Combining deformable templates and neural networks for HCR

Franck Mertzweiller

Master by Research in Pattern Analysis and Neural Networks

ASTON UNIVERSITY

September 1999

ASTON UNIVERSITY

Combining deformable templates and neural networks for HCR

FRANCK MERTZWEILLER

Master by Research in Pattern Analysis and Neural Networks, 1999

Thesis Summary

This thesis describes a method of recognising handwritten digits by using a recurrent neural network to incrementally map a deformed character back to its undeformed template. In the deformable templates approach to handwritten character recognition, a character is described by a set of control points and spline segments are drawn through these points. A forward model of the distribution of characters is then obtained by adding a noise process at the location of these control points. This noise process should model the way characters are actually written. The inversion of the forward model yields a principled approach to HCR.

However, this inversion is computationally expensive and even often intractable. For that reason the general neural network approach to HCR consists of directly mapping characters to classification. Nevertheless, this approach does not give any information about the character and such information would be relevant both to assess the reliability of the classification and to adapt quickly to the characteristics of a single writer.

The aim of this project is, by relaxing control points to their home position, to compute the deformation of the character from the trajectory of the network. This method would resolve the problems described in the above paragraph and in addition would make it possible to estimate the few parameters of the forward model from a small training set if the network provides full inversion of it.

Keywords: Handwritten character recognition, backward propagation, feed forward model, recurrent neural network.

Acknowledgements

I would like first to thank my supervisor Dr. Robert Urbanczik for his help, advice and useful comments during the nine months of this project. I would also like to thank him for answering all my questions with so much kindness during the meetings we had.

I am also grateful to all PhD, Master students as well as to all staff of the NCRG for their help.

And finally, I am very thankful to people I have met this year who have supported me in the realisation of this project. In particular, I would like to thank Nicholas Hughes, Frédéric Viot as well as all the Msc IT students.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Previous research

The goal of this part is not to review the voluminous work on handwritten character recognition that has spanned more than three decades (useful reviews can be found in [Impedovo 1994] and [Suen, Nadal, Legault, Mai, and Lam 1992]). However, it is helpful to summarise the trends. We can distinguish several main different approaches to handwritten character recognition. Many research have been done using statistical tools, comparing examples with models. But more recently methods involving neural network knowledges have been proposed.

1.1.1 Two main approaches

Two main approaches to HCR can be discerned. The first one makes use of a significant amount of prior knowledge. Though in this case only small training set is required, the classification is very slow. The second one which utilises less prior knowledge, requires large training set but performs quick classification. Good performance is obtained using methods of both trends. However, we are more interested in this second trend since the method presented in this paper is a combination of both deformable templates and neural networks approaches which use only little prior knowledge.

1.1.2 Statistical methods

Conventional statistical approach

Many different statistical methods have been experienced. The conventional statistical approach to perform classification is to use a discriminant classifier that constructs boundaries which discriminate between objects of different categories ([Lee 1991]). Although the resource requirements of all those methods differ widely, their classification performance is remarkable similar as long as no estimate of the reliability of the classification is required. Indeed, the misclassification rate is about 1% on the NIST test set for the most reliable methods. Nevertheless, in all cases many patterns must be rejected to achieve low error rates on the remaining ones. In addition, this approach does not permit to say anything about the particular way in which the digit is instantiated.

Optical character recognition

An alternative approach is to use optical character recognition which starts with a structural description of the objects to be recognised and matches it against the image ([Burr 1981]). The classical pattern recognition consists of extracting features before doing the classification. But many various methods have been experimented. Some of them aim at extracting global features whereas some others focus on local ones ([Lam and Suen 1988]). One other method is to use generative models. In the simplest version there is one model for each digit. Given an image of the unidentified digit, the idea is to search for the model that is the most likely to have generated this image. Each of the ten digits has his own elastic model. The image is then recognised by choosing the elastic model which best matches the image. During this matching process, the model is deformed in an attempt to ensure that every piece of ink in the image is close to some part of the model. The fidelity of the final match depends on the amount of deformation of the model, and the distance of remaining ink from the deformed model. In [Revow, Williams, and Hinton 1996], this deformation is computed using a procedure based on the Expectation Maximisation algorithm that maximises the likelihood of the model generating the data.

All these different methods have the attractive property that, in addition to providing a label, they can also, in some sense, explain the image. It may indeed be interesting for example to know which parts of the digit of the image represent the digit and which parts are caused by noise or some incorrectly segmented neighbouring digit. In addition, information about the structural description of the objects to be recognised such that information about position, size, orientation, shear or elongation of the digits should allow us to better estimate the reliability of the match.

However, so far these approaches require large computational resources. In order to speed up the match, methods involving neural networks tools have been proposed.

1.1.3 Using neural networks for HCR

First neural network approach to HCR

The first approach to handwritten character recognition using neural networks consists of directly mapping characters to classification. In this case, given an image of a digit, the network should be able to determine which digit it is. But this method does not provide any information about the pose of the digit either.
Researchers try a different approach to handwritten character recognition involving a feed forward network.

Using a feed forward network

The goal of this method is to train a feed forward network to provide initial values for an elastic match algorithm ([Hinton, Williams, and Revow 1992]). This hybrid method should avoid that the elastic model gets trapped in local optima and perform quicker classification. But unfortunately, this inversion is in general computationally too expensive and even intractable. Hence the first neural networks approach presented in the above paragraph has been rather more successful. What is more, besides requiring huge amounts of training data, this approach does not give much information

about the character.

It was obviously necessary to find a better approach to HCR in order to improve the performance of the classification. The goal of this new method would be first to be computationally cheaper and then to provide much information about the character so as to better estimate the reliability of the match.

1.2 Presentation of the method used

1.2.1 General presentation

As explained in the previous method using a feed forward network, assume we have a dynamical system with attractive fixed points which correspond to the undeformed objects. If the system has suitable attractive basins, it will restore the undeformed object when initialised with a deformed one and thus classify it. We also saw that the main problem in this approach is the large computational data which is required. However, a cheaper procedure can be used, if we can specify the entire trajectory when training the network ([Urbanczik 1991]) and not only initial and final conditions.

1.2.2 Using a recurrent neural network

By relaxing control points to their home position we shall use a forward model to train a recurrent neural network whose goal is to incrementally map a deformed character back to its undeformed template. Indeed, we want the network to undeform digits in several steps. The reason of this choice is that then, by analysing the trajectory of the network, it becomes possible to compute the deformation of the character and consequently to estimate the parameters of the forward model from a small training set which is valuable to assess the reliability of the match.

1.2.3 Combination of both general trends

A stochastic model of deformable templates is also used in this method, but the crucial difference with traditional statistical method is that these deformable templates are to be used to train the network, and not anymore to be directly compared to the real data.

Indeed, once this inversion is achieved by the neural network, the classification consists of merely comparing the final output to the ten different perfect templates, as in the simplest statistical approach.

1.2.4 Structural identification of the objects

The key idea of this method is that the trajectory of the network can be analysed and associated with a deformation of the underlying plane to provide structural identification of a digit. Such a deformation of the plane allows us to provide a vector field which translates the trajectory of the network. If the inversion of the forward model is full, the position of the control points of the deformed character can be located. The advantage of this method is that it would allow the model to adapt quickly to the characteristics of a single writer.

1.3 Thesis overview

In the next section a stochastic model of handwritten digit generation is presented. In this model, characters are described by a set of control points and cubic spline are drawn through these points. A forward model of the distribution of the characters is then obtained by adding a noise process at the location of these control points. The noise process as well as the construction of this forward model will be explained in detail.

These deformed templates built with this stochastic model are to be used to train

a recurrent neural network described in section 3.

Section 4 presents the real data and the various processing techniques achieved to make them compatible with our model.

Section 5 displays the results which have been obtained through the evolution of the different parameters such as the location of the control points of the perfect templates, the parameters of the network or the pre-processing work on data.

The last section discusses present conclusions of the work achieved so far which can be considered as a first part of the whole method to handwritten character recognition presented in this introduction. Indeed, the first aim of this project is to provide a classification with a reasonable rate of error (we need the misclassification rate to be at least less than 10% on real data). Once this is achieved, it would then be possible to compute a reliable vector field corresponding to the trajectory of the network, and by analysing this vector field, make the stochastic model closer to the real data in order to improve again the performance of the classifier.

