast algorithms, based on so-called graph-cuts, for computing the MAP segmentation $\hat{\mathbf{l}} = \arg\max_{\mathbf{l}} p(\mathbf{l}|\mathbf{d}, \hat{\boldsymbol{\theta}})$ [6]. However, we are sometimes more interested in calculating *expectations*, for instance in order to estimate volumes of anatomical structures ((3.3)) or as part of Expectation-Maximization parameter optimizers (which we will cover in Sec. 3.4). Since this involves summing over all possible configurations of $\mathbf{l}$, it is infeasible to do the necessary computations exactly because there are exponentially many configurations of $\mathbf{l}$: a segmentation problem with just $K = 2$ classes of a standard MR image of size $256 \times 256 \times 128$ voxels yields more than $10^{1000000}$ configurations to sum over[2]!

The solution is to resort to approximation schemes, of which one is a variational method based upon the so-called mean field theory in physics. In this

[2] As a comparison, there are approximately 10^{80} atoms in the universe.

method, we seek a distribution $q(\mathbf{l})$ that we design to be of a more tractable form than $p(\mathbf{l}|\mathbf{d}, \hat{\boldsymbol{\theta}})$, while still approximating $p(\mathbf{l}|\mathbf{d}, \hat{\boldsymbol{\theta}})$ as well as possible. One possibility is to impose the factorized form of the posterior of the standard Gaussian mixture model ((3.12)) on $q(\mathbf{l})$, i.e., we chose $q(\mathbf{l})$ to be of the form

$$q(\mathbf{l}) = \prod_{n=1}^{N} q_n(l_n). \tag{3.20}$$

Within this family, our task is to chose the voxel-wise distributions $q_n(k)$ in such a way that the resulting joint distribution $q(\mathbf{l})$ approximates $p(\mathbf{l}|\mathbf{d}, \hat{\boldsymbol{\theta}})$ as accurately as possible. For this purpose, we minimize the so-called Kullback-Leibler (KL) divergence

$$KL\left(q(\mathbf{l}) \,||\, p(\mathbf{l}|\mathbf{d}, \hat{\boldsymbol{\theta}})\right) = -\sum_{\mathbf{l}} q(\mathbf{l}) \log \frac{p(\mathbf{l}|\mathbf{d}, \hat{\boldsymbol{\theta}})}{q(\mathbf{l})}, \tag{3.21}$$

which measures how different $q(\mathbf{l})$ is from $p(\mathbf{l}|\mathbf{d}, \hat{\boldsymbol{\theta}})$: it is always positive, and zero only if our approximation $q(\mathbf{l})$ equals the true posterior $p(\mathbf{l}|\mathbf{d}, \hat{\boldsymbol{\theta}})$ exactly (which will typically be unattainable because we restrict the form of $q(\mathbf{l})$ to (3.20)).

For a given set of distributions $q_{n'}(l_{n'})$ in all voxels $n' \neq n$, it can be shown [7] that the remaining voxel's distribution $q_n(l_n)$ that minimizes the KL-divergence is given by

$$q_n(l_n) = \frac{\mathcal{N}(d_n|\hat{\mu}_{l_n}, \hat{\sigma}^2_{l_n})\gamma_n(l_n)}{\sum_{k=1}^{K} \mathcal{N}(d_n|\hat{\mu}_k, \hat{\sigma}^2_k)\gamma_n(k)} \tag{3.22}$$

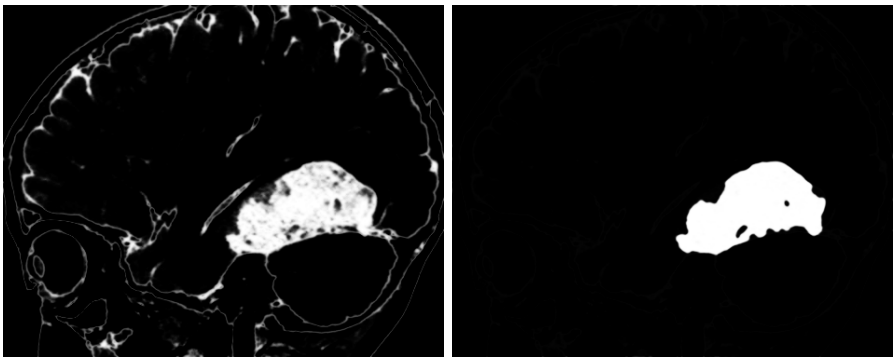
with

$$\gamma_n(k) = \frac{\hat{\pi}_k \cdot \exp\left(-\beta \sum_{n' \in \mathfrak{N}_n} (1 - q_{n'}(k))\right)}{\sum_{k'=1}^{K} \hat{\pi}_{k'} \cdot \exp\left(-\beta \sum_{n' \in \mathfrak{N}_n} (1 - q_{n'}(k'))\right)}. \tag{3.23}$$

Comparing this with the voxel-wise posterior of the standard Gaussian mixture model (3.13), it can be seen that the usual class priors π_k are replaced with altered priors $\gamma_n(k)$ that take the local neighborhood of the voxels into account: as in (3.19), the number of neighboring voxels "assigned" to a different class is "counted" (in a soft, weighted sense) and alters π_k accordingly.

Note that (3.22) gives the optimal distribution $q_n(k)$ for one voxel at a time, but that this result depends in turn on the result in other voxels. Therefore, a common strategy to minimize the KL-divergence is to cycle through the voxels in turn, updating each voxel's $q_n(k)$ based on the current result in the other voxels, and to continue this process until some convergence criterion is satisfied. With such a minimization strategy, the order in which the voxels are visited as well as the initialization of the $q_n(l_n)$'s may affect the local optimum of the KL divergence we arrive at.

A segmentation example on the tumor data of fig. 3.3 is shown in fig. 3.4, for different values of the Markov random field parameter β. It can be seen that the more advanced priors add contextual information that improves the segmentation results.



(a) $\beta = 0.25$ (b) $\beta = 0.55$

Figure 3.4: Visualization of the mean-field approximation to the segmentation posterior for the tumor data of Fig. 3.3, for different settings of the Markov random field parameter β.

Once the voxel-wise distributions $q_n(l_n)$ have been computed, they can be used in the same way as the distributions $p(l_n|d_n, \hat{\boldsymbol{\theta}})$ to approximate expectations, e.g., the expected volume of the structure with label k is approximately given by

$$\sum_{\mathbf{l}} V_k(\mathbf{l}) q(\mathbf{l}) = \sum_{n=1}^{N} q_n(l_n). \tag{3.24}$$

3.4 Parameter optimization using the EM algorithm

So far we have assumed that appropriate values $\hat{\boldsymbol{\theta}}$ of our model parameters are known in advance. In the tumor segmentation example of the previous section, these parameters were estimated by manually clicking on some representative points in the image, and collecting statistics on the intensity of the selected voxels. In general, however, such a strategy is impractical for such a versatile imaging modality as MRI, where intensities do not directly correspond to any physical properties of the tissue being scanned. By merely tweaking the imaging protocol, upgrading the scanner, or collecting images from different scanner models or manufacturers, the values of $\hat{\boldsymbol{\theta}}$ become inappropriate and need to be constructed again using manual interaction.

This difficulty can be avoided by estimating appropriate values for the model parameters automatically for each individual scan, i.e., each scan receives its own, well-suited set of parameters that are computed without requiring any manual interaction. This can be accomplished by estimating the parameters that maximize the so-called *likelihood function* $p(\mathbf{d}|\boldsymbol{\theta})$, which expresses how probable the observed image $\mathbf{d}$ is for different settings of the parameter vector

$\boldsymbol{\theta}$:

$$\begin{aligned} \hat{\boldsymbol{\theta}} &= \arg\max_{\boldsymbol{\theta}} \left[p(\mathbf{d}|\boldsymbol{\theta})\right] \\ &= \arg\max_{\boldsymbol{\theta}} \left[\log p(\mathbf{d}|\boldsymbol{\theta})\right]. \end{aligned} \tag{3.25}$$

The last step is true because the logarithm is a monotonically increasing function of its argument; it is used here because maximizing the log likelihood function instead of the likelihood function directly simplifies the mathematical analysis considerably, and also avoids numerical underflow problems in practical computer implementations. The parameter vector $\hat{\boldsymbol{\theta}}$ resulting from (3.25) is commonly called the *maximum likelihood* (ML) parameter estimate.

Maximizing the log likelihood function in image segmentation problems is a non-trivial optimization problem for which iterative numerical algorithms are needed. Although a variety of standard optimization methods could potentially be used, for the Gaussian mixture model discussed in Sec. 3.2 a dedicated and highly effective optimizer is available in the form of the so-called *expectation-maximization* algorithm (EM)[3]. The EM algorithm belongs to a family of optimization methods that work by repeatedly constructing a lower bound to the objective function, maximizing that lower bound, and repeating the process until convergence [8]. This process is illustrated in Fig. 3.5. For a given starting estimate of the model parameters $\tilde{\boldsymbol{\theta}}$, a function of the model parameters $Q(\boldsymbol{\theta}|\tilde{\boldsymbol{\theta}})$ is constructed that equals the log likelihood function at $\tilde{\boldsymbol{\theta}}$:

$$Q(\tilde{\boldsymbol{\theta}}|\tilde{\boldsymbol{\theta}}) = \log p(\mathbf{d}|\tilde{\boldsymbol{\theta}}), \tag{3.26}$$

but that otherwise never exceeds it:

$$Q(\boldsymbol{\theta}|\tilde{\boldsymbol{\theta}}) \leq \log p(\mathbf{d}|\boldsymbol{\theta}), \ \forall \boldsymbol{\theta}. \tag{3.27}$$

The parameter vector maximizing $Q(\boldsymbol{\theta}|\tilde{\boldsymbol{\theta}})$ is then computed and used as the new parameter estimate $\tilde{\boldsymbol{\theta}}$, after which the whole process is repeated. Critically, because of (3.26) and (3.27), updating the estimate $\tilde{\boldsymbol{\theta}}$ to the parameter vector that maximizes the lower bound automatically guarantees that the log likelihood function increases, by at least the same amount as the lower bound has increased. The consecutive estimates $\tilde{\boldsymbol{\theta}}$ obtained this way are therefore increasingly better estimates of the maximum likelihood parameters – one is *guaranteed* to never move in the wrong direction in parameter space. This is a highly desirable property for a numerical optimization algorithm.

While it is of course always possible to construct a lower bound to an objective function, nothing is gained if optimizing the lower bound is not significantly easier and/or faster to perform than optimizing the objective function directly. However, in the case of the Gaussian mixture model, it turns out it is possible to construct a lower bound for which the parameter vector maximizing it is given directly by analytical expressions. Therefore, the resulting algorithm effectively

[3]For more complex models with Markov random field priors, an approximate EM algorithm is obtained by replacing the voxel-wise posteriors with their mean-field approximations.

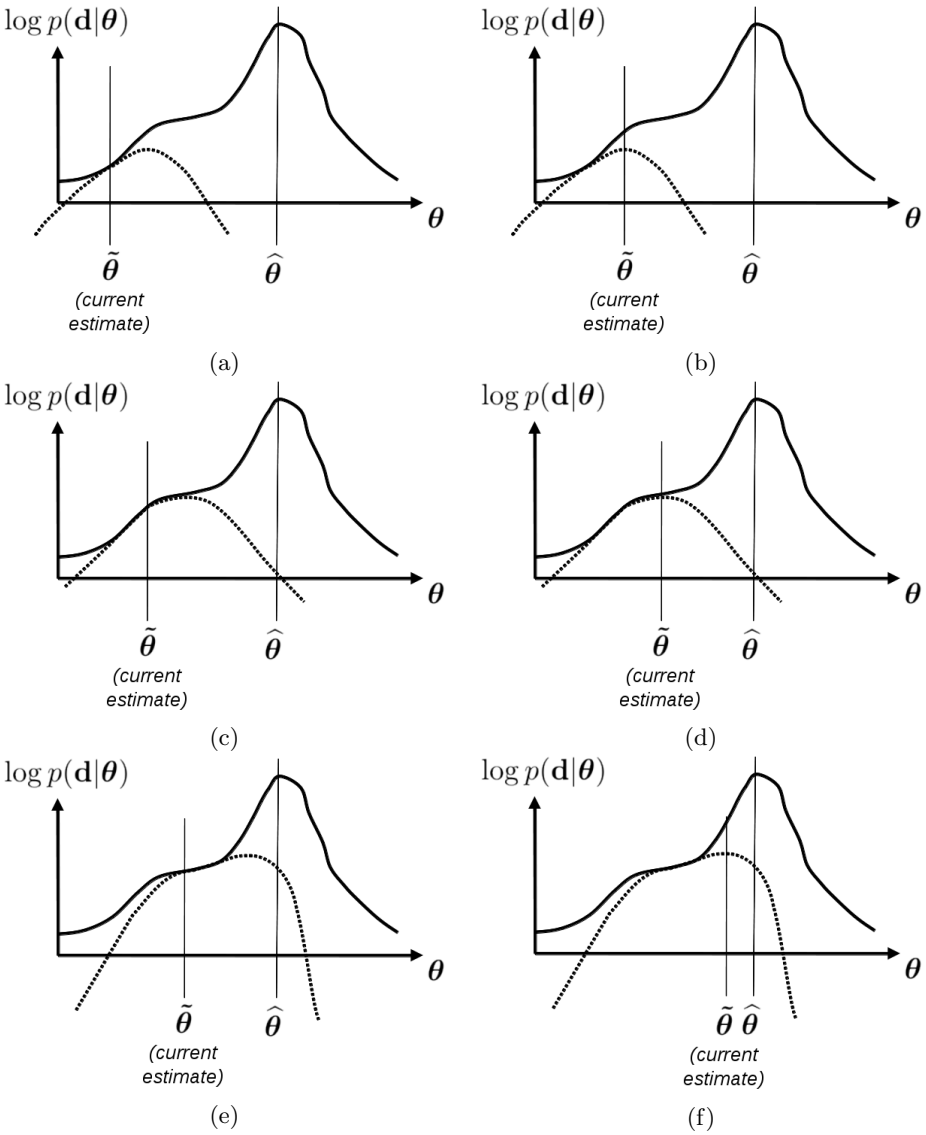


Figure 3.5: In the EM algorithm the maximum likelihood parameters $\hat{\boldsymbol{\theta}}$ are sought by repeatedly constructing a lower bound to the log likelihood function, in such a way that the lower bound touches the log likelihood function exactly at the current parameter estimate $\tilde{\boldsymbol{\theta}}$ (a). Subsequently the parameter estimate $\tilde{\boldsymbol{\theta}}$ is updated to the parameter vector that maximizes the lower bound (b). A new lower bound is then constructed at this new location (c) and maximized again (d), and so forth ((e) and (f)), until convergence. In these plots, the log likelihood function is represented by a full line, and the successive lower bounds with a broken line.

breaks up a difficult maximization problem (of the log likelihood function) into many smaller ones (of the lower bound) that are trivial to solve. Combined with the guarantee of increasingly better estimates of the maximum likelihood parameters as iterations progress, it should be clear why this algorithm is a popular choice for maximum likelihood estimation of Gaussian mixture model parameters.

