22525 Course Note

Medical Image Analysis

Koen Van Leemput
Rasmus Larsen

Kongens Lyngby 2020

Technical University of Denmark
Department of Health Technology
www.healthtech.dtu.dk

Contents

Chapter 1

Fitting image functions

In this chapter we will review some methods for fitting 2- or 3-dimensional functions, since this will be needed on several occasions throughout the remainder of the course. In particular, we will be concerned with situations where we have made (noisy) observations of the function of interest and where we can assume that the function of interest is smooth. In the exercise we will encounter such a situation in the context of for background variation correction in magnetic resonance imaging.

1.1 The linear regression model

The linear regression model aims at predicting an output $y \in \mathbb{R}$ from a vector of inputs $\mathbf{x} = (x_1, x_2, \ldots, x_p)^{\mathrm{T}} \in \mathbb{R}^p$. It is assumed that the regression function $f(\mathbf{x})$ is linear or that it can be approximated with a linear function, i.e.,

$$f(\mathbf{x}) = \beta_0 + \sum_{j=1}^{p} x_j \beta_j. \tag{1.1}$$

Here, the β_j's are the unknown coefficients. These are usually estimated from a training set of observed inputs and outputs, i.e., $\{\mathbf{x}_i, y_i\}_{i=1}^{N}$, where $\mathbf{x}_i =$

$(x_{i1}, x_{i2}, \ldots, x_{ip})^{\mathrm{T}}$ is the vector of observed inputs for the ith case, and y_i the corresponding output.

Further, assume that the noise is i.i.d. Gaussian, i.e.,

$$y = f(\mathbf{x}) + \epsilon, \quad \epsilon \in N(0, \sigma^2). \tag{1.2}$$

The usual least squares estimator picks the coefficients - $\boldsymbol{\beta} = (\beta_0, \beta_1, \ldots, \beta_p)^{\mathrm{T}}$ - that minimizes the residual sum-of-squares (RSS), i.e.,

$$\begin{aligned} \mathrm{RSS}(\boldsymbol{\beta}) &= \sum_{i=1}^{N} (y_i - f(\mathbf{x}_i))^2 & (1.3) \\ &= \sum_{i=1}^{N} (y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2. & (1.4) \end{aligned}$$

The RSS is conveniently expressed in vector-matrix notation by introducing a $N \times (p+1)$ matrix $\mathbf{X}$ where the ith row consist of an initial 1 followed by the elements of the ith input vector, i.e.,

$$\mathbf{X} = \begin{pmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1p} \\ 1 & x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{N1} & x_{N2} & \cdots & x_{Np} \end{pmatrix} \tag{1.5}$$

and a vector $\mathbf{y} = (y_1, y_2, \ldots, y_N)^{\mathrm{T}}$ of all the outputs. Then the residual sum-of-squares is given by

$$\mathrm{RSS} = \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|^2 = (\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^{\mathrm{T}}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}). \tag{1.6}$$

This function is quadratic in $\boldsymbol{\beta}$ and differentiation yields

$$\begin{aligned} \frac{\partial \mathrm{RSS}}{\partial \boldsymbol{\beta}} &= -2\mathbf{X}^{\mathrm{T}}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) \\ \frac{\partial^2 \mathrm{RSS}}{\partial \boldsymbol{\beta}^2} &= 2\mathbf{X}^{\mathrm{T}}\mathbf{X} \end{aligned}$$

If we assume that $\mathbf{X}$ has full rank, then $\mathbf{X}^{\mathrm{T}}\mathbf{X}$ is positive definite, and a unique minimal point is found by setting the first derivative to zero and solving for $\boldsymbol{\beta}$, i.e.,

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y}. \tag{1.7}$$

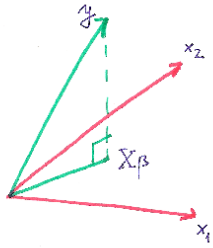
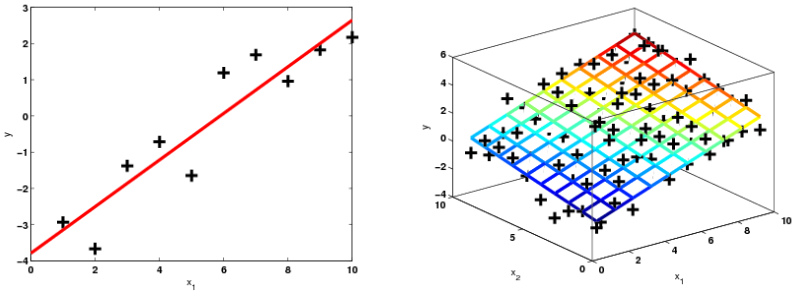


Figure 1.1: The least square estimator is found as the projection of $\mathbf{y}$ onto the subspace spanned by the columns of $\mathbf{X}$



(a) (b)

Figure 1.2: (a) linear fit to a 1D data set; (b) linear fit to a 2D data set.

By inserting this into Eq. (1.1) the estimated function values become

$$\hat{\mathbf{y}} = \mathbf{X}\hat{\boldsymbol{\beta}} = \mathbf{X}(\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y}. \tag{1.8}$$

$\mathbf{H} = \mathbf{X}(\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}$ is called the *hat matrix* because it puts the "hat" on $\mathbf{y}$. Geometrically this procedure is illustrated in Fig. 1.1. $\hat{\mathbf{y}}$ is the projection of $\mathbf{y}$ into the column space of $\mathbf{X}$. Geometrically, it is obvious that the residual sum-of-squares are minimal when the vector of residuals $\mathbf{y} - \mathbf{X}\boldsymbol{\beta}$ is orthogonal to the space spanned by the the columns of $\mathbf{X}$, i.e.,

$$\mathbf{X}^{\mathrm{T}}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) = 0 \tag{1.9}$$

yielding the solution in Eq. (1.7) (when $\mathbf{X}$ has full rank).

1.2 Penalized linear regression

By including a term penalizing the size of the coefficients in the RSS from Eq. (1.3) we get the ridge regression [1, 2] estimator

$$\hat{\boldsymbol{\beta}}^{\text{ridge}} = \arg\min_{\boldsymbol{\beta}} \left\{ \sum_{i=1}^{N} (y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \right\} \tag{1.10}$$

where $\lambda \geq 0$ is a regularization parameter controlling the penalty. If we replace the inputs by their standardized (i.e., zero mean and unit variance) versions - $(x_{ij} - \bar{x}_j)/s_j$, we obtain 1) that the β's act on variable of the same "scale" making the use of one common λ more reasonable; and 2) that the intercept is estimated by the mean of the outputs $\bar{y}$, and the remaining coefficients $\boldsymbol{\beta}$ are estimated by minimization of

$$\text{PRSS}(\lambda) = (\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^{\text{T}}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) + \lambda\boldsymbol{\beta}^{\text{T}}\boldsymbol{\beta} \tag{1.11}$$

Here $\mathbf{X}$ does not include the column of ones as before but only the p columns of standardized inputs. The solution becomes

$$\hat{\boldsymbol{\beta}}^{\text{ridge}} = (\mathbf{X}^{\text{T}}\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^{\text{T}}\mathbf{y}. \tag{1.12}$$

For the ordinary linear model the trace of the hat-matrix $\mathbf{H}$ in Eq. (1.8) equals p - the degrees of freedom of the model. It turns out that the trace of the projection matrix for the ridge is the effective degrees of freedom of the ridge model, i.e.,

$$\hat{\mathbf{y}} = \mathbf{X}\hat{\boldsymbol{\beta}}^{\text{ridge}} = \mathbf{X}(\mathbf{X}^{\text{T}}\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^{\text{T}}\mathbf{y} = \mathbf{H}(\lambda)\mathbf{y} \tag{1.13}$$

$$\text{H}(\lambda) = \mathbf{X}(\mathbf{X}^{\text{T}}\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^{\text{T}} \tag{1.14}$$

$$\text{df}(\lambda) = \text{tr}(\mathbf{X}(\mathbf{X}^{\text{T}}\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^{\text{T}}) \tag{1.15}$$

For $\lambda = 0$ $\text{df}(\lambda) = p$ corresponding to the ordinary linear model. $\lim_{\lambda\to\infty} df(\lambda) = 0$ corresponds to a constant model (where all β's have shrunk to 0).

1.3 Linear basis function models

The models for regression considered so far are not only linear functions of the parameters $\boldsymbol{\beta}$, but also of the input variables x_{ip}. This latter property imposes

significant limitations on the models - for instance in the 1-dimensional case, we can only fit a straight line to a collection of measurements as in figure 1.2(a). In order to obtain more flexible models, we here extend the class of models by considering linear combinations of smooth, nonlinear functions of the input variables, of the form

$$f(\mathbf{x}) = \sum_{m=1}^{M} \beta_m \phi_m(\mathbf{x}), \tag{1.16}$$

where $\phi_m(\mathbf{x})$ are known as *basis functions*. Because of the nonlinearity of the basis functions, the function $f(\mathbf{x})$ is now a nonlinear function of the input vector $\mathbf{x}$. However it is still a linear function of the parameters $\boldsymbol{\beta}$, which makes regression no more difficult than it was with the linear functions used before.

In order to obtain smooth regression models, the basis functions $\phi_m(\mathbf{x})$ are chosen to vary smoothly with changes in the input variables $\mathbf{x}$. Typical choices in medical image analysis include sine or cosine functions (figure 1.3), or so-called *uniform B-spline* basis functions. The latter consist of shifted copies of a symmetrical, bell-shaped function that is constructed (in the one-dimensional case) from the (n+1)-fold convolution of a rectangular pulse B_0 [3]:

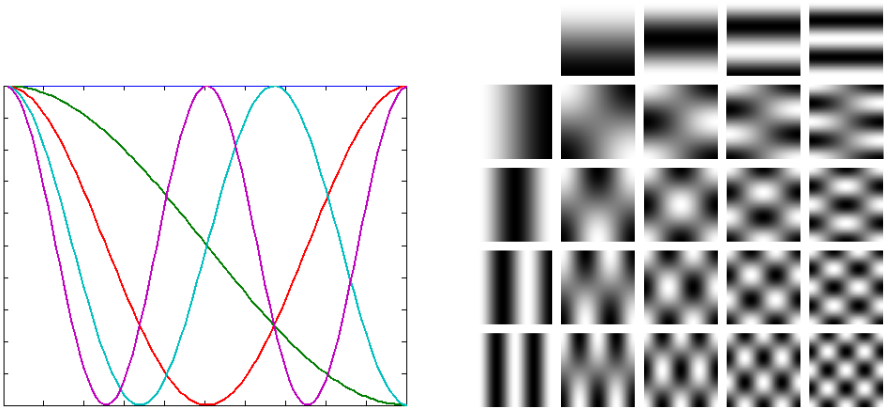
$$B_0(x) = \begin{cases} 1, & -\frac{1}{2} \leq x \leq \frac{1}{2} \\ \frac{1}{2}, & |x| = \frac{1}{2} \\ 0, & \text{otherwise} \end{cases} \tag{1.17}$$

$$B_n(x) = \underbrace{B_0 * B_0 \ldots B_0(x)}_{\text{(n+1) times}}. \tag{1.18}$$

The B-splines of degrees 0 to 3 are shown in figure 1.4(a). Especially the so-called *cubic* B-spline (n=3) has a number of very desirable numerical properties and is often used for interpolation purposes [3]; the basis functions associated with this cubic B-spline are shown in figure 1.4(b).

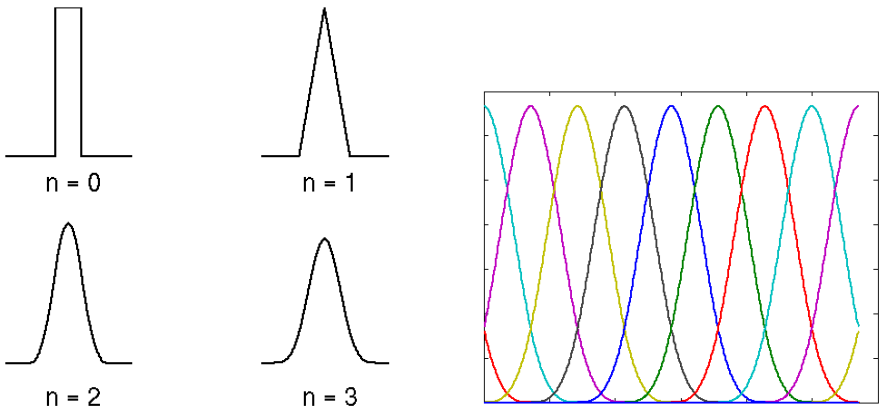
Once an appropriate choice for the basis functions $\phi_m(\mathbf{x})$ has been made, the parameter vector $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_M)^{\mathrm{T}}$ minimizing the residual sum of squares

$$\sum_{i=1}^{N} (y_i - f(\mathbf{x}_i))^2 = (\mathbf{y} - \boldsymbol{\Phi}\boldsymbol{\beta})^{\mathrm{T}} (\mathbf{y} - \boldsymbol{\Phi}\boldsymbol{\beta}) \tag{1.19}$$



(a) (b)

Figure 1.3: (a) Cosine basis functions in 1D; (b) cosine basis functions in 2D.



(a) (b)

Figure 1.4: (a) B-splines of degree 0 to 3; (b) cubic B-spline basis functions.

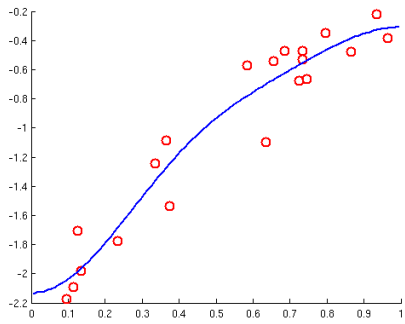


Figure 1.5: 1D regression using the nonlinear basis functions shown in figure 1.3(a).

is sought, where we have defined

$$\boldsymbol{\Phi} = \begin{pmatrix} \phi_1(\mathbf{x}_1) & \phi_2(\mathbf{x}_1) & \cdots & \phi_M(\mathbf{x}_1) \\ \phi_1(\mathbf{x}_2) & \phi_2(\mathbf{x}_2) & \cdots & \phi_M(\mathbf{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(\mathbf{x}_N) & \phi_2(\mathbf{x}_N) & \cdots & \phi_M(\mathbf{x}_N) \end{pmatrix}.$$

Or, more generally, its penalized version:

$$(\mathbf{y} - \boldsymbol{\Phi}\boldsymbol{\beta})^{\mathrm{T}}(\mathbf{y} - \boldsymbol{\Phi}\boldsymbol{\beta}) \; + \; \lambda\boldsymbol{\beta}^{\mathrm{T}}\boldsymbol{\Theta}\boldsymbol{\beta}, \tag{1.20}$$

is minimized, where $\boldsymbol{\Theta}$ is a $M \times M$ positive-semidefinite matrix that penalizes certain parameter vectors $\boldsymbol{\beta}$. The parameter vector minimizing Eq. (1.20) is found by differentation to be

$$\hat{\boldsymbol{\beta}} = (\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi} + \lambda\boldsymbol{\Theta})^{-1}\boldsymbol{\Phi}^{\mathrm{T}}\mathbf{y}. \tag{1.21}$$

Inserting this into Eq. (1.16) yields

$$\hat{\mathbf{y}} = \boldsymbol{\Phi}(\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi} + \lambda\boldsymbol{\Theta})^{-1}\boldsymbol{\Phi}^{\mathrm{T}}\mathbf{y} \tag{1.22}$$

for the estimated function values. Comparing this result to Eq. (1.13) shows that ridge regression is a special case of the linear basis function model, namely when the basis functions are chosen to be the input variables themselves, and $\boldsymbol{\Theta} = \mathbf{I}$.

An example of linear regression using the basis functions of figure 1.3(a) and $\lambda = 0$ is shown in figure 1.5. As before, $\mathrm{tr}(\boldsymbol{\Phi}(\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi} + \lambda\boldsymbol{\Theta})^{-1}\boldsymbol{\Phi}^{\mathrm{T}})$ yields the effective degrees of freedom of the model.

1.4 The cubic smoothing spline

In the previous section we obtained smooth regression models by explicitly representing the models as linear combinations of smooth basis functions. An alternative formulation is to not impose any such limitations on the functional form of $f(\mathbf{x})$, but instead merely discourage non-smooth behavior by penalizing large values of second order derivatives of $f(\mathbf{x})$. In this section, we will analyze these models for functions of one variable $f(x)$ only; section 1.5 will deal with higher-dimensional cases.

Consider the following minimization problem:

$$\text{PRSS}(f, \lambda) = \sum_{i=1}^{N} (y_i - f(x_i))^2 + \lambda \int (f''(t))^2 dt. \tag{1.23}$$

Remarkably, it turns out that the solution to this problem is a function of the form

$$f(x) = \sum_{m=1}^{M} \beta_m \phi_m(x), \tag{1.24}$$

with basis functions $\phi_m(x)$ that are the so-called cubic B-spline basis functions. If the input variables x_i are all uniformly spaced, these basis functions are shifted copies of the uniform cubic B-spline $B_3(x)$ defined in Eq. 1.18, as depicted in figure 1.4(b). In general, however, the basis functions are *non-uniform* cubic B-spline basis functions $B_m(x; \xi)$ corresponding to a sequence of "knots" ξ that consist of the ordered set of inputs x_i, $i = 1, \ldots, N$. Functions of this form can be shown to be third-order polynomials within each knot interval, and their first and second order derivatives are continuous at the knot locations.

Let $\mathbf{B}$ be the $N \times M$ matrix of M cubic spline basis functions evaluated at the N input points x_i with knot sequence ξ, then the $\text{PRSS}(f, \lambda)$ can be expressed

$$\text{PRSS}(f, \lambda) = (\mathbf{y} - \mathbf{B}\boldsymbol{\beta})^{\mathrm{T}} (\mathbf{y} - \mathbf{B}\boldsymbol{\beta}) + \lambda \boldsymbol{\beta}^{\mathrm{T}} \boldsymbol{\Omega}_{\mathbf{B}} \boldsymbol{\beta} \tag{1.25}$$

where $\{\boldsymbol{\Omega}_{\mathbf{B}}\}_{jk} = \int B_j''(t; \xi) B_k''(t; \xi) dt$. This corresponds to the form of Eq. (1.20) with $\boldsymbol{\Phi} = \mathbf{B}$ and $\boldsymbol{\Theta} = \boldsymbol{\Omega}_{\mathbf{B}}$, and therefore the solution is

$$\hat{\boldsymbol{\beta}} = (\mathbf{B}^{\mathrm{T}} \mathbf{B} + \lambda \boldsymbol{\Omega}_{\mathbf{B}})^{-1} \mathbf{B}^{\mathrm{T}} \mathbf{y} \tag{1.26}$$

with estimated function values

$$\begin{aligned} \hat{\mathbf{y}} &= \mathbf{B}(\mathbf{B}^{\mathrm{T}} \mathbf{B} + \lambda \Omega_{\mathbf{B}})^{-1} \mathbf{B}^{\mathrm{T}} \mathbf{y} \\ &= \mathbf{S}_{\lambda} \mathbf{y}. \end{aligned} \tag{1.27}$$

$\mathbf{S}_\lambda$ is called the smoother matrix and by analogy to previous sections we define the effective degrees of freedom to

$$\mathrm{df}_\lambda = \mathrm{tr}(\mathbf{S}_\lambda). \tag{1.28}$$

A simple way of choosing the smoothing parameter is to fix the desired df and use Eq. (1.28) to specify λ. This provides an easy and intuitive way of specifying the model complexity.

1.5 Thin plate splines

As an immediate generalization of the cubic smoothing spline in two and three dimensions we consider the following regularization term

$$J_p(f) = \int_{\mathbb{R}^p} \sum_{i=1}^{p} \sum_{j=1}^{p} \left[\frac{\partial^2 f}{\partial u_i \partial u_j} \right]^2 d\mathbf{u} \tag{1.29}$$

i.e., in two dimensions

$$J_2(f) = \int_{\mathbb{R}^2} \left[\frac{\partial^2 f}{\partial u_1^2} \right]^2 + 2 \left[\frac{\partial^2 f}{\partial u_1 \partial u_2} \right]^2 + \left[\frac{\partial^2 f}{\partial u_2^2} \right]^2 d\mathbf{u}. \tag{1.30}$$

Consider the smoothing problem

$$\mathrm{PRSS}(f, \lambda) = \sum_{i=1}^{N} (y_i - f(\mathbf{x}_i))^2 + \lambda J_p(f). \tag{1.31}$$

Here the $\mathbf{x}_i \in \mathbb{R}^p$ are now points in p-dimensional space. It can be shown [4, 5] that the minimizing f is of the form

$$f(\mathbf{x}) = \beta_0 + \boldsymbol{\beta}_1^{\mathrm{T}} \mathbf{x} + \sum_{i=1}^{N} \alpha_i \eta_p(\|\mathbf{x} - \mathbf{x}_i\|), \tag{1.32}$$

where

$$\begin{aligned} \eta_2(r) &= \begin{cases} r^2 \log r & \text{if } r > 0 \\ 0 & \text{if } r = 0 \end{cases} \\ \eta_3(r) &= \|r\|^3. \end{aligned}$$

The solution has $N + p + 1$ parameters. Because we have only N observations $p + 1$ constraints must be enforced, i.e.,

$$\sum_{i=1}^{N} \alpha_i = 0 \quad \text{and} \quad \sum_{i=1}^{N} \alpha_i x_{ij} = 0, \quad \forall j = 1, \ldots, p. \tag{1.33}$$

We can restate the formulae above in matrix-vector notation. We introduce

$$\begin{aligned}
\boldsymbol{\beta} &= \begin{pmatrix} \beta_0 \\ \boldsymbol{\beta}_1 \end{pmatrix} \\
\boldsymbol{\alpha} &= \begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_N \end{pmatrix} \\
\mathbf{P} &= \begin{pmatrix} 1 & 1 & \cdots & 1 \\ \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_N \end{pmatrix}
\end{aligned}$$

and the $N \times N$ matrix $\mathbf{K}$ with elements $K_{ij} = \eta_p(\|\mathbf{x}_i - \mathbf{x}_j\|)$. Insertion in the objective function in Eq. (1.31) and the constraints in Eq. (1.33) yields

$$\begin{aligned}
\text{PRSS}(f, \lambda) &= \|\mathbf{y} - \mathbf{P}^{\mathrm{T}}\boldsymbol{\beta} - \mathbf{K}\boldsymbol{\alpha}\|^2 + \lambda \boldsymbol{\alpha}^{\mathrm{T}}\mathbf{K}\boldsymbol{\alpha} \\
\mathbf{P}\boldsymbol{\alpha} &= 0.
\end{aligned}$$

By differentiating and setting the objective equal to zero we obtain the following collected set of linear constraints needed to calculate the parameters $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ of the **smoothing thin plate spline**

$$\begin{pmatrix} \mathbf{K} + \lambda\mathbf{I} & \mathbf{P}^{\mathrm{T}} \\ \mathbf{P} & \mathbf{0} \end{pmatrix} \begin{pmatrix} \boldsymbol{\alpha} \\ \boldsymbol{\beta} \end{pmatrix} = \begin{pmatrix} \mathbf{y} \\ \mathbf{0} \end{pmatrix}. \tag{1.34}$$

In turns out that an **interpolating thin plate spline**, i.e., a function that exactly passes through all observations can be estimated by setting $\lambda = 0$ in Eq. (1.34).

When we the insert this in the objective function we obtain

$$\hat{\mathbf{y}} = \begin{pmatrix} \mathbf{K} & \mathbf{P}^{\mathrm{T}} \end{pmatrix} \begin{pmatrix} \hat{\boldsymbol{\alpha}} \\ \hat{\boldsymbol{\beta}} \end{pmatrix} = \begin{pmatrix} \mathbf{K} & \mathbf{P}^{\mathrm{T}} \end{pmatrix} \begin{pmatrix} \mathbf{K} + \lambda\mathbf{I} & \mathbf{P}^{\mathrm{T}} \\ \mathbf{P} & \mathbf{0} \end{pmatrix}^{-1} \begin{pmatrix} \mathbf{y} \\ \mathbf{0} \end{pmatrix} = \mathbf{S}_\lambda^* \begin{pmatrix} \mathbf{y} \\ \mathbf{0} \end{pmatrix}$$

The smoothing matrix – $\mathbf{S}_\lambda$ – is therefore found as the first N columns of $\mathbf{S}_\lambda^*$. Remember that an equivalent number of degrees of freedom of f can be found as the trace (sum of diagonal elements) of $\mathbf{S}_\lambda$

$$\text{df}_\lambda = \text{trace}(\mathbf{S}_\lambda).$$

The degrees of freedom are a characterization of the amount of smoothing applied and can be used to choose λ.

1.A Exercise I - MR bias field correction

Given the following set of four observations

i	y_i	$x_{i,1}$	$x_{i,2}$
1	3	1	1
2	1	1	-1
3	1	-1	1
4	2	-1	-1

- Compute the interpolating TPS ($\boldsymbol{\beta}$ and $\boldsymbol{\alpha}$).
- Plot the function f in a densely sampled grid over the region $[-1.5, 1.5] \times [-1.5, 1.5]$
- compute the smoothing TPS ($\boldsymbol{\beta}$ and $\boldsymbol{\alpha}$) for a sequence of λ's, i.e., (0.1, 1, 10, 100). For each λ plot f as described above and compute the equivalent degrees of freedom. Comment on the functions and the corresponding degrees of freedom. What are the minimum degrees of freedom ($\lambda = \infty$), what is the maximum ($\lambda = 0$)?

HINT! To calculate values $\mathbf{y}$ of f for several values of $\mathbf{x}$ we can use the matrix formulation

$$\mathbf{y} = \begin{pmatrix} \mathbf{K}_x & \mathbf{P}_x^{\mathrm{T}} \end{pmatrix} \begin{pmatrix} \boldsymbol{\alpha} \\ \boldsymbol{\beta} \end{pmatrix}.$$

The parameters $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ you should already have estimated. Now N_x is the number of (grid) points $\mathbf{x}$, $\mathbf{P}_x$ is a $3 \times N_x$ matrix similar to $\mathbf{P}$, and $\mathbf{K}_x$ is a $N_x \times N$ matrix with elements that need to be calculated.

Bias field correction

The image in Fig. 1.6 is severely corrupted by a bias field. We assume a multiplicative model

$$(\text{Image})(x) = (\text{True-image})(x) \times (\text{Bias-field})(x) + (\text{Noise})(x)$$

Now, assume that we have a series of fat pixels (bright tissue) with image values y_i and positions $\mathbf{x}_i$. Then the true image values for these pixels should be the same and therefore any variation is due to the bias field (and noise). So by fitting a smoothing TPS to these values we can estimate the bias field (up to

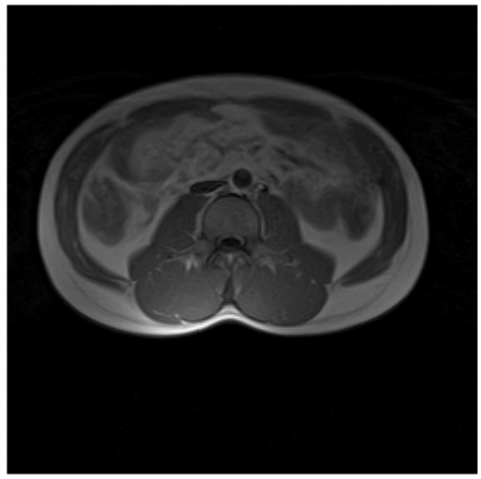


Figure 1.6: Abdominal MR scan with bias field

scaling parameter). The true image can then be recovered by division of the estimated bias field into the image.

The abdominal MR slice is found in the file `abdomen.mat`. I have also include an image `roi` containing a region of interest (ROI). The latter is useful when you apply the bias correction (apply the correction only inside the ROI to avoid unwanted scaling effects).