Chapter 2

Deformable templates

This section describes which stochastic model has been used to generate the deformed templates and what structure has been adopted to store them.

2.1 The deformable template approach

In the deformable templates approach to handwritten character recognition a character is described by a set of control points and line or rather spline segments are drawn through these points. The deformable templates are generated by defining a noise process at the location of these control points. This noise should model the way characters are actually written.

2.2 The control points

2.2.1 How to find the location of the control points?

Obviously the initial location of the control points should describe the perfect templates. However the choice of the position is not an easy task since the idea we have of a perfect character is often quite different to the ones which are likely to be met in real world data. It is indeed important to stress that the main goal of this project is to test the performance of this method against real world data.

In addition, it is also necessary to consider the fact that the digits are not exactly the same all over the world. For example the digit 1 is composed of only one segment in the United States and United Kingdom whereas it is composed of two segments in the rest of Europe. Therefore we have to consider where the real data come from in order to find the most suitable shape for the perfect templates.

2.2.2 Original values

The original values which have been utilised in the beginning of the project are these Dr R. Urbanczik used in previous research about HCR using Neural Network. But as explained just above these values describe the way European people draw the digits, and the real data come from a CDROM provided by *NIST* National Institute. That is one of the reasons why modifications have been made to these values.
The coordinates of the control points which have been originally utilised are reported in the table A.1.

The number of control points depends on the complexity of the digit. For example only five points can perfectly describe the digit 1 but 10 points are required to define the digit 5.
We can also point out that the digits 4 and 5 are composed of two segments since we have to lift the pen to plot one of these digits. We coded the passage from one segment to the other as an artificial control point whose value is '*eos*'[1].

2.2.3 Importance of the location of the control points

Finally it has to be noticed that the definition of the perfect templates is fundamental since these perfect digits are the basis for the deformable templates we want to construct and which are to be used to train the network. Thus the stochastic model should generate as many as possible different shapes of characters which are likely to

[1] 'end of segment'

be met in real world data. Furthermore, it would be even better to respect in the generated distribution the frequency of each different shape. That means that if in the real data, a particular shape of character appears, better performance would be reach if this specific shape is generated in simulated data in the same proportions.

We will see the evolution of these control points in the section 5 as well as the influence the perfect templates have on the results.

2.3 Construction of the complete perfect templates

2.3.1 Structure of the images

The image of a digit can be considered as a 140 by 160 element matrix in which each element corresponds to a pixel of the image. The matrix is binary. The value is 0 if the pixel is off and 1 if the pixel is on. In this matrix, the value whose indices are (1,1) represent the lower-left corner, (1,160) the upper-left and (140,160) the upper-right corner.

2.3.2 Utilisation of cubic spline

Assume we got the coordinates of the control points, the entire digit is built in drawing cubic spline segments through these points as shown in figure 2.1 ([Press, Teukolsky, Vetterling, and Flannery 1992]).

Cubic Spline Interpolation

Given a tabulated function $y_i = y(x_i)$, $i = 1 \ldots N$, linear interpolation in the interval x_j and x_{j+1} gives the interpolation formula

$$y = Ay_j + By_{j+1} \tag{2.1}$$

where

$$A \equiv \frac{x_{j+1} - x}{x_{j+1} - x_j} \qquad B \equiv 1 - A = \frac{x - x_j}{x_{j+1} - x_j} \tag{2.2}$$

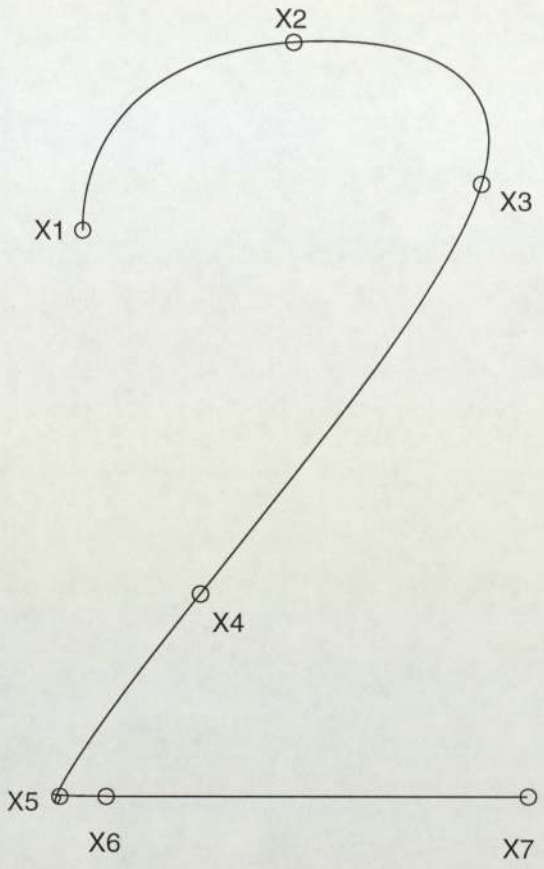

Figure 2.1: A digit is drawn by plotting the cubic spline which interpolates the control points X_i. This figure shows the example of the digit 2 described by 7 control points

The goal of cubic spline interpolations is to get an interpolation formula that is smooth in the first derivative, and continuous in the second derivative, both within an interval and at its boundaries.

Assume we also have tabulated values of the second derivatives y'', that is to say the set of values y_i'', we can then add, within each interval, a cubic polynomial to the right-hand side of equation 2.1. If we construct this cubic polynomial such as its second derivative varies linearly from a value y_j'' on the left to a value y_{j+1}'' on the right, we will have the desired continuous second derivative.

The only way to arrange this construction using 2.1 is

$$y = Ay_j + By_{j+1} + Cy_j'' + Dy_{j+1}'' \tag{2.3}$$

where A and B are defined in 2.2 and

$$C \equiv \frac{1}{6}(A^3 - A)(x_{j+1} - x_j)^2 \qquad D \equiv \frac{1}{6}(B^3 - B)(x_{j+1} - x_j)^2 \tag{2.4}$$

We can check now that y'' is in fact the second derivatives of the new interpolating polynomial. Taking the derivatives of equation 2.3 with respect to x and replacing dA/dx, dB/dx, dC/dx, dD/dx by their expressions found from the definitions of A, B, C, D, we obtain the following result

$$\frac{dy}{dx} = \frac{y_{j+1} - y_i}{x_{j+1} - x_j} - \frac{3A^2 - 1}{6}(x_{j+1} - x_j)y''_j + \frac{3B^2 - 1}{6}(x_{j+1} - x_j)y''_{j+1} \tag{2.5}$$

for the first derivative, and

$$\frac{d^2y}{dx^2} = Ay''_j + By''_{j+1} \tag{2.6}$$

for the second derivative.

The problem is now that we supposed the y''_i's to be known. However we still have to translate the fact that the first derivative, computed from equation 2.5, has to be continuous across the boundaries between two intervals. The key idea of a cubic spline is to require this continuity and use it to get equations for the second derivatives y''_i. These equations are obtained by setting equation 2.5 evaluated for $x = x_j$ in the interval (x_{j-1}, x_j) equal to the same equation evaluated for $x = x_j$ but in the interval (x_j, x_{j+1}). We finally get $N - 2$ linear equations (for $j = 2, \ldots, N - 1$)

$$\frac{x_j - x_{j-1}}{6}y''_{j-1} + \frac{x_{j+1} - x_{j-1}}{3}y''_j + \frac{x_{j+1} - x_j}{6}y''_{j+1} = \frac{y_{j+1} - y_j}{x_{j+1} - x_j} - \frac{y_j - y_{j-1}}{x_j - x_{j-1}} \tag{2.7}$$

These are $N - 2$ linear equations in the N unknowns y''_i, $i = 1, \ldots, N$. That is why we need to specify two more conditions for a unique solution. Nevertheless only the most common way of doing this has been used in the project which consists of setting both y''_1 and y''_N equal to zero, giving the so-called *natural cubic spline*.

Parameterisation of a cubic spline

As an image is represented by a matrix whose elements correspond to the pixels, we need to parameterise the expression of the splines. The goal is so to find the X and

Y-coordinates of each point of the spline segment as a function of a parameter t.