The trick exploited by the EM algorithm to construct its lower bound is based on the property of the logarithm that it is a *concave* function, i.e., every chord connecting two points on its curve lies on or below that curve (see Fig. 3.6). Mathematically, this means that

$$\log\left[wx_1 + (1-w)x_2\right] \;\geq\; w\log x_1 + (1-w)\log x_2$$

for any two points x_1 and x_2 and $0 \leq w \leq 1$. It is trivial to show that this also generalizes to more than two variables, (so-called *Jensen's inequality*):

$$\log(\sum_{k=1}^{K} w_k x_k) \;\geq\; \sum_{k=1}^{K} w_k \log x_k \tag{3.28}$$

where $w_k \geq 0$ and $\sum_{k=1}^{K} w_k = 1$, for any set of points $\{x_k\}$. This can now be used to construct a lower bound to the likelihood function of the Gaussian mixture model as follows. Recalling that $p(\mathbf{d}|\boldsymbol{\theta}) = \prod_{n=1}^{N}\left[\sum_{k=1}^{K}\mathcal{N}(d_n|\mu_k,\sigma_k^2)\pi_k\right]$ ((3.10) and (3.11)), we have that

$$\begin{aligned}
\log p(d|\boldsymbol{\theta}) &= \log\left(\prod_{n=1}^{N}\left[\sum_{k=1}^{K}\mathcal{N}(d_n|\mu_k,\sigma_k^2)\pi_k\right]\right) && (3.29)\\
&= \sum_{n=1}^{N}\log\left[\sum_{k=1}^{K}\mathcal{N}(d_n|\mu_k,\sigma_k^2)\pi_k\right] && (3.30)\\
&= \sum_{n=1}^{N}\log\left[\sum_{k=1}^{K}\left(\frac{\mathcal{N}(d_n|\mu_k,\sigma_k^2)\pi_k}{\omega_{n,k}}\right)\omega_{n,k}\right] && (3.31)\\
&\geq \underbrace{\sum_{n=1}^{N}\left[\sum_{k=1}^{K}\omega_{n,k}\log\left(\frac{\mathcal{N}(d_n|\mu_k,\sigma_k^2)\pi_k}{\omega_{n,k}}\right)\right]}_{Q(\boldsymbol{\theta}|\tilde{\boldsymbol{\theta}})}, && (3.32)
\end{aligned}$$

for any set of weights $\{\omega_{n,k}\}$ that satisfy $\omega_{n,k} \geq 0$ and $\sum_{k=1}^{K}\omega_{n,k} = 1$ (the last step relies on (3.28)). We now have a lower bound function $Q(\boldsymbol{\theta}|\tilde{\boldsymbol{\theta}})$ that satisfies (3.27), but not (3.26), so we are not done yet. Instead of randomly assigning *any* valid K weights $\omega_{n,k}$ to each voxel n (one weight for each label k), we can satisfy (3.26) by choosing the weights so that

$$\omega_{n,k} = \frac{\mathcal{N}(d_n|\tilde{\mu}_k,\tilde{\sigma}_k^2)\tilde{\pi}_k}{\sum_{k'=1}^{K}\mathcal{N}(d_n|\tilde{\mu}_{k'},\tilde{\sigma}_{k'}^2)\tilde{\pi}_{k'}}. \tag{3.33}$$

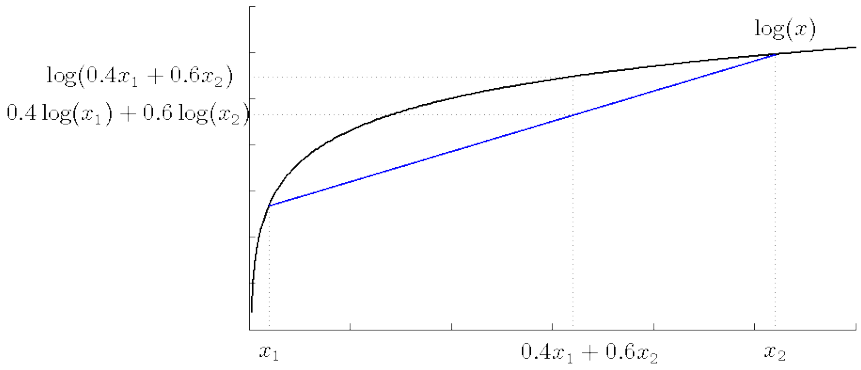


Figure 3.6: Because the logarithm is a concave function, the cord between any two points on its curve lies on or below the curve. An example cord is shown in blue.

By filling these weights into the definition of our lower bound (3.32), it is easy to check that (3.26) is indeed fulfilled with this choice.

Setting the new model parameter estimate $\tilde{\boldsymbol{\theta}}$ to the parameter vector that maximizes the lower bound requires finding the location where

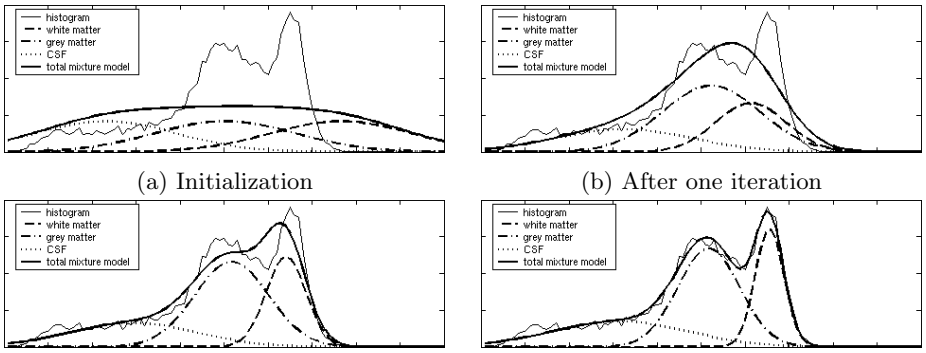
$$\frac{\partial Q(\boldsymbol{\theta}|\tilde{\boldsymbol{\theta}})}{\partial \boldsymbol{\theta}} = 0.$$

Reformulating the lower bound as

$$\begin{aligned} Q(\boldsymbol{\theta}|\tilde{\boldsymbol{\theta}}) \quad = \quad & -\frac{1}{2}\sum_{k=1}^{K}\left[\frac{1}{\sigma_k^2}\sum_{n=1}^{N}\omega_{n,k}\left(d_n-\mu_k\right)^2+\left(\sum_{n=1}^{N}\omega_{n,k}\right)\log\sigma_k^2\right] \\ & +\sum_{k=1}^{K}\left[\left(\sum_{n=1}^{N}\omega_{n,k}\right)\log\pi_k\right] \\ & -\sum_{n=1}^{N}\sum_{k=1}^{K}\omega_{n,k}\log\omega_{n,k}-\frac{N}{2}\log\left(2\pi\right), \end{aligned} \tag{3.34}$$

it is straightforward to show that the parameter updates are given by

$$\begin{aligned} \tilde{\mu}_k \quad &\leftarrow \quad \frac{\sum_{n=1}^{N}\omega_{n,k}d_n}{\sum_{n=1}^{N}\omega_{n,k}} \\ \tilde{\sigma}_k^2 \quad &\leftarrow \quad \frac{\sum_{n=1}^{N}\omega_{n,k}\left(d_n-\tilde{\mu}_k\right)^2}{\sum_{n=1}^{N}\omega_{n,k}} \\ \tilde{\pi}_k \quad &\leftarrow \quad \frac{\sum_{n=1}^{N}\omega_{n,k}}{N}. \end{aligned} \tag{3.35}$$



(a) Initialization (b) After one iteration

(c) After 11 iterations (d) After 30 iterations

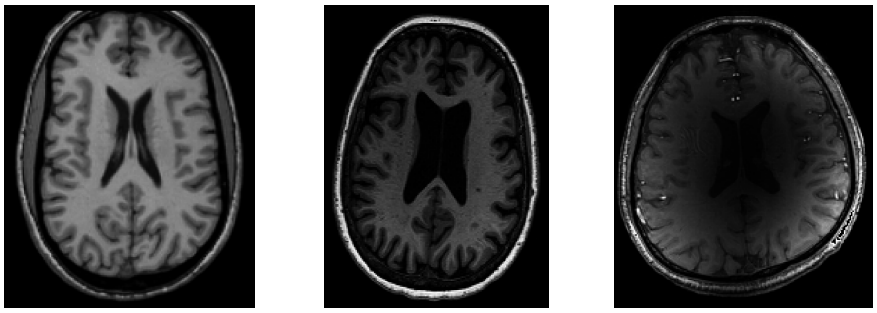
Figure 3.7: Iterative improvement of the Gaussian mixture model parameters for the MR image of Fig. 3.1a, using the EM algorithm.

It is worth spending some time thinking about these equations. The EM algorithm searches for the maximum likelihood parameters of the Gaussian mixture model merely be repeatedly applying the update rules of (3.35), where the weights $\omega_{n,k}$ are defined in (3.33). These weights depend themselves on the current estimate of the model parameters, which explains why the algorithm involves iterating. More importantly, by comparing (3.33) to (3.13), we see that these weights represent nothing but the posterior probability of the segmentation, given the current model parameter estimate! Thus, the EM algorithm repeatedly computes the type of probabilistic segmentation shown in Fig. 3.2 based on its current parameter estimate, and then updates the parameter estimate accordingly. The update rules of (3.35) are remarkably intuitive: the mean and variance of the Gaussian distribution associated with the kth label are simply set to the weighted mean and variance of the intensities of those voxels currently attributed to that label; similarly the prior for each class is set to the fraction of voxels currently attributed to that class.

Fig. 3.7 shows a few iterations of the EM algorithm searching for the maximum likelihood parameters of the brain MR data shown in Fig. 3.1a.

3.5 Modeling MR bias fields

Although the Gaussian mixture model is a very useful tool for image segmentation, and comes with a an dedicated parameter estimation algorithm, it can often not be applied directly to MR images. This is because MR suffers from an imaging artifact that makes some image areas darker and other areas brighter than they should be. This spatially smooth variation of intensities is often referred to as MR "intensity inhomogeneity" or "bias field", and is caused by imaging equipment limitations and electrodynamic interactions with the object being scanned. Interestingly, the bias field artifact is dependent on the anatomy



(a) 1.5 Tesla (b) 3 Tesla (c) 7 Tesla

Figure 3.8: The MR bias field artifact is often more pronounced in scanners operating at higher magnetic field strengths.

being imaged and therefore unique to each scan session, and is much more pronounced in the newest generation of scanners (see Fig. 3.8).

Since the Gaussian mixture model does not account for smoothly varying overall intensity levels within one and the same anatomical structure, it is very susceptible to segmentation errors when applied to typical MR data. However, this problem can be avoided by explicitly taking a model for the bias field artifact into account in the imaging component of the generative model. In particular, we can model the artifact as a linear combination of M spatially smooth basis functions

$$\sum_{m=0}^{M-1} c_m \phi_{n,m}, \tag{3.36}$$

where $\phi_{n,m}$ is shorthand for $\phi_m(\mathbf{x}_n)$, the value of the mth basis function evaluated at voxel n, which has spatial location $\mathbf{x}_n$. Suitable basis functions can be 2D DCT basis functions, 2D B-spline basis functions (such as those shown in Fig. 1.5), or something similar. We can then extend the imaging model of the Gaussian mixture model by still assigning each voxel an intensity drawn from a Gaussian distribution associated with its label, but further *adding*[4] the bias model to the resulting intensity image to obtain the final, bias field corrupted image $\mathbf{d}$. With this model, we have

$$p(\mathbf{d}|\mathbf{l}, \boldsymbol{\theta}_d) = \prod_{n=1}^{N} \mathcal{N}\Big(d_n \,\Big|\, \mu_{l_n} + \sum_{m-1}^{M} c_m \phi_{n,m},\ \sigma_{l_n}^2\Big) \tag{3.37}$$

with parameters $\boldsymbol{\theta}_d = (\mu_1, \ldots, \mu_K, \sigma_1^2, \ldots, \sigma_K^2, c_0, \ldots, c_{M-1})^{\mathrm{T}}$, which consist not only of the parameters associated with the Gaussian distributions, but additionally also the M coefficients of the bias field basis functions, c_m.