- record an observation series (~ 100) fat pixels (values + position)
- fit a smoothing spline with λ corresponding to 5, 10, 20 degrees of freedom.
- for the 3 solutions compute the smoothing spline at all image coordinates inside the region of interest
- for the 3 solutions plot the smoothing spline and compute the corrected image

Hints

Useful Matlab functions include:

- `ginput` an easy way to input clicked coordinates.
- `mldivide` $\mathbf{Ax} = \mathbf{b}$ is solved by $\mathbf{x} = \mathbf{A}\backslash\mathbf{b}$
- `meshgrid` generates a grid of x and y coordinates
- `surf` make surface plots

The following is useful when constructing the $\mathbf{K}$ matrix: to compute a matrix of squared pairwise distances between two sets of points $\mathbf{x}_a$ (size $N_a \times 2$, with the first and the second column containing x- and y-coordinates, respectively) and $\mathbf{x}_b$ (size $N_b \times 2$, same arrangement), do the following:

```
N_a = size( x_a, 1 );
N_b = size( x_b, 1 );

squaredDistance = zeros( N_a, N_b );
for coordinateNumber = 1 : 2
  squaredDistance = squaredDistance + ...
              ( repmat( x_a( :, coordinateNumber ), [ 1 N_b ] ) - ...
                repmat( x_b( :, coordinateNumber )', [ N_a 1 ] ) ).^2;
end
```

Given such a matrix we can compute the $\mathbf{K}$ matrix for the 2D case as

```
K = log( squaredDistance + eps ) .* squaredDistance / 2;
```

Chapter 2

Image registration

Image registration is the process of determining a geometrical transformation that aligns the points of an image with corresponding points in a reference coordinate frame. This reference coordinate frame is often given as another image. However, it can be any arbitrary object coordinate system.

Image registration adds value to medical images by enabling

- monitoring of change in the individual
- fusion of information from different sources in a clinically meaningful way
- comparison of one subject with others
- comparison of groups with others

Image registration occurs frequently in medical image analysis. In our notation an "image" can be both a two-dimensional and a three-dimensional array. Image registration is necessary when we have acquired images of the same patient using different imaging modalities. In Fig. 2.1 corresponding slices of the same brain imaged with a magnetic resonance scanner and a CT scanner, respectively, are shown. The two scanners have different spatial resolution, and the images have slightly different rotation angles with respect to the anatomy due to different

patient positioning in the scanners. If we want to consider the information in the images simultaneously we must provide a geometrical transformation between the two images. For cancer diagnostics a combination of CT and PET imaging is often used - the CT images provide the anatomical information and the PET images the functional information.

In image guided intervention and minimally invasive surgery optical images of surface structures and ultrasound images acquired intra-operatively must be registered to pre-operatively acquired MR and CT images providing the surgeon with an overview of detailed 3D anatomical structure.

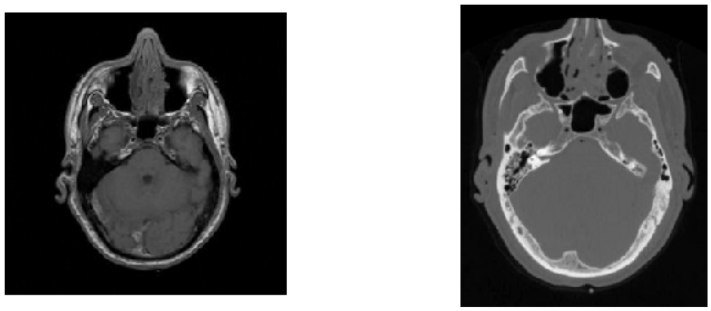


Figure 2.1: Illustration of an MR image and a CT image of the same brain. Notice how the MR image is rotated and of different size and resolution.

Image registration is very important when evaluating the progression of a disease or the effect of a treatment from images acquired at different time points, e.g., days, weeks, years apart.

Sometimes the second image may be a computer-based anatomical "atlas", e.g., in the form of a generic image of a particular anatomy segmented and labeled which allows for automated image interpretation by transferring image labels to the image under study.

And finally for population studies it is often of interest to compare images of many individuals, e.g., in order to quantify the biological variability.

2.1 Elements of image registration

The task of image registration can be phrased as the following [6]: Given a so-called reference $\mathcal{R}$ and a so-called template image $\mathcal{T}$, the basic idea is to find a reasonable geometrical transformation such that a transformed version of the template image becomes similar to the reference image.

Image registration consists of the following elements:

The geometrical transformation required to transform the template to the reference image. In some cases a rigid transformation, i.e., a translation and a rotation suffices, in other situations a non-linear geometrical transformation is necessary.

The similarity measure that describes the goodness of the registration.

The optimization algorithm that controls (determines the parameters of) the geometrical transformations to maximize similarity.

The regularization term securing that only reasonable transformations are obtained.

Each of the items listed is a research area in its own right. A great number of different algorithms and methods for each discipline exists. In the following sections we will take an overview of the most important algorithms and concepts used today.

2.2 Landmark based registration

We will begin by considering image registration in cases where the similarity measure is defined in relation to a set of corresponding landmarks in a pair of images. To be formal a landmark point can be defined as a point of correspondence within and between populations on each object.

In the literature landmarks have been known by various synonyms and been partitioned into various types. We will consider the three types defined by [7] in relation to biological shapes:

Anatomical landmark: a mark assigned by an expert that corresponds between objects in a biologically meaningful way.

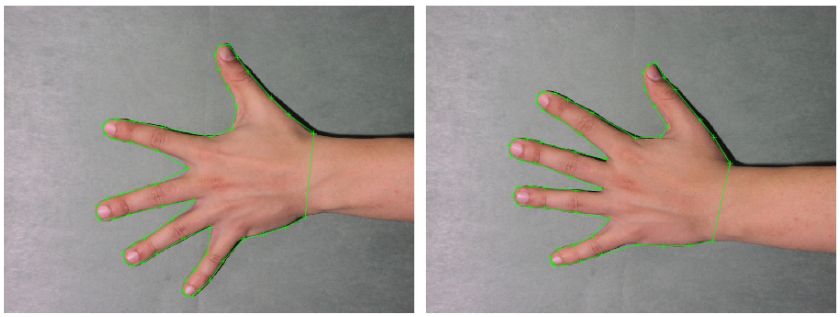


Figure 2.2: Two images of the same hand. Each hand is annotated with 56 landmarks. Some landmarks are anatomical landmarks placed at the finger joints, others are mathematical landmarks placed at points of maximum surface curvatures, and a few of the landmarks are pseudo landmarks placed by sampling part of the curves equidistantly.

Mathematical landmark: a mark that is located on a curve according to some mathematical or geometrical property (e.g., a point of maximum curvature).

Pseudo landmark: a mark that is constructed on a curve based on anatomical or mathematical landmarks (e.g., sampled equidistantly along an outline).

In Fig. 2.2 both anatomical, mathematical, and pseudo landmarks are shown on a set of hands.

Assume that two corresponding point sets have been extracted from a pair of images, i.e., $\mathbf{x}_i, \mathbf{y}_i \in \mathbb{R}^2$ or $\mathbb{R}^3$, $i = 1, \ldots, N$. The $\mathbf{x}_i$ coordinates belong to the space of the reference image $\mathcal{R}$ and the $\mathbf{y}_i$ coordinates belong to the space of the template image $\mathcal{T}$. We aim at determining the parameters $\mathbf{w} \in \mathbb{R}^m$ of a geometrical transformation $\mathbf{y}(\mathbf{x}; \mathbf{w}) : \mathbb{R}^2 \to \mathbb{R}^2$ or $\mathbb{R}^3 \to \mathbb{R}^3$ that map the coordinates of the reference image to the coordinates of the template image such that the sum of squared distances is minimized, i.e.,

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \sum_{i=1}^{N} \|\mathbf{y}(\mathbf{x}_i; \mathbf{w}) - \mathbf{y}_i\|^2 \tag{2.1}$$

We will consider a series of different types of geometrical mappings, i.e.,

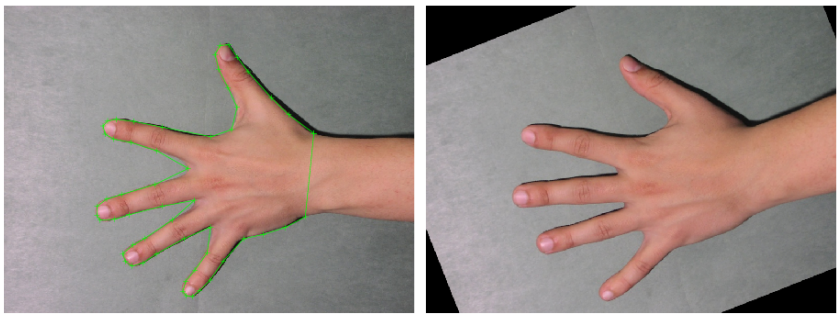


Figure 2.3: The hands from Fig. 2.2 after the first hand has been translated and rotated such that the sum-of-squared distances between landmarks are minimized.

Translation. In special cases it suffices to consider a translatory motion between image acquisitions.

A geometrical transformation consisting of a pure translatory motion $\mathbf{t} = (t_1, t_2)^{\mathrm{T}}$ in 2D and $\mathbf{t} = (t_1, t_2, t_3)^{\mathrm{T}}$ in 3D is represented like this

$$\mathbf{y}(\mathbf{x}; \mathbf{t}) = \mathbf{x} + \mathbf{t} \tag{2.2}$$

By differentiation of the objective function in Eq. (2.1) we obtain

$$\hat{\mathbf{t}} = \bar{\mathbf{y}} - \bar{\mathbf{x}} \tag{2.3}$$

where $\bar{\mathbf{x}} = \sum_{i=1}^{N} \mathbf{x}_i / N$ and $\bar{\mathbf{y}} = \sum_{i=1}^{N} \mathbf{y}_i / N$.

Translation + rotation = rigid transformation. The rigid transformation is often used when we need to compensate for different patient positioning in the same scanner at different times of imaging.

The rigid transformation consists of a translation $\mathbf{t} = (t_1, t_2)^{\mathrm{T}}$ in 2D ($\mathbf{t} = (t_1, t_2, t_3)^{\mathrm{T}}$ in 3D) and a rotation. The rotation is customarily represented by an orthogonal matrix $\mathbf{R}$, i.e., $\mathbf{R}^{\mathrm{T}}\mathbf{R} = \mathbf{I}$. Orthogonal matrices describe both rotations and rotations-reflections. Reflections can be excluded by requiring that $\det(\mathbf{R}) = 1$. The resulting transformation becomes

$$\mathbf{y}(\mathbf{x}; \mathbf{R}, \mathbf{t}) = \mathbf{R}\mathbf{x} + \mathbf{t} \tag{2.4}$$

In 2D the rotation matrix can be parameterized by the angle θ, i.e.,

$$\mathbf{R} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \tag{2.5}$$

In 3D we can for instance parametrize the rotation matrix by the Euler angles – angles of rotation around the respective coordinate axes θ_1, θ_2, and θ_3 – i.e.,

$$\mathbf{R} = \begin{pmatrix} \cos\theta_3 & -\sin\theta_3 & 0 \\ \sin\theta_3 & \cos\theta_3 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos\theta_2 & 0 & -\sin\theta_2 \\ 0 & 1 & 0 \\ \sin\theta_2 & 0 & \cos\theta_2 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_1 & -\sin\theta_1 \\ 0 & \sin\theta_1 & \cos\theta_1 \end{pmatrix}. \tag{2.6}$$

We introduce the centered landmark sets $\tilde{\mathbf{x}}_i = \mathbf{x}_i - \bar{\mathbf{x}}$ and $\tilde{\mathbf{y}}_i = \mathbf{y}_i - \bar{\mathbf{y}}$. By inserting these into the Eq. (2.1) the objective function becomes

$$L_{\mathrm{OBJ}} = \sum_{i=1}^{N} \|\mathbf{R}\tilde{\mathbf{x}}_i - \tilde{\mathbf{y}}_i + \mathbf{t} - (\bar{\mathbf{y}} - \mathbf{R}\bar{\mathbf{x}})\|^2. \tag{2.7}$$

Differentiation with respect to $\mathbf{t}$ yields $\mathbf{t} = \bar{\mathbf{y}} - \mathbf{R}\bar{\mathbf{x}}$ and inserting this in Eq. (2.7) we obtain

$$\begin{aligned} L_{\mathrm{OBJ}} &= \sum_{i=1}^{N} \|\mathbf{R}\tilde{\mathbf{x}}_i - \tilde{\mathbf{y}}_i\|^2 \\ &= \sum_{i=1}^{N} \left(\tilde{\mathbf{x}}_i^{\mathrm{T}}\mathbf{R}^{\mathrm{T}}\mathbf{R}\tilde{\mathbf{x}}_i + \tilde{\mathbf{y}}_i^{\mathrm{T}}\tilde{\mathbf{y}}_i - 2\tilde{\mathbf{x}}_i^{\mathrm{T}}\mathbf{R}^{\mathrm{T}}\tilde{\mathbf{y}}_i\right). \end{aligned} \tag{2.8}$$

Now, using $\mathbf{R}^{\mathrm{T}}\mathbf{R} = \mathbf{I}$ it can be shown[1] [8] that we obtain the minimizing $\mathbf{R}$ by performing a singular value decomposition of $\mathbf{H} = \sum_{i=1}^{N} \tilde{\mathbf{x}}_i\tilde{\mathbf{y}}_i^{\mathrm{T}} = \mathbf{U}\mathbf{D}\mathbf{V}^{\mathrm{T}}$, with $\mathbf{U}^{\mathrm{T}}\mathbf{U} = \mathbf{V}^{\mathrm{T}}\mathbf{V} = \mathbf{I}$, and – in the 3D case[2] – $\mathbf{D} = \mathrm{diag}(\lambda_1, \lambda_2, \lambda_3)$ with $\lambda_1 \geq \lambda_2 \geq \lambda_3$ – and setting

$$\begin{aligned} \hat{\mathbf{R}} &= \mathbf{V}\mathrm{diag}(1, 1, \det(\mathbf{V}\mathbf{U}))\mathbf{U}^{\mathrm{T}} \\ \hat{\mathbf{t}} &= \bar{\mathbf{y}} - \hat{\mathbf{R}}\bar{\mathbf{x}}. \end{aligned} \tag{2.9}$$

The diagonal matrix interposed between $\mathbf{V}$ and $\mathbf{U}$ ensures that we get a proper rotation, i.e., reflections are avoided.

[1] As a curiosity we mention that this problem is also called the ordinary Procrustes problem. Procrustes was a character from Greek mythology. He was an innkeeper who having only one guest bed fitted his guests to it by stretching or squeezing them as appropriate. The analogy refers to our "stretching" or "squeezing" of one set of landmarks to fit to the other set.

[2] In 2D we have $\mathbf{D} = \mathrm{diag}(\lambda_1, \lambda_2)$ and $\hat{\mathbf{R}} = \mathbf{V}\mathrm{diag}(1, \det(\mathbf{V}\mathbf{U}))\mathbf{U}^{\mathrm{T}}$.

Translation + rotation + isotropic scaling = similarity transformation. The similarity transformation adds isotropic scaling to the rigid transformation. This means that in addition to different patient positioning we can also compensate for different spatial resolution in various imaging modalities, i.e., registering MR to CT images. Also in population studies we often would like to compensate for different size of different patients' anatomy.

The similarity adds an isotropic scaling parameter $s > 0$ to the rigid transformation. The resulting transformation becomes

$$\mathbf{y}(\mathbf{x}; s, \mathbf{R}, \mathbf{t}) = s\mathbf{R}\mathbf{x} + \mathbf{t}. \tag{2.10}$$

Computations similar to those for the rigid transformation yield (in 3D)

$$\begin{aligned} \hat{\mathbf{R}} &= \mathbf{V}\mathrm{diag}(1, 1, \det(\mathbf{V}\mathbf{U}))\mathbf{U}^{\mathrm{T}} \\ \hat{s} &= \frac{\sum_{i=1}^{N} \tilde{\mathbf{x}}_i^{\mathrm{T}} \hat{\mathbf{R}}^{\mathrm{T}} \tilde{\mathbf{y}}_i}{\sum_{i=1}^{N} \tilde{\mathbf{x}}_i^{\mathrm{T}} \tilde{\mathbf{x}}_i} \\ \hat{\mathbf{t}} &= \bar{\mathbf{y}} - \hat{s}\hat{\mathbf{R}}\bar{\mathbf{x}}. \end{aligned} \tag{2.11}$$

Affine transformation. The affine transformation adds anisotropic scaling and skewing to the similarity transformation. It is often applied in order to compensate for faulty image acquisition. The resulting transformation becomes

$$\mathbf{y}(\mathbf{x}; \mathbf{A}, \mathbf{t}) = \mathbf{A}\mathbf{x} + \mathbf{t}, \tag{2.12}$$

where there are no restrictions on the elements of the matrix $\mathbf{A}$. The solution is obtained by standard linear least squares regression.

Thin plate spline based geometrical transformation The thin plate spline based geometrical transformation is usually applied in situations where the warp field is to be determined from given correspondences between a smaller number (< 1000) of landmark points in two images. Each coordinate k of the geometrical transformation is modeled as a smoothing thin plate spline transformation, i.e., according to Eq. 1.32 for coordinates in p-dimensional space

$$y^k(\mathbf{x}; \mathbf{w}) = \beta_0^k + (\boldsymbol{\beta}_1^k)^{\mathrm{T}}\mathbf{x} + \sum_{i=1}^{N} \alpha_i^k \eta_p(\|\mathbf{x} - \mathbf{x}_i\|) \tag{2.13}$$

where

$$\begin{aligned}\eta_2(r) &= \begin{cases} r^2 \log r & \text{if } r > 0 \\ 0 & \text{if } r = 0 \end{cases} \\ \eta_3(r) &= \|r\|^3.\end{aligned}$$

The solution is found as the solution to the linear system (cf. Eq.1.34):

$$\begin{pmatrix} \mathbf{K} + \lambda \mathbf{I} & \mathbf{P}^{\mathrm{T}} \\ \mathbf{P} & \mathbf{0} \end{pmatrix} \begin{pmatrix} \boldsymbol{\alpha}^k \\ \boldsymbol{\beta}^k \end{pmatrix} = \begin{pmatrix} \mathbf{y}^k \\ \mathbf{0} \end{pmatrix}.$$

In Fig. 2.4 an example of a thin plate spline warp is shown: the landmarks from two metacarpal bones are transformed using a pair of smoothing thin plate splines.

2.2.1 Fiducial localization and registration error

The term "fiducial point" is often used interchangeable with "landmark". The "fiducial localization error" (FLE) is a measure quantifying the uncertainty in the landmarks. Landmarks are often obtained by manual annotation. The manual annotation process is in itself a stochastic process where the same individual will annotate slightly differently when repeating the annotation (repeatability) and a different individual will deviate even more (reproducibility). The fiducial localization error obviously depends on the expertise of the annotating individual. However, it also depends on the image noise and the particular image structure to be annotated, i.e., at high curvature points the fiducial localization error is small in all directions, but along linear structures we may have a low fiducial error across the linear structure and high error along the linear structure.

Given J repeat attempts $\mathbf{x}_{i,j}, j = 1, \ldots, J$ at localizing a specific point $\mathbf{x}_i$ in an image (either by the same or a different individual), the fiducial localization error can be characterized by the covariance matrix

$$\boldsymbol{\Sigma}_{FLE} = \frac{1}{NJ} \sum_{i=1}^{N} \sum_{j=1}^{J} (\mathbf{x}_{i,j} - \boldsymbol{\mu}_i)(\mathbf{x}_{i,j} - \boldsymbol{\mu}_i)^{\mathrm{T}},$$

where $\boldsymbol{\mu}_i = \sum_{j=1}^{J} \mathbf{x}_{i,j}/J$ denotes the average location of the i^{th} landmark across the J localization attempts, and N is the number of landmarks that are being

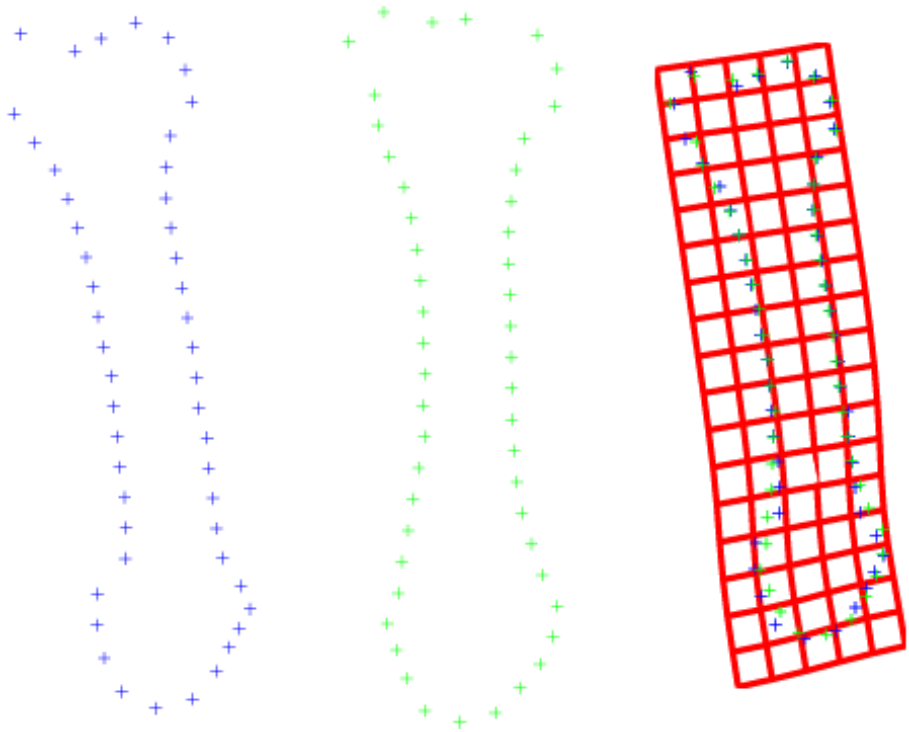


Figure 2.4: The two leftmost plots show the outlines of two metacarpal bones outlined by 50 landmarks each. A pair of smoothing thin plate splines are applied to geometrically transform the green landmarks to the blue landmarks. The resulting geometrical transformation is shown in the rightmost plot as a warped regular grid.

annotated. If the localization error is isotropic, i.e., similar and independent errors are typically made in different directions, the covariance matrix will be more or less diagonal. In the 2-dimensional case ($\mathbf{x}_i \in \mathbb{R}^2$):

$$\mathbf{\Sigma}_{FLE} \simeq \begin{pmatrix} \sigma^2_{FLE} & 0 \\ 0 & \sigma^2_{FLE} \end{pmatrix} \quad \text{with} \quad \sigma^2_{FLE} = \frac{1}{2NJ} \sum_{i=1}^{N} \sum_{j=1}^{J} \|\mathbf{x}_{i,j} - \boldsymbol{\mu}_i\|^2. \quad (2.14)$$

A related concept is the "fiducial registration error" (FRE), which quantifies how far corresponding points in the reference and template images remain removed from each other *after* applying point-wise registration. Again in 2D:

$$\sigma^2_{FRE} = \frac{1}{2N} \sum_{i=1}^{N} \|\mathbf{y}(\mathbf{x}_i; \hat{\mathbf{w}}) - \mathbf{y}_i\|^2, \quad (2.15)$$

where $\hat{\mathbf{w}}$ are the optimal parameters given by Eq. (2.1). (Note that these are also the parameters that yield the *minimal* fiducial registration error.)

2.2.2 Image transformation and interpolation

When we have estimated the parameters of the appropriate geometrical transformation by applying one of the algorithms shown above to a set of corresponding landmarks we must transform one of the images to the space of the other one.

The way we have designed our algorithms, we can for each (integer) image coordinate $\mathbf{x}$ in the reference image $\mathcal{R}$ compute the corresponding point $\mathbf{y}$ in the template image using the geometrical transformation $\mathbf{y}(\mathbf{x}; \mathbf{w})$ function. Now, the resulting $\mathbf{y}$ is likely to be a fractional number (pointing somewhere between pixels in the template image). Therefore we must find the template image value using a interpolation procedure, e.g., bi-(tri)-linear interpolation.

2.3 Intensity based registration

In a more realistic setting we will want to perform the image registration without the manual interaction of identifying sets of corresponding landmarks, i.e., based directly on the observed intensity patterns. The measure of dissimilarity between the intensity pattern of the transformed template image and the reference image must be chosen according to the nature of the images. In this section we will review a series of intensity based dissimilarity measures.

2.3.1 Sum-of-squared-differences and the correlation coefficient

If we can assume that the difference between the template and the reference after registration can be modelled by a zero-mean, constant variance Gaussian process, then the appropriate dissimilarity measure is the **sum-of-squared-differences (SSD)**[3], i.e.

$$\mathcal{D}_{\text{SSD}}(\mathbf{w}) \quad = \quad \frac{1}{2}\sum_{i\in\Omega}\left(\mathcal{T}(\mathbf{y}(\mathbf{x}_i;\mathbf{w})) - \mathcal{R}(\mathbf{x}_i)\right)^2. \tag{2.16}$$

Here, as before $\mathbf{y}(\mathbf{x}_i;\mathbf{w})$ is the geometrical transformation function returning the coordinate in the template image $\mathcal{T}$ corresponding to $\mathbf{x}_i$ in the reference image $\mathcal{R}$. $\mathbf{w}$ are the parameters of the geometrical transformation. The template refers to our model image and the reference to the image that we have (most recently) measured. Given this nature of the template and the reference, it is fair that the dissimilarity should be calculated across the sample positions $\mathbf{x}_i$ in $\mathcal{R}$. Ω denotes the space over which we calculate the dissimilarity. It could be the region of overlap of the two images or in some cases it could be restricted to a particular anatomy inside the images, i.e., for the CT and MR images in Fig. 2.1, Ω could be restricted to pixels/voxels inside the head (i.e., excluding the background) or brain voxels/pixels.

The assumption of Gaussian noise often works well, e.g., for registration of serial MR images[4].

In other situations, we see a violation of the assumption of constant variance of the noise. This would be the case if the measured intensity values were a realization of a Poisson process, e.g., counts as in PET images. For a Poisson distributed variable $X \in P(\lambda)$ we have that $E\{X\} = V\{X\} = \lambda$, i.e., the variance is higher in high intensity areas than in low intensity areas. This issue can often be resolved by a variance stabilizing transformation like the $\sqrt{\cdot}$ or the $\log\cdot$ prior to applying the SSD.

In some situations it can be assumed that intensity values in the template and the reference image are related by a linear transformation. This can be due to

[3] Sometimes in literature we see that the SSD is defined as an integral over the squared differences between two image functions. Here, we prefer to formulate the dissimilarity measures in terms of the sampled signals that we have measured.

[4] The noise in MR images is usually modelled by a Rice distribution which is approximately Gaussian for high intensity areas but deviates in low intensity areas. In spite of this fact SSD is often the choice dissimilarity for registration of MR images.

inaccurate calibration of the instruments used to acquire the image or simply by the fact that pixel quantization is different in the two images. In this case the appropriate dissimilarity measure is the **correlation coefficient**, i.e.,

$$\begin{aligned} \mathcal{D}_{\mathrm{CC}}(\mathbf{w}) &= \frac{\sum_{i\in\Omega}\left(\mathcal{T}(\mathbf{y}(\mathbf{x}_i;\mathbf{w}))-\bar{\mathcal{T}}\right)\left(\mathcal{R}(\mathbf{x}_i)-\bar{\mathcal{R}}\right)}{\sqrt{\sum_{i\in\Omega}\left(\mathcal{T}(\mathbf{y}(\mathbf{x}_i;\mathbf{w}))-\bar{\mathcal{T}}\right)^2\sum_{i\in\Omega}\left(\mathcal{R}(\mathbf{x}_i)-\bar{\mathcal{R}}\right)^2}} \\ \bar{\mathcal{T}} &= \frac{1}{|\Omega|}\sum_{i\in\Omega}\mathcal{T}(\mathbf{y}(\mathbf{x}_i;\mathbf{w})) \\ \bar{\mathcal{R}} &= \frac{1}{|\Omega|}\sum_{i\in\Omega}\mathcal{R}(\mathbf{x}_i). \end{aligned} \tag{2.17}$$

Standardization of the template and reference images to zero mean and unit variance before applying the SSD dissimilarity measure is equivalent to using the correlation coefficient dissimilarity measure on the raw data. This will often be desirable because of the simple calculations involved in computing the SSD.

Finally, a method for overcoming the problems of intensity inhomogeneity as seen for instance in the MR image in Fig. 1.6 should be mentioned. Obviously we could make a correction step first as was done in the bias correction exercise in Sect. 1.A. However, we can often circumvent such problems given simple assumptions. If we for instance assume that the bias fields are slowly varying, then they will not contribute much to the spatial derivatives of the image. In this case we can then carry out the registration on the image gradients, e.g.,

$$\mathcal{D}_{\mathrm{SSD}}(\mathbf{w}) = \frac{1}{2}\sum_{i\in\Omega}\left\|\nabla\mathcal{T}(\mathbf{y}(\mathbf{x}_i;\mathbf{w}))-\nabla\mathcal{R}(\mathbf{x}_i)\right\|^2. \tag{2.18}$$

This would be a sensible thing to do since gradients encode information of precise localization of edges and lines.