Given a segment with N points (x_i, y_i), for $i = 1, \ldots, N$, we aim at constructing two cubic splines $x(t)$ and $y(t)$ such that

$$x(t_i) = x_i \tag{2.8}$$

$$y(t_i) = y_i \tag{2.9}$$

with suitable choice of t_i, $i = 1, \ldots, N$.
The N values of the parameter t have been chosen recursively as following (for $i = 1, \ldots, N-1$)

$$t_1 = 0 \tag{2.10}$$

$$t_{i+1} = t_i + d((x_i, y_i), (x_{i+1}, y_{i+1})) \tag{2.11}$$

where $d((x_i, y_i), (x_j, y_j))$ is the Euclidean distance between both points whose coordinates are (x_i, y_i) and (x_j, y_j), that is to say

$$d((x_i, y_i), (x_j, y_j)) = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2} \tag{2.12}$$

If we have chosen the parameter t as described just above, it is in order to find uniformly spaced points along the curve even if two following control points are either close or distant to each other.

Once the parameters of these two splines are computed, it is easy to find any coordinates of a point of the curve by taking different values of the parameter t. It is sufficient to consider for t a range of values lying between 0 and t_n with a small enough step to obtain a continuous curve.

2.3.3 Problem of the line width

Now we found the curve of the perfect templates, we have to deal with the problem of line width. Besides, we shall see in another section that it may be useful to be able to modify this parameter in order to quickly adapt the deformable templates to the

real data. For that reason, in this project, the line width has to be a parameter which is easily modifiable.

If l is the value of the line width we finally want to obtain in the map of the template, a line segment whose length is l has to be drawn at each point of the curve. This segment has to be drawn perpendicularly to the curve at the considered point.
If we consider two close consecutive points of the curve (x_1, y_1) and (x_2, y_2), the perpendicular vector to the curve at the first point can be defined as $V = (V_1, V_2) = (y_2 - y_1, x_1 - x_2)$. The line width is finally drawn in plotting the line segment limited by both points whose coordinates are

$$(x_1 - \frac{l}{2}\frac{V_1}{\|V\|}, y_1 - \frac{l}{2}\frac{V_2}{\|V\|}) \quad and \quad (x_1 + \frac{l}{2}\frac{V_1}{\|V\|}, y_1 + \frac{l}{2}\frac{V_2}{\|V\|}) \tag{2.13}$$

2.3.4 Map of the perfect templates

We saw now how to construct the perfect templates. The result is stored in a 140 by 160 element matrix. Figure 2.2 shows the result of the plot of the perfect templates using the original values of the coordinates of the control points.

2.4 Noise process

We defined in the previous section the perfect templates. Those templates are described by control points. We shall explain in this section how to build deformed digits by adding a noise process at the location of these control points. Once this noise is added, deformed digits are obtained by merely drawing spline segments through these new control points as it has been done for perfect templates.

2.4.1 Choice of the noise process

The noise process should model the way characters are actually written. First of all it would be a too rough approximation to use identically independent distributed noise. This idea is easy to be understood. If a writer, starting drawing a character,

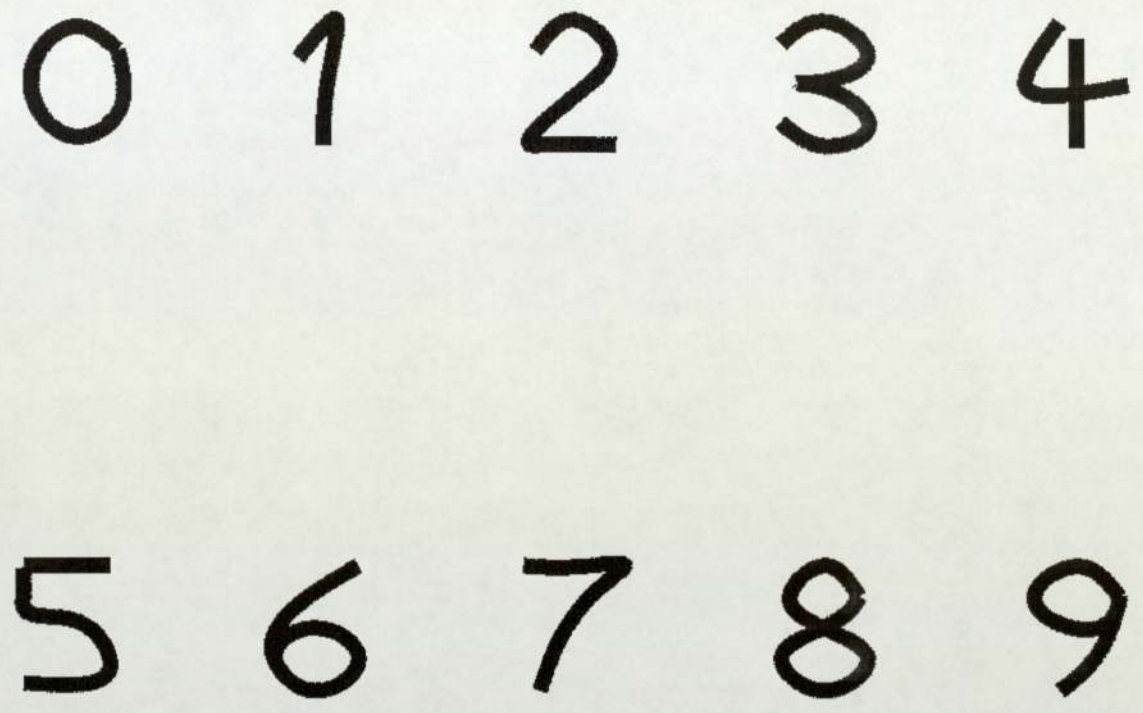

Figure 2.2: Perfect templates

deviates from the perfect curve, he will obviously try to correct this error in order to respect the general shape of the whole digit. That explains why we have to consider dependent noise at the different control points.

2.4.2 Definition of the noise process at a point X of the curve

If a template is described by N control points, and assume the noise at these control points is equal to V_i, for $i = 1, \ldots, N$ this previous idea leads to consider, at a point X of the curve, a noise which is defined by

$$f(X) = \frac{V_1 d(X, X_1)^{-1} + V_2 d(X, X_2)^{-1} + \cdots + V_N d(X, X_N)^{-1}}{d(X, X_1)^{-1} + d(X, X_2)^{-1} + \ldots + d(X, X_N)^{-1}} \tag{2.14}$$

where X_i, for $i = 1, \cdots, N$ are the N control points of the digit, and $d(X, Y)$ is the Euclidean distance between both points X and Y as defined in equation 2.12.

We can first notice that obviously if equation 2.14 is evaluated at a point X which tends to the location of the control point X_i, the limit of the result is V_i, that is to say equal to the noise at this control point. That can be written as

$$for\ i = 1, \dots, N \quad \lim f(X) = V_i \quad as\ X \to X_i \tag{2.15}$$

That result can be explained by the fact that all the terms of the sum in equation 2.14 are negligible with respect to the term $d(X, X_i)^{-1}$ which tends to infinity in both numerator and denominator. After simplification, only the term V_i is left.

In addition it is important to point out that the closer to one of the control points X is, the more influent the noise at this control point is. That is still due to the term $d(X, X_i)^{-1}$ of the sum which dominates over all the others as the point X is close to X_i.

2.4.3 Definition of the noise process at the control points

General expression

We aim at defining a noise process at the location of the control points. The method employed is to define the noise process recursively. In other words, although we saw in equation 2.14 that the noise at a point X on the curve depends on the noise defined at all the control points, we assume that the noise at control point X_i only depends on the noise which has been defined at the previous points $X_1, X_2, \cdots, X_{i-1}$. That can be understood by considering a writer who is drawing a digit. The deviation from the perfect curve at a point X only depends on what has already been drawn.

We note the actually drawn points by $\hat{X}'_i$. $\hat{X}'_i$ is so the former control point X_i to which we added the noise V_i, that is to say

$$\hat{X}'_i = X_i + V_i \tag{2.16}$$

The location $\hat{X}'_{i+1}$ for the next point, given that the previous points have been drawn

at $\hat{X}'_j$, $j = 1, \ldots, i-1$, is

$$\hat{X}'_{i+1} = X_{i+1} + V_{i+1} \tag{2.17}$$

The ideal location does not occur for $V_{i+1} = 0$, but the best choice is to choose V_{i+1} as a weighted average of the previous V_i (2.3).