[4]Because of the physics of MR, the bias field is better modeled as a multiplicative rather than an additive artifact. This can be taken into account by working with logarithmically transformed intensities in the models, instead of using directly the original MR intensities.

As was the case with the Gaussian mixture model, model parameter estimation can be performed conveniently by iteratively constructing a lower bound to the log likelihood function, and working with that lower bound instead. For constructing the lower bound in this case, we follow the exact same procedure as in the previous section to obtain

$$\begin{aligned} Q(\boldsymbol{\theta}|\tilde{\boldsymbol{\theta}}) &= -\frac{1}{2}\sum_{k=1}^{K}\left[\frac{1}{\sigma_k^2}\sum_{n=1}^{N}\omega_{n,k}\left(d_n-\mu_k-\sum_{m-1}^{M}c_m\phi_{n,m}\right)^2+\left(\sum_{n=1}^{N}\omega_{n,k}\right)\log\sigma_k^2\right] \\ &\quad+\sum_{k=1}^{K}\left[\left(\sum_{n=1}^{N}\omega_{n,k}\right)\log\pi_k\right] \\ &\quad-\sum_{n=1}^{N}\sum_{k=1}^{K}\omega_{n,k}\log\omega_{n,k}-\frac{N}{2}\log\left(2\pi\right), \end{aligned} \tag{3.38}$$

where now the weights satisfying (3.26) are given by

$$\omega_{n,k}=\frac{\mathcal{N}\left(d_n\ \middle|\tilde{\mu}_k+\sum_{m-1}^{M}\tilde{c}_m\phi_{n,m},\ \tilde{\sigma}_k^2\right)\tilde{\pi}_k}{\sum_{k'}\mathcal{N}\left(d_n\ \middle|\tilde{\mu}_{k'}+\sum_{m-1}^{M}\tilde{c}_m\phi_{n,m},\ \tilde{\sigma}_{k'}^2\right)\tilde{\pi}_{k'}}. \tag{3.39}$$

Maximizing this lower bound is more complicated than in the Gaussian mixture model, however, because setting the derivative with respect to the parameter vector $\boldsymbol{\theta}$ to zero no longer yields analytical expressions for the parameter update rules. If we keep the bias field parameters fixed at their current values $\tilde{c}_m$, and only maximize the lower bound with respect to the Gaussian mixture model parameters, we easily obtain

$$\begin{aligned} \tilde{\mu}_k &\leftarrow \frac{\sum_{n=1}^{N}\omega_{n,k}\left(d_n-\sum_{m-1}^{M}\tilde{c}_m\phi_{n,m}\right)}{\sum_{n=1}^{N}\omega_{n,k}} \\ \tilde{\sigma}_k^2 &\leftarrow \frac{\sum_{n=1}^{N}\omega_{n,k}\left(d_n-\sum_{m-1}^{M}\tilde{c}_m\phi_{n,m}-\tilde{\mu}_k\right)^2}{\sum_{n=1}^{N}\omega_{n,k}} \\ \tilde{\pi}_k &\leftarrow \frac{\sum_{n=1}^{N}\omega_{n,k}}{N}. \end{aligned} \tag{3.40}$$

Similarly, keeping the Gaussian mixture model parameters fixed at their current values, it is easy to see that the bias field parameters maximizing the lower bound are given in analytical form as well: the lower bound is a quadratic function of the coefficients c_m, and finding their optimal values is therefore merely a linear basis function regression problem (see Sec. 1.1). In particular, the solution is given by

$$\tilde{\mathbf{c}} \leftarrow (\boldsymbol{\Phi}^{\mathrm{T}}\mathbf{P}\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^{\mathrm{T}}\mathbf{P}\mathbf{r} \tag{3.41}$$

where $\mathbf{\Phi}$ is given by (1.2) and

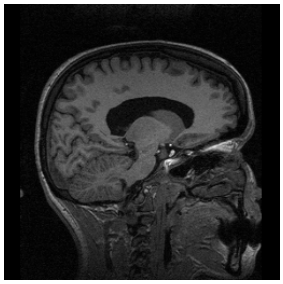
$$p_{n,k} = \frac{\omega_{n,k}}{\tilde{\sigma}_k^2}, \quad p_n = \sum_{k=1}^{K} p_{n,k}, \quad \mathbf{P} = \mathrm{diag}(p_1, \ldots, p_N),$$

$$\tilde{d}_n = \frac{\sum_{k=1}^{K} p_{n,k}\tilde{\mu}_k}{\sum_{k=1}^{K} p_{n,k}}, \quad \boldsymbol{r} = \begin{pmatrix} d_1 - \tilde{d}_1 \\ \vdots \\ d_N - \tilde{d}_N \end{pmatrix}.$$

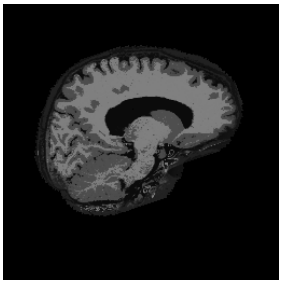
Since (3.40) and (3.41) depend on one another, one could in principle try to maximize the lower bound by cycling through these two equations, one at a time, until some convergence criterion is met. However, the desirable property of the EM algorithm to never decrease the value of the likelihood function with each new iteration still holds even when the lower bound is not *maximized* but merely *improved*. Therefore, a more efficient strategy is to construct the lower bound by computing the weights $\omega_{n,k}$ ((3.39)) and then update the Gaussian mixture model parameters ((3.40)) and subsequently the bias field parameters ((3.41)) only *once* to merely improve it. After that a new lower bound is constructed by recomputing the weights, which is again improved by updating the model parameters, etc, until convergence. Such an optimization strategy of only partially optimizing the EM lower bound is known as so-called *generalized expectation-maximization*.

The interpretation of the update equations is again very intuitive. As was the case in the Gaussian mixture model, the weights $\omega_{n,k}$ represent again a statistical classification of the image voxels as in Fig. 3.2, and the mixture model parameters are again updated accordingly. The only difference now is that instead of the original MR intensities, d_n, the bias field corrected intensities, $d_n - \sum_{m-1}^{M} \tilde{c}_m \phi_{n,m}$, i.e., the intensities after the estimated bias field has been subtracted, are used. Regarding the bias field update, the algorithm tries to make a reconstruction $(\tilde{d}_1, \ldots, \tilde{d}_N)^{\mathrm{T}}$ of what the image should look without the bias field artifact (shown in Fig. 3.9b), subtracts that from the MR scan to obtain a (noisy) estimate of the bias field (image $\mathbf{r}$, shown in Fig. 3.9c), and smoothes the result to obtain an estimate of the bias field (shown in Fig. 3.9e). For the smoothing, each voxel has a weight p_n (shown in Fig. 3.9d) that depends on the variance of the class it is attributed to, reflecting the confidence in the local value of $\mathbf{r}$ (classes with tighter variances are more trustable than classes with very large variances).

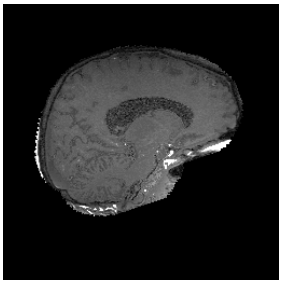
By extending the Gaussian mixture model with an explicit model for the bias field artifact this way, it is possible to obtain high-quality segmentations of MR scans without errors caused by intensity inhomogeneities, as shown in Fig. 3.10.



(a) MR scan

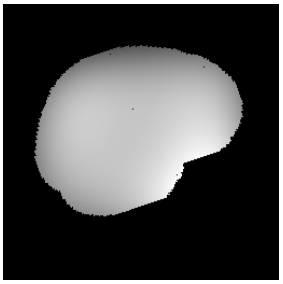


(b) Reconstruction



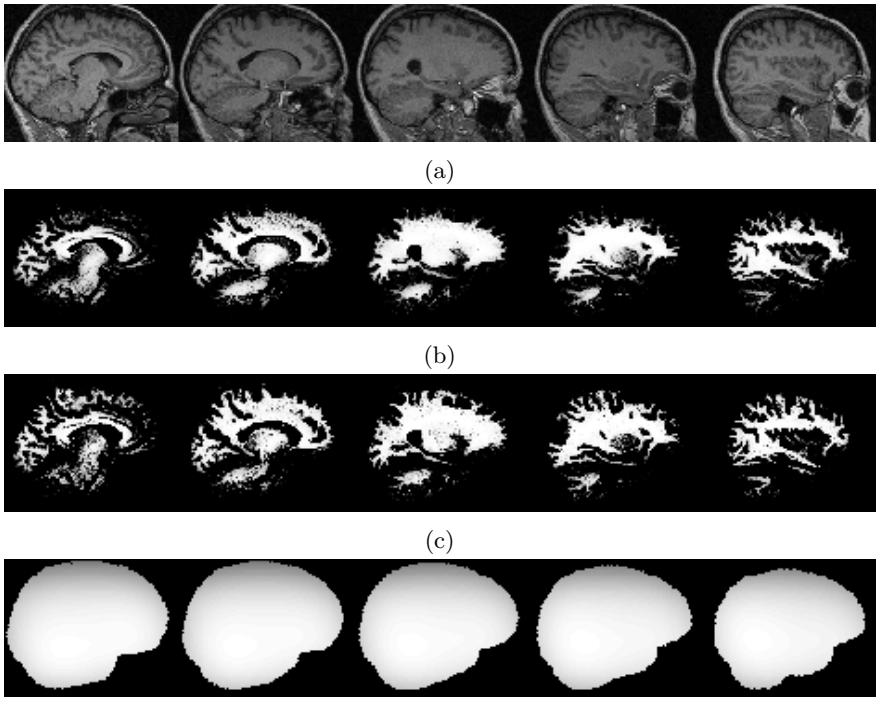
(c) Difference between the MR scan and the reconstruction

(d) Weight image



(e) Estimated bias field

Figure 3.9: Illustration of the bias field estimation within a generalized expectation-maximization algorithm.



(a) (b) (c) (d)

Figure 3.10: Explicitly modeling and estimating the bias field artifact in MR scans often improves segmentation results considerably. Shown are a few sagittal slices from a brain MR scan (a); the posterior probability for white matter using the standard Gaussian mixture model (b); the same when a bias field model is explicitly taken into account (c); and the automatically estimated bias field model (d). Note the marked improvement in segmentation accuracy in the upper parts of the brain.

Chapter 4

Neural Networks

The methods for registration and segmentation that we have seen so far are all based on *models* that somehow encode prior knowledge of the problem at hand. For instance, mutual information-based registration exploits the fact that one image is predictive of another image only when the two are well aligned; while the Gaussian mixture model for segmentation encodes the knowledge that voxels with the same label typically have similar intensities.

A completely different approach to solving a problem is not to try to understand it, but rather to simply emulate successful example solutions. As opposed to the generative models of Chapter 3, this approach to model-free "machine learning" is called *discriminative* learning. In this chapter we will review a specific class of discriminative methods based on artificial neural networks. Although such networks can also be used for other purposes, such as image registration and computer aided diagnosis, they are especially successful in the area of image segmentation, which we will focus on in this chapter.

In general, it should be noted that sidestepping the difficulty of building models is both the main strength and the most important weakness of neural networks. As long as sufficient example solutions are available, they are very easy to deploy without requiring any domain-specific knowledge, and can be orders of magnitude faster than model-based methods. On the other hand, however, example solutions are often in short supply in medical image analysis, as manually annotating a large set of images can be excruciatingly time consuming. Especially in versatile imaging modalities such as MRI, this problem is exacerbated by the differences in scanning hardware, software and protocols that exist even within the same hospital, which can make carefully curated example solutions useless overnight.

4.1 Logistic regression

In Chapter 3 we saw how the Gaussian mixture model can be used to classify each image voxel into one of K different classes (tissue types) based on its inten-

sity alone. Specifically, we modeled the prior probability of a voxel belonging to class k as π_k, and the intensity distribution associated with class k as a Gaussian with mean μ_k and variance σ_k^2. Using Bayes' rule, we then obtained the probability of a voxel having label l as (cf. ((3.13)))

$$p(l = k|d, \boldsymbol{\theta}) = \frac{\mathcal{N}(d|\mu_k, \sigma_k^2)\pi_k}{\sum_{k'} \mathcal{N}(d|\mu_{k'}, \sigma_{k'}^2)\pi_{k'}}, \tag{4.1}$$

where d is the voxel's intensity and $\boldsymbol{\theta}$ are the parameters of the model (collecting π_k, μ_k and σ_k^2 for all classes).

Rather than using such models, another way of obtaining a voxel-wise classifier $p(l|d, \boldsymbol{\theta})$ is by simply mimicking the behavior seen in ("learning from") examples. For reasons that will soon become clear, for the remainder of this chapter we will switch to the notation used for regression in Chapter 1, indicating a training set of observed example inputs and outputs as $\{\mathbf{x}_n, t_n\}_{n=1}^N$, where $\mathbf{x}_n$ is the D-dimensional vector of observed inputs for the nth case, and t_n the corresponding output. For voxel-wise classification, each input is simply an intensity $\mathbf{x}_n = d_n$, i.e., the input dimensionality is $D = 1$, but we will soon see cases where $\mathbf{x}_n$ is higher-dimensional. Unlike in the regression case, where each output t_n was a continuous value, for classification the outputs can only take K discrete values, indicating which class each example belongs to. To simplify notation, here we only consider scenarios with $K = 2$ classes[1]. The outputs can then be denoted as taking binary values $t_n \in \{0, 1\}$, where values 1 and 0 correspond to the class assignments $l_n = 1$ and $l_n = 2$, respectively.