We could of course also employ other differential operators.

2.3.2 Information theoretic dissimilarities

When the image types of the template and the reference are different it may not be meaningful to use sum-of-squared-differences or the correlation coefficient as a dissimilarity measure. Consider for instance the CT and MR image pair in

Fig. 2.1. Here we can see that the skull appears bright in the CT image and dark in the MR image i.e., violating the assumption that the differences can be modelled by Gaussian noise. A series of dissimilarity measures derived from information theory are applicable in this situation.

First, we must consider the joint histogram or 2D histogram of a pair of images. In Fig. 2.5(a-b) the MR and CT images from Fig. 2.1 after a landmark based registration based on a similarity transformation are shown. In Fig. 2.5(c-e) we see the histograms of the MR and CT images as well as the joint histogram of the two images. In the joint histogram we can read out the number of pixels having a particular combination of intensities in the two images. If the histograms were normalized by the number of pixels in the region (Ω) they would be estimators of the corresponding probability distribution functions. Note that the resolution on the axis, i.e., the number of bins of the histograms is a user specified parameter. Here we have used 32 bins. Let the histogram values be $\mathrm{HIST}(j,k)$ then the estimator of the joint probability function is

$$\mathrm{PDF}(j,k) \quad = \quad \frac{\mathrm{HIST}(j,k)}{\sum_{j',k'} \mathrm{HIST}(j',k')}. \tag{2.19}$$

Now let us consider what happens when the reference and template move out of perfect registration. In Fig. 2.6 we see the joint histogram for the perfect registration of the MR and CT images in (a). In (b) the CT image has been translated 80 mm with respect to the MR image. We see that the joint histograms become "blurry": The clustered configuration we observed for the perfect registration becomes more diffuse.

This effect of the joint histogram diffusing with mis-registration is captured by the entropy of the joint histogram. The Shannon entropy is given by

$$H = -\sum_i p(i) \log p(i) \tag{2.20}$$

where $p(i)$ is the probability of event i. The entropy is minimal if only one event has a non-zero probability of occurrence (i.e., that event occurs with probability one), and it is maximal if all events have equal probability. In our image registration setting we can use the entropy as a dissimilarity measure because in perfect registration the joint histogram is concentrated to a few events (the entropy is small) and when we move out of registration the histogram becomes more diffuse and the entropy increases. Using our notation the joint

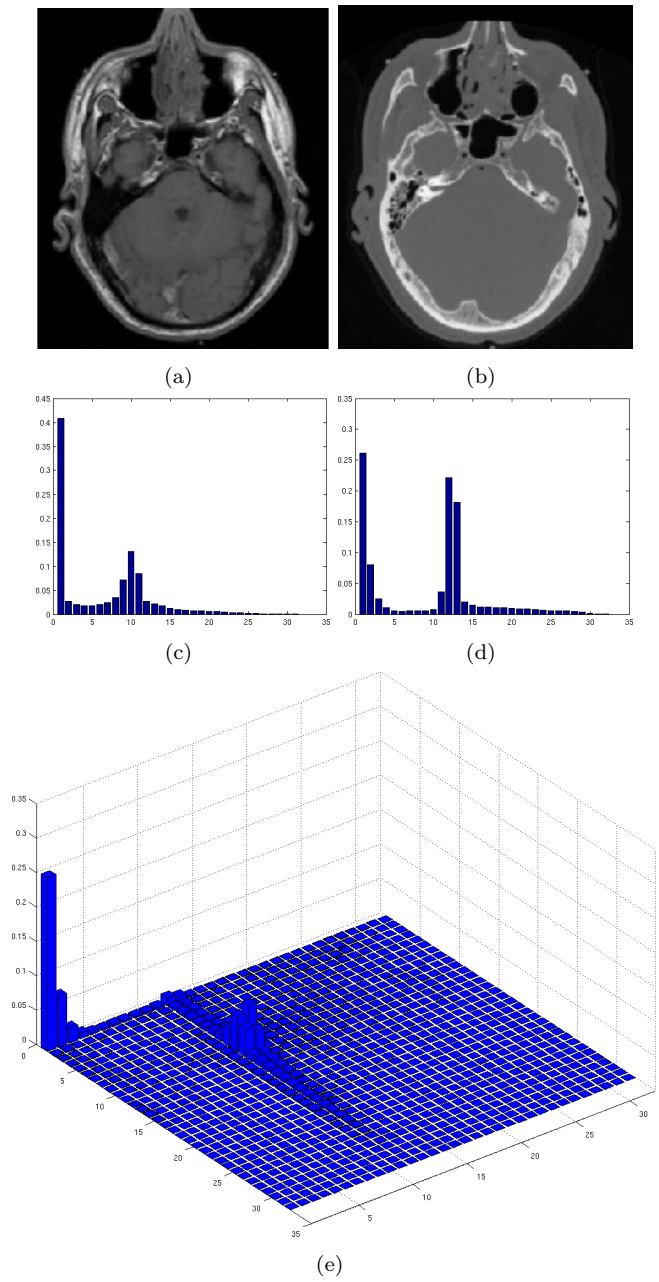
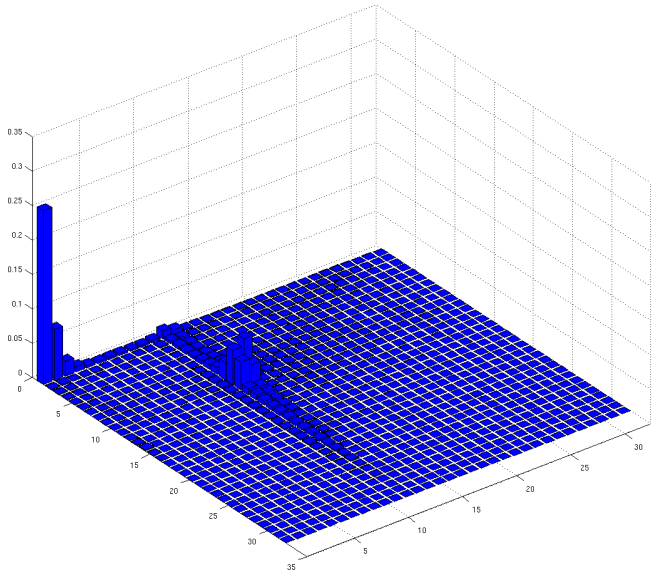
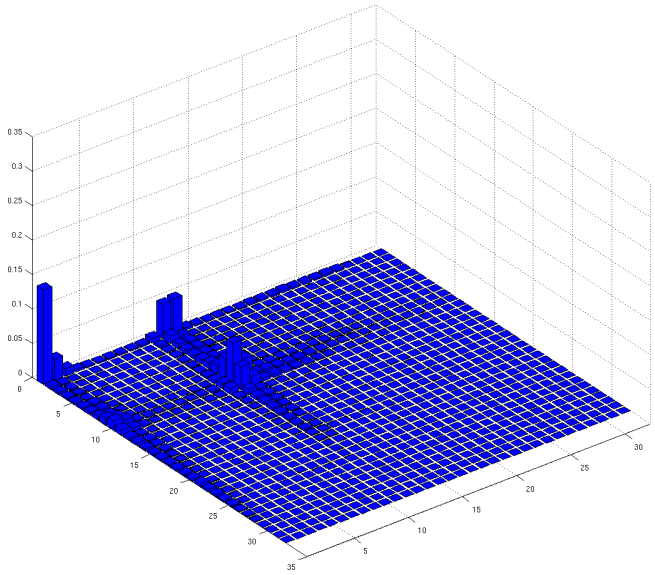


Figure 2.5: The MR and CT images from Fig. 2.1 after a landmark based registration based on a similarity transformation are shown in (a) and (b). (c) and (d) are the histogram of the MR and CT intensities, respectively. We have used 32 bins. (e) is the joint histogram of the MR and CT intensities.



(a)



(b)

Figure 2.6: (a) is the joint histogram for the perfect registration of the MR and CT images. In (b) the CT image has been translated 80 mm with respect to the MR image. We see that the joint histogram becomes "blurry": The clustered configuration we observed for the perfect registration becomes more diffuse.

entropy dissimilarity measure is

$$\mathcal{D}_{\mathrm{JE}}(\mathbf{w}) = H(\mathcal{T}(\mathbf{y}(\mathbf{x}_i;\mathbf{w})),\mathcal{R}(\mathbf{x}_i)) \tag{2.21}$$

$$H(\mathcal{T}(\mathbf{y}(\mathbf{x}_i;\mathbf{w})),\mathcal{R}(\mathbf{x}_i)) = -\sum_{j,k}\mathrm{PDF}(j,k)\log(\mathrm{PDF}(j,k)) \tag{2.22}$$

where the PDF is calculated for the $\mathcal{T}(\mathbf{y}(\mathbf{x}_i;\mathbf{w}))$ and $\mathcal{R}(\mathbf{x}_i)$ and $\mathbf{x}_i \in \Omega$, the region of joint overlap.

Now, the joint entropy has a downside that makes it unsuitable for image registration in some situations. Imagine that the two images only overlapped in a same-intensity region in both images. Then the joint entropy estimated in the overlapped region is exactly 0 and thus optimal.

The solution to this problem lies in balancing the joint entropy with the marginal entropies in each of the images. This is in information theory called the *mutual information*, i.e.,

$$\begin{aligned}
\mathcal{D}_{\mathrm{MI}} &= H(\mathcal{T}(\mathbf{y}(\mathbf{x}_i;\mathbf{w})),\mathcal{R}(\mathbf{x}_i)) - H(\mathcal{T}(\mathbf{y}(\mathbf{x}_i;\mathbf{w}))) - H(\mathcal{R}(\mathbf{x}_i)) \\
H(\mathcal{T}(\mathbf{y}(\mathbf{x}_i;\mathbf{w}))) &= -\sum_j\left[\Big(\sum_k \mathrm{PDF}(j,k)\Big)\log\Big(\sum_k \mathrm{PDF}(j,k)\Big)\right] \\
H(\mathcal{R}(\mathbf{x}_i)) &= -\sum_k\left[\Big(\sum_j \mathrm{PDF}(j,k)\Big)\log\Big(\sum_j \mathrm{PDF}(j,k)\Big)\right].
\end{aligned}$$

Experiments have shown that an even more robust combination is the so-called normalization mutual information, i.e.,

$$\mathcal{D}_{\mathrm{NMI}} = \frac{H(\mathcal{T}(\mathbf{y}(\mathbf{x}_i;\mathbf{w})),\mathcal{R}(\mathbf{x}_i))}{H(\mathcal{T}(\mathbf{y}(\mathbf{x}_i;\mathbf{w}))) + H(\mathcal{R}(\mathbf{x}_i))}.$$

For both the joint entropy, the mutual information, and the normalized mutual information a critical parameter is the number of bins used in the histograms.

2.3.3 Principal axes transform

The dissimilarity measures listed above all may result in multiple local minima. In fact, it may be that the optimum we are looking for is indeed a local minimum. The reason for this is that spurious minima (even the global minimum) can result from some uninteresting combinations of parameters of the geometrical transformation. In order to resolve this we are almost always faced with the task of providing an initial image registration by some heuristic method. Sometimes we know in advance, e.g., from the way the patient is placed in the scanner, that we are fairly close to the solution we are looking for.

In the following we will review one particular method of arriving at an initial estimate of a similarity transform matching two images. This method is called the **principal axes transform**. The principal axes transform is easily understood by a physical analogue. Imagine that the image intensities represent densities. Then we can compute the moments of these density distributions for each of the pair of images.

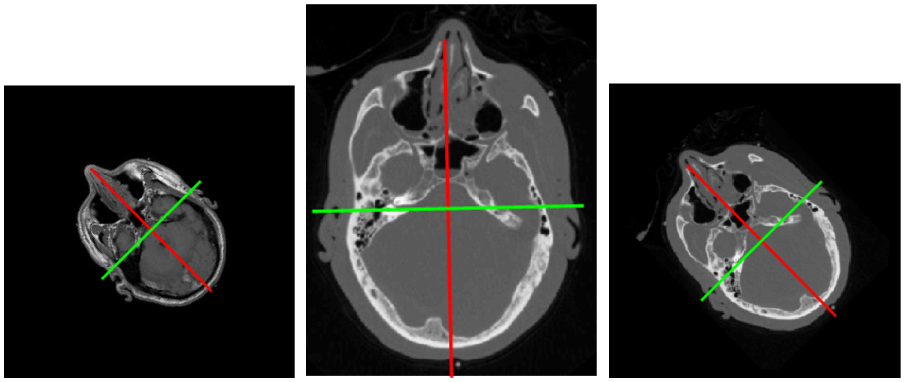
The formulae for estimating the center-of-mass and the covariance matrix are

$$\begin{aligned} \boldsymbol{\mu}_{\mathcal{R}} &= \frac{\sum_{\Omega_{\mathcal{R}}} \mathcal{R}(\mathbf{x}_i)\mathbf{x}_i}{\sum_{\Omega_{\mathcal{R}}} \mathcal{R}(\mathbf{x}_i)} \\ \boldsymbol{\Sigma}_{\mathcal{R}} &= \frac{\sum_{\Omega_{\mathcal{R}}} \mathcal{R}(\mathbf{x}_i)(\mathbf{x}_i - \boldsymbol{\mu}_{\mathcal{R}})(\mathbf{x}_i - \boldsymbol{\mu}_{\mathcal{R}})^{\mathrm{T}}}{\sum_{\Omega_{\mathcal{R}}} \mathcal{R}(\mathbf{x}_i)} \end{aligned} \tag{2.23}$$

$$\begin{aligned} \boldsymbol{\mu}_{\mathcal{T}} &= \frac{\sum_{\Omega_{\mathcal{T}}} \mathcal{T}(\mathbf{y}_i)\mathbf{y}_i}{\sum_{\Omega_{\mathcal{T}}} \mathcal{T}(\mathbf{y}_i)} \\ \boldsymbol{\Sigma}_{\mathcal{T}} &= \frac{\sum_{\Omega_{\mathcal{T}}} \mathcal{T}(\mathbf{y}_i)(\mathbf{y}_i - \boldsymbol{\mu}_{\mathcal{T}})(\mathbf{y}_i - \boldsymbol{\mu}_{\mathcal{T}})^{\mathrm{T}}}{\sum_{\Omega_{\mathcal{T}}} \mathcal{T}(\mathbf{y}_i)}. \end{aligned} \tag{2.24}$$

The difference between the centers-of-masses is an estimator for the translation that must be applied to reference image. The matrix of eigenvectors of the covariance matrices yields in each case a rotation matrix and the set of the square roots of the eigenvalues are anisotropic scaling factors for each of the principal axes. The method cannot provide a shearing component.

If we are only interested in an isotropic scaling we can in 2D for instance base this on a transformation that matches two points in each image: namely the centers of masses and the end points of the first eigenvector offset from the center-of-masses and scaled by the square root of the corresponding eigenvalues.



(a) (b) (c)

Figure 2.7: (a) MR image with principal axes; (b) CT image with principal axes; (c) transformed CT image using the principal axes transform.

An example is shown in Fig. 2.7.

2.4 Geometrical transformations

In Sec. 2.2 we already considered a series of geometrical transformations including translation, rigid transformation, Euclidean similarity transformation, and the affine transformation. In this section we will introduce some parameterized non-linear transformations. However, we will restrict ourselves to such geometrical transformations that can be represented by linear splines, such as the thin plate spline and the tensor B-spline. Moreover, we will put these geometrical transformation models on a canonical form. Putting the transformation on the same form eases the introduction to optimization in a later section.

The canonical form splits the geometrical transformation into an identity part and a deformation part, i.e.,

$$\mathbf{y}(\mathbf{x};\mathbf{w}) = \mathbf{x} + \mathbf{u}(\mathbf{x};\mathbf{w}). \tag{2.25}$$

In Fig. 2.8 a template image $\mathcal{T}$ in the upper left corner contains a black square. It is registered to a reference image $\mathcal{R}$ containing a black circle, shown in the upper right corner. In the lower left corner the resulting geometrical transformation

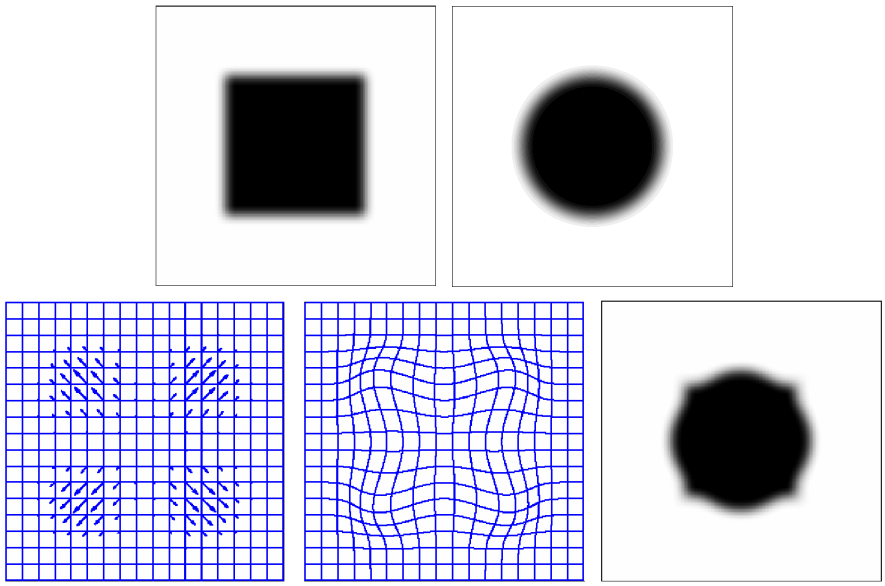


Figure 2.8: Illustration of the canonical form of a non-linear warp. From left to right, top to bottom: template image $\mathcal{T}$; reference image $\mathcal{R}$; deformation; warped grid; warped template image.

is visualized as a set of deformation vectors applied to a set of grid points. The grid points represent the identity part $\mathbf{x}$ of the transformation and the vectors the deformation part $\mathbf{u}(\mathbf{x};\cdot)$. In the lower middle is the resulting warped grid $\mathbf{y}(\mathbf{x};\cdot)$. The lower right is the template shown in the coordinate system of the reference image.

2.4.1 The affine transformation

Now, consider the affine transformation in Eq. (2.12) parameterized by

$$\begin{aligned} \mathbf{A} &= \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \\ \mathbf{t} &= \begin{pmatrix} t_1 \\ t_2 \\ t_3 \end{pmatrix} \\ \mathbf{y}(\mathbf{x}_i;\mathbf{A},\mathbf{t}) &= \mathbf{A}\mathbf{x}_i + \mathbf{t}. \end{aligned}$$

We can now rewrite the kth coordinate of $\mathbf{y}(\mathbf{x}_i;\cdot)$ using a new set of parameters $\mathbf{w}^1 = (t_1,\ a_{11}-1,\ a_{12},\ a_{13})^{\mathrm{T}}$, $\mathbf{w}^2 = (t_2,\ a_{21},\ a_{22}-1,\ a_{23})^{\mathrm{T}}$ and $\mathbf{w}^3 = (t_3,\ a_{31},\ a_{32},\ a_{33}-1)^{\mathrm{T}}$:

$$y^k(\mathbf{x}_i;\mathbf{A},\mathbf{t}) \quad = \quad x_i^k + \mathbf{q}(\mathbf{x}_i)^{\mathrm{T}}\mathbf{w}^k \tag{2.26}$$

with x_i^k being the kth coordinate of $\mathbf{x}_i$, and $\mathbf{q}(\mathbf{x}_i) = (1,\ x_i^1,\ x_i^2,\ x_i^3)^{\mathrm{T}}$. $\mathbf{q}(\mathbf{x}_i)$ is the same for y^1, y^2, and y^3.

In the following we are going to consider a large vector $\tilde{\mathbf{x}}$ containing the pixel coordinates of all pixels in the reference image, as well as the corresponding warped grid of output points $\tilde{\mathbf{y}}$ (the coordinates where the template image should be sampled), defined as follows:

$$\tilde{\mathbf{x}} = \begin{pmatrix} \mathbf{x}^1 \\ \mathbf{x}^2 \\ \mathbf{x}^3 \end{pmatrix} \quad \text{and} \quad \tilde{\mathbf{y}} = \begin{pmatrix} \mathbf{y}^1 \\ \mathbf{y}^2 \\ \mathbf{y}^3 \end{pmatrix},$$

where

$$\mathbf{x}^k = \begin{pmatrix} x_1^k \\ x_2^k \\ \vdots \\ x_N^k \end{pmatrix} \quad \text{and} \quad \mathbf{y}^k = \begin{pmatrix} y_1^k \\ y_2^k \\ \vdots \\ y_N^k \end{pmatrix}.$$

That is, $\tilde{\mathbf{x}}$ and $\tilde{\mathbf{y}}$ are organized as stacks of coordinates of the inputs, i.e., given N input points the first N elements are the first coordinate of these input, the next N elements are the 2nd coordinates, and so on[5]. In a vector-matrix notation the affine geometrical transformation for each of the coordinates of the entire set of points can be written as

$$\mathbf{y}^k \quad = \quad \mathbf{x}^k + \mathbf{Q}'\mathbf{w}_k, \tag{2.27}$$

where $\mathbf{Q}'$ is a matrix made by stacking the row-vectors $\mathbf{q}(\mathbf{x}_i)^{\mathrm{T}} = (1,\ x_i^1,\ x_i^2,\ x_i^3)$.

We introduce a construction called the Kronecker matrix product. Let $\mathbf{A}$ be an $n \times m$ matrix with elements $\{a_{ij}\}$ and $\mathbf{B}$ be a $k \times l$ matrix with elements $\{b_{ij}\}$

[5]Alternatively we could have organized the input and output points as matrices of dimension $N \times 3$. However, for the optimization procedure we want the parameters $\mathbf{w}$ in a column vector and not a matrix, wherefore the chosen organization is most convenient.

then the Kronecker product is the $nk \times ml$ block matrix

$$\mathbf{A} \otimes \mathbf{B} = \begin{pmatrix} a_{11}\mathbf{B} & \dots & a_{1m}\mathbf{B} \\ \vdots & & \vdots \\ a_{n1}\mathbf{B} & \dots & a_{nm}\mathbf{B} \end{pmatrix}.$$

Using the Kronecker notation and stacking the column vectors $\mathbf{w}^k$ in a long vector

$$\mathbf{w} = \begin{pmatrix} \mathbf{w}^1 \\ \mathbf{w}^2 \\ \mathbf{w}^3 \end{pmatrix},$$

we can write the affine transformation for all grid points on the canonical vector-matrix form

$$\tilde{\mathbf{y}} = \tilde{\mathbf{x}} + \begin{pmatrix} \mathbf{Q}' & 0 & 0 \\ 0 & \mathbf{Q}' & 0 \\ 0 & 0 & \mathbf{Q}' \end{pmatrix} \mathbf{w} = \tilde{\mathbf{x}} + \mathbf{I}_3 \otimes \mathbf{Q}'\mathbf{w} = \tilde{\mathbf{x}} + \mathbf{Q}\mathbf{w}, \tag{2.28}$$

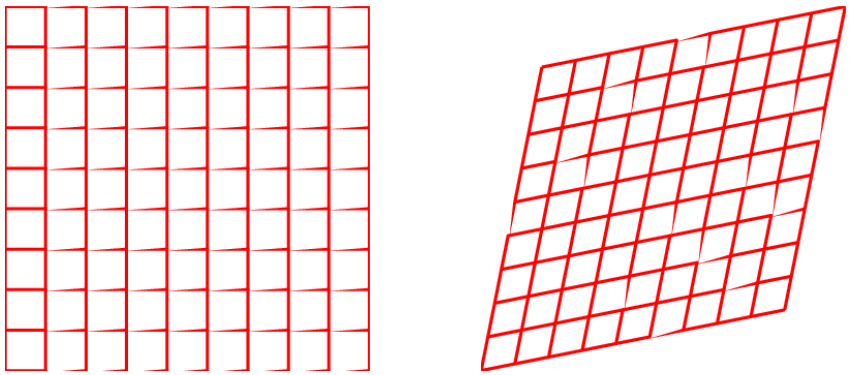
where $\mathbf{I}_3$ is the 3×3 identity matrix.

In Fig. 2.9 a regular grid is warped using an affine geometrical transformation with

$$\mathbf{t} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$
$$\mathbf{A} = \begin{pmatrix} 1 & 0.2 \\ 0.2 & 1 \end{pmatrix}$$

The following matlab code was used to generate the transformation:

```
n = 10; m=10;

[x1 x2] = meshgrid(1:n,1:m);

plotgrid(x1,x2);
```



(a) (b)

Figure 2.9: The regular grid in (a) is warped to (b) by an affine transformation.

```
Q = [ones(size(x1(:))) x1(:) x2(:)];

w = [0 0 0.2 0 0.2 0]';

y = [x1(:); x2(:)] + kron(eye(2),Q)*w;

y1 = reshape(y(1:end/2),n,m);

y2 = reshape(y(end/2+1:end),n,m);

plotgrid(y1,y2);
```

In the following we show this procedure can be used to transform a template image. We have used a nearest neighbor interpolation. The resulting transformed image is shown in Fig. 2.10:

```
[m n p ] = size(hand);

[x1 x2] = meshgrid(1:n,1:m);

Q = [ones(size(x1(:))) x1(:) x2(:)];

w = [0 0 0.2 0 0.2 0]';

y = [x1(:); x2(:)] + kron(eye(2),Q)*w;
```

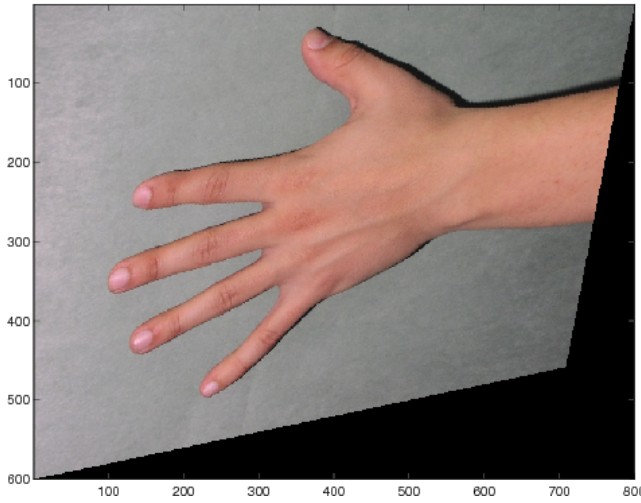


Figure 2.10: Affinely transformed hand image.

```
y1 = reshape(y(1:end/2),n,m);

y2 = reshape(y(end/2+1:end),n,m);

handx = [hand zeros(600,200,3); zeros(400,1000,3)];

ind = sub2ind(size(handx),round(y2(:)),round(y1(:)));

R = handx(:,:,1); G = handx(:,:,2); B = handx(:,:,3);

rx = R(ind); gx = G(ind); bx = B(ind);

RGB = reshape(rx,size(hand(:,:,1)));

RGB(:,:,2) = reshape(gx,size(hand(:,:,1)));

RGB(:,:,3) = reshape(bx,size(hand(:,:,1)));

figure; image(RGB); axis image; axis off;
```

2.4.2 Polynomial transformation

We can now generalize to more complex transformations. If we want a polynomial transformation of 2nd degree this is achieved by updating $\mathbf{q}_i$ in Eq. (2.26) to include quadratic terms, i.e.,

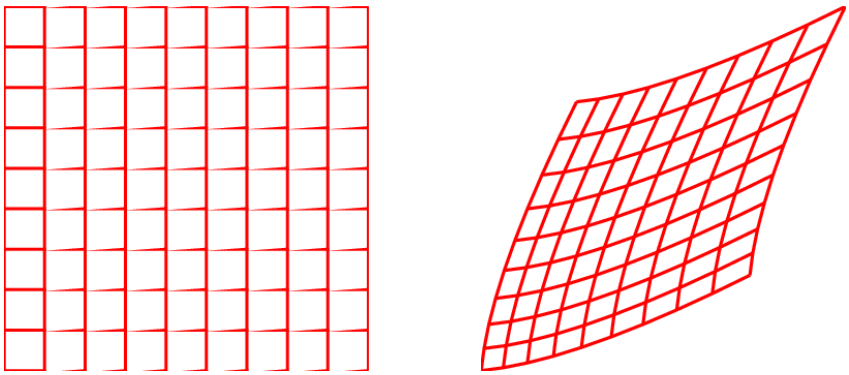
$$\mathbf{y}^k(\mathbf{x}_i; \mathbf{A}, \mathbf{t}) \quad = \quad x_i^k + \mathbf{q}(\mathbf{x}_i)^{\mathrm{T}} \mathbf{w}^k$$

with $\mathbf{q}(\mathbf{x}_i) = (\ 1 \quad x_i^1 \quad x_i^2 \quad x_i^3 \quad x_i^1 x_i^1 \quad x_i^2 x_i^2 \quad x_i^3 x_i^3 \quad x_i^1 x_i^2 \quad x_i^1 x_i^3 \quad x_i^2 x_i^3\ )^{\mathrm{T}}$ and a corresponding extension of $\mathbf{w}^k$.