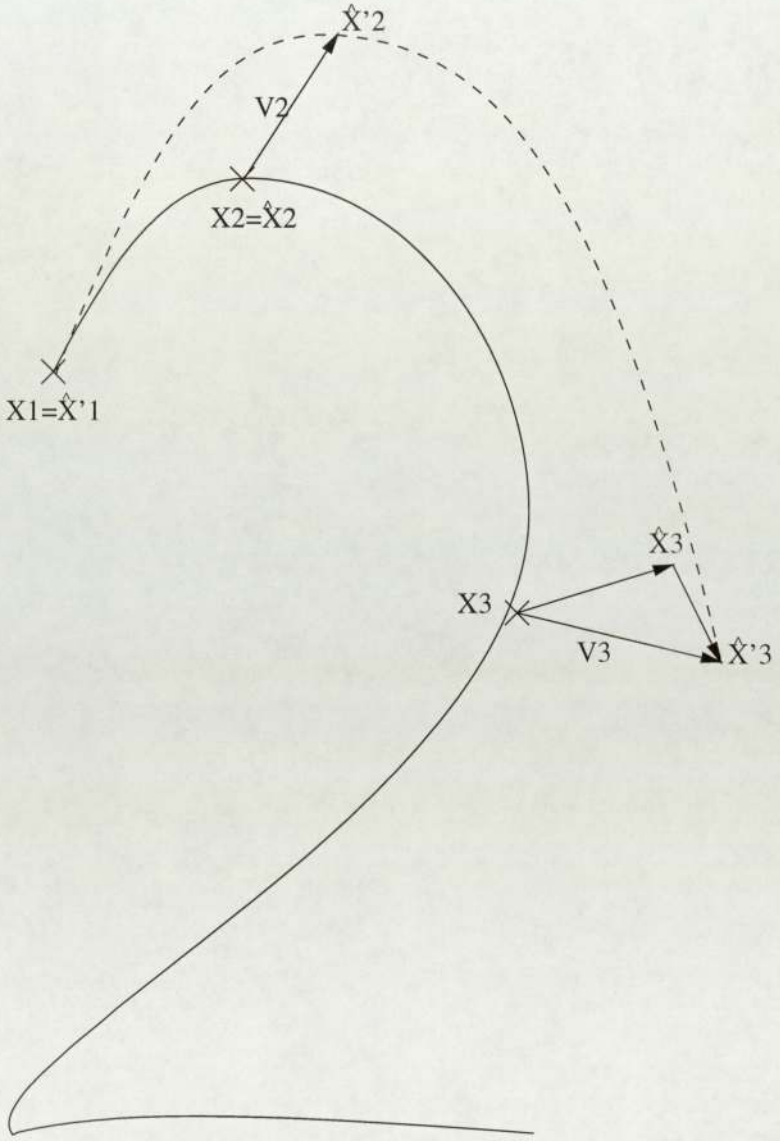

Figure 2.3: The noise is defined recursively at the control points. Assuming the noise V_2 at the control point X_2, we want the ideal location of the third point to be chosen in order to respect the general shape of the digit. For that reason, this point is not X_3, but $\hat{X}_3$ which is located by adding a weighted average of the noise at the previous control points to X_3. Gaussian noise is then added which translates the possible deviation that the writter may make. We finally find the actual drawn point $\hat{X}_3{}'$

Thus, if we consider a digit described by N control points, the expression of the

noise V_i, $i = 1, \ldots, N$ is defined by using the following expressions

$$\hat{X}_{i+1} = X_i + f(X_{i+1} \mid (X_1, V_1), (X_2, V_2), \ldots, (X_i, V_i)) \tag{2.18}$$

where f is the function described by expression 2.14 computed with i control points. $\hat{X}_{i+1}$ is a weighted average of the noises defined at the previous control points.

$$\hat{X}'_{i+1} = \hat{X}_{i+1} + \eta_j \, d(X_i, X_{i+1}) \tag{2.19}$$

where η_j is independent Gaussian random variable with zero mean and variance σ to be determined. $\hat{X}'_{i+1}$ is the actually drawn point. The noise V_i at the point X_i is finally

$$V_{i+1} = \hat{X}'_{i+1} - X_{i+1} \tag{2.20}$$

We can see in expression 2.18 that the term $\hat{X}_{i+1}$ of the noise only depends on the noise at the previous control points as explained just above.
In expression 2.19 we add to the noise another term which is the product of Gaussian noise and the Euclidean distance between X_i and X_{i+1}. If we chose the noise at the point X_{i+1} proportional to the distance separating this point from the previous one, it is because the longer the drawn distance is, the higher the risk of deviation from the perfect curve is.

Expression of the noise at the first three control points

In order to make this description of the noise clearer, we shall explain in this paragraph the expression of the noise at the first three control points.

- At the first point X_1, $V_1 = 0$

  That result means we chose that no noise is added to the first control point X_1. Indeed adding noise to X_1 would be equivalent to merely translating the whole character. Besides, we shall see how this problem can be resolved in aligning characters on their center of mass.

- At the second point X_2, $V_2 = \eta \, d(X_1, X_2) - X_2$

As $V_1 = 0$, $\hat{X}_2$ is equal to zero in 2.18. The first term in the expression of V_2 comes from 2.19 and the second one from 2.20. As we said in the above paragraph, Gaussian noise process which is added is proportional to the distance drawn from the previous control point X_1.

- At the third point X_3, we obtain

$$\hat{X}_3 = X_3 + \frac{d(X_3, X_2)^{-1} V_2}{d(X_3, X_1)^{-1} + d(X_3, X_2)^{-1}} \tag{2.21}$$

$$\hat{X}'_3 = \hat{X}_3 + \eta\, d(X_3, X_2) \tag{2.22}$$

$$V_3 = \frac{d(X_3, X_2)^{-1} V_2}{d(X_3, X_1)^{-1} + d(X_3, X_2)^{-1}} + \eta\, d(X_3, X_2) - X_3 \tag{2.23}$$

In expression 2.23 we see how the noise V_3 depends on the noises defined at the previous control points and, like for the noise V_2, on the distance between X_2 and X_3.

2.4.4 Value of the amplitude of the noise

The noise process which is added at the location of the control points is presented in equation 2.18, 2.19, and 2.20. The parameter to be defined is the standard deviation σ of Gaussian noise η_j which represents the amplitude of the noise. The initial value has been found in considering a sample of simulated data. This value has to be great enough to generate as many different shapes of characters as possible, but of course characters have to be recognisable, at least for human. The initial value we chose is 0.20. That means that if for example the distance drawn from the previous control point corresponds to 100 pixels, the typical deviation from the original position is about 20 pixels, in a 140 by 160 pixel frame.

2.5 Sample of simulated data

Figure 2.4 shows a sample of the distribution built with the stochastic model presented in this section. The coordinates of the control points used to plot these digits

are the original ones, and are given in table A.1.