To model $p(y|\mathbf{x}, \boldsymbol{\theta})$ – which is equivalent to modeling $p(l|d, \boldsymbol{\theta})$ in our voxel-wise classification example – we start from a parametric curve of the form

$$f(\mathbf{x}) = \sigma\left(\sum_{m=0}^{M-1} w_m \phi_m(\mathbf{x})\right), \tag{4.2}$$

which is similar to the linear basis function model for regression we saw in Sec. 1.1: A linear combination of M nonlinear basis functions $\phi_m(\mathbf{x})$ with tunable coefficients w_m, which we here collect in the parameter vector $\boldsymbol{\theta} = (w_0, \ldots, w_{M-1})^{\mathrm{T}}$. Unlike in the regression case, however, where the quantity

$$a = \sum_{m=0}^{M=1} w_m \phi_m(\mathbf{x}) \tag{4.3}$$

was used directly, here it is subject to a "squashing" function (illustrated in Fig. 4.1)

$$\sigma(a) = \frac{1}{1 + \exp(-a)} \tag{4.4}$$

that maps the result into the interval $[0, 1]$. Because of this mapping, $f(\mathbf{x})$ can be interpreted as a probability: $p(t = 1|\mathbf{x}, \boldsymbol{\theta}) = f(\mathbf{x})$. Since $p(t = 0|\mathbf{x}, \boldsymbol{\theta}) =$

[1] The generalization to several classes is rather straightforward (cf. [9], chapter 4).

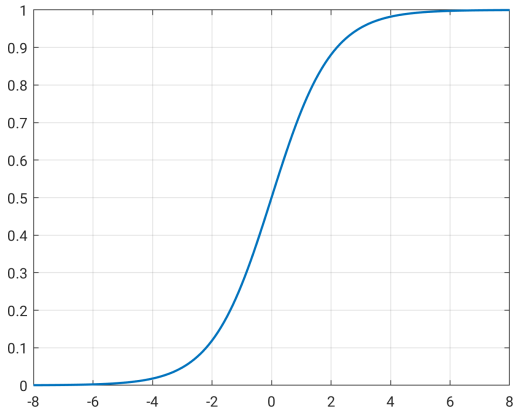


Figure 4.1: The logistic function $\sigma(a)$ defined in ((4.4)) maps the domain $[-\infty, \infty]$ into $[0, 1]$.

$1 - p(t = 1|\mathbf{x}, \boldsymbol{\theta}) = 1 - f(\mathbf{x})$, we can then write (Bernoulli distribution):

$$p(t|\mathbf{x}, \boldsymbol{\theta}) = f(\mathbf{x})^t [1 - f(\mathbf{x})]^{1-t}. \tag{4.5}$$

Because the mapping function $\sigma(a)$ in ((4.4)) is called the "logistic function", the model of ((4.2)) is known as "logistic regression".

Given our training set of N input observations $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ with corresponding outputs $\mathbf{t} = \{t_1, \ldots, y_N\}$, appropriate values for the model parameters $\hat{\boldsymbol{\theta}}$ can be found by numerically maximizing the likelihood function

$$p(\mathbf{t}|\mathbf{X}, \boldsymbol{\theta}) = \prod_{n=1}^{N} p(t_n|\mathbf{x}_n, \boldsymbol{\theta}) \tag{4.6}$$

with respect to $\boldsymbol{\theta}$. Once fitted this way, the model can subsequently be used to classify new voxels simply by evaluating

$$p(l = 1|d, \hat{\boldsymbol{\theta}}) = p(t = 1|\mathbf{x}, \hat{\boldsymbol{\theta}}) = \sigma\left(\sum_{m=0}^{M-1} \hat{w}_m \phi_m(\mathbf{x})\right),$$

and assigning a voxel to class 1 if $p(l = 1|d, \hat{\boldsymbol{\theta}}) > 0.5$ and to class 2 otherwise.

Fig. 4.2 illustrates this procedure on an MRI scan of the kidneys, using the five 1D cosines of Fig. 1.1a as basis functions $\phi_m(\mathbf{x})$ and $N = 50$ training points.

4.2 Training with stochastic gradient descent

So far we have not specified how to numerically optimize ((4.6)) during training. For the specific case of logistic regression, there exists a dedicated algorithm called *iterative reweighted least squares (IRLS)* that finds $\hat{\boldsymbol{\theta}}$ by iteratively

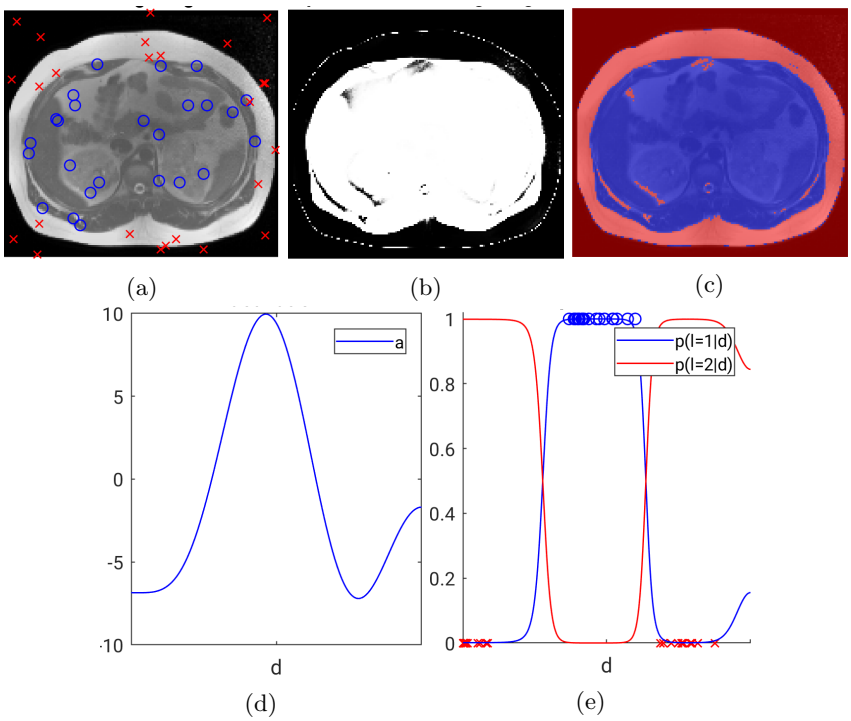


Figure 4.2: Example of a logistic regression classifier trained to discern pixels in the inner organs from pixels in other areas. (a) Image to be segmented, along with training data consisting of 25 manually selected points inside ($l = 1$, blue circles) and 25 points outside ($l = 2$, red crosses) the area of interest. (b) The output $p(l = 1|d, \hat{\boldsymbol{\theta}})$ of the trained classifier, evaluated in each pixel. (c) Final segmentation produced by the trained classifier, overlayed on top of the input image. (d) The quantity a of ((4.3)) for various intensity levels d. (e) The classifier resulting from subjecting a to the squashing function $\sigma(a)$ of ((4.4)). For reference, the intensity levels and labels of the 50 training points are also shown.

mapping the problem to a regression problem, which is then solved using tools similar to those described in Chapter 1. However, for the more general case of feed-forward neural networks, which we will soon discuss, IRLS cannot be used and optimization is typically performed using variants of *stochastic gradient descent*, described next.

Maximizing the likelihood function $p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta})$ is equivalent to minimizing its negative logarithm. Using ((4.5)) and ((4.6)), training therefore consists of minimizing the energy function

$$E_N(\boldsymbol{\theta}) = -\log p(\mathbf{t}|\mathbf{X}, \boldsymbol{\theta}) = -\sum_{n=1}^{N} \{t_n \log f(\mathbf{x}_n) + (1 - t_n) \log [1 - f(\mathbf{x}_n)]\}\,, \tag{4.7}$$

which is known in the machine learning community as *cross-entropy*. Starting from some (e.g., random) initial estimate $\boldsymbol{\theta}^{(0)}$, standard gradient descent proceeds by iteratively stepping in the direction of steepest descent:

$$\boldsymbol{\theta}^{(\tau+1)} = \boldsymbol{\theta}^{(\tau)} - \nu \, \nabla E_N(\boldsymbol{\theta}^{(\tau)}),$$

where τ denotes the iteration number, $\nabla E_N(\boldsymbol{\theta}) = \frac{\partial E_N}{\partial \boldsymbol{\theta}}$ is the gradient of the energy function, and ν is a step size that needs to be provided by the user. In practice, significant computational speed-ups can be achieved by noticing that $E_N(\boldsymbol{\theta})$ and therefore $\nabla E_N(\boldsymbol{\theta})$ involves summing over the contribution of all N training points. Since typically there is considerable redundancy in the training data set, i.e., many training points will contribute similarly, in *stochastic* gradient descent the true gradient is approximated by

$$\nabla E_N(\boldsymbol{\theta}) \simeq \frac{N}{N'} \, \nabla E_{N'}(\boldsymbol{\theta}),$$

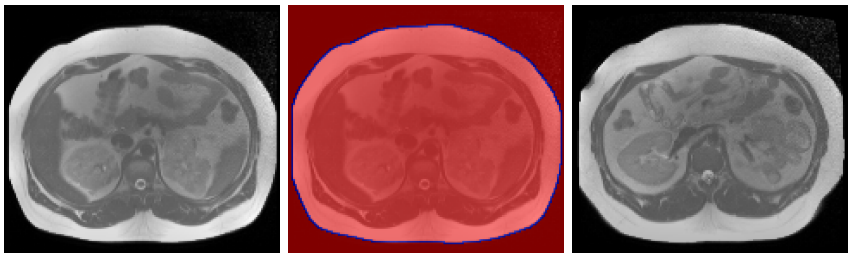
where $\nabla E_{N'}(\boldsymbol{\theta})$ is the gradient computed from only $N' \ll N$ randomly sampled training points. Optimizing then proceeds by

$$\boldsymbol{\theta}^{(\tau+1)} = \boldsymbol{\theta}^{(\tau)} - \nu' \, \nabla E_{N'}(\boldsymbol{\theta}^{(\tau)}) \tag{4.8}$$

with step size $\nu' = N\nu/N'$, after which a new subset of size N' is sampled for the next iteration.

4.3 Feed-forward network functions

Although we have so far only considered a problem where the dimensionality of $\mathbf{x}$ is $D = 1$, classifiers can also be used in cases where $D \gg 1$. As an example, considering again the same MRI slice of the kidneys as before, but this time we are given a full manual delineation of the border just outside of the body as training data (shown in blue in Fig. 4.3b). Our task is now to automatically segment the same border in other MRI slices and/or patients (an example is shown in Fig. 4.3c). Since the border cannot be discerned based on intensity alone, a simple voxel-wise classifier will no longer suffice. Instead, we can take



(a) (b) (c)

Figure 4.3: Training image (a) and corresponding segmentation (b) used for training the network shown in Fig. 4.4, as well as the type of image (c) that the network should be able to segment after training.

the local context into account, by having as input $\mathbf{x}$ not just the intensity of the pixel we aim to classify, but also that of its immediate 8 neighbors in the 2D pixel grid. The input to the classifier is therefore not a scalar intensity, but rather an entire 3×3 image patch, so that the dimensionality is $D = 9$.

In order to use logistic regression as before, we now face the difficulty of choosing appropriate basis functions $\phi_m(\mathbf{x})$ in our 9-dimensional input space. For low-dimensional cases (e.g., $D = 2$ or $D = 3$) we could follow the same procedure as in Sec 1.3 and use *separable* basis functions, which are simply the product of 1D basis funcions as in (1.11). However, this construct quickly becomes impractical in higher dimensions since the number of basis functions increases exponentially with the dimensionality ("curse of dimensionality"): with our five 1D cosines we would obtain 5^9 basis functions when $D = 9$, which is almost 2 million basis functions!

Rather than using a set of fixed basis functions, one solution is to choose basis functions that are *adaptive*, by having their form depend on extra parameters that are also optimized during training. A typical choice of such adaptive basis functions is as follows:

$$\phi_m(\mathbf{x}) = \begin{cases} 1 & \text{if } m = 0, \\ \sigma\left(\sum_{d=1}^{D} \beta_{m,d} x_d + \beta_{m,0}\right) & \text{otherwise,} \end{cases} \tag{4.9}$$

where $\sigma(\cdot)$ is the logistic function defined in ((4.4)), and the weights $\beta_{m,d}$ are extra parameters that together with the coefficients w_m form the parameters $\boldsymbol{\theta}$ of the model. Fig. 4.4 has a graphical representation of the resulting model for the case $D = 9$ and $M = 4$. Loosely based on a (now completely outdated) notion of similarity with how information is processed in the brain, it presents a simple *feed-forward neural network* in which information moves from the left of the figure to the right: all the components x_d (called *input units* in the neural network literature) of $\mathbf{x}$ are linearly combined and then non-linearly transformed through the logistic function to evaluate the basis functions $\phi_m(\mathbf{x})$'s (called *hidden units*). All the resulting $\phi_m(\mathbf{x})$'s are then themselves linearly combined

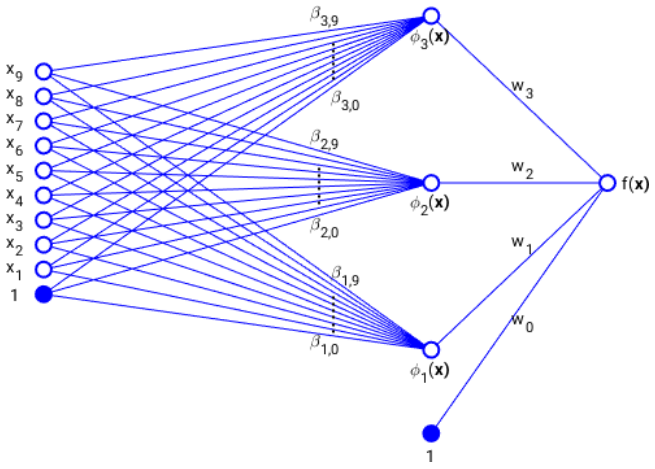


Figure 4.4: Graphical representation of a feed-forward neural network with a 9-dimensional input, a single hidden layer with three hidden units, and a single output unit. Since the parameters w_0 and $\{\beta_{m,0}\}$ effectively add mere constants during the computations, they are shown to be connected to additional units with values clamped to 1 (indicated by filled circles). The information flows from the left to the right through the network.

and non-linearly transformed in the same way to obtain $f(\mathbf{x})$ (called the *output unit*). It is easy to see that this type of model can be readily extended by inserting more layers of hidden units, each taking the previous hidden layer as input, resulting in "deep" networks for models with many such layers. Because of their specific structure, the gradient $\nabla E_N(\boldsymbol{\theta})$ that is needed for training these networks can be computed very efficiently, even for very large networks with many parameters, using an algorithm known as *backpropagation* [10]).