In Fig. 2.11 a regular grid is warped using an quadratic polynomial geometrical transformation. The following matlab code was used to generate the transformation

```
n = 10; m=10;

[x1 x2] = meshgrid(1:n,1:m);

plotgrid(x1,x2);

Q = [ones(size(x1(:))) x1(:) x2(:) x1(:).*x1(:) x2(:).*x2(:)
x1(:).*x2(:)];

 w = [0 1 0 0.1 0.1 0 0 0 1 0.1 0.1 0]';

y = [x1(:); x2(:)] + kron(eye(2),Q)*w;

y1 = reshape(y(1:end/2),n,m);

y2 = reshape(y(end/2+1:end),n,m);

plotgrid(y1,y2);
```

The polynomial geometrical transformation can be extended to degrees higher than two by including higher order terms in the $\mathbf{Q}$ matrix.



(a) (b)

Figure 2.11: The regular grid in (a) is warped to (b) by a 2nd order polynomial transformation.

2.5 B-spline based geometrical transformations

The global polynomial transformations are rarely used for modelling geometrical image transformations. A more popular way of introducing non-linearity is to use splines. In Sec. 1.4 we introduced the cubic spline and represented it as a linear combination of a set of cubic B-spline basis functions. One of the main features of these basis functions is that they have only local support, i.e., if we change the coefficient of one B-spline basis function it only affects a local neighborhood. This is an attractive property for an optimization scheme. In Fig. 2.12 a set of cubic B-spline basis functions are shown.

However, we need a function over the image domain, i.e., in 2D or 3D in order to model our deformation field. A set of the 2D equivalent of the one-dimensional B-spline functions is shown in Fig. 2.13.

We will briefly discuss the construction of 2D and 3D B-spline basis functions. Let $B_{m_k}(x^k;\xi_k)$ be a set of 1D B-spline basis functions indexed by $m_k = 1,\ldots,M_k$ defined along the kth coordinate axis by the knot sequence ξ_k[6]. Then the set of 3D B-spline basis functions are given by

$$q_m(\mathbf{x};\xi_1,\xi_2,\xi_3) \quad = \quad B_{m_1}(x^1;\xi_1)B_{m_2}(x^2;\xi_2)B_{m_3}(x^3;\xi_3)$$

[6]Recall from Sec. 1.4 that cubic B-spline basis functions are specific third-order polynomials within each interval of an ordered set of "knots", with continuous first and second order derivatives at the knot locations. For our application the knots will be spaced equidistantly, covering the entire image domain, except perhaps at the boundaries.

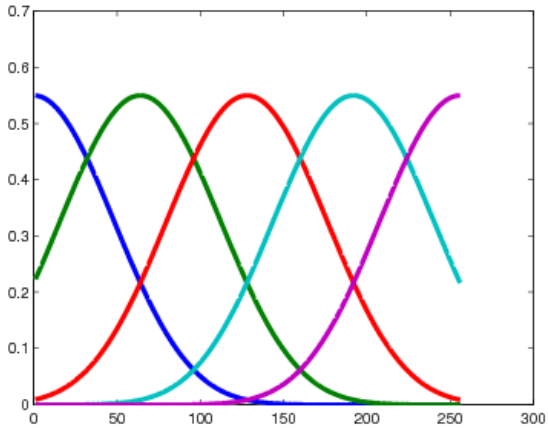


Figure 2.12: Here a set of 1-D cubic B-spline basis functions are shown.

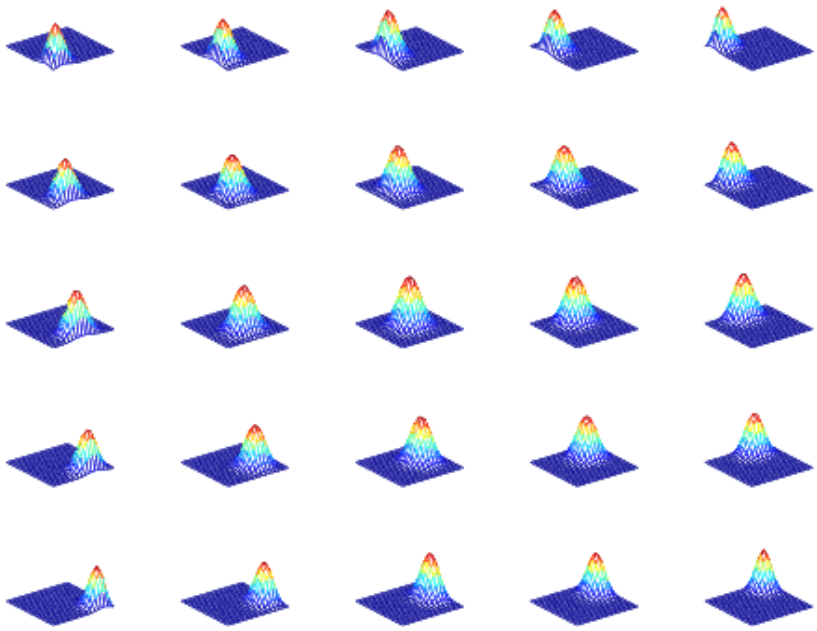


Figure 2.13: A set of 2D cubic B-spline basis functions.

where $m = 1, \ldots, M_1 M_2 M_3$. The 2D and 3D basis functions are outer products of the 1D basis function and are referred to as tensor B-spline functions.

We can now use the tensor B-spline function to model the deformation field, i.e.,

$$y^k(\mathbf{x}_i; \mathbf{w}^k) \quad = \quad x_i^k + \mathbf{q}(\mathbf{x}_i)^{\mathrm{T}} \mathbf{w}^k$$

with

$$\mathbf{q}(\mathbf{x}_i) \quad = \quad \begin{pmatrix} q_1(\mathbf{x}_i) \\ q_2(\mathbf{x}_i) \\ \vdots \\ q_{M_1 M_2 M_3}(\mathbf{x}_i) \end{pmatrix}. \tag{2.29}$$

In vector-matrix notation we again find for the kth coordinate of $\mathbf{y}(\mathbf{x}_i; \mathbf{w})$ that

$$\mathbf{y}^k \quad = \quad \mathbf{x}^k + \mathbf{Q}' \mathbf{w}^k$$

and by stacking the $\mathbf{y}^k$'s we get

$$\tilde{\mathbf{y}} \quad = \quad \tilde{\mathbf{x}} + \begin{pmatrix} \mathbf{Q}' & 0 & 0 \\ 0 & \mathbf{Q}' & 0 \\ 0 & 0 & \mathbf{Q}' \end{pmatrix} \mathbf{w} = \tilde{\mathbf{x}} + \mathbf{I}_3 \otimes \mathbf{Q}' \mathbf{w} = \tilde{\mathbf{x}} + \mathbf{Q} \mathbf{w} \tag{2.30}$$

where $\mathbf{I}_3$ is the 3×3 identity matrix.

Now because our tensor B-spline basis functions are separable (they are written as functions of one coordinate at a time) the $\mathbf{Q}$ matrix will have a special structure. In particular, if we consider $\mathbf{q}(\mathbf{x}_i)$ it turns out that

$$\mathbf{q}(\mathbf{x}_i) \quad = \quad \begin{pmatrix} B_1(x_i^3; \xi_3) \\ \vdots \\ B_{M_3}(x_i^3; \xi_3) \end{pmatrix} \otimes \begin{pmatrix} B_1(x_i^2; \xi_2) \\ \vdots \\ B_{M_2}(x_i^2; \xi_2) \end{pmatrix} \otimes \begin{pmatrix} B_1(x_i^1; \xi_1) \\ \vdots \\ B_{M_1}(x_i^1; \xi_1) \end{pmatrix}.$$

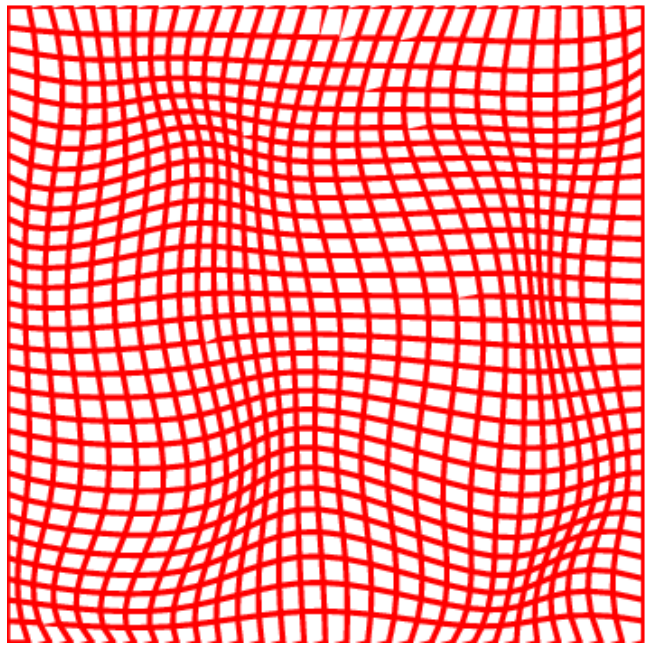
More generally, we can write

$$\mathbf{Q}' \quad = \quad \mathbf{Q}'_{\mathbf{3}} \otimes \mathbf{Q}'_{\mathbf{2}} \otimes \mathbf{Q}'_{\mathbf{1}}$$

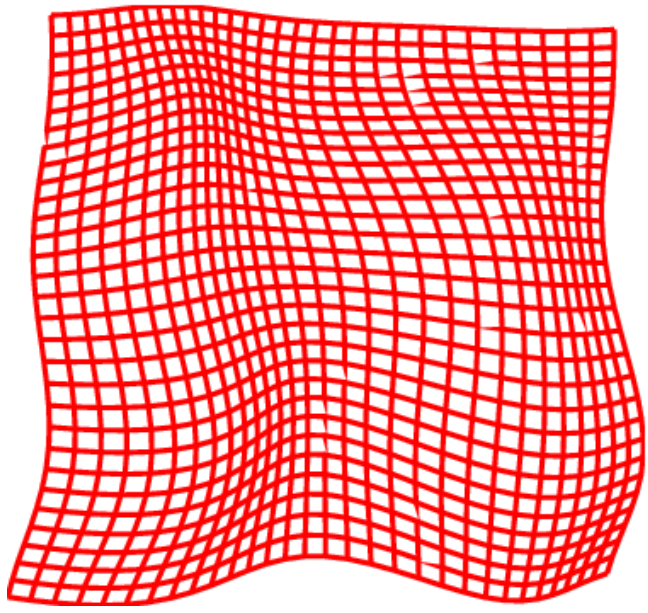
where the (i, m_k)th element of $\mathbf{Q'}_k$ is $\{B_{m_k}(x_i^k; \xi_k)\}$, i.e., the $\mathbf{Q'}_k$ matrix holds the values of the 1D B-spline functions along the kth coordinate axis for all input points $\mathbf{x}_i$.

As an example we will try to warp a grid using a set of tensor B-spline basis functions. The following Matlab commands will produce the warped grid shown in Fig. 2.14(a). Note here that the deformation is zero along the grid borders. This is achieved by letting the border knots have multiplicity 3 (i.e., three knots are placed at the same border spot) in the 1D cubic B-spline functions, as shown in Fig. 2.15. In this way the basis function are forced to 0 at the borders. Alternatively, in Fig. 2.14(b) we have augmented the knot sequence with knots placed equidistantly outside the image borders, resulting in the 1D cubic B-spline functions shown in Fig. 2.12. In this way we get a free border effect.

```
%image dimensions
m = [31 31];
% number of knots
p1 = 5; p2 = 5;
% knot sequence 1
k1 = linspace(1,m(1),p1); k1 = augknt(k1,3);
% knot sequence 2
k2 = linspace(1,m(2),p2); k2 = augknt(k2,3);
% Form B-spline matrix Q which contains basic functions
B1 = spmak(k1,eye(p1)); Q1 = fnval(B1,1:m(1))';
%
B2 = spmak(k2,eye(p2)); Q2 = fnval(B2,1:m(2))';
%
Q = kron(speye(2),kron(Q2,Q1));

[x1 x2] = meshgrid(1:m(2),1:m(1));

y = [x1(:); x2(:)] + Q*w*0.4;

y1 = reshape(y(1:end/2),size(x1,1),size(x1,2));

y2 = reshape(y(end/2+1:end),size(x1,1),size(x1,2));

plotgrid(y1,y2);
```



(a)



(b)

Figure 2.14: Two tensor B-spline based deformation grids.

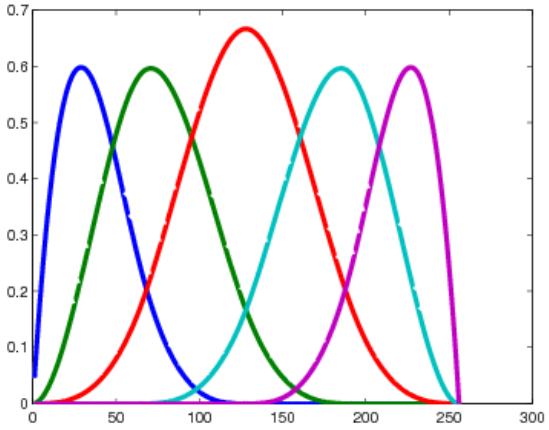


Figure 2.15: As in Fig. 2.12, a set of 1-D cubic B-spline basis functions are shown, but now using overlapping knot locations at the boundaries. As a result, the spline functions evaluate to zero outside of the image domain.

2.6 Optimization of image registration

In this section we will consider the optimization of parameterized image registration. We will use the sum-of-squared-differences dissimilarity measure and the canonical form for parameterized geometrical transformations derived in the previous two sections.

The sum-of-squared-differences dissimilarity measure $\mathcal{D}_{\mathrm{SSD}}(\mathbf{w})$ was defined in Eq. (2.16). In order to ensure that only reasonable transformations are found, a *regularized* objective function

$$\mathcal{J}(\mathbf{w}) = \mathcal{D}_{\mathrm{SSD}}(\mathbf{w}) + \alpha\mathcal{S}(\mathbf{w})$$

is often used instead, where the regularizer is of the form

$$\mathcal{S}(\mathbf{w}) = \frac{1}{2}\|\boldsymbol{\Gamma}\mathbf{w}\|^2 \tag{2.31}$$

and $\alpha \geq 0$ is a scalar determining the strength of the regularization. In Eq. (2.31) $\boldsymbol{\Gamma}$ is a matrix chosen in such a way that the product $\boldsymbol{\Gamma}\mathbf{w}$ is a vector containing large numerical values for weights $\mathbf{w}$ that are somehow deemed "undesirable". Examples could be $\boldsymbol{\Gamma} = \mathbf{I}$ (simply penalizing large weights), $\boldsymbol{\Gamma} = \mathbf{Q}$ (penalizing large deformations), or $\boldsymbol{\Gamma} = \nabla\mathbf{Q}$ (penalizing e.g., large gradients or curvature

in the resulting deformations if ∇ implements appropriate finite difference operations). Putting everything together, we finally obtain the following objective function to be minimized:

$$\begin{aligned}\mathcal{J}(\mathbf{w}) &= \frac{1}{2}\sum_{i\in\Omega}\left(\mathcal{T}(\mathbf{y}(\mathbf{x}_i;\mathbf{w}+\mathbf{s}))-\mathcal{R}(\mathbf{x}_i)\right)^2 + \frac{\alpha}{2}\|\mathbf{\Gamma}\mathbf{w}\|^2 \\ &= \frac{1}{2}\left\|\mathcal{T}(\tilde{\mathbf{y}})-\mathcal{R}(\tilde{\mathbf{x}})\right\|^2 + \frac{\alpha}{2}\|\mathbf{\Gamma}\mathbf{w}\|^2, \end{aligned} \tag{2.32}$$

where $\mathcal{R}(\tilde{\mathbf{x}})$ is a vector of reference image values collected at the input points $\mathbf{x}_i$, $i = 1,\ldots,N$, and $\mathcal{T}(\tilde{\mathbf{y}})$ is a vector of the corresponding, interpolated image values in the template image collected at the points $\tilde{\mathbf{y}} = \tilde{\mathbf{x}} + \mathbf{Q}\mathbf{w}$ (canonical form, cf. Eq. (2.30)).

The Gauss-Newton idea now consists of linearizing (by a Taylor expansion to first order) the function under the norm in the first term of Eq. (2.32), and thus approximating the objective function with a quadratic function. In this way the parameter update can be found by solving a set linear equations obtained from setting the first order derivatives equal to 0. In particular, the value of the objective function after we have updated the geometrical transformation parameters $\mathbf{w}$ by a small amount $\Delta\mathbf{w}$ is given by

$$\begin{aligned}\mathcal{J}(\mathbf{w}+\Delta\mathbf{w}) &= \frac{1}{2}\|\mathcal{T}(\tilde{\mathbf{x}}+\mathbf{Q}(\mathbf{w}+\Delta\mathbf{w}))-\mathcal{R}(\tilde{\mathbf{x}})\|^2 + \frac{\alpha}{2}\|\mathbf{\Gamma}\mathbf{w}+\mathbf{\Gamma}\Delta\mathbf{w}\|^2 \\ &= \frac{1}{2}\|\mathcal{T}(\tilde{\mathbf{y}}+\mathbf{Q}\Delta\mathbf{w})-\mathcal{R}(\tilde{\mathbf{x}})\|^2 + \frac{\alpha}{2}\|\mathbf{\Gamma}\mathbf{w}+\mathbf{\Gamma}\Delta\mathbf{w}\|^2 \\ &\approx \frac{1}{2}\|\mathcal{T}(\tilde{\mathbf{y}})+\nabla\mathcal{T}(\tilde{\mathbf{y}})\mathbf{Q}\Delta\mathbf{w}-\mathcal{R}(\tilde{\mathbf{x}})\|^2 + \frac{\alpha}{2}\|\mathbf{\Gamma}\mathbf{w}+\mathbf{\Gamma}\Delta\mathbf{w}\|^2, \end{aligned}$$

where where the notation

$$\nabla\mathcal{T}(\tilde{\mathbf{y}}) = \begin{pmatrix}\mathbf{G}_1 & \mathbf{G}_2 & \mathbf{G}_3\end{pmatrix}, \quad \mathbf{G}_k = \mathrm{diag}\left(\left.\frac{\partial\mathcal{T}}{\partial y^k}\right|_{\mathbf{y}_1},\ldots\left.\frac{\partial\mathcal{T}}{\partial y^k}\right|_{\mathbf{y}_N}\right) \tag{2.33}$$

was used.

Differentiation yields

$$\frac{\partial\mathcal{J}}{\partial\Delta\mathbf{w}} \approx [\nabla\mathcal{T}(\tilde{\mathbf{y}})\mathbf{Q}]^{\mathrm{T}}\Big(\mathcal{T}(\tilde{\mathbf{y}})+\nabla\mathcal{T}(\tilde{\mathbf{y}})\mathbf{Q}\Delta\mathbf{w}-\mathcal{R}(\tilde{\mathbf{x}})\Big)+\alpha\mathbf{\Gamma}^{\mathrm{T}}\mathbf{\Gamma}(\mathbf{w}+\Delta\mathbf{w}).$$

Let $\mathbf{A} = \nabla\mathcal{T}(\tilde{\mathbf{y}})\mathbf{Q}$. Setting the derivatives equal to 0 yields

$$(\mathbf{A}^{\mathrm{T}}\mathbf{A} + \alpha\boldsymbol{\Gamma}^{\mathrm{T}}\boldsymbol{\Gamma})\Delta\mathbf{w} = \mathbf{A}^{\mathrm{T}}(\mathcal{R}(\tilde{\mathbf{x}}) - \mathcal{T}(\tilde{\mathbf{y}})) - \alpha\boldsymbol{\Gamma}^{\mathrm{T}}\boldsymbol{\Gamma}\mathbf{w}, \tag{2.34}$$

which can now be solved for $\Delta\mathbf{w}$.

The resulting Gauss-Newton optimization of parameterized image registration is as follows:

1. choose initial value $\mathbf{w}$
2. while not STOP
3. calculate transformation $\tilde{\mathbf{y}} = \tilde{\mathbf{x}} + \mathbf{Q}\mathbf{w}$
4. calculate transformed image $\mathcal{T}(\tilde{\mathbf{y}})$ and its spatial derivatives $\nabla\mathcal{T}(\tilde{\mathbf{y}})$
5. solve for update $\Delta\mathbf{w}$ using Eq. (2.34)
6. update $\mathbf{w} := \mathbf{w} + \Delta\mathbf{w}$
7. end while

Because we are using an approximation it can be advisable to change the update step to $\mathbf{w} := \mathbf{w} + \lambda\Delta\mathbf{w}$, and include a step to find an optimal step length to avoid over and under stepping.

The stop criterion is typically a combination of the decrease of the cost function $\mathcal{J}(\mathbf{w}_{k-1}) - \mathcal{J}(\mathbf{w}_k) < \epsilon_J$, the size of the gradients $\|\mathbf{A}\| < \epsilon_A$, the change of the parameter vector $\|\Delta\mathbf{w}_k\| < \epsilon_\Delta$, and the number of iterations.

2.A Exercise II: Landmark based registration

In this exercise you are going to experiment with point based image registration. The methods will be applied to four 2D images from the *visible human*[7].

[7] See http://www.nlm.nih.gov/research/visible/visible_human.html

2.A.1 The Visible Human Project

The Visible Human Project is an outgrowth of the US National Library of Medicine's 1986 Long-Range Plan. It is the creation of complete, anatomically detailed, three-dimensional representations of the normal male and female human bodies. Acquisition of transverse CT, MR and cryosection images of representative male and female cadavers has been completed. The male was sectioned at one millimeter intervals, the female at one-third of a millimeter intervals.

The long-term goal of the Visible Human Project is to produce a system of knowledge structures that will transparently link visual knowledge forms to symbolic knowledge formats such as the names of body parts.

Four images are available for today's exercise. All images are a slice through the visible male head:

- `head`– color cryosection.
- `head_fresh`– CT scan of fresh cadaver.
- `head_frozen`– CT scan of frozen cadaver.
- `head_mri`– MR scan.

2.A.2 Tasks

1. Select 10 fiducial points from the color crysosection. Evaluate the fiducial localization error (FLE) by letting all members of your group annotate the 10 points in the image. Report the FLE variance σ^2_{FLE} given in Eq. (2.14). You may want to use the Matlab control point select tool `cpselect`. The point coordinates are saved in the file menu into arrays `base_points` and `input_points`.

2. Find coordinates of the 10 points in all images.

3. What is the appropriate transformation between the MR and frozen CT scan? Estimate the parameters that map the MR scan to the CT scan; compute the fiducial registration error (FRE) σ^2_{FRE} given by Eq. (2.15). Also estimate the parameters that map the CT scan to the MR scan and the determine the FRE σ^2_{FRE}. Compare and discuss the results of the two registrations.

4. Apply the estimated transformation to the MR image and generate a RGB image with CT in the red channel and the transformed MR image in the green channel.

Your report should contain control point coordinates, images, and matlab code that will allow to check your results.

Hints

You may want to use the Matlab functions

- cpselect - control point selection tool.
- meshgrid - generates a grid of x and y coordinates.
- svd - computes the matrix singular value decomposition.
- interp2 - interpolates intensity values at specified coordinates given intensity values on a grid.

2.B Exercise III: Mutual information based registration

In this exercise you should experiment with mutual information based image registration. The methods are applied to the same 2D images of the *Visible Human Project* described on page 47.

2.B.1 Tasks

We will be concerned with registering the MR image and the frozen cadaver CT scan. First we will bring the two images into partial alignment using the principal axes transform. Secondly, we will make a final adjustment using mutual information registration.

1. Compute mean and principal axes for the two images.

2. Match the center point and the 1st principal axis end-point from the two images to each other, by similarity transformation.

3. Perform the transformation as in exercise 2. Show the result as a color image with one image in red and the other in green. Describe the remaining transformation required to map the images.

4. Find the translation that maps the principal-axes transformed images to each other by optimization of the mutual information between the images. One way would be to compute the mutual information for a grid of translations and read out the translation that leads to the optimum.

5. (For the enthusiastic student) Optimize also over rotation (and scale).

The provided Matlab function `histogram2` computes the 2D histogram; the mutual information between images `im1` and im2 can then be computed like this:

```
jointHistogram = histogram2( double( im1(:)' ), double( im2(:)' ), ...
                             [ 0 256 64; 0 256 64 ] );
jointPdf = jointHistogram / sum( jointHistogram( : ) );
firstMarginalPdf = sum( jointPdf, 1 );
secondMarginalPdf = sum( jointPdf, 2 );
MI = log( jointPdf( : ) + eps )' * jointPdf( : ) - ...
     log( firstMarginalPdf( : ) + eps )' * firstMarginalPdf( : ) - ...
     log( secondMarginalPdf( : ) + eps )' * secondMarginalPdf( : );
```

2.C Exercise IV: Non-linear intensity-based registration

In this exercise you are going to implement and test the Gauss-Newton based optimization scheme for tensor B-spline based image registration described in Sec. 2.6.

2.C.1 Data

The data for the exercise consists of two mid-sagittal brain MR slices - `mr.mat`.

2.C.2 Tasks

1. Construct a set of 2D tensor B-spline basis functions as combinations of three 1D basis functions in each coordinate direction, i.e., construct a tensor B-spline matrix with $3 \times 3 = 9$ 2D basis functions. Augment the 1D boundary knots by placing 2 additional knots on top of them (cf. Fig. 2.15 and the Matlab code on page 42).

2. Deform a grid by setting all elements of $\mathbf{w}^1$ to the same non-zero value, and all elements of $\mathbf{w}^2$ to zero. Do the same when the roles of $\mathbf{w}^1$ and $\mathbf{w}^2$ are reversed. Deform the grid as well by setting all values of $\mathbf{w}^1$ and $\mathbf{w}^2$ to zero, except for the elements corresponding to one of the 9 basis functions, which are set to a non-zero value.

3. Implement a non-linear intensity-based registration algorithm with B-splines as basis functions, the sum-of-squared differences as the similarity measure, and regularizer matrix $\mathbf{\Gamma} = \mathbf{I}$. Use the Gauss-Newton solver as the optimizer.

4. Non-linearly register the second MR slice to the first. Try to vary the number of knots, e.g. 3×3, 5×5 and 7×7; and the regularization parameter α, e.g., $\alpha = 0.1, 1$ and 10. Compare the results.

Hints

Matlab code for generating 2D tensor B-spline basis functions, and for deforming a grid using such basis functions, is given on page 42.

Before updating in step 6 in the Gauss-Newton algorithm described on page 46 you can check if the update results in a decreasing objective function. If not, try $\Delta\mathbf{w}/2$, etc.

Interpolation is done by `interp2` and gradients can be computed using `gradient`.

The function `plotgrid`, provided on CampusNet, can be used to plot a deformed grid.

Chapter 3

Surface based analysis

In the previous chapter we considered the registration of pairs of images, where the registration process is driven by the intensity content in corresponding voxels. Another important set of techniques in medical image analysis relate to so-called *segmentation*, where the aim is to outline specific anatomical structures of interest in the images. Although it is possible to segment images by determining whether individual voxels belong to a a specific structure (so-called voxel-based segmentation, analyzed in detailed in chapter 4), many powerful segmentation methods instead aim at locating the *boundaries* between structures. Similarly, we often encounter situations in which we have to align the outer boundaries of corresponding objects after they have been extracted from a set of images, for instance when we are analyzing the objects' *shape* properties.

In this chapter, we will consider a class of registration and segmentation algorithms that explicitly work with the boundaries of anatomical structures rather than directly on image voxels. In two dimensions, such boundaries are *curves*, which are often represented by a connected series of line segments, whereas in three dimensions they are given by *surfaces*, often represented by a mesh of triangles that are connected by their common edges.