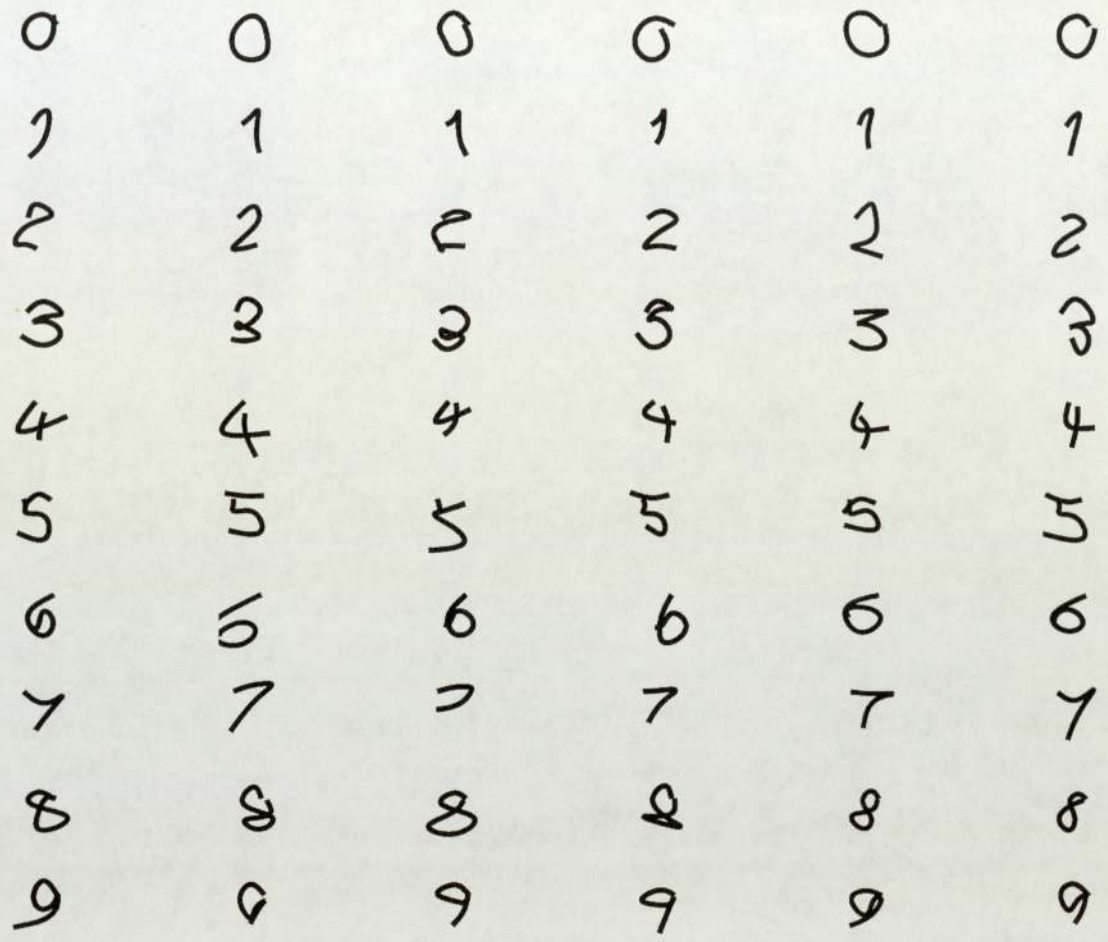

Figure 2.4: Sample of simulated data using original values of the coordinates of the control points

These deformed templates are built in order to train a recurrent network which is presented in the next section.

Chapter 3

The recurrent network

We shall first explain what the input and the required output to the network are. Then we shall discuss the different possible networks which could have been used to achieve the desired result and the reasons which led us to the final choice.

3.1 Goal of the neural network

As we saw in the introduction, the desired behaviour of the network is to incrementally undeform characters. In other words that means that, at each step, we want the output to the network be an image whose shape is slightly closer to the corresponding perfect template than at the previous step. The data which is used to train the neural network is built according to the stochastic model described in the previous part.

3.2 Input to the neural network

We saw that the image of a character is stored in a 140 by 160 element matrix, that is to say 22400 elements. Even if the matrix is binary[1], it would not be efficient to use this matrix as an input to the network because of huge computational resources which would be required. That is the reason why we decided to utilise a smaller and coarser gray-scale of this image as an input to the network.

[1] The value of each element of the matrix is 1 if the element corresponds to an inked pixel and 0 if it corresponds to an uninked one

3.2.1 Utilisation of a gray-scale

Dimensions of the gray-scale

The main problem in the construction of the gray-scale is to find the most suitable dimensions. Obviously this parameter can strongly affect the performance of the network. If the size is to small, not enough information will be given to the network. Indeed characters (and particularly smallest ones) would be then indecipherable, even for humans. The network would be so unable to learn correctly. On the other hand, if we would choose a too large gray-scale size, the problem would become computationally impossible to solve. One reason is that we have to consider that each element of the gray-scale corresponds to an input unit of the neural network.
Consequently the choice of the dimensions of the gray-scale has been made by choosing the smallest values which preserve the main features of characters. According to the complexity of the characters, we can estimate that a gray-scale size of 14 by 16 is a reasonable and suitable choice. However it is important to keep in mind that this parameter can still be modified and increasing the values of the dimensions would allow us to obtain better results.
We shall see now how this gray-scale has been constructed from the original frame.

Structure of the gray-scale

We aim at transforming the original frame containing the image of a character and whose dimensions are 140 by 160 into a gray-scale format whose dimensions are 14 by 16. This gray-scale image can so be considered as a 14 by 16 element matrix. Like for the original one, the element whose indices are (1,1) corresponds to the lower-left corner and (14,16) to the upper-right corner. Besides, we have to conserve as much information as possible in the gray-scale. Thus we did the two following choices:

- We will not use anymore binary type element in the gray-scale matrix, but each value lies between -1 and 1 according to the percentage of inked pixels in the input window[2].

[2] the value is close to 1 if most of pixels in the window are inked and close to -1 if most of them are uninked

- The gray-scale image of the digit is centered on its center of mass. Thanks to this process, we avoid the problem of position of the image in the frame. It may be especially useful to achieve this transformation on real data for which we do not have any information about the position of characters in the frame.

3.2.2 Construction of the gray-scale

How to center the image on the center of mass?

Assume the original image of a character is stored in the two dimensional matrix whose name is $frame(i,j)$, where $1 \leq i \leq 140$ and $1 \leq j \leq 160$. The center of mass (G_X, G_Y) of the figure is computed as follows

$$G_X = \frac{\sum_{i=1}^{140}\sum_{j=1}^{160} i\ frame(i,j)}{N} \qquad G_Y = \frac{\sum_{i=1}^{140}\sum_{j=1}^{160} j\ frame(i,j)}{N} \tag{3.1}$$

where $N = \sum_{i=1}^{140}\sum_{j=1}^{160} frame(i,j)$ is the weight of the whole image. Once we found the center of mass of the character, we just have to consider this point as the center of the middle window of the gray-scale. In our example, the center of mass will be aligned with the middle pixel of the window whose indices are $(7,8)$.

Values of the gray-scale

Each element of the gray-scale matrix represents a 10 by 10 pixel window in the original image. The first value which is calculated for each window is the number of inked pixels in the window. For example, if we denote M the number of inked pixels of the window whose coordinates of the central pixel are (X_M, Y_M), we obtain

$$M = \sum_{i=-5}^{i=4}\sum_{j=-5}^{j=4} frame(X_M + i, Y_M + j) \tag{3.2}$$

We can compute all the different values of M for each window in replacing the values of X_M and Y_M in equation 3.2 by

$$X_M = G_X + (k-7).10 \tag{3.3}$$

$$Y_M = G_Y + (l-8).10 \tag{3.4}$$

for $k = 1, \ldots, 14$ and $l = 1, \ldots, 16$, and where G_X and G_Y are defined in equation 3.1.

The number M of inked pixels in one window obviously lies between 0 and 100. However we want every values of the gray-scale to lie between -1 and 1. Thus the final value actually stored in the matrix is $\tanh(A.M + B)$ where A and B are two constant parameters to be determined. We use the tangent hyperbolic function in order to obtain values between -1 and 1. Both parameters A and B have to be adjusted to actually find a complete range of values lying into the interval $(-1, 1)$.

3.2.3 Example of transformation

Figure 3.1 shows an example of transformation of a deformed character into the gray-scale format.

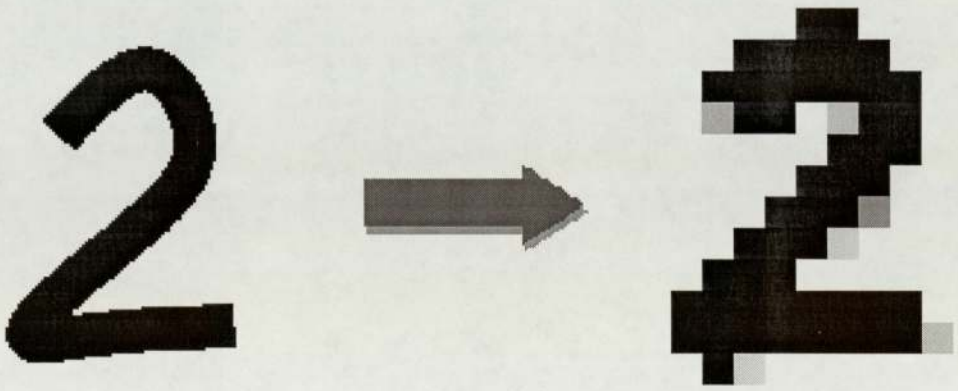

Figure 3.1: Example of the transformation of a character into the gray-scale format

The 14 by 16 element matrix built from the original 140 by 160 element matrix is the input to the neural network. An example of matrix corresponding to figure 3.1 is presented in table 3.1.