Fig. 4.5 shows the result of applying our 3×3 patch-based network on a new image, after training it on the image-segmentation pair of Fig. 4.3b using stochastic gradient descent. The network output $f(\mathbf{x})$ for each pixel is shown in Fig. 4.5b, and the basis function evaluations $\{\phi_m(\mathbf{x})\}_{m=1}^{M-1}$ (called *feature maps*) and their coefficients $\{w_m\}_{m=1}^{M-1}$ are shown in Fig. 4.5d. Since we are working with 3×3 images patches, the set of weights $\{\beta_{m,d}, d = 1, \ldots, 9\}$ for each feature $m = 1, \ldots, 3$ has a spatial structure and can be visualized as a 3×3 image itself, shown in Fig. 4.5e. Since computing the term $\sum_{d=1}^{D} \beta_{m,d} x_d$ in ((4.9)) for each pixel in the input image can be implemented as a *convolution* of the image with a 3×3 spatial filter, our network is a simple instance of what is called a *convolutional neural network* [11]. In practical applications, such networks typically have many more hidden layers than just the one used here, effectively using the feature maps shown in Fig. 4.5d as input to the next convolutional network layer etc, yielding increasingly higher-level features that can be used to solve increasingly hard segmentation tasks. The price to be paid for this increased ability is that the network then has many more parameters to

be estimated, which necessitates access to much larger amounts of (often very hard to get) annotated training data.

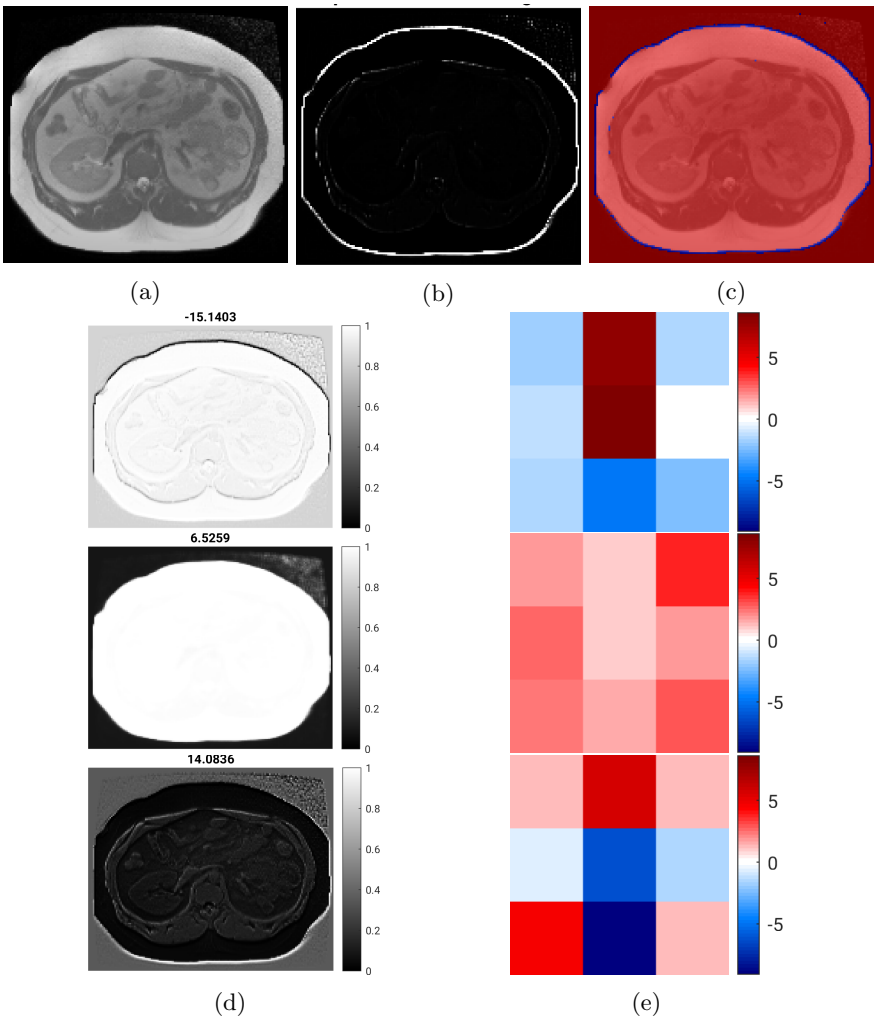


Figure 4.5: Segmentation of a new image using the patch-based network of Fig. 4.4, after training it on the data of Fig. 4.3b: (a) image to be segmented; (b) posterior probability generated by the network; (c) final segmentation generated by the network; (d) the feature maps used by the network to generate the segmentation, along with the corresponding coefficients; and (e) the 3×3 weight patterns used to generate the feature maps.

Chapter 5

Atlases

In this chapter we discuss the concept of so-called *atlases* in biomedical image analysis. An *atlas* is broadly defined as an image that has somehow been augmented with additional information beyond the voxel intensities alone. The exact form of this additional information depends on the specific application the atlas is used for, but typical examples include detailed manual segmentations of all the structures that are visible in the image, or a reference coordinate system that allows to compare anatomical or functional characteristics across different individuals. Atlases are often used as a teaching tool, helping medical students understand the complicated three-dimensional shape, configuration, and relations of different anatomical structures, or as an anatomical reference in surgical planning. They also frequently serve as a means of incorporating detailed anatomical knowledge into automated segmentation algorithms, or as a substrate to report novel findings in the scientific literature.

In this chapter we only concentrate on two types of atlases, namely so-called *reference templates* and *atlases for automated segmentation*. Although the examples in this chapter are all from brain MRI, the same concepts apply equally well to other imaging modalities such as CT and PET, and different anatomical structures, including the heart and lungs.

5.1 Reference templates

A *reference template* $\mathcal{T}$ is an image that serves as a reference coordinate system. By registering other images to this template and transferring relevant functional or anatomical information accordingly, findings can be compared across individuals, with subject-specific anatomical differences removed. Such a spatial normalization of information has many applications, including understanding the normal variation in anatomy between different individuals, comparing anatomy or function between patient populations to understand disease effects, or communicating scientific findings to researchers working in different laboratories.

Mathematically, once a template image $\mathcal{T}$ has been selected, it can be used

as a reference coordinate system by mapping any image under study, denoted by $\mathcal{I}$, into this standard coordinate system. Using the notation from Chapter 2, a geometrical transformation $\mathbf{y}(\mathbf{x}, \mathbf{w})$ is computed that maps the coordinates $\mathbf{x}_n$ of the template's voxels to coordinates $\mathbf{y}(\mathbf{x}_n, \mathbf{w})$ in the image. Resampled intensity values $\mathcal{I}(\mathbf{y}(\mathbf{x}_n, \mathbf{w}))$ can then be extracted, effectively deforming the image into the reference coordinate system.

5.1.1 Intensity averaging

In its simplest form, we can just pick some representative scan of a randomly chosen subject to serve as the template $\mathcal{T}$. However, it is often desirable to use a template that is more representative of a whole *population*, because the anatomy of a single individual cannot faithfully represent the complex structural variability between different people. This can be accomplished by taking a collection of images $\{\mathcal{I}_1, \ldots, \mathcal{I}_Q\}$ acquired from Q different individuals, performing $Q-1$ registrations between the scans of individuals $2 \ldots Q$ and the scan of the first individual, and averaging the resulting images. Specifically, let the parameter vector of the geometrical transformation mapping individual 1 to individual q be denoted by $\mathbf{w}_q$. For each voxel n in the scan of individual 1, with coordinates $\mathbf{x}_n$ and intensity $\mathcal{I}_1(\mathbf{x}_n)$, we can then obtain the corresponding intensity $\mathcal{I}_q(\mathbf{y}(\mathbf{x}_n, \mathbf{w}_q))$ in each subject $q = 2 \ldots Q$. The template $\mathcal{T}$ is then computed by assigning as intensity of voxel n the average intensity over all Q available images:

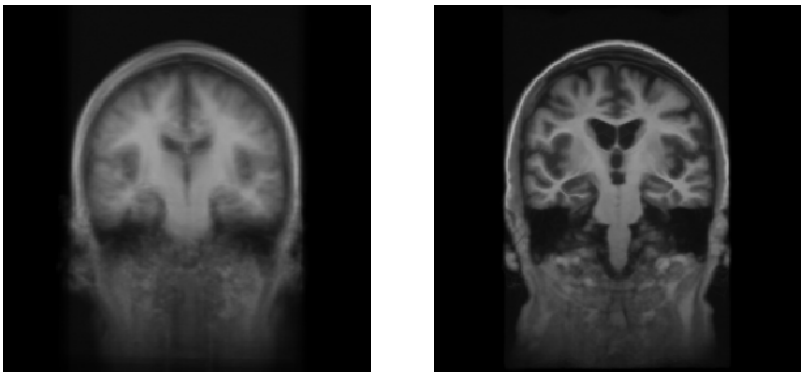
$$\mathcal{T}(\mathbf{x}_n) \quad \leftarrow \quad \frac{\mathcal{I}_1(\mathbf{x}_n) + \sum_{q=2}^{Q} \mathcal{I}_q(\mathbf{y}(\mathbf{x}_n, \mathbf{w}_q))}{Q}.$$

The level of anatomical detail in such average templates depends on the degrees of freedom in the geometrical transformation $\mathbf{y}(\mathbf{x}, \mathbf{w})$: models with many degrees of freedom will be able to match corresponding anatomical locations across all Q individuals better than simpler models, yielding "sharper" templates. Example reference templates illustrating this effect are shown in Fig. 5.1.

5.1.2 Group-wise registration

Although the procedure outlined above effectively averages over the intensity levels across Q different scans, the geometrical *shape* of the template is still entirely defined by the shape of the first individual, since the scan of that first individual was used to establish the coordinate system to which all other scans were deformed. This can be problematic in studies comparing two different groups of subjects, if the shape of the first individual is somehow closer to the average shape of one of the groups: the registration process will then be more difficult to perform for the other group, resulting in an uneven distribution of registration errors and ultimately false interpretations about the inferred differences between the groups.

In order to avoid this situation, it is better to compute a template that is average both in terms of *intensities* and anatomical *shape*. This can be accomplished by considering the following generative model for the Q available scans.



(a) (b)

Figure 5.1: Reference template obtained by intensity-averaging the brain MR scans of 20 different individuals after affine (a) and deformable (a) registration.

First, a template image $\mathcal{T}$ is assumed to be drawn from some prior distribution $p(\mathcal{T})$. Since we typically have no reason to prefer certain templates over others, we will use a uniform prior, i.e., $p(\mathcal{T}) \propto 1$. Subsequently, each scan $\mathcal{I}_q$ is assumed to be obtained independently from this template by (1) drawing a sample $\mathbf{w}_q$ from some distribution $p(\mathbf{w}) \propto \exp(-\mathcal{S}(\mathbf{w}))$ governing the deformation model parameters, where $\mathcal{S}(\mathbf{w})$ is a regularizer penalizing unlikely deformations; (2) deforming the template $\mathcal{T}$ accordingly; and (3) adding random, zero-mean Gaussian noise with variance σ^2 to each voxel independently.