3.1 Surface based registration

In surface based registration the aim is to determine the parameters $\mathbf{w}$ of a geometrical transformation $\mathbf{y}(\mathbf{x};\mathbf{w}) : \mathbb{R}^3 \rightarrow \mathbb{R}^3$ that maps the coordinates of one surface to the coordinates of a corresponding one so that both surfaces are maximally aligned. Since it is straightforward to extract a set of points lying on a surface for most surface representations (for instance, the corners of the triangles in triangular meshes are points that lie on the surface), it is tempting to re-formulate this registration problem as a landmark based registration, which is something we already know how to do (see section 2.2). However, in contrast to landmark based registration, here we are not given the *correspondence* between the sets of points representing the two surfaces: Which location on the second surface each point on the first surface corresponds to, is therefore something we will need to estimate automatically.

3.1.1 Iterative closest point

The solution proposed in the **iterative closest point** (ICP) algorithm [9] is to iterate between assigning correspondences and updating the transformation, each in turn, until convergence. The algorithm works as follows: Let two surfaces (or curves in two dimensions) be denoted X and Y, and let $\mathbf{x}_i$, $i = 1, \ldots, N$ be a point representation of X. Then the ICP algorithm consist of minimizing

$$\mathcal{D}(\mathbf{w}, \mathcal{C}) \quad = \quad \sum_{i=1}^{N} \|\mathbf{y}(\mathbf{x}_i;\mathbf{w}) - \mathbf{y}_i(\mathcal{C})\|^2 \tag{3.1}$$

where $\mathbf{y}_i(\mathcal{C})$ is the point on surface Y that is deemed to correspond to point $\mathbf{x}_i$ on surface X according to the correspondence assignments $\mathcal{C}$. The algorithm solves this optimization problem by iterating between (1) determining the point correspondences $\mathcal{C}$ that minimize eq. (3.1) while keeping the registration parameters $\mathbf{w}$ fixed at their current values; and (2) computing the registration parameters $\mathbf{w}$ that minimize eq. (3.1) while keeping the point correspondences $\mathcal{C}$ fixed to their current values. The solution to the first step is given by assigning each point $\mathbf{x}_i$ on X to the point on Y that is geometrically closest to its mapped location $\mathbf{y}(\mathbf{x}_i;\mathbf{w})$; while the solution to the second step is simply computing a landmark based registration as in section 2.2.

The transformation $\mathbf{y}(\mathbf{x};\mathbf{w})$ can be any of the geometrical transformations that we have covered in the preceding sections. A popular choice is to use sets of

smoothing thin plate splines. However, the geometrical transformation needs to be heavily regularized in order to minimize the effect of false matches in the early stages of the iteration. Regarding the assignment of corresponding points on Y, it should be noted that we typically require the closest point on the *curve/surface* and not just the closest point in a fixed set of points lying on that curve/surface; determining the closest point on a curve/surface from any given point requires some non-trivial computations the details of which depend on the surface representation that is used.

The ICP is a local method and it usually requires a good initialization to converge to the global optimum.

3.1.2 Shape context

Instead of re-estimating point correspondences based on the physical distance between the surface Y and the mapped locations $\mathbf{y}(\mathbf{x}_i; \mathbf{w})$ of the points $\mathbf{x}_i$ under the current transformation estimate, we can also consider matching local properties of the curves/surfaces. These properties could be of a differential-geometrical nature, e.g., local curve/surface curvature, but here we will consider an alternative property, namely the so-called *shape context* [10].

The concept of shape context is illustrated in Fig. 3.1. Around each point in X (and similarly for Y) we place a circular area that is subdivided into K so-called "bins". By counting the number of points in X (respectively Y) that fall within each bin, we obtain a histogram that is called the *shape context* of the point under investigation. This shape context acts as a compact descriptor of the local shape of X (respectively Y) that is relatively insensitive to small perturbations of the shape. Note that the bins are uniform in log-polar space, making the descriptor more sensitive to positions of nearby sample points than to those of points farther away. Fig. 3.1(c) and (d) show the shape context for the points marked in red and green in Fig. 3.1(a), respectively. Here the histograms are displayed in the Cartesian coordinate system, with the polar angle on the x-axis and the radius on the y-axis; high intensities correspond to large point counts.

Since shape contexts are local distributions, a relevant measure for comparing two shape contexts is the ξ^2 test statistics for equal distributions:

$$C_{ij} = \frac{1}{2}\sum_{k=1}^{K}\frac{(h_i(k) - h_j(k))^2}{h_i(k) + h_j(k)}, \tag{3.2}$$

where $h_i(k)$ and $h_j(k)$ denote the entry in the k-th bin of the normalized his-

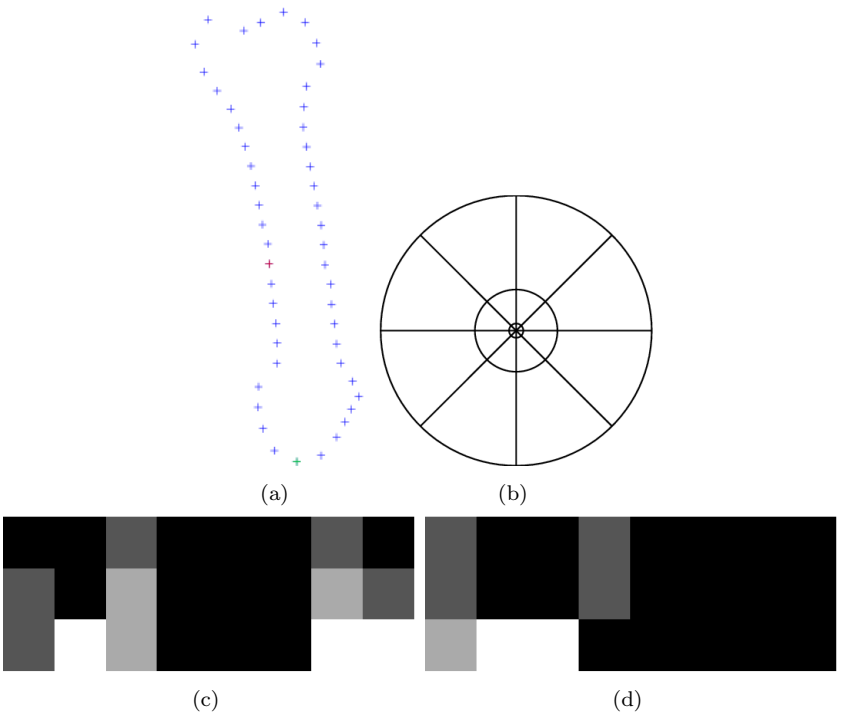


Figure 3.1: (a) metacarpal bone with two points marked in red and green; (b) the histogram bins used in computing the shape context; (c,d) the the shape context around the red and green point respectively (the histograms are displayed in the Cartesian coordinate system, with the polar angle on the x-axis and the radius on the y-axis; high intensities correspond to large point counts).

togram at points $\mathbf{x}_i$ and $\mathbf{y}_j$, respectively. The squared differences are weighted by the sum of the histogram values because bins with many neighbors will have a correspondingly higher variance.

From these test statistics a so-called *cost matrix* $\mathbf{C}$ can be computed that evaluates the matching cost between all possible pairs of points in the two shapes X and Y, i.e.,

$$\mathbf{C} = \begin{pmatrix} C_{11} & \dots & C_{1M} \\ \vdots & \ddots & \vdots \\ C_{N1} & \dots & C_{NM} \end{pmatrix},$$

where N and M are the number of points in X and Y, respectively. Each point $\mathbf{x}_i$ in X can now be matched with a point $\mathbf{y}_j$ in Y by looking up the column j that has the lowest cost in the i-th row of this matrix. The obtained point matching can then be used to update the registration parameters $\mathbf{w}$ as in the ICP algorithm, after which the cost matrix and the corresponding point matching can be updated, etc, until convergence.

3.2 Surface based segmentation

An important set of algorithms for segmenting biomedical images concentrates on accurately locating the *boundaries* between relevant objects. The power of these methods derives mainly from the ability to incorporate detailed prior knowledge about the type of *shape* we are looking for in the images. Such prior information makes it possible to accurately locate object boundaries even in image areas where the boundaries are difficult to discern based on local intensity properties alone, which happens frequently in medical imaging.

There exists a wide variety of boundary based segmentation methods that differ mainly in the way they represent curves/surfaces and encode prior knowledge about their expected shape. Well-known examples include so-called level set methods [11], medial shape representations [12], and active shape models [13]. Here we only consider a simple path tracing algorithm that is based on a graph search algorithm using so-called *dynamic programming*. Details of this method are explained in section 6.2 of the course notes of the DTU course *Introduction to Medical Image Analysis* (Rasmus R. Paulsen, 2010).

3.A Exercise V: Contour registration

In this exercise you should experiment with contour registration where the point-to-point correspondences between the contours are unknown.

3.A.1 Data

The data for this exercise consists of points sampled equidistantly along curves. These curves are hand drawn on images. However, we do not know the correspondence between points of the curves.

- meta.mat - contour of metacarpal bones
- hands.mat - contour of hands
- lung.mat - contour of lungs annotated from a lung radiograph.

The point sets are stored as complex numbers with the real part being the x-coordinate and the complex part being the y-coordinate.

3.A.2 Method

We will consider two similarity measures between curves in this exercise: 1) the sum of squared Euclidean distances between pairs of corresponding points; 2) the shape context based similarity measure described in (Belongie, Malik, & Puzicha, 2002) section 3.1. All points in one shape should have exactly one match in the other point set. This is know as a linear assignment problem and efficient algorithms to solve it exists. One of these is the so-called Hungarian algorithm, for which a matlab implementation is provided (hungarian.m).

3.A.3 Hints

3.A.3.1 Euclidean distance based similarity measure

The Hungarian algorithm takes a cost matrix $\mathbf{A}$ as input. This matrix should hold the pairwise Euclidean distances between all pairs of points in the two

curves to be aligned. Let **c** and **d** be the vector of points for each of the shapes, and N the number of points in each vector. The squared Euclidean distance matrix is then computed as follows:

```
Nc = size(c,1);
Nd = size(d,1);
a = repmat(c,1,Nd);
b = repmat(d.',Nc,1);
e = a-b;
A = e.*conj(e);
```

The Hungarian algorithm returns the correspondences in a vector C. In order to plot the result you can do as follows:

```
plot(c,'b');
plot(d,'g')
plot([c(C) d].','r')
```

Centroid of a shape `c`: `mean(c)`
Centered shape: `c0 = c-mean(c)`
Size of shape: `S = sqrt(c0'*c0)`
Centered and unit size shape: `cs0 = c0/S`
A smoothing TPS warp consist of a pair of TPS; one for the x-coordinates and one for the y-coordinates.

3.A.3.2 Shape context similarity measure

The shape context similarity is a more advanced similarity measure. The following matlab functions are provided:

- shapecontext.m - the shape context is computed for every point in a shape (set of points). The output is a matrix in which each row contains the shape context of a single point. The shape context is 3 radial levels by 8 angular levels. One parameter is required namely the radius of the shape context given in proportion to the maximum range of the data.
- computecostmatrix.m - the cost matrix for the linear assignment is computed from the shape context for the points in two shapes. Additional parameters are the number of dummy points and the dummy cost (see below).

3.A.4 Tasks

Find correspondences for the first 2 shapes in the meta datasets.

1. Make a crude alignment by centering and scaling each shape to unit size.
2. Compute correspondences using the Hungarian algorithm based on the Euclidean distance similarity measure.

For the lung dataset the outlines are annotated equidistantly with 200 points each. In order to allow for some local stretching and compression of the curves we will try to match a subsampled version of one lung outline to the other, i.e.,

```
c = lung1;
d = lung2(1:4:200);
```

Now, the Euclidean distance matrix **A** becomes non-square and we must expand with a number of dummy points:

```
B = [A 0.5*ones(200,150)];
```

In order to plot actual matches (not dummy matches) only we can do:

```
plot(c,'b+');
plot(d,'g+');
plot([c(C(1:50)) d].','r')
```

3. Try this on two shapes from the lung data set.

This latter case will in all likelihood fail at some parts of the shapes. Try solving the lung problem using the shape context method.

4. Compute shape contexts (rfrac = 0.25) for the two lung shapes.
5. Plot the context of some points and check that corresponding points have similar shape contexts (cf. fig. 3 in (Belongie et al., 2002)).
6. Compute the cost matrix.

7. Compute correspondences using the Hungarian algorithm.

Now, we have some extra information we can use, namely that the lung dataset consist of a left and a right part and the points are sampled equidistantly along the outlines. Implement these constraints by setting the cost in the cost matrix such that points from the left lung cannot match to the right lung and vice versa, and such that points from one lung dataset cannot shift more than $\pm 20\%$ along the outline.

8. Plot the new cost matrix as an image.
9. Carry out the matching.
10. Warp the second shape to the first one with a smoothing TPS warp, using the current estimates of the correspondences.
11. Re-compute correspondences between the first shape and the warped second shape using the Hungarian algorithm.
12. Go to 10 until convergence

References

S. Belongie, J. Malik, and J. Puzicha, "Shape matching and object recognition using shape contexts," *IEEE Transactions on Patter Analysis and Machine Intelligence*, vol. 24, no. 24, pp. 509–522, 2002.

3.B Exercise VI: Path tracing using dynamic programming

In this exercise you should use the abdominal MR scan in the file `abdomen-biascorrected.mat`. You should use dynamic programming to extract the outer contour of the subcutaneous fat layer as well as the inner contour of this fat layer.

3.B.1 Tasks

1. Extract a rectangular region of interest containing the entire abdominal section.
2. Make a polar representation of the image (see below).
3. Extract the radial derivatives (see below).
4. Extract the outer contour of the subcutaneous fat by dynamic programming.
5. Plot the results both in the polar image and the original image. Hint: in the code below `lcoords` contains the original image coordinates for each point in polar coordinates.
6. Apply dynamic programming to find the inner fat border. In order to discern this border from the previously found outer border, you can try to exploit the sign (negative/positive) of the edges, or alter the intensities in the polar image based on knowledge of the location of the outer border.
7. Plot the result of the inner border extraction.

3.B.2 Hints

3.B.2.1 Polar representation

```
load abdomen_biascorrected
IM = abdomencorrected( 37:183, 15:232 );

% Allocate arrays for pixel values and coordinates of lines extending
% from image center to the border as the spokes of a wheel
[nrows, ncols] = size(IM);
```

```
maxlen = ceil(0.5*norm([nrows ncols]));
nang = 360;
lines = zeros(nang,maxlen); % Pixel values along wheel spokes
lcoords = zeros(nang,maxlen,2);
for I=1:nang,
    % Compute unit vector in direction of spoke
    t = [cos(2*pi*I/nang+pi/2) sin(2*pi*I/nang+pi/2)];

    % Compute coordinates of pixels along spoke
    linecoords = ones(maxlen,1)*[ncols nrows]/2 + (0:maxlen-1)'*t;

    % Store coordinates as pixel index in original image
    lcoords(I,:,:) = linecoords;

    % Extract pixel values using bilinear interpolation
    lines(I,:) = interp2(IM,linecoords(:,1),linecoords(:,2),'linear')';
end

% Make sure we don't have NaN values anywhere
lines(isnan(lines)) = 0;
```

3.B.2.2 Radial derivatives

```
signedEdges = diff(lines')';
```

Chapter 4

Voxel-based segmentation

In the previous chapter, we briefly introduced methods for segmentation that aim at accurately locating models of relevant anatomical *boundaries* in the images. Another important class of segmentation algorithms concentrates on individual *voxels* instead, and aims at deciding exactly which voxels in the image belong to a specific structure of interest. Once such a voxel-based segmentation is obtained, a surface mesh representing the structure's outer boundary can easily be computed using fast meshing algorithms [14], if such a surface model is what is desired.

This chapter gives an overview of some of the most important concepts of voxel-based segmentation in biomedical image analysis.

4.1 Generative modeling framework

Voxel-based 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 $\mathbf{l} = (l_1, \ldots, l_I)^{\mathrm{T}}$ be a (vectorized) label image with a total of I voxels, with $l_i \in \{1, \ldots, K\}$ denoting the one of K possible labels assigned to voxel i, 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 $\mathbf{d} = (d_1, \ldots, d_I)^{\mathrm{T}}$ 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

$$p(\mathbf{l}|\mathbf{d}, \hat{\boldsymbol{\theta}}) = \frac{p(\mathbf{d}|\mathbf{l}, \hat{\boldsymbol{\theta}}_d)p(\mathbf{l}|\hat{\boldsymbol{\theta}}_l)}{p(\mathbf{d}|\hat{\boldsymbol{\theta}})}, \tag{4.1}$$

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:

$$\hat{\mathbf{l}} = \arg\max_{\mathbf{l}} p(\mathbf{l}|\mathbf{d}, \hat{\boldsymbol{\theta}}), \tag{4.2}$$

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

$$\sum_{\mathbf{l}} V_k(\mathbf{l})p(\mathbf{l}|\mathbf{d}, \hat{\boldsymbol{\theta}}) \tag{4.3}$$

[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 simply compares alternate segmentations by evaluating $p(\mathbf{d}|\mathbf{l}, \hat{\boldsymbol{\theta}}_d)p(\mathbf{l}|\hat{\boldsymbol{\theta}}_l)$ instead.

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

4.2 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

$$p(\mathbf{l}|\boldsymbol{\theta}_l) = \prod_i p(l_i|\boldsymbol{\theta}_l) \tag{4.4}$$

$$= \prod_i \pi_{l_i} \tag{4.5}$$

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 $\sum_k \pi_k = 1$. 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 (eq. 4.4), and each label occurs, on average, with a relative frequency of π_k (eq. 4.5).

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:

$$p(\mathbf{d}|\mathbf{l}, \boldsymbol{\theta}_d) = \prod_i p(d_i|l_i, \boldsymbol{\theta}_d), \tag{4.6}$$

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

$$p(d_i|l_i, \boldsymbol{\theta}_d) = \mathcal{N}(d_i|\mu_{l_i}, \sigma_{l_i}^2), \tag{4.7}$$

where

$$\mathcal{N}(d|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\big[-\frac{(d-\mu)^2}{2\sigma^2}\big] \tag{4.8}$$

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 $\mathbf{d}$:

$$\begin{aligned} p(\mathbf{d}|\boldsymbol{\theta}) &= \sum_{\mathbf{l}} p(\mathbf{d}|\mathbf{l},\boldsymbol{\theta}_d)p(\mathbf{l}|\boldsymbol{\theta}_l) \\ &= \sum_{\mathbf{l}} \left[\prod_i \mathcal{N}(d_i|\mu_{l_i},\sigma^2_{l_i}) \prod_i \pi_{l_i}\right] \qquad (4.9) \\ &= \prod_i p(d_i|\boldsymbol{\theta}) \qquad (4.10) \end{aligned}$$

with

$$p(d|\boldsymbol{\theta}) = \sum_k \mathcal{N}(d|\mu_k,\sigma^2_k)\pi_k. \qquad (4.11)$$

(Although the transition from eq. 4.9 to eq. 4.10 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 eq. 4.10.) Eq. 4.11 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 figure 4.1.

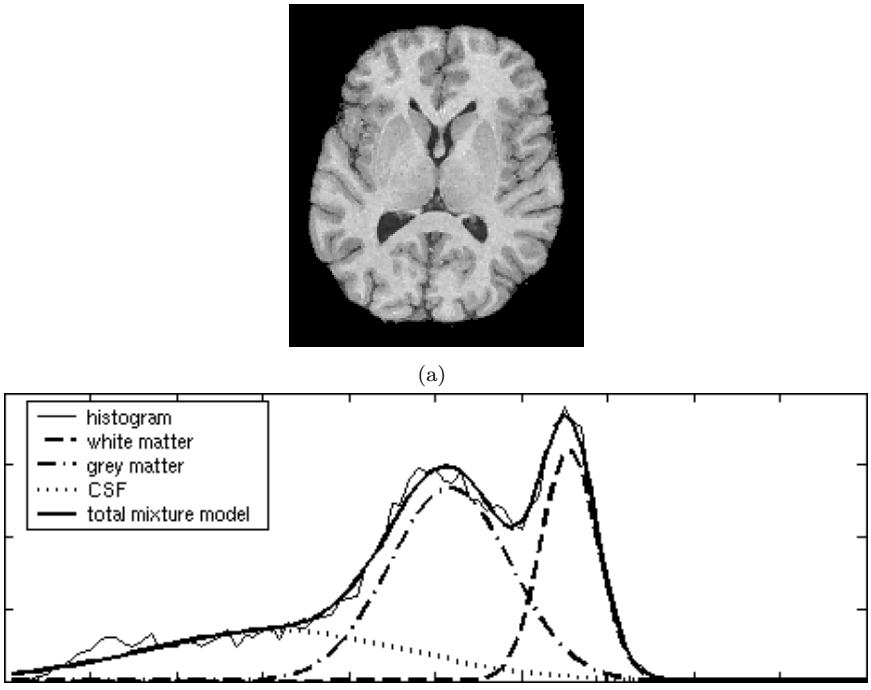
Because of the assumption of statistical independence between voxels, the segmentation posterior (eq. 4.1) reduces to a simple form that is factorized (i.e., appears as a product) over the voxels:

$$\begin{aligned} p(\mathbf{l}|\mathbf{d},\hat{\boldsymbol{\theta}}) &= \frac{p(\mathbf{d}|\mathbf{l},\hat{\boldsymbol{\theta}}_d)p(\mathbf{l}|\hat{\boldsymbol{\theta}}_l)}{p(\mathbf{d}|\hat{\boldsymbol{\theta}})} \\ &= \frac{\prod_i \mathcal{N}(d_i|\hat{\mu}_{l_i},\hat{\sigma}^2_{l_i}) \prod_i \hat{\pi}_{l_i}}{\prod_i \sum_k \mathcal{N}(d_i|\hat{\mu}_k,\hat{\sigma}^2_k)\hat{\pi}_k} \\ &= \prod_i p(l_i|d_i,\hat{\boldsymbol{\theta}}), \qquad (4.12) \end{aligned}$$

where

$$p(l_i|d_i,\hat{\boldsymbol{\theta}}) = \frac{\mathcal{N}(d_i|\hat{\mu}_{l_i},\hat{\sigma}^2_{l_i})\hat{\pi}_{l_i}}{\sum_k \mathcal{N}(d_i|\hat{\mu}_k,\hat{\sigma}^2_k)\hat{\pi}_k}. \qquad (4.13)$$

Therefore, the segmentation posterior is fully specified by each voxel's K posterior probabilities of belonging to each structure; such segmentation posteriors



(a)

(b)

Figure 4.1: 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.

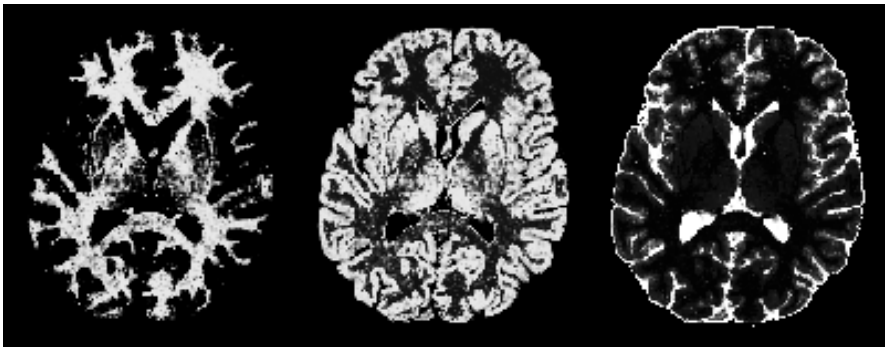


Figure 4.2: Visualization of the segmentation posterior corresponding to the data and model of figure 4.1. High intensities correspond to high probabilities and vice versa.

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 figure 4.1 is visualized in 4.2 this way. Evidently, the sum of all the structures' posterior probabilities add to one in each voxel: $\sum_k p(k|d_i, \hat{\boldsymbol{\theta}}) = 1, \forall i$.

Because of the factorized form of the segmentation posterior, the MAP segmentation (eq. 4.2) is simply given by

$$\hat{\mathbf{l}} = \arg\max_{\mathbf{l}} p(\mathbf{l}|\mathbf{d}, \hat{\boldsymbol{\theta}}) = \arg\max_{l_1,\ldots,l_I} p(l_i|d_i, \hat{\boldsymbol{\theta}}), \tag{4.14}$$

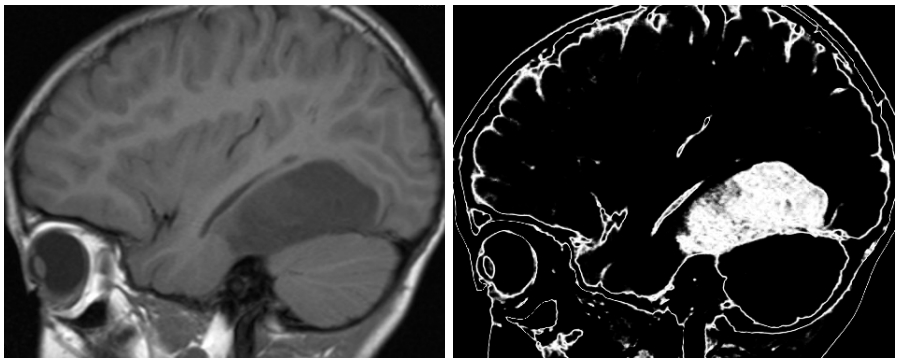
i.e., each voxel is assigned exclusively to the label with the highest posterior probability. Similarly, the expected volume of the anatomical structure corresponding to label k is given by (eq. 4.3)

$$\sum_{\mathbf{l}} V_k(\mathbf{l}) p(\mathbf{l}|\mathbf{d}, \hat{\boldsymbol{\theta}}) = \sum_i p(k|d_i, \hat{\boldsymbol{\theta}}), \tag{4.15}$$

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

4.3 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 (eq. 4.13). Although this works quite



(a) (b)

Figure 4.3: 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.

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 figure 4.3, 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.

4.3.1 Markov random field model

An often-used improvement to the simplistic prior of eq. 4.5 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

$$p(\mathbf{l}|\boldsymbol{\theta}_l) = \frac{1}{Z(\boldsymbol{\theta}_l)} \exp(-U(\mathbf{l}|\boldsymbol{\theta}_l)), \tag{4.16}$$

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 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

$$U(\mathbf{l}|\boldsymbol{\theta}_l) = \beta \sum_{(i,j)} \delta(l_i \neq l_j), \tag{4.17}$$

where the sum is running over all the voxel pairs (i, j) 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:

$$U(\mathbf{l}|\boldsymbol{\theta}_l) = \beta \sum_{(i,j)} \delta(l_i \neq l_j) - \sum_i \log(\pi_{l_i}). \tag{4.18}$$

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 eq. 4.5.

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 $\mathbf{l}_{\setminus i} = (l_1, \ldots, l_{i-1}, l_{i+1}, \ldots, l_I)^{\mathrm{T}}$

to denote the vector of labels in all voxels except voxel i, and $\mathfrak{N}_i$ the set of voxels that form neighboring pairs with voxel i, we have (Bayes' rule)

$$\begin{aligned} p(l_i|\mathbf{l}_{\setminus i}) &= \frac{p(\mathbf{l})}{p(\mathbf{l}_{\setminus i})} \\ &= \frac{p(\mathbf{l})}{\sum_{l_i} p(\mathbf{l})} \\ &= \frac{\exp(-U(\mathbf{l}|\boldsymbol{\theta}_l))}{\sum_{l_i} \exp(-U(\mathbf{l}|\boldsymbol{\theta}_l))} \\ &= \frac{\exp\big(-\beta \sum_{j\in\mathfrak{N}_i} \delta(l_i \neq l_j) + \log \pi_{l_i}\big)}{\sum_k \exp\big(-\beta \sum_{j\in\mathfrak{N}_i} \delta(l_j \neq k) + \log \pi_k\big)} \\ &= \frac{\pi_{l_i} \cdot \exp\big(-\beta \sum_{j\in\mathfrak{N}_i} \delta(l_i \neq l_j)\big)}{\sum_k \pi_k \cdot \exp\big(-\beta \sum_{j\in\mathfrak{N}_i} \delta(l_j \neq k)\big)}. \end{aligned} \tag{4.19}$$

The second to last step is explained by the fact that all the remaining terms of eq. 4.18 cancel out in the numerator and the denominator.

Comparing eq. 4.19 to the standard Gaussian mixture prior, the probability of having label k in voxel i 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 i will be higher than π_k.

4.3.2 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 (eq. 4.6 and 4.7), 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 (eq. 4.12).