-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0	-1.0	-0.9	**1.0**	**1.0**	-0.9	-1.0	-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0	**0.7**	**1.0**	**1.0**	**1.0**	**1.0**	-1.0	-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	**0.5**	**1.0**	**1.0**	**0.9**	**1.0**	**1.0**	**0.9**	-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-0.7	**1.0**	**0.8**	-1.0	-0.8	**1.0**	**1.0**	-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	**1.0**	**1.0**	**1.0**	-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	**0.9**	**1.0**	**1.0**	-0.9	-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0	-1.0	**1.0**	**1.0**	**1.0**	-0.5	-1.0	-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0	**1.0**	**1.0**	**1.0**	-0.9	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-0.9	**1.0**	**1.0**	**1.0**	-0.9	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	-0.9	-1.0	-1.0	-1.0
-1.0	-1.0	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	-0.8	-1.0	-1.0	-1.0
-1.0	-1.0	-0.9	**0.8**	-0.9	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0

Table 3.1: Example of 14 by 16 gray-scale matrix. The bold font values correspond to the values greater than 0. It so represents the more inked windows of the gray-scale. Thus we can discern through these values the deformed digit 2 whose map is drawn in figure 3.1

3.2.4 The gray-scale matrix is the input to the network

The gray-scale matrix of the character is to be utilised as an input to neural network. Thus we have to consider a recurrent network with 224 input units[3]. As we want the output to the network to be a slightly less deformed image of the character, the number of output units is 224 as well.

Note that this dimension is high. This method would be computationally less expensive if it could be possible to reduce the dimension of the feature space by using PCA methods for example. Nevertheless, it is important to keep in mind that the goal of this method is not merely to provide a classification but also to construct a vector field corresponding to the trajectory of the network. For that reason, the geometrical structure of the image has to be preserved.

We shall now explain the different possible networks which could have been used, and which choice has finally been made.

[3] the dimensions of the gray-scale matrix are 14 by 16 which represents 224 elements

3.3 Different possible networks

According to previous work, several different recurrent networks could have been used to achieve what is expected. We shall see in this section different ways and the reasons which led us to the final choice.

3.3.1 Presentation of the problem

We want the network to progressively undeform the image of characters. Considering the stochastic model we described, deformed templates are generated by adding noise called μ at the control points of the perfect templates. We shall call $\hat{I}^{\mu,t}$ the perfect template to which we have added the attenuated noise $0.9^t.\mu$. A sample of simulated data is so constructed with different values of the noise μ. The corresponding output to the network at the step t is called $O^{\mu,t}$.

The idea is that, given an example of deformed character $\hat{I}^{\mu,t}$, we expect the output to the network to be $\hat{I}^{\mu,t+1}$ at the next step. We assume the full inversion of the deformed digit to the corresponding perfect template is supposed to be achieved in 20 steps. Indeed the character $\hat{I}^{\mu,20}$ is very close to the perfect template since $0.9^{20} = 0.12$.

Consequently that means that we shall train the neural network using a sample of distribution of characters and, at each step, the target $\sigma^{\mu,t}$ for the input $\hat{I}^{\mu,t}$ shall be $\hat{I}^{\mu,t+1}$.

Besides, in order to find a progressive transformation of the digit, we choose to define the dynamic of the network at the step t as following

$$\hat{O}^t = (1 - \tilde{\lambda})I^t + \tilde{\lambda}O^t \tag{3.5}$$

where $\tilde{\lambda}$ is a parameter lying between 0 and 1 to be determined and I^t is the input to the network at the step t. We can see that, at each step, a fraction of the previous input is added to the output in order to slow down the transformation. The error

measure is defined as

$$E = \frac{1}{2}(\hat{O}^t - \hat{I}^{\mu,t+1})^2 \tag{3.6}$$

3.3.2 Three different possibilities of learning

First possibility

The first method would be to train the network with the ideal input at each step. That means that at the step t, for a character $\hat{I}^\mu$, the input is the perfect template to which the noise $0.9^t\mu$ has been added (figure 3.2). In other words, the input at the step t is

$$I^t = \hat{I}^{\mu,t} \tag{3.7}$$

and the target is

$$\sigma^{\mu,t} = \hat{I}^{\mu,t+1} \tag{3.8}$$

In this case, the output at each step is not considered anymore in the next input.

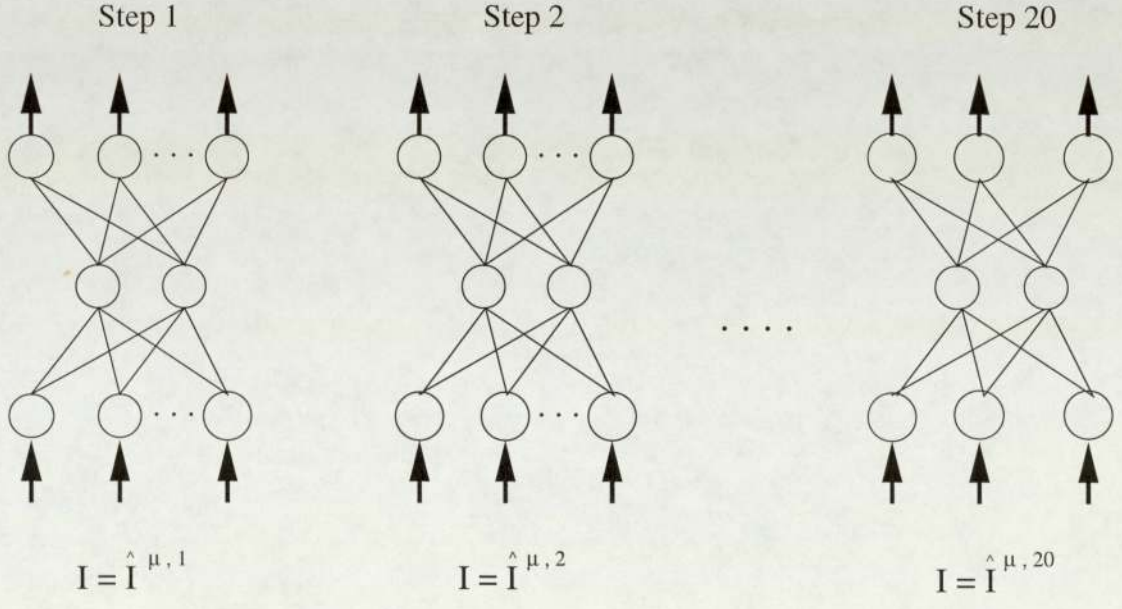

Figure 3.2: First possibility of recurrent network. We can note that in this case the successive inputs are independent and equal to the target at the previous step.

The main disadvantage of this method is that one does not have any control on the output. It is in fact difficult to understand the way the network learns. No information

is provided before testing the performance of the network against data. Indeed we do not exactly know the behaviour of the trained network if the input at each step is not the target at the previous step anymore as during training, but the previous output of the network. This fact means that if at one step, the output $O^{\mu,t}$ is not similar to what was expected (that is to say $\hat{I}^{\mu,t+1}$), the network could never manage to correct the former error, and the inversion would not be achieved.

Second possibility

An other idea is to train the network using as input the previous output (figure 3.3). At the step t, the input is

$$I^1 = \hat{I}^{\mu,1} \ (t = 1) \tag{3.9}$$

$$I^t = \hat{O}^{t-1} \ for \ t \geq 2 \tag{3.10}$$

where $\hat{O}$ is defined in 3.5. Of course the target at the step t is still defined by $\hat{I}^{\mu,t+1}$.

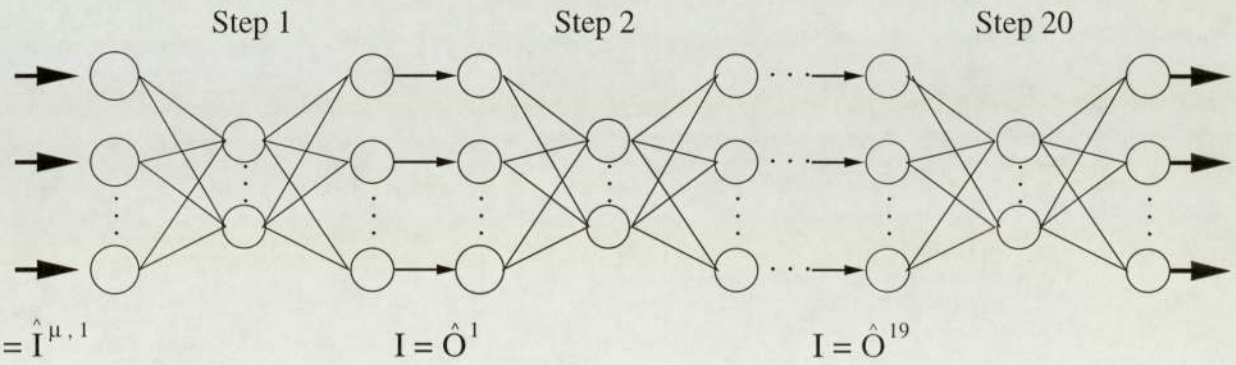

Figure 3.3: Second possibility of recurrent network. At each step, the input is equal to the previous output.