With this model, the template $\mathcal{T}$ and the deformation parameters $\{\mathbf{w}_q\}$ can be estimated from the Q available scans by maximizing their joint posterior distribution

$$\begin{aligned} & p(\mathcal{T}, \mathbf{w}_1, \ldots, \mathbf{w}_Q | \mathcal{I}_1, \ldots \mathcal{I}_Q) \\ & \quad \propto \quad p(\mathcal{I}_1, \ldots \mathcal{I}_Q | \mathcal{T}, \mathbf{w}_1, \ldots, \mathbf{w}_Q) p(\mathcal{T}, \mathbf{w}_1, \ldots, \mathbf{w}_Q) \\ & \quad = \quad p(\mathcal{T}) \prod_{q=1}^{Q} p(\mathcal{I}_q | \mathcal{T}, \mathbf{w}_q) p(\mathbf{w}_q) \\ & \quad \propto \quad \prod_{q=1}^{Q} \left[\prod_{n=1}^{N} \exp\left(- \frac{(\mathcal{I}_q(\mathbf{y}(\mathbf{x}_n, \mathbf{w}_q)) - \mathcal{T}(\mathbf{x}_n))^2}{2\sigma^2} \right) \right] \exp(-\mathcal{S}(\mathbf{w}_q)) \,. \end{aligned} \tag{5.1}$$

This is equivalent to minimizing $-\log[p(\mathcal{T}, \mathbf{w}_1, \ldots, \mathbf{w}_Q | \mathcal{I}_1, \ldots \mathcal{I}_Q)]$, which can be re-written as:

$$\begin{aligned} & \{\hat{\mathcal{T}}, \hat{\mathbf{w}}_1, \ldots, \hat{\mathbf{w}}_Q\} = \\ & \qquad \arg \min_{\{\mathcal{T}, \mathbf{w}_1, \ldots, \mathbf{w}_Q\}} \sum_{q=1}^{Q} \left[\frac{1}{2\sigma^2} \sum_{n=1}^{N} (\mathcal{I}_q(\mathbf{y}(\mathbf{x}_n, \mathbf{w}_q)) - \mathcal{T}(\mathbf{x}_n))^2 + \mathcal{S}(\mathbf{w}_q) \right] . \end{aligned} \tag{5.2}$$

Starting from some initial deformation parameters $\tilde{\mathbf{w}}_q$, typically chosen to cor-

respond to no deformation at all, the optimization of (5.2) can be performed by updating the estimate of the template $\tilde{\mathcal{T}}$ while keeping the deformation parameters fixed to their current values, and subsequently updating the deformation parameters while keeping the template fixed, each in turn, until convergence. The update for the template that minimizes the objective function of (5.2) for a given set of deformation parameters is given by

$$\tilde{\mathcal{T}}(\mathbf{x}_n) \quad \leftarrow \quad \frac{\sum_{q=1}^{Q} \mathcal{I}_q(\mathbf{y}(\mathbf{x_n}, \tilde{\mathbf{w}}_q))}{Q} \tag{5.3}$$

for each template voxel n, and the corresponding update for each of the Q deformation parameter vectors is given by

$$\tilde{\mathbf{w}}_q \quad \leftarrow \quad \arg\min_{\mathbf{w}_q} \left[\frac{1}{2\sigma^2} \sum_{n=1}^{N} \left(\mathcal{I}_q(\mathbf{y}(\mathbf{x}_n, \mathbf{w}_q)) - \tilde{\mathcal{T}}(\mathbf{x}_n) \right)^2 + \mathcal{S}(\mathbf{w}_q) \right], \tag{5.4}$$

which can be solved using the optimization procedure discussed in Sec. 2.4.1. In summary, the algorithm computes an initial, fuzzy average template ((5.3)) to which all Q scans are subsequently registered ((5.4)), after which the template is re-computed etc, until convergence. The template resulting from this procedure will show increasingly more anatomical detail as the iterations progress and the deformations to it become more precise. Upon convergence, a template is obtained that it is not biased to any of the Q available scans in particular, but rather represents the average, both in shape and intensity, among *all* the scans simultaneously.

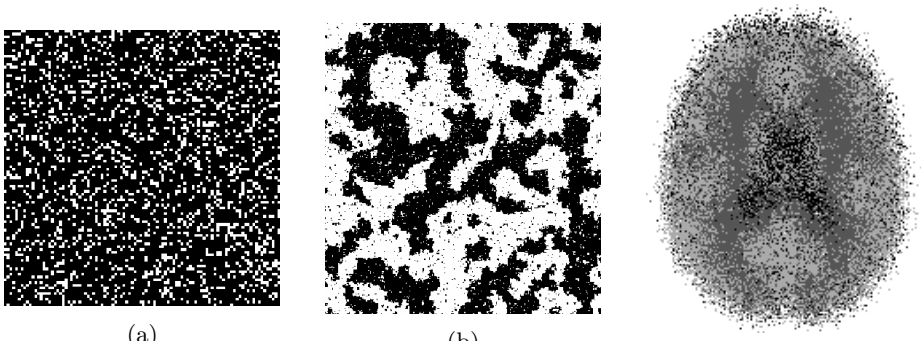
5.2 Atlases for segmentation

Another class of atlases is used to infuse prior anatomical knowledge into computational image segmentation algorithms. In this setting, one starts not only from Q medical images $\{\mathcal{I}_1, \ldots, \mathcal{I}_Q\}$, but also from a corresponding number of detailed manual segmentations $\{\mathcal{L}_1, \ldots, \mathcal{L}_Q\}$ of those images. To keep notation consistent with the one used in Chapter 3, we will assume that each voxel in those segmentations is assigned a unique label $k \in \{1, \ldots, K\}$ that represents the anatomical structure the voxel belongs to. For each individual q, we thus have an intensity $\mathcal{I}_q(\mathbf{x})$ and a corresponding label $\mathcal{L}_q(\mathbf{x})$ in each location $\mathbf{x}$.

Atlases for segmentation purposes come in two distinct forms. So-called *probabilistic* atlases contain pre-computed statistics about the Q label images $\{\mathcal{L}_q\}$, whereas atlases used for so-called *label propagation* use the original label images directly instead. Below we will discuss the basic principles of both types of segmentation atlases.

5.2.1 Probabilistic atlases

In Chapter 3 we introduced two priors that are often used in model-based segmentation. The first, stated in (3.5), simply assumes that label k occurs with



(a) (b) (c)

Figure 5.2: Samples from three different priors often used in model-based segmentation: the prior used in the standard Gaussian mixture model (a); a Markov random field prior (b); and a probabilistic atlas prior (c).

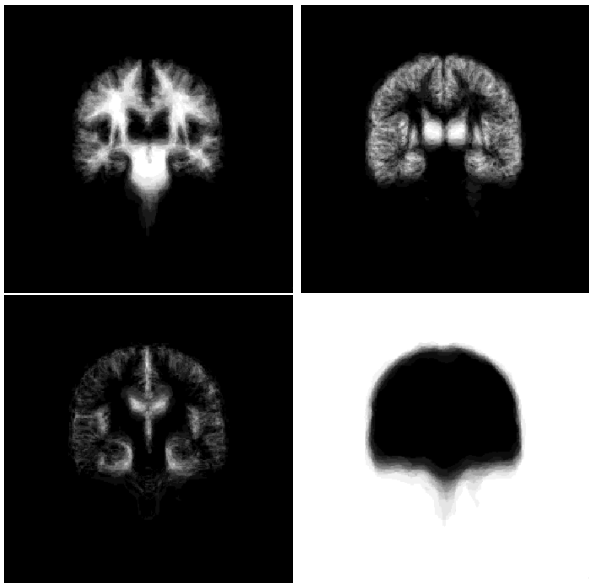
probability π_k in any given voxel, without any further constraints on the spatial organization of the labels. The second, stated in (3.16) and (3.18), is a Markov random field model that additionally encourages the different labels to occur in spatial clusters, rather than being scattered randomly throughout the image area. Although these priors are computationally convenient to work with, they do not encode any information about the shape, organization, and spatial relationships of real anatomical structures, as demonstrated in Fig. 5.2a and 5.2b).

Here we consider a third class of priors for model-based segmentation, namely *probabilistic atlases*, that do encode such prior knowledge of anatomy while still being computationally attractive. A sample from this type of anatomical prior is shown in Fig. 5.2c. The prior is constructed as follows. First, an intensity template $\mathcal{T}$ is computed by co-registering all the Q intensity images $\{\mathcal{I}_q\}$ to a common reference and averaging the intensities, using one of the techniques discussed in Sec. 5.1. Subsequently, the qth warp $\mathbf{y}(\mathbf{x}, \mathbf{w}_q)$ mapping the intensity image $\mathcal{I}_q$ to the template is used to warp the corresponding segmentation $\mathcal{L}_q$ into the template space as well. Finally, at every voxel n with location $\mathbf{x}_n$ in the template, one counts the frequency with which each label k occurred at that location across the Q warped label images:

$$\pi_{n,k} = \frac{\sum_{q=1}^{Q} \delta\left(\mathcal{L}_q(\mathbf{y}(\mathbf{x}_n, \mathbf{w}_q)) = k\right)}{Q}, \tag{5.5}$$

where $\delta(k = l)$ equals one if $k = l$ and zero otherwise. The resulting $\pi_{n,k}$ represents roughly the prior probability that label k occurs in voxel n in the template space: it varies spatially and always satisfies $\sum_{k=1}^{K} \pi_{n,k} = 1$ in all voxels. Examples of such prior probability maps derived from manual segmentation of the brain's white matter, gray matter and CSF are shown in Fig. 5.3.

Once the template $\mathcal{T}$ and the prior probabilities $\pi_{n,k}$ have been computed,



(a)

(b)

Figure 5.3: Probabilistic atlas of the main brain tissue types using affine registration (a) and deformable registration (b). The atlases represent spatially varying prior probability maps of white matter, gray matter, CSF, and everything else. Bright and dark intensities correspond to high and low probabilities, respectively.

they can be used to segment a new image $\mathcal{I}$ as follows. First, the geometric transformation between $\mathcal{T}$ and $\mathcal{I}$ is computed, which is then applied to the prior probability maps, yielding interpolated prior probabilities $\pi_{n,k}$ for every class k in every voxel n in $\mathcal{I}$. These prior probabilities $\pi_{n,k}$ are then used in place of the generic π_k in every equation of the model-based segmentation models of Chapter 3, yielding voxel classifications that no longer depends solely on the voxels' local intensity alone, but also on their spatial location. Furthermore, the priors $\pi_{n,k}$ unambiguously associate segmentation classes to pre-defined anatomical structures, and can be used to automatically initialize the iterative update equations of the EM optimizers, even in multi-channel data (vector-valued voxel intensities) where initialization is otherwise difficult. Finally, the spatial priors are also typically used to discard voxels that are of no interest, such as muscle, skin, or fat in brain MR scans. As a result, the use of the spatial priors $\pi_{n,k}$ contributes greatly to the overall robustness and practical value of the model-based segmentation models discussed in Chapter 3.

5.2.2 Label propagation

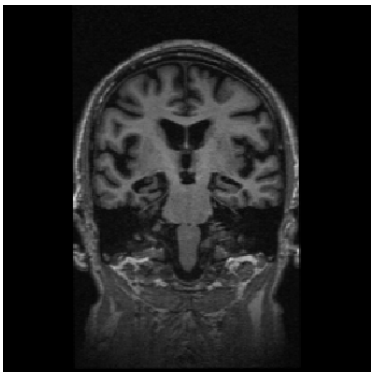
In so-called *label propagation*, an image $\mathcal{I}$ is segmented by computing a geometrical transformation $\mathbf{y}(\mathbf{x}_n, \mathbf{w}_q)$ that maps each voxel location $\mathbf{x}_n$ in $\mathcal{I}$ into a corresponding location in the qth pre-segmented image $\mathcal{I}_q$. Each voxel's label is then simply propagated from the manual segmentation $\mathcal{L}_q$ associated with $\mathcal{I}_q$, i.e., the label assigned to voxel n in $\mathcal{I}$ is given by

$$\mathcal{L}(\mathbf{x}_n) \quad \leftarrow \quad \mathcal{L}_q(\mathbf{y}(\mathbf{x}_n, \mathbf{w}_q)).$$

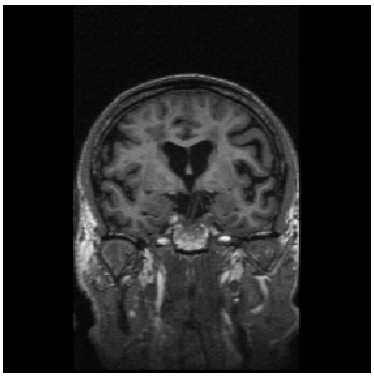
This process is illustrated in Fig. 5.4.

Of course, the same procedure can be followed for each of the Q pairs $\{\mathcal{I}_q, \mathcal{L}_q\}$, yielding Q possible segmentations of the same image $\mathcal{I}$. In practice, it is found that *combining* all Q segmentations in some way yields a consensus segmentation that is of a much higher quality than any of the Q original segmentations considered individually. Although there are many possible ways to combine Q segmentations of the same image, a very simple but still effective method is so-called *majority voting*, where each voxel is assigned to the label that occurred most frequently among all Q individual segmentations:

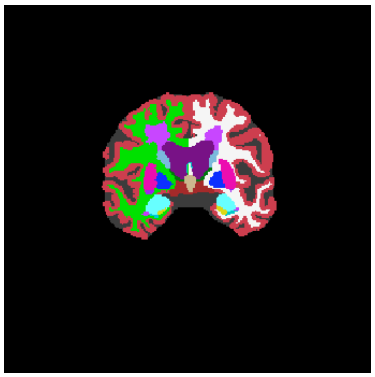
$$\mathcal{L}(\mathbf{x}_n) \quad \leftarrow \quad \arg\max_k \left[\sum_{q=1}^{Q} \delta\left(\mathcal{L}_q(\mathbf{y}(\mathbf{x}_n, \mathbf{w}_q)) = k\right) \right].$$



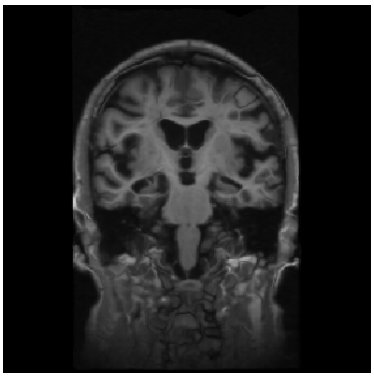
(a)



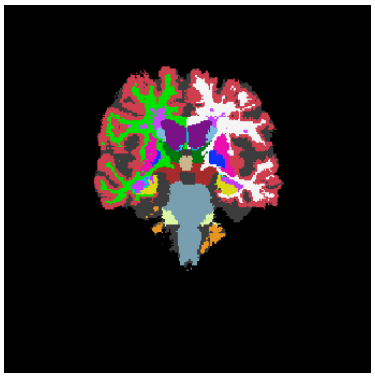
(b)



(c)



(d)



(e)

Figure 5.4: In label propagation an image (a) is automatically segmented by taking an image from another subject (b) that has been manually segmented (c), computing the deformation that warps the pre-segmented image to the to-be-segmented image (d), and applying the same deformation to the manual segmentation (e). The result (e) is an automatic segmentation of the to-be-segmented image (a).