There exist fast algorithms, based on so-called graph-cuts, for computing the MAP segmentation $\hat{\mathbf{l}} = \arg\max_{\mathbf{l}} p(\mathbf{l}|\mathbf{d}, \hat{\boldsymbol{\theta}})$ [15]. However, we are sometimes more interested in calculating *expectations*, for instance in order to estimate volumes of anatomical structures (eq. 4.3) or as part of Expectation-Maximization parameter optimizers (which we will cover in section 4.4). 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 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 (eq. 4.12) on $q(\mathbf{l})$, i.e., we chose $q(\mathbf{l})$ to be of the form

$$q(\mathbf{l}) = \prod_i q_i(l_i). \tag{4.20}$$

Within this family, our task is to chose the voxel-wise distributions $q_i(k)$ 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

$$KL\left(q(\mathbf{l}) \,||\, p(\mathbf{l}|\mathbf{d}, \hat{\boldsymbol{\theta}})\right) = -\sum_{\mathbf{l}} q(\mathbf{l}) \log \frac{p(\mathbf{l}|\mathbf{d}, \hat{\boldsymbol{\theta}})}{q(\mathbf{l})}, \tag{4.21}$$

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 eq. 4.20).

For a given set of distributions $q_j(l_j)$ in all voxels $j \neq i$, it can be shown [16] that the remaining voxel's distribution $q_i(l_i)$ that minimizes the KL-divergence is given by

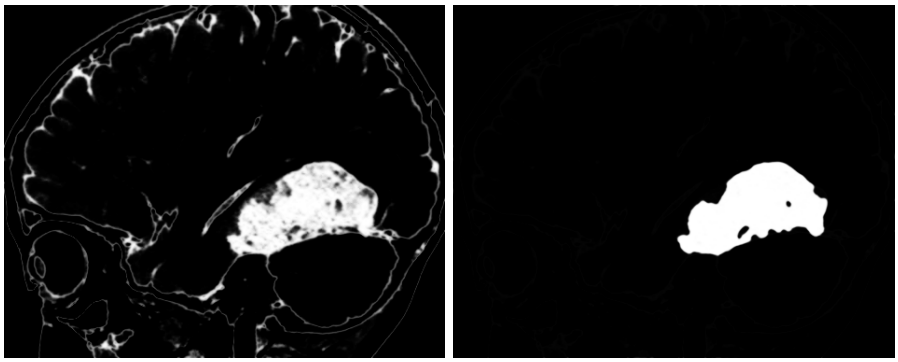
$$q_i(l_i) = \frac{\mathcal{N}(d_i|\hat{\mu}_{l_i}, \hat{\sigma}^2_{l_i})\gamma_i(l_i)}{\sum_k \mathcal{N}(d_i|\hat{\mu}_k, \hat{\sigma}^2_k)\gamma_i(k)} \tag{4.22}$$

with

$$\gamma_i(k) = \frac{\hat{\pi}_k \cdot \exp\left(-\beta \sum_{j \in \mathfrak{N}_i}(1 - q_j(k))\right)}{\sum_{k'} \hat{\pi}_{k'} \cdot \exp\left(-\beta \sum_{j \in \mathfrak{N}_i}(1 - q_j(k'))\right)}. \tag{4.23}$$

Comparing this with the voxel-wise posterior of the standard Gaussian mixture model (eq. 4.13), it can be seen that the usual class priors π_k are replaced with altered priors $\gamma_i(k)$ that take the local neighborhood of the voxels into account: as in eq. 4.19, the number of neighboring voxels "assigned" to a different class is "counted" (in a soft, weighted sense) and alters π_k accordingly.

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



(a) (b)

Figure 4.4: Visualization of the mean-field approximation to the segmentation posterior for the tumor data of figure 4.3: (a) with the Markov random field parameter β set to 0.25 and (b) 0.55.

Note that eq. 4.22 gives the optimal distribution $q_i(k)$ 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 $q_i(k)$ 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 $q_i(l_i)$'s may affect the local optimum of the KL divergence we arrive at.

A segmentation example on the tumor data of fig. 4.3 is shown in fig. 4.4, 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.

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

$$\sum_{\mathbf{l}} V_k(\mathbf{l}) q(\mathbf{l}) = \sum_i q_i(l_i). \tag{4.24}$$

4.4 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 $\boldsymbol{\theta}$:

$$\begin{aligned} \hat{\boldsymbol{\theta}} &= \arg\max_{\boldsymbol{\theta}} \left[p(\mathbf{d}|\boldsymbol{\theta}) \right] \\ &= \arg\max_{\boldsymbol{\theta}} \left[\log p(\mathbf{d}|\boldsymbol{\theta}) \right] . \end{aligned} \tag{4.25}$$

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 eq. 4.25 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 section 4.2 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 op-

[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.

timization methods that work by repeatedly constructing a lower bound to the objective function, maximizing that lower bound, and repeating the process until convergence [17]. This process is illustrated in figure 4.5. 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 $\tilde{\boldsymbol{\theta}}$:

$$Q(\tilde{\boldsymbol{\theta}}|\tilde{\boldsymbol{\theta}}) = \log p(\mathbf{d}|\tilde{\boldsymbol{\theta}}), \tag{4.26}$$

but that otherwise never exceeds it:

$$Q(\boldsymbol{\theta}|\tilde{\boldsymbol{\theta}}) \leq \log p(\mathbf{d}|\boldsymbol{\theta}), \ \forall \boldsymbol{\theta}. \tag{4.27}$$

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 eq. 4.26 and 4.27, 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 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 figure 4.6). 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*):

$$\log\left(\sum_k w_k x_k\right) \ \geq \ \sum_k w_k \log x_k \tag{4.28}$$

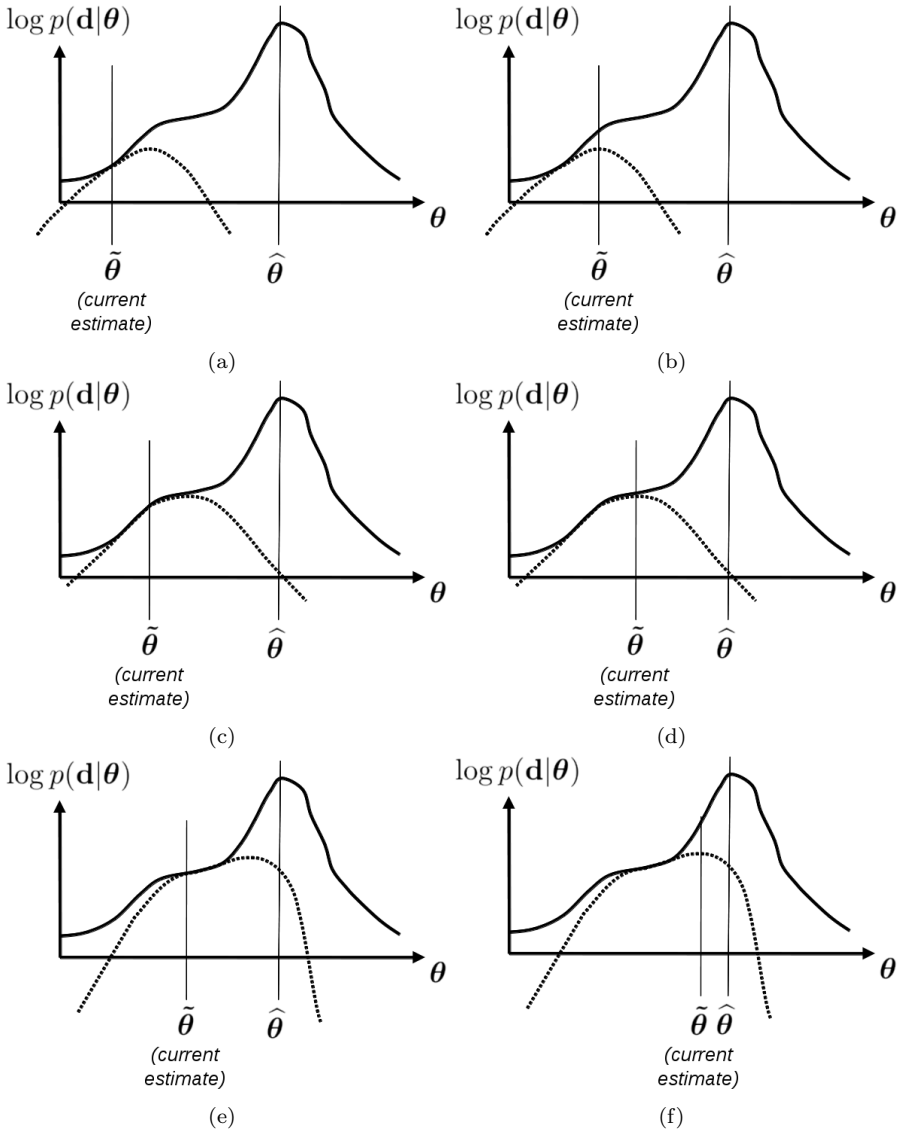


Figure 4.5: In the EM algorithm the maximum likelihood parameters $\hat{\boldsymbol{\theta}}$ are sought by repeatedly constructing a lower bound to the log likelihood function, in such a way that the lower bound touches the log likelihood function exactly at the current parameter estimate $\tilde{\boldsymbol{\theta}}$ (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.

where $w_k \geq 0$ and $\sum_k w_k = 1$, 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 $p(\mathbf{d}|\boldsymbol{\theta}) = \prod_i \left[\sum_k \mathcal{N}(d_i|\mu_k, \sigma_k^2)\pi_k\right]$ (eq. 4.10 and 4.11), we have that

$$\begin{aligned}
\log p(d|\boldsymbol{\theta}) &= \log\left(\prod_i \left[\sum_k \mathcal{N}(d_i|\mu_k, \sigma_k^2)\pi_k\right]\right) && (4.29)\\
&= \sum_i \log\left[\sum_k \mathcal{N}(d_i|\mu_k, \sigma_k^2)\pi_k\right] && (4.30)\\
&= \sum_i \log\left[\sum_k \left(\frac{\mathcal{N}(d_i|\mu_k, \sigma_k^2)\pi_k}{w_k^i}\right) w_k^i\right] && (4.31)\\
&\geq \underbrace{\sum_i \left[\sum_k w_k^i \log\left(\frac{\mathcal{N}(d_i|\mu_k, \sigma_k^2)\pi_k}{w_k^i}\right)\right]}_{Q(\boldsymbol{\theta}|\tilde{\boldsymbol{\theta}})}, && (4.32)
\end{aligned}$$

for any set of weights $\{w_k^i\}$ that satisfy $w_k^i \geq 0$ and $\sum_k w_k^i = 1$ (the last step relies on eq. 4.28). We now have a lower bound function $Q(\boldsymbol{\theta}|\tilde{\boldsymbol{\theta}})$ that satisfies eq. 4.27, but not eq. 4.26, so we are not done yet. Instead of randomly assigning *any* valid K weights w_k^i to each voxel i (one weight for each label k), we can satisfy eq. 4.26 by choosing the weights so that

$$w_k^i = \frac{\mathcal{N}(d_i|\tilde{\mu}_k, \tilde{\sigma}_k^2)\tilde{\pi}_k}{\sum_{k'} \mathcal{N}(d_i|\tilde{\mu}_{k'}, \tilde{\sigma}_{k'}^2)\tilde{\pi}_{k'}}. \tag{4.33}$$

By filling these weights into the definition of our lower bound (eq. 4.32), it is easy to check that eq. 4.26 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

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

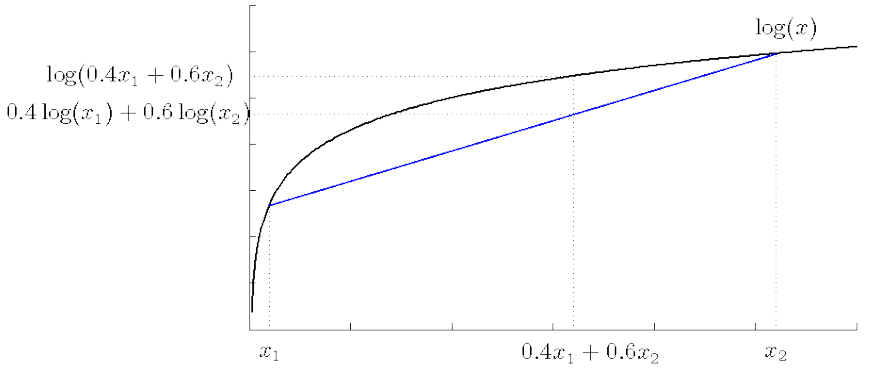


Figure 4.6: Because the logarithm is a concave function, the cord between any two points on its curve lies on or below the curve. An example cord is shown in blue.

Reformulating the lower bound as

$$\begin{aligned} Q(\boldsymbol{\theta}|\tilde{\boldsymbol{\theta}}) &= -\frac{1}{2}\sum_k \left[\frac{1}{\sigma_k^2}\sum_i w_k^i \left(d_i - \mu_k\right)^2 + \left(\sum_i w_k^i\right)\log\sigma_k^2\right] \\ &\quad + \sum_k \left[\left(\sum_i w_k^i\right)\log\pi_k\right] \\ &\quad - \sum_i \sum_k w_k^i \log w_k^i - \frac{N}{2}\log\left(2\pi\right), \end{aligned} \tag{4.34}$$

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

$$\begin{aligned} \tilde{\mu}_k &\leftarrow \frac{\sum_i w_k^i d_i}{\sum_i w_k^i} \\ \tilde{\sigma}_k^2 &\leftarrow \frac{\sum_i w_k^i \left(d_i - \tilde{\mu}_k\right)^2}{\sum_i w_k^i} \\ \tilde{\pi}_k &\leftarrow \frac{\sum_i w_k^i}{N}. \end{aligned} \tag{4.35}$$

It is worth spending some time thinking about these equations. The EM algorithm searches for the maximum likelihood parameters of the Gaussian mixture

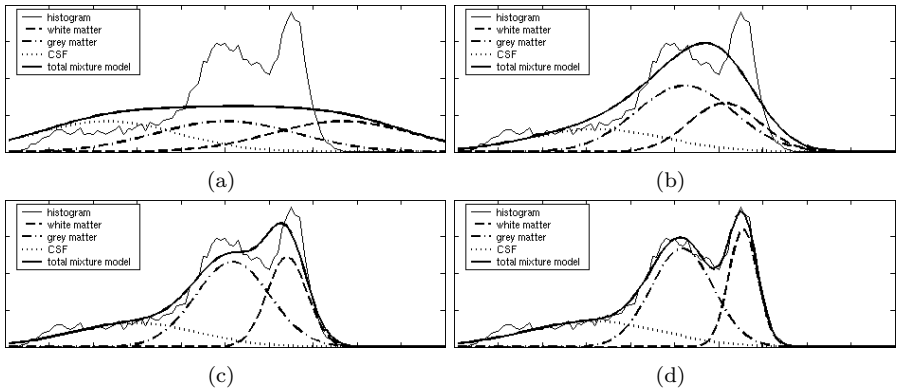


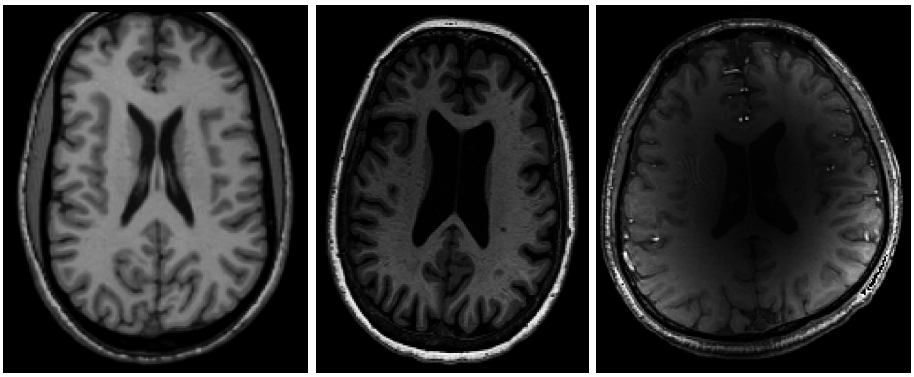
Figure 4.7: Iterative improvement of the Gaussian mixture model parameters for the MR image of figure 4.1(a), using the EM algorithm: initialization (a) and parameter estimate after one (b), 10 (c) and 30 (d) iterations.

model merely be repeatedly applying the update rules of eq. 4.35, where the weights w_k^i are defined in eq. 4.33. These weights depend themselves on the current estimate of the model parameters, which explains why the algorithm involves iterating. More importantly, by comparing eq. 4.33 to eq. 4.13, 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 figure 4.2 based on its current parameter estimate, and then updates the parameter estimate accordingly. The update rules of eq. 4.35 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.

Figure 4.7 shows a few iterations of the EM algorithm searching for the maximum likelihood parameters of the brain MR data shown in figure 4.1(a).

4.5 Modeling MR bias fields

Although the Gaussian mixture model is a very useful tool for voxel-based 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



(a) (b) (c)

Figure 4.8: The MR bias field artifact is more pronounced in more recent scanners operating at higher magnetic field strengths: typical brain scan acquired on a 1.5 (a), 3 (b), and 7 (c) Tesla machine.

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 being imaged and therefore unique to each scan session, and is much more pronounced in the newest generation of scanners (see figure 4.8).

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

$$\sum_{m=1}^{M} c_m \phi_m^i, \tag{4.36}$$

where ϕ_m^i is shorthand for $\phi_m(\mathbf{x}_i)$, the value of the mth basis function evaluated at voxel i, which has spatial location $\mathbf{x}_i$. Suitable basis functions can be the sine or cosine functions shown in figure 1.3, uniform B-spline basis functions, 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

[4]Because of the physics of MR, the bias field is better modeled as a multiplicative rather

resulting intensity image to obtain the final, bias field corrupted image $\mathbf{d}$. With this model, we have

$$p(\mathbf{d}|\mathbf{l}, \boldsymbol{\theta}_d) = \prod_i \mathcal{N}\Big(d_i - \sum_m c_m \phi_m^i \;\Big|\mu_{l_i}, \sigma_{l_i}^2\Big) \tag{4.37}$$

with parameters $\boldsymbol{\theta}_d = (\mu_1, \ldots, \mu_K, \sigma_1^2, \ldots, \sigma_K^2, c_1, \ldots, c_M)^{\mathrm{T}}$, 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.

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

$$\begin{aligned} Q(\boldsymbol{\theta}|\tilde{\boldsymbol{\theta}}) \;=\; & -\frac{1}{2}\sum_k \left[\frac{1}{\sigma_k^2}\sum_i w_k^i \left(d_i - \mu_k - \sum_m c_m \phi_m^i\right)^2 + \left(\sum_i w_k^i\right)\log\sigma_k^2\right] \\ & + \sum_k \left[\left(\sum_i w_k^i\right)\log\pi_k\right] \\ & - \sum_i \sum_k w_k^i \log w_k^i - \frac{N}{2}\log(2\pi)\,, \end{aligned} \tag{4.38}$$

where now the weights satisfying eq. 4.26 are given by

$$w_k^i = \frac{\mathcal{N}\Big(d_i - \sum_m \tilde{c}_m \phi_m^i \;\Big|\tilde{\mu}_k, \tilde{\sigma}_k^2\Big)\tilde{\pi}_k}{\sum_{k'} \mathcal{N}\Big(d_i - \sum_m \tilde{c}_m \phi_m^i \;\Big|\tilde{\mu}_{k'}, \tilde{\sigma}_{k'}^2\Big)\tilde{\pi}_{k'}}. \tag{4.39}$$

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

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.

parameters, we easily obtain

$$\begin{aligned}
\tilde{\mu}_k &\leftarrow \frac{\sum_i w_k^i \left(d_i - \sum_m \tilde{c}_m \phi_m^i\right)}{\sum_i w_k^i} \\
\tilde{\sigma}_k^2 &\leftarrow \frac{\sum_i w_k^i \left(d_i - \sum_m \tilde{c}_m \phi_m^i - \tilde{\mu}_k\right)^2}{\sum_i w_k^i} \\
\tilde{\pi}_k &\leftarrow \frac{\sum_i w_k^i}{N}.
\end{aligned} \tag{4.40}$$

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 section 1.3). In particular, the solution is given by

$$\tilde{\mathbf{c}} \leftarrow (\mathbf{\Phi}^{\mathrm{T}} \mathbf{S} \mathbf{\Phi})^{-1} \mathbf{\Phi}^{\mathrm{T}} \mathbf{S} \mathbf{r} \tag{4.41}$$

where

$$\mathbf{\Phi} = \begin{pmatrix} \phi_1^1 & \phi_2^1 & \dots & \phi_M^1 \\ \phi_1^2 & \phi_2^2 & \dots & \phi_M^2 \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1^N & \phi_2^N & \dots & \phi_M^N \end{pmatrix}$$

and

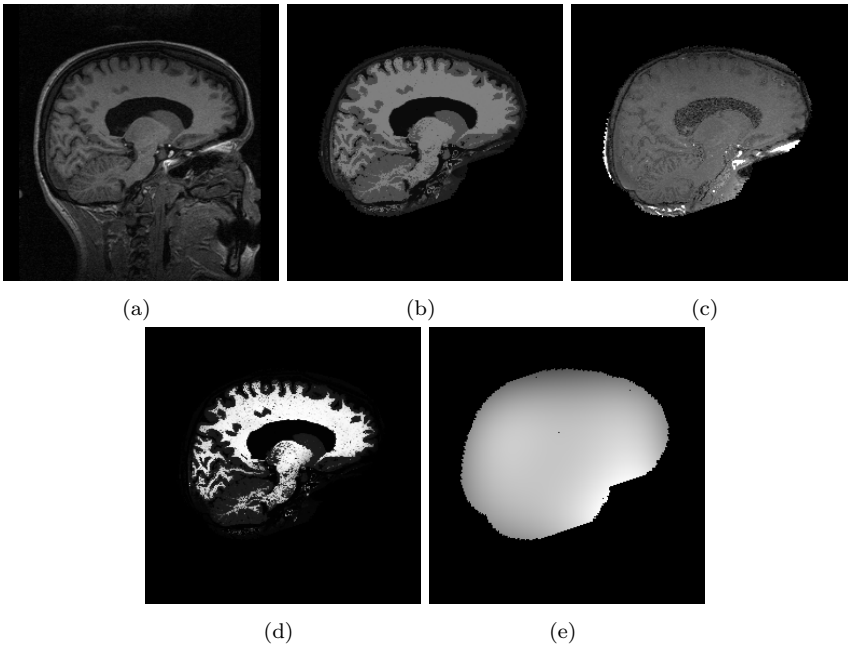
$$s_k^i = \frac{w_k^i}{\tilde{\sigma}_k^2}, \quad s_i = \sum_k s_k^i, \quad \boldsymbol{S} = \mathrm{diag}(s_i), \quad \tilde{d}_i = \frac{\sum_k s_k^i \tilde{\mu}_k}{\sum_k s_k^i}, \quad \boldsymbol{r} = \begin{pmatrix} d_1 - \tilde{d}_1 \\ \vdots \\ d_N - \tilde{d}_N \end{pmatrix}.$$

Since eq. 4.40 and eq. 4.41 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 w_k^i (eq. 4.39) and then update the Gaussian mixture model parameters (eq. 4.40) and subsequently the bias field parameters (eq. 4.41) 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 w_k^i represent again a statistical classification of the image voxels as in figure 4.2, and the mixture model parameters are again updated accordingly. The only difference now is that instead of the original MR intensities, d_i, the bias field corrected intensities, $d_i - \sum_m \tilde{c}_m \phi_m^i$, 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 figure 4.9(b)), subtracts that from the MR scan to obtain a (noisy) estimate of the bias field (image $\mathbf{r}$, shown in figure 4.9(c)), and smoothes the result to obtain an estimate of the bias field (shown in figure 4.9(e)). For the smoothing, each voxel has a weight s_i (shown in figure 4.9(d)) 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).

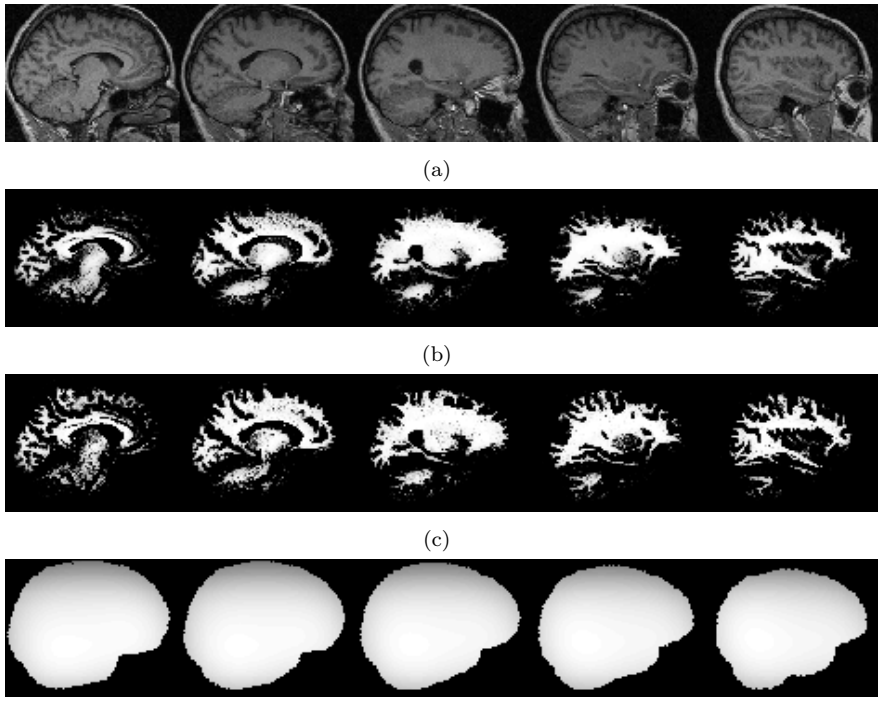
By extending the Gaussian mixture model with an explicit model for the bias field artifact this way, it is possible to obtain high-quality segmentations of MR scans without errors caused by intensity inhomogeneities, as shown in figure 4.10.



(a) (b) (c)

(d) (e)

Figure 4.9: Illustration of the bias field estimation within a generalized expectation-maximization algorithm: MR scan (a); reconstruction (b); difference between the MR scan and the reconstruction (c); weight image (d); and estimated bias field (e). See text for more details.



(a)

(b)

(c)

(d)

Figure 4.10: 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.

4.A Exercise VII: Brain tumor segmentation

In this exercise you will automatically segment a brain tumor from a head MR scan using a Gaussian mixture model. You will also augmented this simple model with a Markov random field prior, for which you will be using the mean-field approximation to do approximate inference on the segmentation posterior.

4.A.1 Data

The MR data for this exercise (a single 2-dimensional slice) is given in the file `tumorData.mat`.

4.A.2 Tasks

1. Display the image and manually click on about 20 points that are inside the tumor. Compute the mean and variance of the intensities of the selected pixels.

2. Compute the mean and variance of *all* the image intensities as well.

3. Using a two-component Gaussian mixture model (i.e., $K = 2$) with means and variances as computed above, and the prior probability associated with the Gaussian representing the tumor set to around 0.2, compute and display the pixels' posterior probabilities of the two classes. Also display the posterior probabilities of the tumor class overlaid on the MR scan (see below).

4. For a Markov random field prior with parameter $\beta = 0.3$, compute the mean-field approximation of the segmentation posterior using the iterative minimization algorithm described in section 4.3.2. You should use a neighborhood of 8 pixels (i.e., all the pixels in a 3×3 window except for the central one are considered to be neighbors of the central pixel), and employ the Markov random field penalty functional defined in eq. 4.18. Initialize the algorithm with the standard Gaussian mixture model's posterior probabilities computed above, and use enough iterations (e.g., 100) to make sure you have reached a (local) optimum, where one iteration is defined as visiting all pixels in the image once. Display the resulting pixel-specific $q_i(k)$'s as images, and describe the visual difference with these and the pixels' posterior probabilities using the standard Gaussian mixture model.