In order to understand the problem of this method, we have to consider an untrained network, at the beginning of the learning period. For an example of deformed character, at the second step ($t = 2$), the input to the network is $\hat{O}^1$. However, $\hat{O}^1$ is probably totally different to the target $\sigma^{\mu,3}$ since the network is not trained yet. That would mean that at the next step, we will make the network learn something incoherent. Actually, such a network is similar to a 20 hidden layers network, if we suppose to achieve the inversion of the forward model into 20 steps. We do not say that this

method does not work, but it would probably require huge amounts of training data to reach acceptable results.

Third possibility: a combination of the last two approaches

This third method, which is a combination of the last two approaches, consists of using an input composed of a fraction of the previous output and of a fraction of the previous target (figure 3.4). The expression of the input is

$$I^1 = \hat{I}^{\mu,1} \quad (t = 1) \tag{3.11}$$

$$I^t = (1-\lambda)\hat{O}^{t-1} + \lambda\hat{I}^{\mu,t} \quad for\ t \geq 2 \tag{3.12}$$

where λ is a constant parameter lying in the interval $(0, 1)$ which shall be determined empirically. We also shall see what the influence of this parameter is on the performance of the classifier in chapter 5.

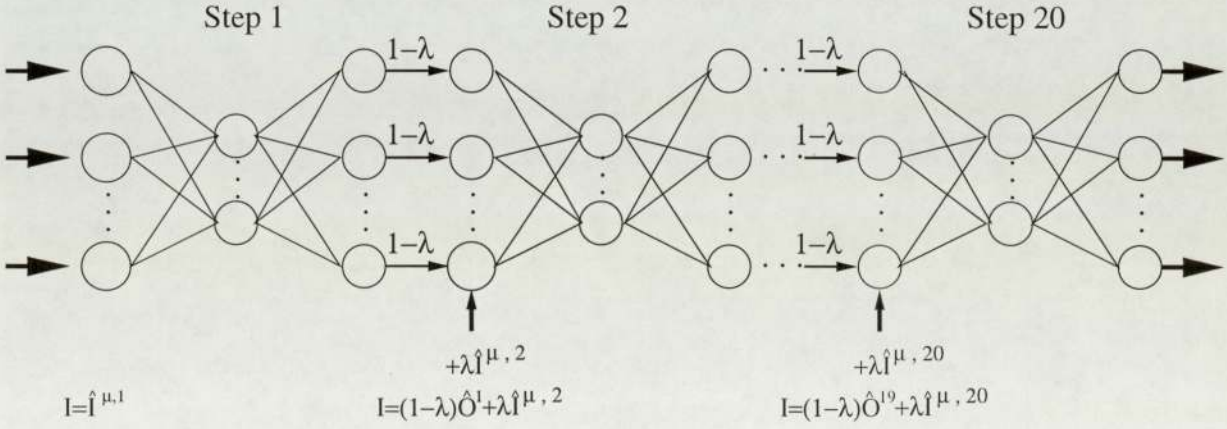

Figure 3.4: Third possibility of recurrent network. At each step, the input is composed of a fraction of the previous output to which a fraction of the previous target is added.

This third method has been used in this project since it should allow us to avoid the problems we exposed in the last two methods. Indeed, in including in the input a fraction of the output, the network is allowed to deviate from the expected trajectory but it also learns how to correct these errors. Assume that during the learning period, after the first pass through the neural network, the output is different from the expected target. What it is learnt by the network at the next step is not incoherent because

of the fraction of the target we added to the next input. Indeed, the main difference compared to the second possibility is that in this present case a fraction of the target is included in the new input in order not to lose the main features of the deformed character we want the network to learn.

3.4 The neural network

We have just explained in the above paragraph what the input and expected output to the neural network are. We shall see now in detail what kind of neural network has been used to achieve the inversion of the forward model.

3.4.1 Presentation of the network

We consider a network with n_{input} input units as well as n_{input} output units. This value corresponds to the number of element of the gray-scale matrix which contains the image of the character we want to classify and thus is equal to 224[4]. However the only way to find the most appropriate number of hidden units is to try different values and to test the performance of the network against real data. There is no algorithm to determine a priori what would be the best number of hidden units. The performance of the classifier as a function of the number of hidden units n_{hidden} is displayed in the next part.

We aim at training a neural network whose first layer weights are stored in the matrix A and whose dimensions are n_{hidden} by n_{input}. The second layer weights are stored in the matrix B whose dimensions are n_{input} by n_{hidden}. Figure 3.5 gives a representation of the network we used.

[4] The dimensions of the gray-scale matrix are 14 by 16

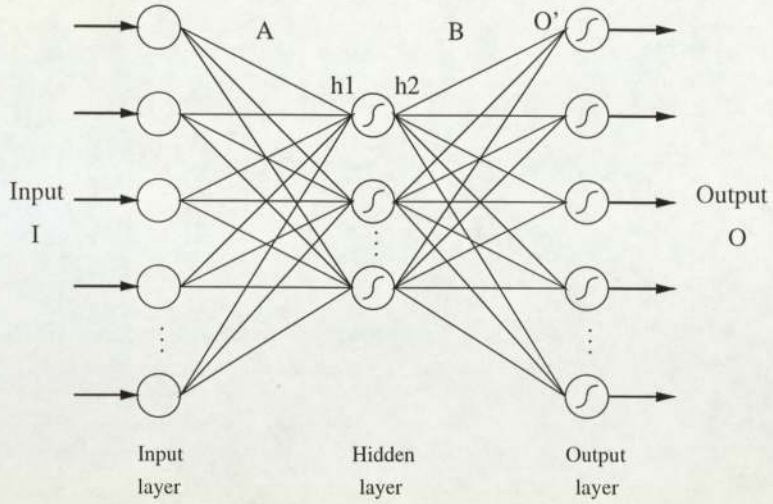

Figure 3.5: Representation of the neural network. We used 224 input and output units. A and B respectively correspond to the weights of the first and second layers.

3.4.2 Explanation of the algorithm used

We shall see now how to find the output O to the network given an input I, and the two matrices A and B corresponding to the weights of both first and second layers. The first vector to compute is the value h_1 of the input just after the first layer

$$h_1 = A.I \tag{3.13}$$

We use sigmoidal hidden units. Thus the result after the hidden layer is :

$$h_2 = tanh(h_1) = [tanh(h_1(1)), tanh(h_1(2)), \ldots, tanh(h_1(n_{hidden}))] \tag{3.14}$$

Consequently we obtain before the output layer the vector

$$O' = B.h_2 \tag{3.15}$$

and finally the output to the network is

$$O = tanh(O') = [tanh(O'_1), tanh(O'_2), \ldots, tanh(O'_{n_{hidden}})] \tag{3.16}$$

We know now what is the output to the network given an input. We shall now explain which method has been used for learning.

3.4.3 Learning period

The choice of the learning method

First of all, it is important to notice that on-line learning has been used to train the network. The main reason of this choice is the amount of data which is required. The memory to store a set of deformed characters is so important that on-line learning is the most adapted optimisation method. Furthermore we saw that we want the inversion of the forward model to be achieved in several steps. That would mean that to make the network learn one character, several different maps are utilised which would increase again the memory which would be required to store a set of data.
Thus deformed characters are built one after the other during learning period and the weights of both layers A and B of the network are updated after each example.