Chapter 6

Validation

An important aspect of developing medical image analysis algorithms is demonstrating that the resulting algorithms actually work. This is called *validation*, and it often entails quantitatively evaluating how automatically generated results compare to those obtained by a trained human expert.

Although validation is vital in both *registration* and *segmentation*, in this chapter we will consider only the most important validation strategies for segmentation applications. Specifically, we will discuss two different validation scenarios: in the first scenario automatic segmentations are compared directly to manual segmentations that are considered to perfectly reflect the underlying anatomy, whereas in the second scenario we first need to estimate the underlying anatomy from several imperfect manual segmentations.

6.1 Validation against a known ground truth

Consider a scenario where someone has developed a new automated segmentation algorithm, and wants to demonstrate its performance. For a number of different images that are representative of what is encountered in the target application area, (s)he has access to manual segmentations that have been carefully performed by a trained human expert. The task now is to evaluate how well the automatically generated segmentations compare to the manual ones.

For the remainder of this section, we will concentrate on how to compare a single manual segmentation with a corresponding automated one, keeping in mind that in practice one would analyze dozens of different cases and summarize the results, for instance by averaging the performance over the individual cases.

6.1.1 Confusion matrix, sensitivity, and specificity

To establish notation, let $\mathbf{t} = (t_1, \ldots, t_N)^{\mathrm{T}}$ denote a manual segmentation of an image with N voxels, where $t_n \in \{0, 1\}$ denotes the label assigned to voxel with index n. We will refer to $\mathbf{t}$ as the *ground truth* segmentation. By convention,

$t_n = 1$ if the voxel belongs to the structure of interest (so-called "foreground"), and $t_n = 0$ otherwise ("background"). Similarly, the automated algorithm to be evaluated generates a segmentation $\mathbf{s} = (s_1, \ldots, s_N)^{\mathrm{T}}, s_n \in \{0, 1\}$ that is (hopefully) similar to $\mathbf{t}$, but not exactly the same.

The number of true positives (TP) is defined as the number of voxels that are assigned to the *foreground* in both $\mathbf{s}$ and $\mathbf{t}$, i.e.,

$$TP = \sum_{n=1}^{N} s_n t_n. \tag{6.1}$$

Similarly, the number of true negatives (TN) is defined as the number of voxels assigned to the *background* in both $\mathbf{s}$ and $\mathbf{t}$:

$$TN = \sum_{n=1}^{N} (1 - s_n)(1 - t_n). \tag{6.2}$$

In contrast to these counts of voxels where both segmentations agree with one another, the number of false positives (FP) and the number of false negatives (FN) refer to segmentation errors made by the automated algorithm compared to the ground truth manual segmentation: in the former case, voxels are erroneously assigned to the foreground when they really belong to the background, and in the latter case the exact opposite occurs:

$$FP = \sum_{n=1}^{N} s_n (1 - t_n) \tag{6.3}$$

and

$$FN = \sum_{n=1}^{N} (1 - s_n) t_n. \tag{6.4}$$

The concepts of true positives, true negatives, false positives and false negatives are illustrated in Fig. 6.1.

The four quantities defined in (6.1)– (6.4) can be summarized in a 2×2 table called the *confusion matrix*, defined as

		t	
		0	1
s	0	TN	FN
	1	FP	TP

.

For segmentations that correspond perfectly with the ground truth, the off-diagonal elements FN and FP in the confusion matrix will be zero.

We can also define the following *relative* quantities, ranging between 0 and 1:

- The **true positive rate** $\frac{\mathrm{TP}}{\mathrm{TP+FN}}$ expresses the fraction of foreground voxels that were correctly identified as such.

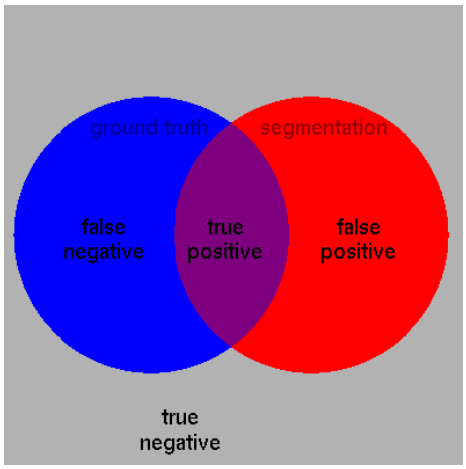


Figure 6.1: Illustration of the concepts true and false positives and negatives. The "ground truth" segmentation **t** is shown in blue, and the to-be-evaluated segmentation **s** in red.

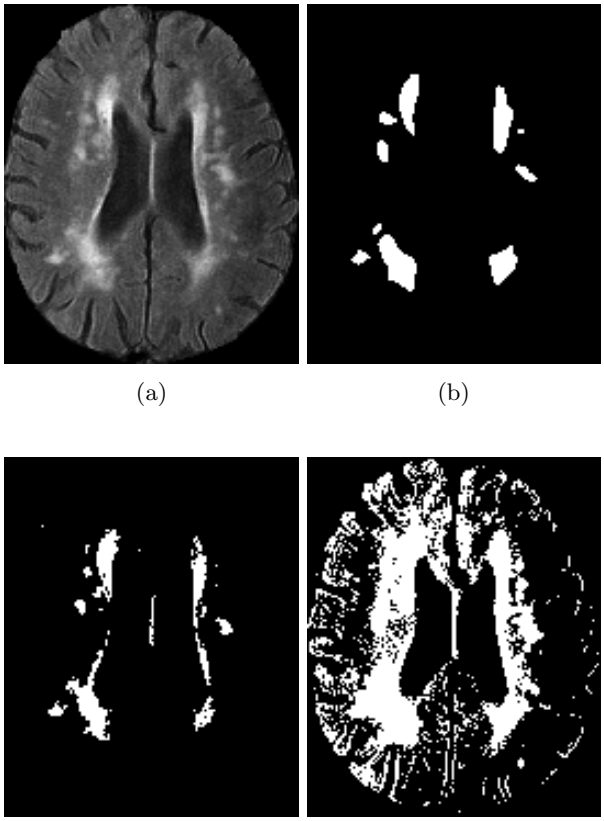
- The **true negative rate** $\frac{TN}{TN+FP}$ expresses the fraction of background voxels that were correctly identified as such.
- The **false positive rate** $\frac{FP}{TN+FP}$ expresses the fraction of background voxels that were incorrectly identified as foreground.
- The **false negative rate** $\frac{FN}{TP+FN}$ expresses the fraction of foreground voxels that were incorrectly identified as background.

The *true positive rate* is also commonly referred to as the **sensitivity** of a segmentation, and the *true negative rate* as the **specificity**. For segmentations that correspond perfectly with the ground truth, they will both have the maximum value of 1.

6.1.2 ROC curve

In binary classifiers, aiming at correctly assigning binary labels to some input data, there is often a threshold value that determines the exact location of the classification boundary in feature space. In some image segmentation problems, for instance, a viable segmentation strategy might be to assign all voxels with an intensity above a certain threshold value to the foreground class, and the rest to the background. An illustration of this process is shown in Fig. 6.2, where some white matter affected by small vessel disease appears very bright in the MRI scan, and can therefore be segmented (to some extent) by simple thresholding.

As shown in Fig. 6.2, the exact threshold value that is used plays a decisive role in the quality of the resulting segmentations. If the threshold value is chosen too high, only very bright voxels are selected, resulting in a very low *false*



(a) (b)

(c) (d)

Figure 6.2: In MR images acquired with the so-called FLAIR protocol, lesions in the white matter appear bright and can be segmented, to some extent, by simple intensity thresholding. Shown are a skull-stripped FLAIR scan (a); a manual segmentation that functions as the ground truth $\mathbf{t}$ (b); a segmentation $\mathbf{s}$ obtained by thresholding with a high threshold value (c); and with a low threshold value (d).

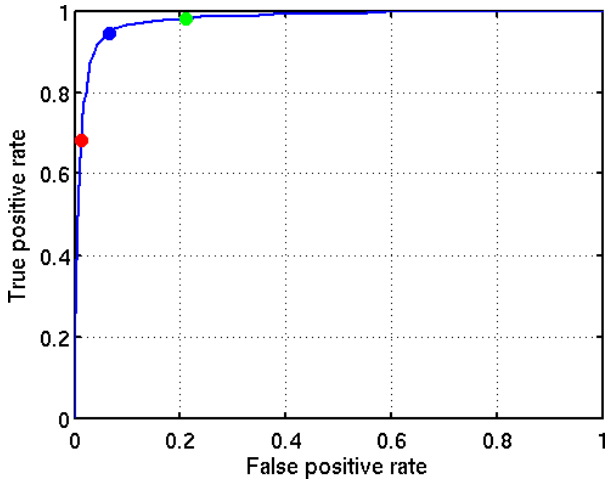


Figure 6.3: The ROC curve obtained by intensity thresholding the MR scan of Fig. 6.2a using a wide range of threshold values, and comparing the results to the manual segmentation shown in Fig. 6.2b. The red dot corresponds to the segmentation shown in Fig. 6.2c, the green dot to Fig. 6.2d, and the blue dot to the threshold setting that minimizes the Euclidean distance to the perfect classifier (point (0,1)).

positive rate (which is good because it means very few voxels in the background are erroneously selected), but also in a low *true positive rate* (which is bad because it means many foreground voxels have been missed). On the other hand, choosing the threshold value too low results in an excellent *true positive rate* but at the expensive of an elevated *false positive rate*.

For each setting of the threshold value, we can obtain a pair (*false positive rate*, *true positive rate*) by comparing the resulting segmentation to a manual segmentation that is considered to be the ground truth (shown in Fig. 6.2b). The plot of these pairs over a whole range of threshold values is called the receiver operating characteristic (ROC), or simply ROC curve. The ROC curve corresponding to the thresholding experiment of Fig. 6.2 is shown in Fig. 6.3. Note that the point (0,1) corresponds to the (unattainable) perfect classifier: it classifies all foreground and background voxels correctly. In practice, the best threshold setting is always a compromise between the *false positive rate* and the *true positive rate* performance. If we do not know anything about the cost of misclassification (of either type) or the prior distribution of the classes, one strategy is to choose the threshold value that minimizes the Euclidean distance between (0,1) (the perfect classifier) and the corresponding location on the ROC curve.

6.1.3 Dice score

An often used segmentation performance metric that summarizes the overall spatial overlap between a segmentation $\mathbf{s}$ and a ground truth segmentation $\mathbf{t}$ is the so-called *Dice score*. It is defined as simply the volume of those voxels that are deemed foreground in both segmentations simultaneously, divided by the mean volume of the foreground in both segmentations. Since the volume of the foreground in $\mathbf{s}$ is given by $\mathrm{TP} + \mathrm{FP}$, and by $\mathrm{TP} + \mathrm{FN}$ for $\mathbf{t}$, the Dice score can be computed as

$$\begin{aligned} \mathrm{Dice} &= \frac{\mathrm{TP}}{\frac{(\mathrm{TP}+\mathrm{FP})+(\mathrm{TP}+\mathrm{FN})}{2}} \\ &= \frac{2\mathrm{TP}}{2\mathrm{TP}+\mathrm{FP}+\mathrm{FN}}. \end{aligned}$$

It always lies between 0 (no spatial overlap between $\mathbf{s}$ and $\mathbf{t}$ at all) and 1 (perfect correspondence), and penalizes both false negatives and false positives at the same time. It is therefore very easy to report and interpret, making it a popular overall segmentation performance metric in the medical image segmentation literature.

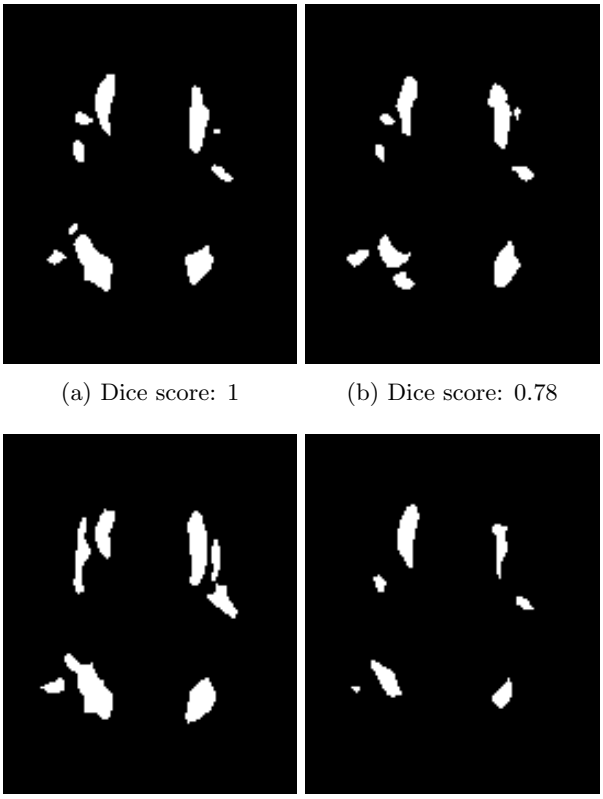
Several examples of alternate manual segmentations of the MR data of Fig. 6.2a are shown in Fig. 6.4, along with their Dice overlap scores with the first manual segmentation (Fig. 6.4a), which was considered the ground truth $\mathbf{t}$ for this purpose. Note that lower spatial correspondence (e.g., Fig. 6.4d) indeed results in a lower Dice overlap score.