5. Repeat the same experiment for $\beta = 0$, $\beta = 0.6$, and $\beta = 2.5$. Display and comment on the results.

6. For $\beta = 2.5$, report the expected area of the tumor (given in number of pixels).

4.A.3 Hints

4.A.3.1 Efficient Matlab implementation of $\sum_{j \in \mathfrak{N}_i}(1 - q_j(k))$

In order to implement the mean-field approximation equations, you will need to repeatedly compute the quantity $\sum_{j \in \mathfrak{N}_i}(1-q_j(k))$, which counts (in a soft sense) the number of neighbors of pixel i that are currently classified to something else than class k. This poses something of a challenge because implementing this in a naive way (by looping over all the pixels and each time looking up the classification of the current pixel's neighbors) is extremely slow in Matlab. The following code snippet will compute $\sum_{j \in \mathfrak{N}_i}(1 - q_j(k))$ in all pixels simultaneously, in a manner that is optimized for Matlab processing:

```
nonPadded = 1 - qOfClassK;
padded = zeros( size( nonPadded ) + 2 );
padded( 2 : end-1, 2: end-1 ) = nonPadded;
numberOfNeighborsAssignedDifferently = ...
                  padded( [ 2 : end-1 ] - 1, [ 2 : end-1 ] ) + ...
                  padded( [ 2 : end-1 ] + 1, [ 2 : end-1 ] ) + ...
                  padded( [ 2 : end-1 ], [ 2 : end-1 ] + 1 ) + ...
                  padded( [ 2 : end-1 ], [ 2 : end-1 ] - 1 ) + ...
                  padded( [ 2 : end-1 ] - 1, [ 2 : end-1 ] - 1 ) + ...
                  padded( [ 2 : end-1 ] + 1, [ 2 : end-1 ] - 1 ) + ...
                  padded( [ 2 : end-1 ] - 1, [ 2 : end-1 ] + 1 ) + ...
                  padded( [ 2 : end-1 ] + 1, [ 2 : end-1 ] + 1 );
```

4.A.3.2 Efficient Matlab implementation of visiting every pixel in turn

Recall that the mean-field update given by eq. 4.22 is valid only when you are updating one pixel at a time. Again this is tricky to implement in Matlab because visiting all pixels in turn is hugely inefficient in Matlab. Notice, however, that we can define sets of pixels that are not neighbors of one another and that

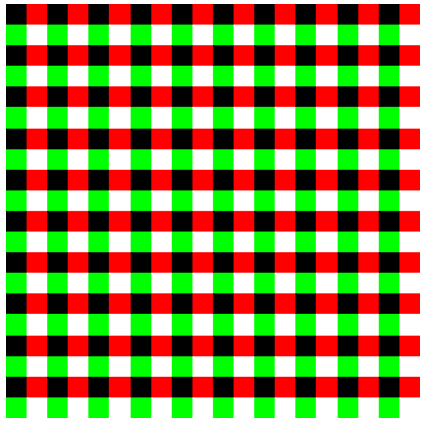


Figure 4.11: In a 2-D Markov random field model with an 8-neighborhood system, the pixels depicted in the same color here can be updated simultaneously.

we can therefore update simultaneously – as long as we keep the estimates in all other pixels fixed at their current values. An example of such a division of pixels is shown in figure 4.11: note that all the white pixels can be updated at the same time since they are not neighbors of one another, and that the same is true for all black, green, and red pixels.

The following is a Matlab snippet computing the indices of the pixels corresponding to the various colors:

```
tmp1 = ones( size( data, 1 ), 1 );
tmp1( 1 : 2 : end ) = 0;
tmp2 = ones( 1, size( data, 2 ) );
tmp2( 1 : 2 : end ) = 0;
whiteIndices = find( kron( tmp1, tmp2 ) );
blackIndices = find( kron( ~tmp1, ~tmp2 ) );
redIndices = find( kron( ~tmp1, tmp2 ) );
greenIndices = find( kron( tmp1, ~tmp2 ) );
```

Applying the update of eq. 4.22 by visiting each pixel exactly once can now be achieved by doing

```
indicesToUpdateCell{1} = whiteIndices;
indicesToUpdateCell{2} = blackIndices;
indicesToUpdateCell{3} = redIndices;
indicesToUpdateCell{4} = greenIndices;
```

```
for colorIndex = 1 : 4
  indicesToUpdate = indicesToUpdateCell{ colorIndex };

  % Here some work in all indicesToUpdate simultaneously
end
```

4.A.3.3 Overlaying a probabilistic tumor segmentation on MR intensities

```
tmp = repmat( data, [ 1 1 3 ] );
red = zeros( [ size( data ) 3 ] );
red( :, :, 1 ) = 255;
tmp = tmp .* repmat( classification( :, :, 1 ), [ 1 1 3 ] ) + ...
      red .* repmat( classification( :, :, 2 ), [ 1 1 3 ] );
imshow( uint8(tmp) )
```

4.B Exercise VIII: Expectation-maximization algorithm

In this exercise you will estimate the maximum likelihood parameters of a 3-component Gaussian mixture model for a brain MR scan, using the expectation-maximization algorithm.

4.B.1 Data

The MR data for this exercise consists of a 2-dimensional slice of a brain scan in the axial orientation. The file `segmentData.mat` contains the original slice as well as a mask that excludes non-brain tissues such as eyeballs, fat, skin, etc. The file `correctedData.mat` contains the same image, but after it has been corrected for the MR bias field artifact using the method described in section 4.5. It is this image you will be modeling.

4.B.2 Tasks

1. Display the image and plot its histogram.
2. Compute the minimum and maximum intensity in the image (non-zero pixels), and divide the intensity range up into three equally wide intervals. Initialize the parameters of a 3-component Gaussian mixture model by setting the means μ_k of the Gaussians to the centers of the intensity intervals, the variances σ_k^2 to the square of the width of the intervals, and the prior weights π_k to $1/3$ each.
3. Overlay the resulting Gaussian mixture model on the histogram (see below) by plotting each Gaussian weighted by its π_k, as well as the total weighted sum of all three Gaussian distributions (as in fig. 4.1(b)).
4. Compute and display the values of w_k^i defined by eq. 4.33.
5. Estimate the maximum likelihood parameters by iterating between updating the model parameter estimate according to eq. 4.35, and recomputing the w_k^i's according to eq. 4.33. Make sure to perform enough iterations (e.g., 100) for the algorithm to converge. As the iterations progress, plot the evolution of the log likelihood function, and update each time the display of the w_k^i's as well as the Gaussian mixture model plot overlaid on the histogram. Include the evolution of the log likelihood function and the plot of the final w_k^i's and the final mixture model in your report.

6. Keeping all other parameters fixed to their estimated values, vary only μ_2 (the mean of the middle Gaussian distribution) between the estimated values of μ_1 and μ_3 in about 100 steps, and plot for each step the log likelihood function.

7. On the same figure, also plot the lower bound $Q(\boldsymbol{\theta}|\tilde{\boldsymbol{\theta}})$ corresponding to the parameter vector $\tilde{\boldsymbol{\theta}}$ in which all parameters are set to their estimated values, except μ_2 which is set to the estimated value of μ_1.

8. Compute the value for μ_2 that maximizes this lower bound (first line of eq. 4.35), indicate its location on the figure, and comment on the result.

4.B.3 Hints

The following Matlab code snippet plots the histogram of the bias field corrected image with a Gaussian distribution overlaid:

```
% Get the data
load correctedData
mask = ( correctedData > 0 );

% Compute some reasonable mean and variance for the Gaussian distribution
intensities = correctedData( find( mask ) );
mean = sum( intensities ) / length( intensities );
variance = sum( ( intensities - mean ).^2 ) / length( intensities );

% Compute the histogram and its properties
[ histogram binCenters ] = hist( intensities, 64 );
pdf = histogram / sum( histogram );
binSize = binCenters(2) - binCenters(1);

% Display
figure
bar( binCenters, pdf );
grid
hold on
gauss = 1/sqrt( 2 * pi * variance ) * ...
        exp( -( binCenters - mean ).^2 / 2 / variance );
plot( binCenters, gauss * binSize, 'g' )
```

Chapter 5

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 4, 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 an-

notating 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.

5.1 Logistic regression

In Chapter 4 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 intensity 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. Eq. (4.13))

$$p(l = k|d, \boldsymbol{\theta}) = \frac{\mathcal{N}(d|\mu_k, \sigma_k^2)\pi_k}{\sum_{k'} \mathcal{N}(d|\mu_{k'}, \sigma_{k'}^2)\pi_{k'}}, \tag{5.1}$$

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 $\{\mathbf{x}_i, y_i\}_{i=1}^N$, where $\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots, x_{ip})^{\mathrm{T}}$ is the vector of observed inputs for the ith case, and y_i the corresponding output. For voxel-wise classification, each input is simply an intensity $\mathbf{x}_i = d_i$, i.e., the input dimensionality is $p = 1$, but we will soon see cases where $\mathbf{x}_i$ is higher-dimensional. Unlike in the regression case, where each output y_i 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 $y_i \in \{0, 1\}$, where values 1 and 0 correspond to the class assignments $l_i = 1$ and $l_i = 2$, 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

$$f(\mathbf{x}) = \sigma\left(\sum_{m=1}^{M} \beta_m \phi_m(\mathbf{x})\right), \tag{5.2}$$

[1] The generalization to several classes is rather straightforward (cf. [18], chapter 4).

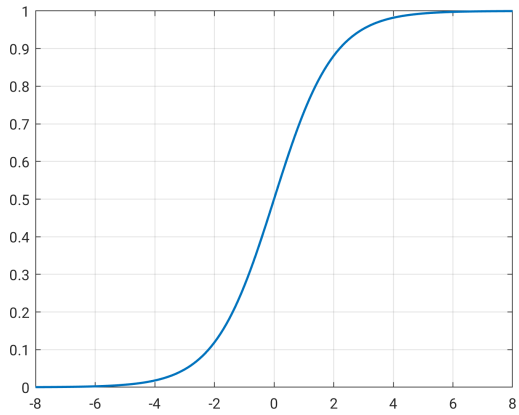


Figure 5.1: The logistic function $\sigma(a)$ defined in Eq. (5.4) maps the domain $[-\infty, \infty]$ into $[0, 1]$.

which is similar to the linear basis function model for regression we saw in Section 1.3: A linear combination of M nonlinear basis functions $\phi_m(\mathbf{x})$ with tunable coefficients β_m, which we here collect in the parameter vector $\boldsymbol{\theta} = (\beta_1, \ldots, \beta_M)^{\mathrm{T}}$. Unlike in the regression case, however, where the quantity

$$a = \sum_{m=1}^{M} \beta_m \phi_m(\mathbf{x}) \tag{5.3}$$

was used directly, here it is subject to a "squashing" function (illustrated in Fig. 5.1)

$$\sigma(a) = \frac{1}{1 + \exp(-a)} \tag{5.4}$$

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

$$p(y|\mathbf{x}, \boldsymbol{\theta}) = f(\mathbf{x})^y [1 - f(\mathbf{x})]^{1-y}. \tag{5.5}$$

Because the mapping function $\sigma(a)$ in Eq. (5.4) is called the "logistic function", the model of Eq. (5.2) 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 $\mathbf{y} = \{y_1, \ldots, y_N\}$, appropriate values for the model parameters $\hat{\boldsymbol{\theta}}$ can be found by numerically maximizing the likelihood function

$$p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta}) = \prod_{i=1}^{N} p(y_i|\mathbf{x}_i, \boldsymbol{\theta}) \tag{5.6}$$

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

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

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

Figure 5.2 illustrates this procedure on an MRI scan of the kidneys, using the five 1D cosines of Fig. 1.3(a) as basis functions $\phi_m(\mathbf{x})$ and $N = 50$ training points.

5.2 Training with stochastic gradient descent

So far we have not specified how to numerically optimize Eq. (5.6) 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 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 Eq. (5.5) and Eq. (5.6), training therefore consists of minimizing the cost function

$$E_N(\boldsymbol{\theta}) = -\log p(\mathbf{y}|\mathbf{X},\boldsymbol{\theta}) = -\sum_{i=1}^{N}\left\{y_i \log f(\mathbf{x}_i) + (1-y_i)\log\left[1-f(\mathbf{x}_i)\right]\right\}, \quad (5.7)$$

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 cost 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

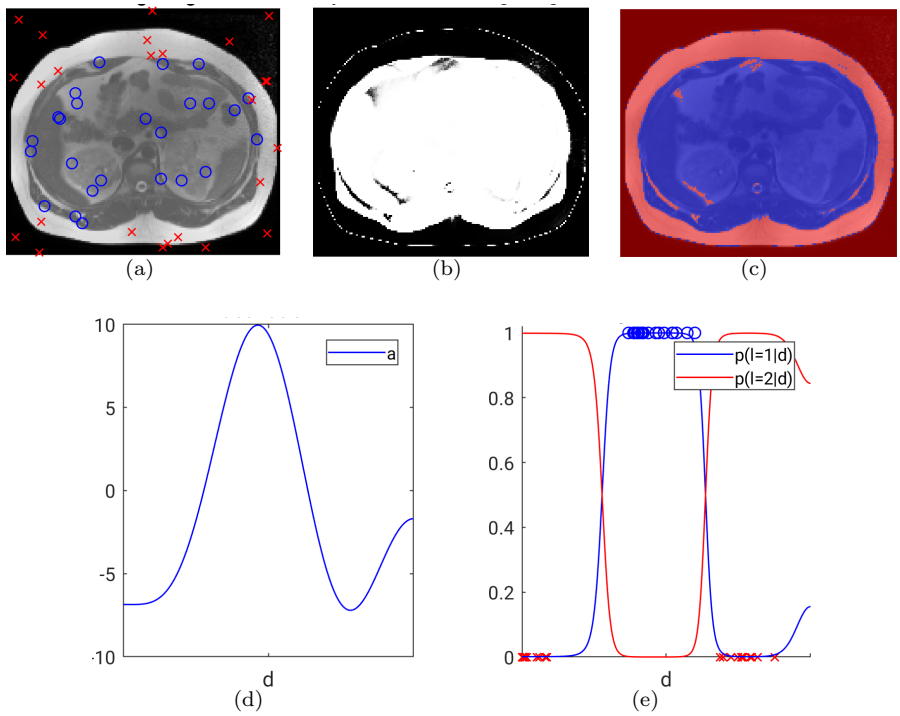


Figure 5.2: Example of a logistic regression classifier trained to discern pixels in the inner organs from pixels in other areas. (a) Image to be segmented, along with training data consisting of 25 manually selected points inside ($l = 1$, blue circles) and 25 points outside ($l = 0$, red crosses) the 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 Eq. (5.3) for various intensity levels d. (e) The classifier resulting from subjecting a to the squashing function $\sigma(a)$ of Eq. (5.4). For reference, the intensity levels and labels of the 50 training points are also shown.

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

$$\boldsymbol{\theta}^{(\tau+1)} = \boldsymbol{\theta}^{(\tau)} - \nu' \nabla E_{N'}(\boldsymbol{\theta}^{(\tau)}) \tag{5.8}$$

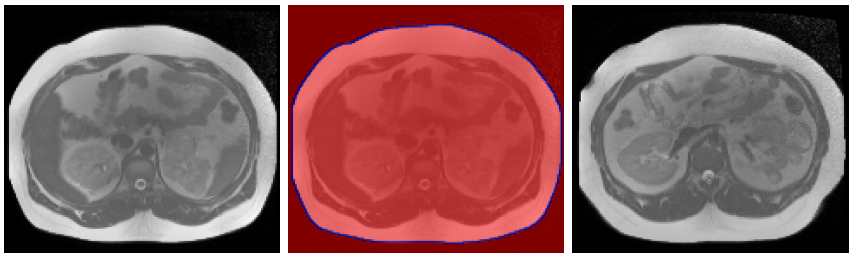
with step size $\nu' = N\nu/N'$, after which a new subset of size N' is sampled for the next iteration.

5.3 Feed-forward network functions

Although we have so far only considered a problem where the dimensionality of $\mathbf{x}$ is $p = 1$, classifiers can also be used in cases where $p \gg 1$. 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. 5.3(b)). Our task is now to automatically segment the same border in other MRI slices and/or patients (an example is shown in Fig. 5.3(c)). Since the border cannot be discerned based on intensity alone, a simple voxel-wise classifier will no longer suffice. Instead, we can take 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 $p = 9$.

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., $p = 2$ or $p = 3$) we could follow the same procedure as in Section 2.5 for obtaining tensor B-splines: a 2D B-spline is obtained by taking the outer product of two 1D B-splines. For the five 1D cosines used in the earlier example (shown in Fig. 1.3(a)) this procedure would result in the 25 2D cosines shown in Fig. 1.3(b). 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 $p = 9$, 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



(a) (b) (c)

Figure 5.3: Training image (a) and corresponding segmentation (b) used for training the network shown in Fig. 5.4, as well as the type of image (c) that the network should be able to segment after training.

that are also optimized during training. A typical choice of such adaptive basis functions is as follows:

$$\phi_m(\mathbf{x}) = \begin{cases} 1 & \text{if } m = 1, \\ \sigma\left(\sum_{j=1}^{p} w_{m,j} x_j + w_{m,0}\right) & \text{otherwise,} \end{cases} \tag{5.9}$$

where $\sigma(\cdot)$ is the logistic function defined in Eq. (5.4), and the weights $\{w_{m,j}\}$ are extra parameters that together with the coefficients $\{\beta_m\}$ form the parameters $\boldsymbol{\theta}$ of the model. Fig. 5.4 has a graphical representation of the resulting model for the case $p = 9$ 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_j (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 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* [19]).

Fig. 5.5 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. 5.3(b) using stochastic gradient descent. The network output $f(\mathbf{x})$ for each pixel is shown in Fig. 5.5(b), and the basis function evaluations $\{\phi_m(\mathbf{x})\}_{m=2}^{M}$ (called *feature maps*) and their coefficients $\{\beta_m\}_{m=2}^{M}$ are shown in Fig. 5.5(d). Since we are working with 3×3 images patches, the set of weights $\{w_{m,j}, j = 1, \ldots, 9\}$ for

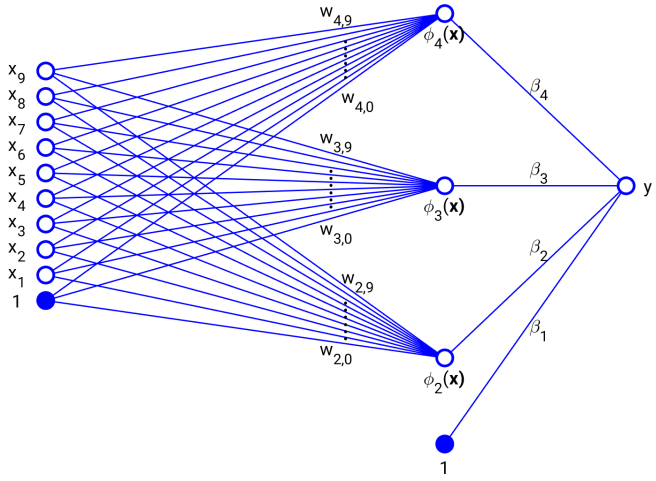
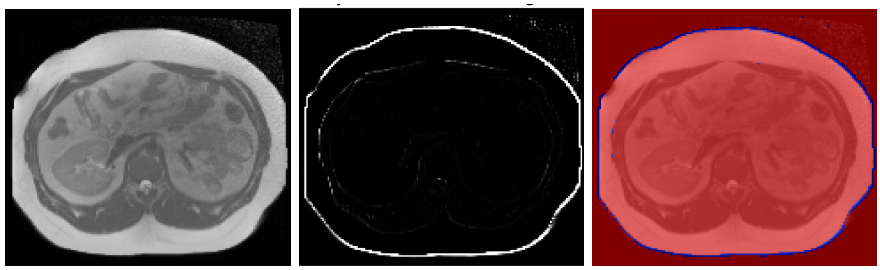
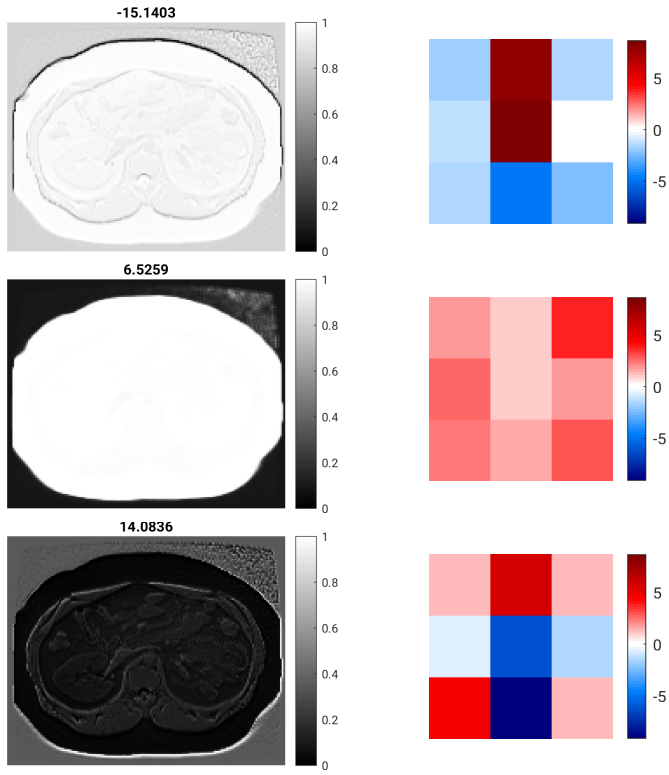


Figure 5.4: Graphical representation of a feed-forward neural network with a 9-dimensional input, a single hidden layer with three hidden units, and a single output unit. Since the parameters β_1 and $\{w_{m,0}\}$ effectively add mere constants during the computations, they are shown to be connected to additional units with values clamped to 1 (indicated by filled circles). The information flows from the left to the right through the network.

each feature $m = 2, \ldots, 4$ has a spatial structure and can be visualized as a 3×3 image itself, shown in Fig. 5.5(e). Since computing the term $\sum_{j=1}^{p} w_{m,j} x_j$ in Eq. (5.9) 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* [20]. 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. 5.5(d) 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.



(a) (b) (c)



(d) (e)

Figure 5.5: Segmentation of a new image using the patch-based network of Fig. 5.4, after training it on the data of Fig. 5.3(b): (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.

5.4 Exercise XI: Neural networks

In this exercise you will implement the neural network of Fig. 5.4 in several steps. In each step you will train the model on an image-segmentation pair that is provided, and use it to segment an unseen scan. You will also be asked to critically analyze the behavior of the model in different scenarios.

5.4.1 Data

The data for this exercise consists of the files `dataForNN_inside_clip0.mat`, `dataForNN_foreBorder_clip0.mat` and `dataForNN_foreBorder_clip1.mat`. In each file you will find the images `trainingImage` and `trainingSegmentation`, which you should use to train your models, and another image `testImage` that you should automatically segment with the trained models. The pixels belonging to the foreground vs. background in the segmentations have values 1 and 0, respectively.

5.4.2 Tasks

For each of the tasks below, you should implement models with $M = 6$ basis functions $\phi_m(\mathbf{x})$, and optimize the model parameters $\boldsymbol{\theta}$ on the training data using stochastic gradient descent (Eq. (5.8)). Reasonable settings for the optimization are $N' = 200$ and $\nu' = 0.005$, and you should run the algorithm for a sufficiently large number of iterations, e.g., 5000. It is good practice to plot the evolution of the cross-entropy (Eq. (5.7)) across iterations, as this will allow you to fine-tune N' and ν' if needed. Since you will draw N' new samples in each iteration, the evolution of the cross-entropy will be noisy – however the overall trend should be that it decreases as the iterations progress (and stabilizes before the training is ended). It is also recommended to visualize the output of the classifier on the training and/or test image during training, as in Fig. 5.2(b).

1. For the data file `dataForNN_inside_clip0.mat`, implement a pixel-wise logistic regression classifier $p(l|d, \boldsymbol{\theta})$, using fixed basis functions of the form

$$\phi_m(d) = \cos\left[\pi(m-1)d\right].$$

   You can use $\beta_m = 0, \forall m$ for initializing the stochastic gradient descent algorithm.

Include in your report a plot of the evolution of the cross-entropy across iterations, the final segmentation of the test image as visualized in Fig. 5.2(c), and the quantity a and the obtained classifier as in Fig. 5.2(d) and Fig. 5.2(e), respectively. For the latter, please include a visualization of where the training samples $\{\mathbf{x}_i, y_i\}$ are located (e.g., showing the N' samples of the last iteration of the stochastic gradient descent algorithm).

2. Create a new test image by replacing the intensities in the original test image using the rule
$$d_{new} = d^{1.3}.$$
Use the classifier trained above to segment this new test image, and reflect on the result you obtain. Do the two test images look similar? What about their segmentations?

3. Now replace the fixed cosine basis functions in your model with adaptive ones of the type Eq. (5.9), and train the classifier again. The extra parameters $\{w_{m,j}\}$ can be initialized randomly, distributed according to a zero-mean Gaussian distribution with unit variance. For the trained model, visualize again the obtained classifier and the resulting segmentation of the original test image. This time, also include a plot of the estimated basis functions $\phi_m(d)$.

4. Using the data in the file `dataForNN_foreBorder_clip1.mat`, re-train and re-apply your classifier, but this time not just based on the intensity in the pixel being classified, but also on that of the pixel in the next row (i.e., the input $\mathbf{x}$ is now a vector of dimension $p = 2$). In addition to showing the classifier output on the test image, also include in your report a visualization of the learned basis functions $\phi_m(\mathbf{x})$ in the 2D input space, along with the learned classifier (again in the 2D input space). For the latter, please include a visualization of where the training samples $\{\mathbf{x}_i, y_i\}$ are located (cf. the first task).

5. Using the data in the file `dataForNN_foreBorder_clip0.mat`, re-train and re-apply the same classifier, but now using 3×3 patches as input, i.e., the input $\mathbf{x}$ is now a vector of dimension $p = 9$. In your report you should include the classifier output on the test image, as well as the feature maps and the corresponding model weights $\{w_{m,j}\}_{j=1}^{p}$ as visualized in Fig. 5.5(d) and Fig. 5.5(e), respectively.

6. *For the enthusiast student:* In order to provide a more challenging task to the neural network, do task 4 again but this time using the data in the file `dataForNN_foreBorder_clip0.mat`. Concentrate on the visualization of the learned basis functions and especially that of the classifier with respect to the training samples. Can you explain why this scenario is more challenging than the one of task 4? You can also try to add another hidden layer in the network – what do the basis functions look like now?

5.4.3 Hints

The Matlab functions `getInputFeatures.m`, `getSamples.m` and `showLocationOfSamples.m` are provided. Using the variable `numberOfNeighbors` to denote the number of neighboring pixels to use in the classifier input (1 for task 4 and 8 for task 5), the following code snippet extracts `numberOfSamples` training points (cf. $\mathbf{X}$ and $\mathbf{y}$ in Section 5.1), and then visualizes their location on top of the image as in Fig. 5.2(a):

```
featureImage = getInputFeatures( image, numberOfNeighbors );
[ X, y, rowAndColNumbers ] = getSamples( featureImage, ...
                                         segmentation, numberOfSamples );
showLocationOfSamples( image, rowAndColNumbers, y )
```

The following code snippet overlays a segmentation on top of an image as in Fig. 5.2(c) and Fig. 5.3(b):

```
figure
tmp = zeros( size( segmentation, 1 ), size( segmentation, 2 ), 3 );
tmp( :, :, 1 ) = 1-segmentation;
tmp( :, :, 3 ) = segmentation;
imshow( image * 0.5 + tmp * 0.5, 'init', 'fit' )
```

For the sake of simplicity, you should compute the gradient $\nabla E_{N'}(\boldsymbol{\theta})$ using the method of finite differences. If the statement `cost = getCost( theta )` computes the cost for parameters `theta`, the following Matlab snippet computes the gradient:

```
cost = getCost( theta );
delta = 0.001;
gradient = zeros( size( theta ) );
for i = 1 : length( theta )
  newTheta = theta;
  newTheta( i ) = newTheta( i ) + delta
  newCost = getCost( newTheta )
  gradient( i ) = ( newCost - cost ) / delta;
end
```

Chapter 6

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.

6.1 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.

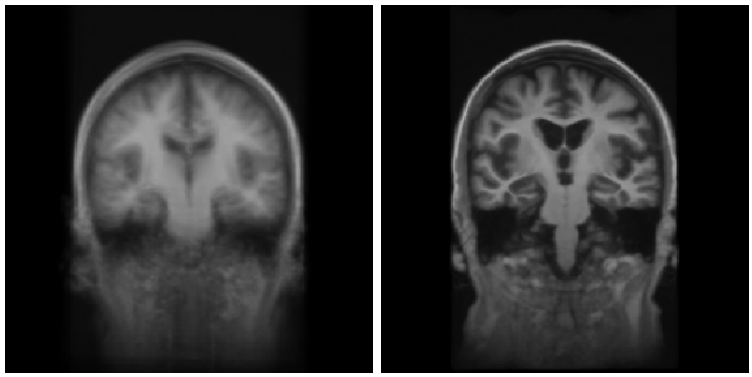
Mathematically, once a template image $\mathcal{T}$ has been selected, it can be used as a reference coordinate system by mapping any image under study, denoted by $\mathcal{I}$, into this standard coordinate system. Using the notation from chapter 2, a geometrical transformation $\mathbf{y}(\mathbf{x};\mathbf{w}) : \mathbb{R}^2 \rightarrow \mathbb{R}^2$ or $\mathbb{R}^3 \rightarrow \mathbb{R}^3$ is computed that maps the coordinates $\mathbf{x}_i$ of the template's voxel locations to coordinates $\mathbf{y}(\mathbf{x}_i;\mathbf{w})$ in the image. Resampled intensity values $\mathcal{I}(\mathbf{y}(\mathbf{x}_i;\mathbf{w}))$ can then be extracted, effectively deforming the image into the reference coordinate system.