6.2 Estimating the ground truth

So far, we have assumed that a single manual segmentation performed by a human expert is the perfect ground truth segmentation, i.e., corresponds perfectly to the underlying biology. In practice, however, there is often considerable disagreement between even highly trained experts on what the perfect segmentation of a given medical image should be. We already saw an example of this in Fig. 6.4, where different human raters segmented the same MRI scan and differed quite a bit in their judgment, in this case because the diffuseness of the lesions makes deciding on exact boundary locations particularly challenging.

In scenarios like this, it is not clear which ground truth segmentations we should compare the results of new automated algorithms to. We know that some human experts might produce more accurate segmentations than others: some might "over-segment" (i.e., have high sensitivity but low specificity), others might "under-segment" (high specificity but low sensitivity), while yet others might just be sloppy (both low sensitivity and specificity).

It turns out we can estimate the underlying ground truth $\mathbf{t}$ from M imperfect manual segmentations by explicitly modeling the errors human experts are likely to make [12]. Let $\mathbf{s}_m = (s_{1,m}, \ldots, s_{N,m})^{\mathrm{T}}$ denote the mth available segmentation, where $s_{n,m} \in \{0, 1\}$ indicates whether or not voxel n is assigned



(a) Dice score: 1 (b) Dice score: 0.78

(c) Dice score: 0.77 (d) Dice score: 0.66

Figure 6.4: Four alternate manual segmentations of the MR scan of Fig. 6.2a. The Dice scores are computed with respect to the first manual segmentation (a).

to the foreground by rater m. Assuming that rater m has sensitivity p_m and specificity q_m, and that (s)he makes labeling errors in each voxel independently, we have

$$p(\mathbf{s}_m|\mathbf{t}, p_m, q_m) = \prod_{n=1}^{N} p(s_{n,m}|t_n, p_m, q_m) \tag{6.5}$$

for the probability of $\mathbf{s}_m$, where

$$p(s|t, p, q) = \begin{cases} p^s(1-p)^{(1-s)} & \text{if } t = 1 \\ q^{(1-s)}(1-q)^s & \text{if } t = 0 \end{cases} . \tag{6.6}$$

(6.6) simply re-expresses the definition of *sensitivity* and *specificity*: if the ground truth label is 1, there is a probability p that the segmentation label will also be 1. Similarly, if the ground truth label is 0, the segmentation label will also be 0 with probability q.

Let $\mathbf{S} = (\mathbf{s}_1, \ldots, \mathbf{s}_M)$ denote all M available segmentations, and similarly $\boldsymbol{\theta} = (p_1, \ldots, p_M, q_1, \ldots, q_M)^{\mathrm{T}}$ all the parameters of the model, consisting of the sensitivity and specificity of all raters, respectively. Since we can assume that each rater makes his/her segmentation independently of the other raters, we have that

$$\begin{aligned} p(\mathbf{S}|\mathbf{t}, \boldsymbol{\theta}) &= \prod_{m=1}^{M} p(\mathbf{s}_m|\mathbf{t}, p_m, q_m) \\ &= \prod_{m=1}^{M} \prod_{n=1}^{N} p(s_{n,m}|t_n, p_m, q_m) \\ &= \prod_{n=1}^{N} p(\mathbf{s}_n|t_n, \boldsymbol{\theta}), \end{aligned} \tag{6.7}$$

where we have defined

$$p(\mathbf{s}_n|t_n, \boldsymbol{\theta}) = \prod_{m=1}^{M} p(s_{N,m}|t_n, p_m, q_m) \quad \text{with} \quad \mathbf{s}_n = (s_{n,1}, \ldots s_{n,M})^{\mathrm{T}} \tag{6.8}$$

for mathematical convenience later on.

In order to complete the model, we also specify a prior $p(\mathbf{t})$ that expresses our prior expectations about the ground truth segmentation $\mathbf{t}$. Similar to the prior used in the Gaussian mixture model (cf. (3.5)), we will use a simple prior of the form $p(\mathbf{t}) = \prod_{n=1}^{N} \pi_{t_n}$, where π_1 and $\pi_0 = (1 - \pi_1)$ govern the frequency with which voxels are expected to belong to foreground and background, respectively. In the remainder, we will assume that reasonable estimates of these parameters are known in advance, and keep their values fixed throughout.

As was the case with the model-based segmentation models discussed in Chapter 3, suitable values for the model parameters $\boldsymbol{\theta}$ can be obtained through maximum likelihood estimation. Also here we will perform the optimization

$$\hat{\boldsymbol{\theta}} = \arg\max_{\boldsymbol{\theta}} \log p(\mathbf{S}|\boldsymbol{\theta})$$

by using the EM algorithm, i.e., by repeatedly constructing a lower bound $Q(\boldsymbol{\theta}|\tilde{\boldsymbol{\theta}})$ that touches the log likelihood function at the current parameter estimate $\tilde{\boldsymbol{\theta}}$, and maximizing that lower bound, until convergence. Since the likelihood function is given by

$$\begin{aligned} p(\mathbf{S}|\boldsymbol{\theta}) &= \sum_{\mathbf{t}} p(\mathbf{S}|\mathbf{t},\boldsymbol{\theta})p(\mathbf{t}) \\ &= \sum_{\mathbf{t}} \left[\prod_{n=1}^{N} p(\mathbf{s}_n|t_n,\boldsymbol{\theta}) \prod_{n=1}^{N} \pi_{t_n}\right] \\ &= \prod_{n=1}^{N} \Big[p(\mathbf{s}_n|t_n=0,\boldsymbol{\theta})\pi_0 + p(\mathbf{s}_n|t_n=1,\boldsymbol{\theta})\pi_1\Big], \end{aligned}$$

we have that

$$\begin{aligned} \log p(\mathbf{S}|\boldsymbol{\theta}) &= \sum_{n=1}^{N} \log \Big[p(\mathbf{s}_n|t_n=0,\boldsymbol{\theta})\pi_0 + p(\mathbf{s}_n|t_n=1,\boldsymbol{\theta})\pi_1\Big] \\ &= \sum_{n=1}^{N} \log \left[\left(\frac{p(\mathbf{s}_n|t_n=0,\boldsymbol{\theta})\pi_0}{\omega_{n,0}}\right)\omega_{n,0} + \left(\frac{p(\mathbf{s}_n|t_n=1,\boldsymbol{\theta})\pi_1}{\omega_{n,1}}\right)\omega_{n,1}\right] \\ &\geq \underbrace{\sum_{n=1}^{N} \left[\omega_{n,0}\log\left(\frac{p(\mathbf{s}_n|t_n=0,\boldsymbol{\theta})\pi_0}{\omega_{n,0}}\right) + \omega_{n,1}\log\left(\frac{p(\mathbf{s}_n|t_n=1,\boldsymbol{\theta})\pi_1}{\omega_{n,1}}\right)\right]}_{Q(\boldsymbol{\theta}|\tilde{\boldsymbol{\theta}})}, \end{aligned}$$

for any pair of weights $\{\omega_{n,0},\omega_{n,1}\}$ in each voxel that satisfy $\omega_{n,0}+\omega_{n,1}=1$ and $\omega_{n,0},\omega_{n,1}\geq 0$ (the last step uses (3.28)). In order for $Q(\boldsymbol{\theta}|\tilde{\boldsymbol{\theta}})$ to additionally touch the log likelihood function at $\tilde{\boldsymbol{\theta}}$, these weights have to be chosen so that

$$\omega_{n,0} \propto p(\mathbf{s}_n|t_n=0,\tilde{\boldsymbol{\theta}})\pi_0 \qquad \text{and} \qquad \omega_{n,1} \propto p(\mathbf{s}_n|t_n=1,\tilde{\boldsymbol{\theta}})\pi_1, \tag{6.9}$$

as can be easily verified by substituting these weights into the definition of the lower bound and observing that then $Q(\tilde{\boldsymbol{\theta}}|\tilde{\boldsymbol{\theta}}) = \log p(\mathbf{S}|\tilde{\boldsymbol{\theta}})$.

The EM algorithm now dictates that the parameter estimate $\tilde{\boldsymbol{\theta}}$ be updated to the parameter vector that maximizes the lower bound. Re-writing the lower bound as

$$\begin{aligned} Q(\boldsymbol{\theta}|\tilde{\boldsymbol{\theta}}) &= \sum_{m=1}^{M} \left[\left(\sum_{n=1}^{N} \omega_{n,0}(1-s_{n,m})\right)\log q_m + \left(\sum_{n=1}^{N} \omega_{n,0}s_{n,m}\right)\log(1-q_m)\right] \\ &+ \sum_{m=1}^{M} \left[\left(\sum_{n=1}^{N} \omega_{n,1}s_{n,m}\right)\log p_m + \left(\sum_{n=1}^{N} \omega_{n,1}(1-s_{n,m})\right)\log(1-p_m)\right] \\ &+ \sum_{n=1}^{N} \left[\omega_{n,0}\log\left(\frac{\pi_0}{\omega_{n,0}}\right) + \omega_{n,1}\log\left(\frac{\pi_1}{\omega_{n,1}}\right)\right] \end{aligned} \tag{6.10}$$

and requiring that

$$\frac{\partial Q(\boldsymbol{\theta}|\tilde{\boldsymbol{\theta}})}{\partial \boldsymbol{\theta}} = 0$$

yields

$$\tilde{p}_m \leftarrow \frac{\sum_{n=1}^{N} \omega_{n,1} s_{n,m}}{\sum_{n=1}^{N} \omega_{n,1}} \tag{6.11}$$

$$\tilde{q}_m \leftarrow \frac{\sum_{n=1}^{N} \omega_{n,0} (1 - s_{n,m})}{\sum_{n=1}^{N} \omega_{n,0}} \tag{6.12}$$

for the updates of the model parameters.

In summary, the EM parameter optimizer iteratively computes the probability with which each voxel belongs to the foreground or background based on the available segmentations and the current estimates of each rater's sensitivity and specificity ((6.9)), and then updates each rater's sensitivity and specificity accordingly ((6.11) and (6.12)). Interesting, if some rater is estimated to have low sensitivity and specificity, that rater's segmentation is automatically down-weighted in the estimation of the foreground/background assignment probabilities, making the algorithm effectively focus on the most trustable segmentations only.

Upon convergence of the EM algorithm, an estimate of the ground truth corresponding to the estimated parameters $\hat{\boldsymbol{\theta}}$ can be found by looking for the maximum a posteriori ground truth

$$\hat{\mathbf{t}} = \arg\max_{\mathbf{t}} p(\mathbf{t}|\mathbf{S}, \hat{\boldsymbol{\theta}}),$$

which is obtained by assigning each voxel n to the foreground if $\omega_{n,1} \geq 0.5$ and to the background otherwise. This estimated ground truth can then be used as an unbiased reference segmentation to compare automated segmentation results to.

Fig. 6.5 shows the underlying ground truth $\hat{\mathbf{t}}$ estimated from the four manual segmentations shown in Fig. 6.4. The raters' sensitivity $\hat{p}_m$ and specificity $\hat{q}_m$ were estimated by the algorithm to be 0.867 and 0.997 for the segmentation of Fig. 6.4a, 0.782 and 0.998 for Fig. 6.4b , 0.935 and 0.988 for Fig. 6.4c , and 0.512 and 0.999 for Fig. 6.4d. The absolute values of all the specificity estimates are high because of the sheer size of the background; it is their relative magnitudes that really matter.

Figure 6.5: Estimated ground truth $\hat{\mathbf{t}}$ from the four manual segmentations of Fig. 6.4. The parameter π_1 was set to the average fraction of foreground voxels across all four manual segmentations, and $\pi_0 = (1 - \pi_1)$.

Bibliography

[1] M. Unser, "Splines: A perfect fit for signal and image processing," *IEEE Signal Processing Magazine*, vol. 16, no. 6, pp. 22–38, 1999.

[2] P. H. Schönemann, "A generalized solution of the orthogonal procrustes problem," *Psychometrika*, vol. 31, no. 1, pp. 1–10, 1966.

[3] P. Thévenaz, T. Blu, and M. Unser, "Interpolation revisited [medical images application]," *IEEE Transactions on medical imaging*, vol. 19, no. 7, pp. 739–758, 2000.

[4] W. M. Wells III, P. Viola, H. Atsumi, S. Nakajima, and R. Kikinis, "Multimodal volume registration by maximization of mutual information," *Medical image analysis*, vol. 1, no. 1, pp. 35–51, 1996.

[5] F. Maes, A. Collignon, D. Vandermeulen, G. Marchal, and P. Suetens, "Multimodality image registration by maximization of mutual information," *IEEE transactions on Medical Imaging*, vol. 16, no. 2, pp. 187–198, 1997.

[6] D. Greig, B. Porteous, and A. Seheult, "Exact maximum a posteriori estimation for binary images," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 51, no. 2, pp. 271–279, 1989.

[7] T. Jaakkola, *Advanced Mean Field Methods: Theory and Practice*, ch. Tutorial on Variational Approximation Methods. The MIT Press, 2001.

[8] D. Hunter and K. Lange, "A tutorial on MM algorithms," *The American Statistician*, vol. 58, no. 1, pp. 30–37, 2004.

[9] C. M. Bishop, *Pattern Recognition and Machine Learning*. springer, 2006.

[10] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[11] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989.

[12] S. Warfield, K. Zou, and W. Wells, "Simultaneous truth and performance level estimation (STAPLE): An algorithm for the validation of image segmentation," *IEEE Transactions on Medical Imaging*, vol. 23, no. 7, pp. 903–921, 2004.