6.1.1 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 i in the scan of individual 1, with coordinates $\mathbf{x}_i$ and intensity $\mathcal{I}_1(\mathbf{x}_i)$, we can then obtain the corresponding intensity $\mathcal{I}_q(\mathbf{y}(\mathbf{x}_i;\mathbf{w}_q))$ in each subject $q = 2\ldots Q$. The template $\mathcal{T}$ is then computed by assigning as intensity of voxel i the average intensity over all Q available images:

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

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



(a) (b)

Figure 6.1: Reference template obtained by intensity-averaging the brain MR scans of 20 different individuals after affine registration (a) and deformable registration (b).

individuals better than simpler models, yielding "sharper" templates. Example reference templates illustrating this effect are shown in figure 6.1.

6.1.2 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. 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

$$\begin{aligned} & p(\mathcal{T}, \mathbf{w}_1, \ldots, \mathbf{w}_Q | \mathcal{I}_1, \ldots \mathcal{I}_Q) \\ & \quad \propto \quad p(\mathcal{I}_1, \ldots \mathcal{I}_Q | \mathcal{T}, \mathbf{w}_1, \ldots, \mathbf{w}_Q) p(\mathcal{T}, \mathbf{w}_1, \ldots, \mathbf{w}_Q) \\ & \quad = \quad p(\mathcal{T}) \prod_{q=1}^{Q} p(\mathcal{I}_q | \mathcal{T}, \mathbf{w}_q) p(\mathbf{w}_q) \\ & \quad \propto \quad \prod_{q=1}^{Q} \left[\prod_{i=1}^{N} \exp\left(- \frac{(\mathcal{I}_q(\mathbf{y}(\mathbf{x}_i; \mathbf{w}_q)) - \mathcal{T}(\mathbf{x}_i))^2}{2\sigma^2} \right) \right] \exp(-\mathcal{S}(\mathbf{w}_q)) . \quad (6.1) \end{aligned}$$

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:

$$\begin{aligned} & \{\hat{\mathcal{T}}, \hat{\mathbf{w}}_1, \ldots, \hat{\mathbf{w}}_Q\} = \\ & \qquad \arg \min_{\{\mathcal{T}, \mathbf{w}_1, \ldots, \mathbf{w}_Q\}} \sum_{q=1}^{Q} \left[\frac{1}{2\sigma^2} \sum_{i=1}^{N} (\mathcal{I}_q(\mathbf{y}(\mathbf{x}_i; \mathbf{w}_q)) - \mathcal{T}(\mathbf{x}_i))^2 + \mathcal{S}(\mathbf{w}_q) \right] . \quad (6.2) \end{aligned}$$

Starting from some initial deformation parameters $\tilde{\mathbf{w}}_q$, typically chosen to correspond to no deformation at all, the optimization of eq. 6.2 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 eq. 6.2 for a given set of deformation parameters is given by

$$\tilde{\mathcal{T}}(\mathbf{x}_i) \quad \leftarrow \quad \frac{\sum_{q=1}^{Q} \mathcal{I}_q(\mathbf{y}(\mathbf{x_i}; \tilde{\mathbf{w}}_q))}{Q} \quad (6.3)$$

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

$$\tilde{\mathbf{w}}_q \quad \leftarrow \quad \arg\min_{\mathbf{w}_q} \left[\frac{1}{2\sigma^2} \sum_{i=1}^{N} \left(\mathcal{I}_q(\mathbf{y}(\mathbf{x}_i; \mathbf{w}_q)) - \tilde{\mathcal{T}}(\mathbf{x}_i) \right)^2 + \mathcal{S}(\mathbf{w}_q) \right] , \quad (6.4)$$

which can be solved using the optimization procedure discussed in section 2.6. In summary, the algorithm computes an initial, fuzzy average template (eq. 6.3) to which all Q scans are subsequently registered (eq. 6.4), 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.

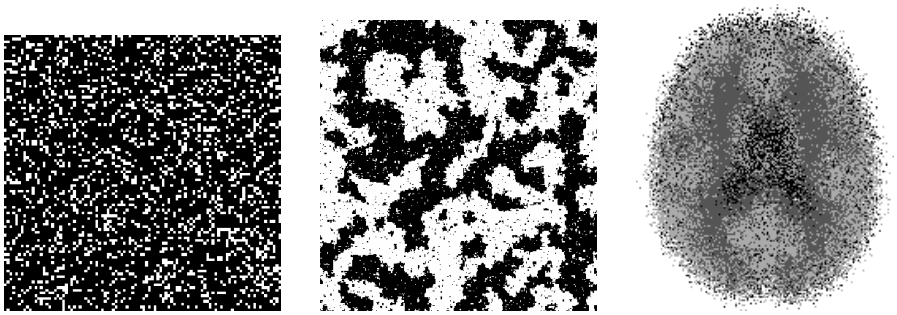
6.2 Atlases for segmentation

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 4, 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.

6.2.1 Probabilistic atlases

In chapter 4 we introduced two priors that are often used in voxel-based segmentation. The first, stated in eq. 4.5, simply assumes that label k occurs with probability π_k in any given voxel, without any further constraints on the spatial organization of the labels. The second, stated in eq. 4.16 and 4.18, 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 figure 6.2(a) and (b).



(a) (b) (c)

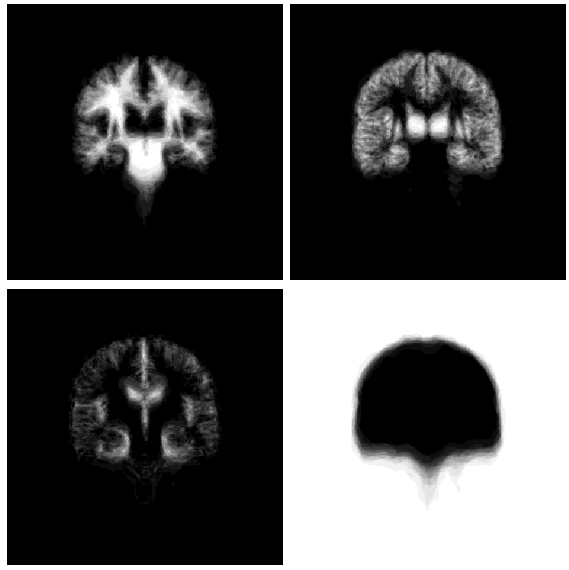
Figure 6.2: Samples from three different priors often used in voxel-based segmentation: the prior used in the standard Gaussian mixture model (a); a Markov random field prior (b); and a probabilistic atlas prior (c).

Here we consider a third class of priors for voxel-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 figure 6.2(c). 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 section 6.1. 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 i with location $\mathbf{x}_i$ in the template, one counts the frequency with which each label k occurred at that location across the Q warped label images:

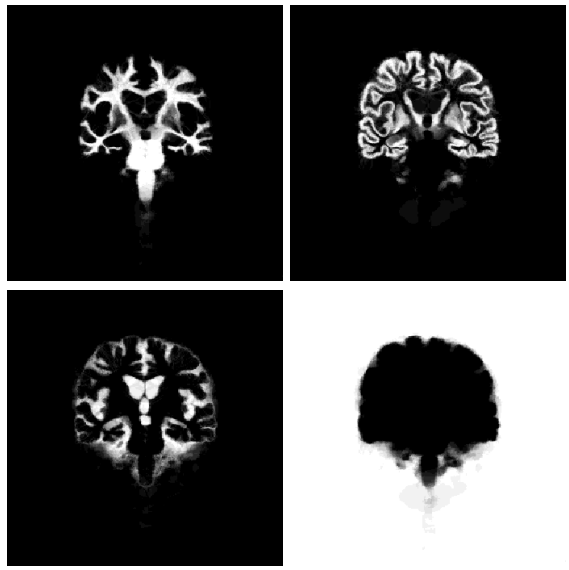
$$\pi_k^i = \frac{\sum_{q=1}^{Q} \delta\left(\mathcal{L}_q(\mathbf{y}(\mathbf{x}_i; \mathbf{w}_q)) = k\right)}{Q}, \tag{6.5}$$

where $\delta(k = l)$ equals one if $k = l$ and zero otherwise. The resulting π_k^i represents roughly the prior probability that label k occurs in voxel i in the template space: it varies spatially and always satisfies $\sum_{k=1}^{K} \pi_k^i = 1$ 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 figure 6.3.

Once the template $\mathcal{T}$ and the prior probabilities π_k^i have been computed, 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 reformatted prior probabilities π_k^i for every class k in every voxel i in $\mathcal{I}$. These prior probabilities π_k^i are then used in place of the generic π_k in every equation of the voxel-based segmentation models of chapter 4, yielding voxel classifications that no longer depends solely on the



(a)



(b)

Figure 6.3: 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.

voxels' local intensity alone, but also on their spatial location. Furthermore, the priors π_k^i 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 π_k^i contributes greatly to the overall robustness and practical value of the voxel-based segmentation models discussed in chapter 4.

6.2.2 Label propagation

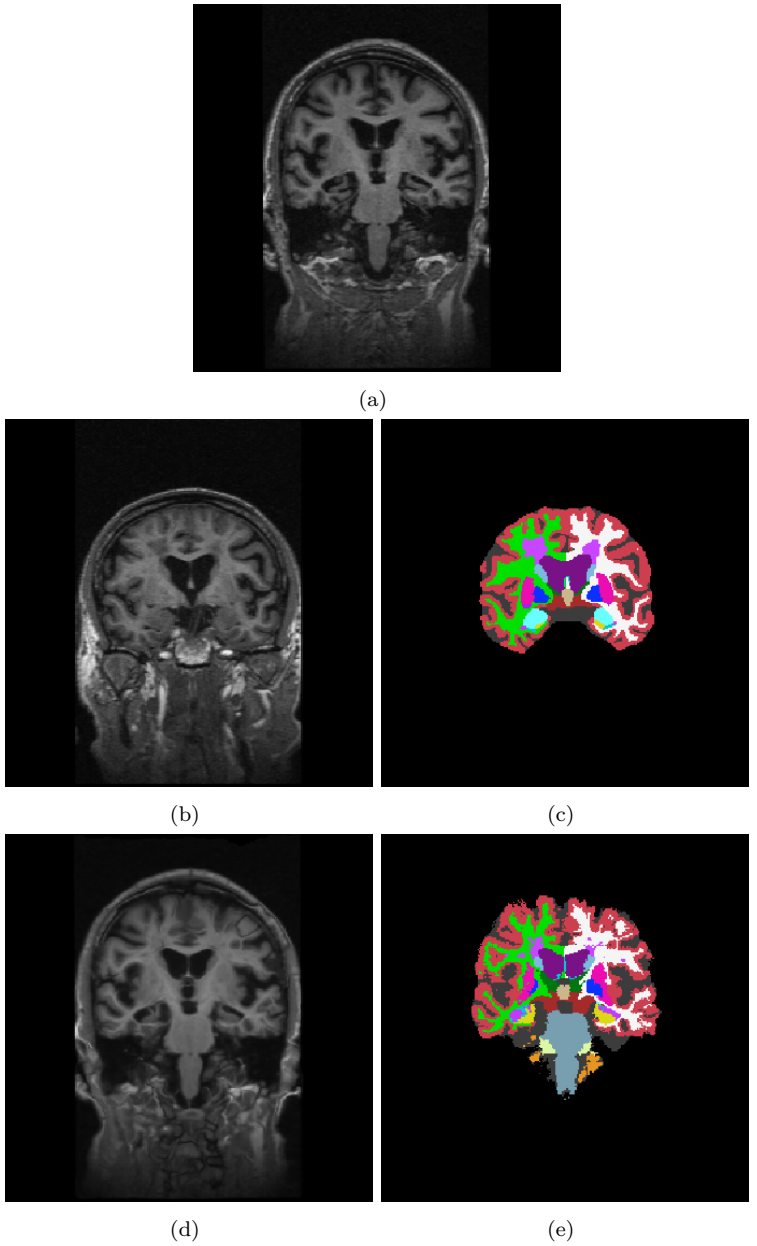
In so-called *label propagation*, an image $\mathcal{I}$ is segmented by computing a geometrical transformation $\mathbf{y}(\mathbf{x}_i; \mathbf{w}_q)$ that maps each voxel location $\mathbf{x}_i$ in $\mathcal{I}$ into a corresponding location in the qth pre-segmented image $\mathcal{I}_q$. Each voxel's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

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

This process is illustrated in figure 6.4.

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:

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



(a)

(b)

(c)

(d)

(e)

Figure 6.4: 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 7

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.

7.1 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.

7.1.1 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 $t_i \in \{0, 1\}$ denotes the label assigned to voxel with index i. We will refer to $\mathbf{t}$ as the *ground truth* segmentation. By convention, $t_i = 1$ if the voxel belongs to the structure of interest (so-called "foreground"), and $t_i = 0$ otherwise ("background"). Similarly, the automated algorithm to be evaluated generates a segmentation $\mathbf{s} = (s_1, \ldots, s_N)^{\mathrm{T}}, s_i \in \{0, 1\}$ 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.,

$$TP = \sum_{i=1}^{N} s_i t_i. \tag{7.1}$$

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

$$TN = \sum_{i=1}^{N} (1 - s_i)(1 - t_i). \tag{7.2}$$

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:

$$FP = \sum_{i=1}^{N} s_i (1 - t_i) \tag{7.3}$$

and

$$FN = \sum_{i=1}^{N} (1 - s_i) t_i. \tag{7.4}$$

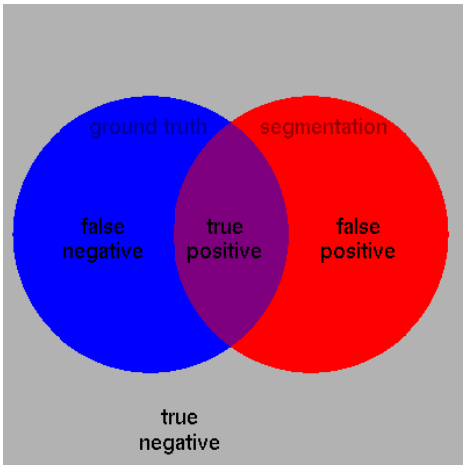


Figure 7.1: Illustration of the concepts true and false positives and negatives. The "ground truth" segmentation **t** is shown in blue, and the to-be-evaluated segmentation **s** in red.

The concepts of true positives, true negatives, false positives and false negatives are illustrated in figure 7.1.

The four quantities defined in eq. 7.1– 7.4 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.
- The **true negative rate** $\frac{\mathrm{TN}}{\mathrm{TN+FP}}$ 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.

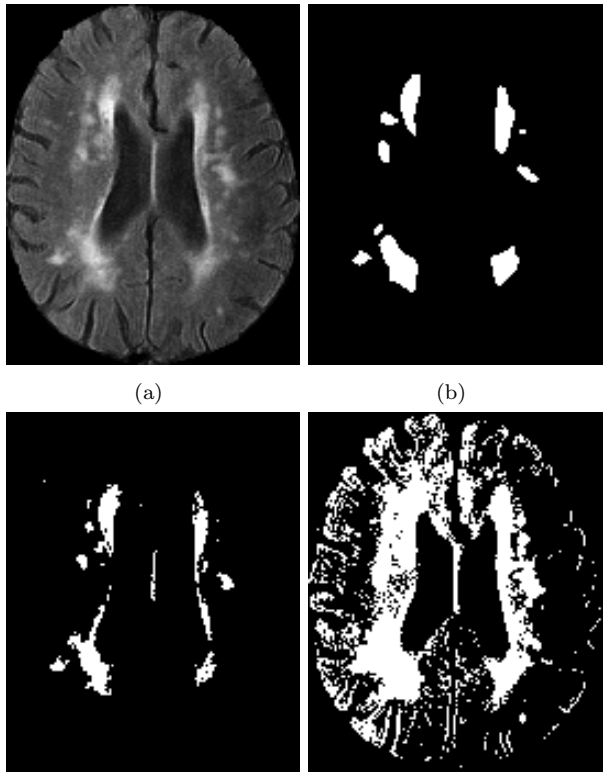
The *true positive rate* is also commonly referred to as the **sensitivity** of a segmentation, and the *true negative rate* as the **specificity**. For segmentations that correspond perfectly with the ground truth, they will both have the maximum value of 1.

7.1.2 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 figure 7.2, 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 figure 7.2, 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 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 figure 7.2(b)). 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 figure 7.2 is shown in figure 7.3. 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



(a) (b)

(c) (d)

Figure 7.2: 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).

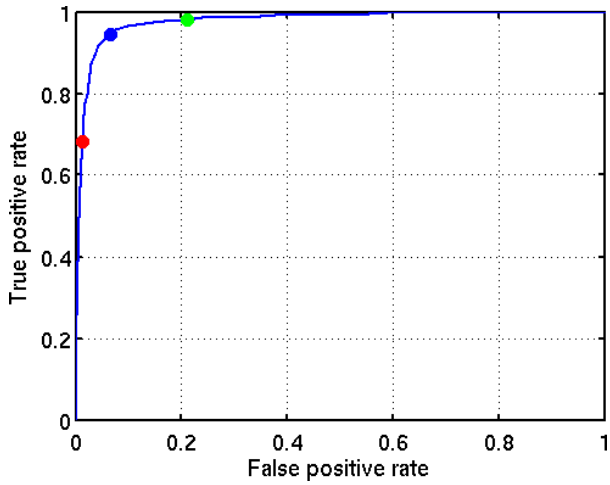


Figure 7.3: The ROC curve obtained by intensity thresholding the MR scan of figure 7.2(a) using a wide range of threshold values, and comparing the results to the manual segmentation shown in figure 7.2(b). The red dot corresponds to the segmentation shown in figure 7.2(c), the green dot to figure 7.2(d), and the blue dot to the threshold setting that minimizes the Euclidean distance to the perfect classifier (point (0,1)).

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.

7.1.3 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

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

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.

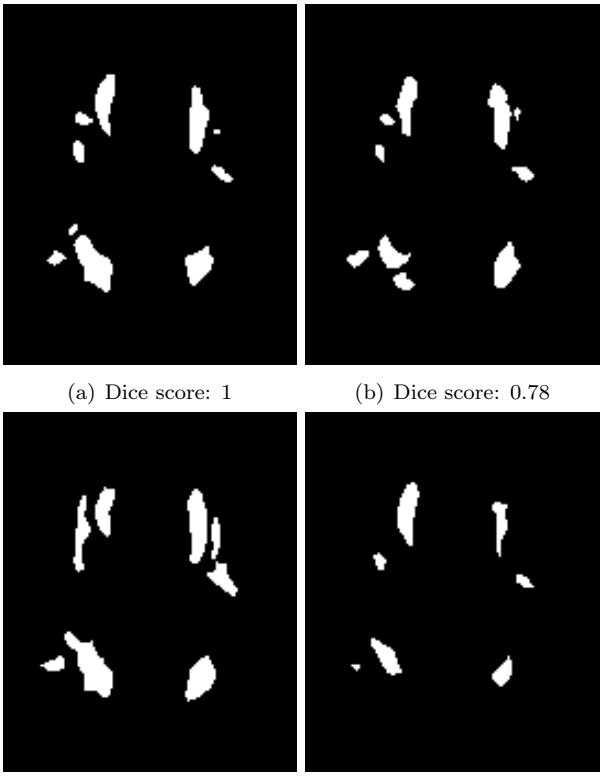
Several examples of alternate manual segmentations of the MR data of figure 7.2(a) are shown in figure 7.4, along with their Dice overlap scores with the first manual segmentation (figure 7.4(a)), which was considered the ground truth $\mathbf{t}$ for this purpose. Note that lower spatial correspondence (e.g., figure 7.4(d)) indeed results in a lower Dice overlap score.

7.2 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 figure 7.4, 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 [21]. Let $\mathbf{s}^m = (s_1^m, \ldots, s_N^m)^{\mathrm{T}}$ denote the mth available segmentation,



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

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

Figure 7.4: Four alternate manual segmentations of the MR scan of figure 7.2(a). The Dice scores are computed with respect to the first manual segmentation (a).

where $s_i^m \in \{0, 1\}$ indicates whether or not voxel i is assigned 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

$$p(\mathbf{s}^m|\mathbf{t}, p^m, q^m) = \prod_{i=1}^{N} p(s_i^m|t_i, p^m, q^m) \tag{7.5}$$

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

$$p(s|t, p, q) = \begin{cases} p^s(1-p)^{(1-s)} & \text{if } t = 1 \\ q^{(1-s)}(1-q)^s & \text{if } t = 0 \end{cases} . \tag{7.6}$$

Eq. 7.6 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

$$\begin{aligned} p(\mathbf{S}|\mathbf{t}, \boldsymbol{\theta}) &= \prod_{m=1}^{M} p(\mathbf{s}^m|\mathbf{t}, p^m, q^m) \\ &= \prod_{m=1}^{M} \prod_{i=1}^{N} p(s_i^m|t_i, p^m, q^m) \\ &= \prod_{i=1}^{N} p(\mathbf{s}_i|t_i, \boldsymbol{\theta}), \end{aligned} \tag{7.7}$$

where we have defined

$$p(\mathbf{s}_i|t_i, \boldsymbol{\theta}) = \prod_{m=1}^{M} p(s_i^m|t_i, p^m, q^m) \quad \text{with} \quad \mathbf{s}_i = (s_i^1, \ldots s_i^M)^{\mathrm{T}} \tag{7.8}$$

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. eq. 4.5), we will use a simple prior of the form $p(\mathbf{t}) = \prod_{i=1}^{N} \pi_{t_i}$, 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 voxel-based segmentation models discussed in chapter 4, 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

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

we have that

$$\begin{aligned} \log p(\mathbf{S}|\boldsymbol{\theta}) &= \sum_{i=1}^{N} \log \Big[p(\mathbf{s}_i|t_i=0,\boldsymbol{\theta})\pi_0 + p(\mathbf{s}_i|t_i=1,\boldsymbol{\theta})\pi_1 \Big] \\ &= \sum_{i=1}^{N} \log \left[\left(\frac{p(\mathbf{s}_i|t_i=0,\boldsymbol{\theta})\pi_0}{w_0^i} \right) w_0^i + \left(\frac{p(\mathbf{s}_i|t_i=1,\boldsymbol{\theta})\pi_1}{w_1^i} \right) w_1^i \right] \\ &\geq \underbrace{\sum_{i=1}^{N} \left[w_0^i \log \left(\frac{p(\mathbf{s}_i|t_i=0,\boldsymbol{\theta})\pi_0}{w_0^i} \right) + w_1^i \log \left(\frac{p(\mathbf{s}_i|t_i=1,\boldsymbol{\theta})\pi_1}{w_1^i} \right) \right]}_{Q(\boldsymbol{\theta}|\tilde{\boldsymbol{\theta}})}, \end{aligned}$$

for any pair of weights $\{w_0^i, w_1^i\}$ in each voxel that satisfy $w_0^i + w_1^i = 1$ and $w_0^i, w_1^i \geq 0$ (the last step uses eq. 4.28). 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

$$w_0^i \propto p(\mathbf{s}_i|t_i=0,\tilde{\boldsymbol{\theta}})\pi_0 \qquad \text{and} \qquad w_1^i \propto p(\mathbf{s}_i|t_i=1,\tilde{\boldsymbol{\theta}})\pi_1, \tag{7.9}$$

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

$$\begin{aligned} Q(\boldsymbol{\theta}|\tilde{\boldsymbol{\theta}}) \quad = \quad & \sum_{m=1}^{M} \left[\left(\sum_{i=1}^{N} w_0^i (1 - s_i^m) \right) \log q^m + \left(\sum_{i=1}^{N} w_0^i s_i^m \right) \log(1 - q^m) \right] \\ & + \sum_{m=1}^{M} \left[\left(\sum_{i=1}^{N} w_1^i s_i^m \right) \log p^m + \left(\sum_{i=1}^{N} w_1^i (1 - s_i^m) \right) \log(1 - p^m) \right] \\ & + \sum_{i=1}^{N} \left[w_0^i \log \left(\frac{\pi_0}{w_0^i} \right) + w_1^i \log \left(\frac{\pi_1}{w_1^i} \right) \right] \end{aligned} \tag{7.10}$$

and requiring that

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

yields

$$\tilde{p}^m \quad \leftarrow \quad \frac{\sum_{i=1}^{N} w_1^i s_i^m}{\sum_{i=1}^{N} w_1^i} \tag{7.11}$$

$$\tilde{q}^m \quad \leftarrow \quad \frac{\sum_{i=1}^{N} w_0^i (1 - s_i^m)}{\sum_{i=1}^{N} w_0^i} \tag{7.12}$$

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 (eq. 7.9), and then updates each rater's sensitivity and specificity accordingly (eq. 7.11 and eq. 7.12). Interesting, if some rater is estimated to have low sensitivity and specificity, that rater's segmentation is automatically downweighted 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 maxi-

Figure 7.5: Estimated ground truth $\hat{\mathbf{t}}$ from the four manual segmentations of figure 7.4. The parameter π_1 was set to the average fraction of foreground voxels across all four manual segmentations, and $\pi_0 = (1 - \pi_1)$.

mum 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 i to the foreground if $w_i^1 \geq 0.5$ and to the background otherwise. This estimated ground truth can then be used as an unbiased reference segmentation to compare automated segmentation results to.

Figure 7.5 shows the underlying ground truth $\hat{\mathbf{t}}$ estimated from the four manual segmentations shown in figures 7.4(a)-(d). 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 figure 7.4(a), 0.782 and 0.998 for figure 7.4(b) , 0.935 and 0.988 for figure 7.4(c) , and 0.512 and 0.999 for figure 7.4(d). 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.

Bibliography

[1] A. E. Hoerl and R. W. Kennard, "Ridge regression. Biased estimation for nonorthogonal problems," *Technometrics*, vol. 12, no. 1, pp. 55–67, 1970.

[2] D. W. Marquardt, "Generalized inverses, ridge regression, biased linear estimation, and nonlinear estimation," *Technometrics*, vol. 12, no. 3, 1970.

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

[4] G. Wahba, *Spline models for observational data*. SIAM, 1990.

[5] P. J. Green and B. W. Silverman, *Nonparametric regression and generalized linera models. A roughness penalty approach*. Chapman & Hall/CRC, 1994.

[6] E. Haber and J. Modersitzki, "A multilevel method for image registration," *SIAM Journal on Scientific Computing*, vol. 27, no. 5, pp. 1594–1607, 2006.

[7] I. L. Dryden and K. Mardia, *Statistical Shape Analysis*. Chichester: John Wiley & Sons, 1998. xx + 347 pp.

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

[9] P. J. Besl and N. D. McKay, "A method for registration of 3D shapes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 2, pp. 239–256, 1992.

[10] S. Belongie, J. Malik, and J. Puzicha, "Shape matching and object recognition using shape contexts," *IEEE Transactions on Patter Analysis and Machine Intelligence*, vol. 24, no. 24, pp. 509–522, 2002.

[11] J. Sethian, *Level Set Methods and Fast Marching Methods – Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Science*. Cambridge Monographs on Applied and Computational Mathematics (No. 3), Cambridge University Press, 2 ed., 1999.

[12] S. M. Pizer, P. T. Fletcher, S. Joshi, A. Thall, J. Z. Chen, Y. Fridman, D. S. Fritsch, A. G. Gash, J. M. Glotzer, and M. R. Jiroutek et al., "Deformable M-reps for 3D medical image segmentation," *International Journal of Computer Vision*, vol. 55, no. 2-3, pp. 85–106, 2003.

[13] T. F. Cootes, C. J. Taylor, D. H. Cooper, and J. Graham, "Active shape models – their training and application," *Computer Vision and Image Understanding*, vol. 61, no. 1, pp. 38–59, 1995.

[14] W. Lorensen and H. Cline, "Marching cubes: A high resolution 3D surface construction algorithm," *Computer Graphics*, vol. 21, no. 4, pp. 163–169, 1987.

[15] 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.

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

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

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

[19] 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.

[20] 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.

[21] 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.