The optimisation function

To update the weights of the neural network, a gradient descent method computing by back-propagation has been used. We saw that after each step, and for each example, the error E is computed as follows

$$E = \frac{1}{2} \sum_{i=1}^{224} (\hat{O}_i - \sigma_i)^2 \tag{3.17}$$

where $\hat{O}_i$ and σ_i, for $i = 1, \ldots, N$ represent respectively the output $\hat{O}$ and the target σ to the network. Both are 224 element vectors. We want now to update both matrices A and B using

$$A = A - \Delta A \, \eta \tag{3.18}$$

$$B = B - \Delta B \, \eta \tag{3.19}$$

where η is the learning rate of the network and where ΔB and ΔA are computed using the formulas

$$\Delta B = T_B \, h_2^T \tag{3.20}$$

$$\Delta A = T_A \, I^T \tag{3.21}$$

where Z^T denotes the transpose matrix of Z. The matrix T_B is defined as

$$T_B = \begin{pmatrix} (\hat{O}_1 - \sigma_1)(1 - \hat{O}_1^2) \\ (\hat{O}_2 - \sigma_2)(1 - \hat{O}_2^2) \\ \vdots \\ (\hat{O}_{224} - \sigma_{224})(1 - \hat{O}_{224}^2) \end{pmatrix} \tag{3.22}$$

The vector T_A in equation 3.21 is computed using the value of the vector T_B defined in equation 3.22

$$T_A = B^T T_B D \tag{3.23}$$

where D is the diagonal matrix

$$D = \begin{pmatrix} (1 - h_2(1))^2 & 0 & \cdots & 0 \\ 0 & (1 - h_2(2))^2 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & (1 - h_2(n_{hidden}))^2 \end{pmatrix} \tag{3.24}$$

3.4.4 Visual example

Figure 3.6 shows an example of transformation of a character in 20 steps. Each figure corresponds to an output to the neural network at a different step. We can see that the inversion can be considered as total with this number of steps.

3.5 The classification

We saw how the network is used to transform deformed characters to their corresponding templates. However the output to the recurrent network is the gray-scale matrix of the corresponding perfect template if we assume that the inversion has been completely achieved. Thus the final classification is done by computing the quadratic error between the final output, that is to say after the 20 passes through the network, and the ten perfect templates. In other words, if O denotes the final output to the network and T the gray-scale of one of the ten perfects templates, the error is computed

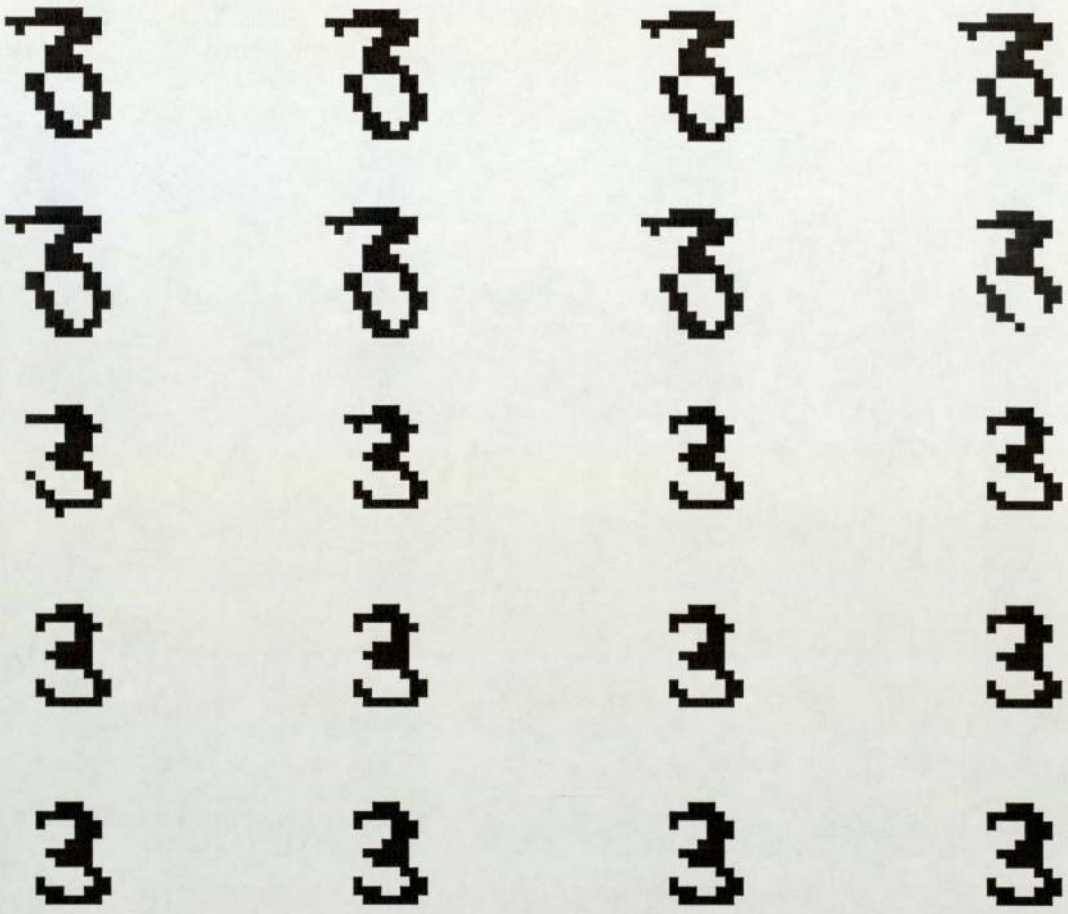

Figure 3.6: Example of transformation of a deformed character to its corresponding template. We can see that the inversion of the forward model (from left to right and from top to bottom) is progressively achieved in 20 steps. The last figure represents the gray-scale image of the corresponding perfect template.

as

$$\frac{1}{2}\sqrt{\sum_{i=1}^{14}\sum_{j=1}^{16}(O_{i,j}-T_{i,j})^2} \tag{3.25}$$

Obviously, we assume that the digit which corresponds to the template for which the value of this error is the lowest is the final result of the classification.

So far, we constructed a forward model to build deformed characters. These characters were to be used to train the network network we presented in this chapter. We shall now explain how this method can be applied to classify real world data.

Chapter 4

The real data

This chapter gives information about the real data which has been used to test the classification. We shall also see which modifications have been made to this data to make it compatible to the model which has been presented so far in this thesis.

4.1 The real data

4.1.1 General presentation

The real data used to test the performance of the networks comes from the NIST Handwritten Segmented Characters database. This database has been created in scanning 2100 forms. The fields on these forms have been isolated and the characters within those fields have been segmented, extracted, and placed into individual 128 by 128 pixel images, one character per image.

The structure of the real data is so quite similar to the structure of the digits we built. The image is stored in a binary matrix representing the frame. Indeed, as for the simulated data, the value of an element in the matrix is 1 if the corresponding pixel is on and 0 otherwise. However the size of the frame, and consequently of the matrix, is now 128 by 128, whereas a 140 by 160 element matrix is used for the simulated data. The first processing on the real data is to convert the size of the matrix into the desired

format.

4.1.2 Conversion of the format

It has been noticed in the first approach that the general size of characters is very small compared to the size of the frame which is used. Actually, the digits which have been pre-centered on the middle of the frame are only drawn on half the frame. The main problem we would have met if the size of the digits would have been left unchanged is that, in computing the gray-scale, much information would have been lost. Indeed the digit would have been drawn only on a few central windows of the gray-scale. Such an image would have been indecipherable even for human and it is of course not likely to be classified by the network. That is the reason why the conversion of the format has been done in doubling the size of the image to obtain a 256 by 256 element matrix. The required 140 by 160 matrix is built in only keeping a central window of this new frame.

4.1.3 Sample of distribution of the real data

It is relevant to compare real data (whose a sample is shown in figure 4.1) to the simulated data (in figure 2.4) in order to check if the stochastic model we have defined in the second chapter really models the way characters are written. It is also useful to adjust different parameters of the model such as the amplitude of the noise at the control points, or the average of the size of the digits. However it is hardly possible to assess this kind of values in just considering a sample of data. It was indeed necessary to find some reliable estimates of these values on real data and then to adapt the simulated data.

Besides, the main different shapes of the characters in the real data have to be generated by the stochastic model. That means in other words that, if a particular shape of a digit often appears in the real data, it has to be generated in the simulated data as well. The ideal model would be to be able to generate each specific shape in

the same proportions than in the real data. But of course this is impossible to achieve. Nevertheless, we can compute different estimates to assess the main features of the real data and then to be able to make the simulated data closer to them.

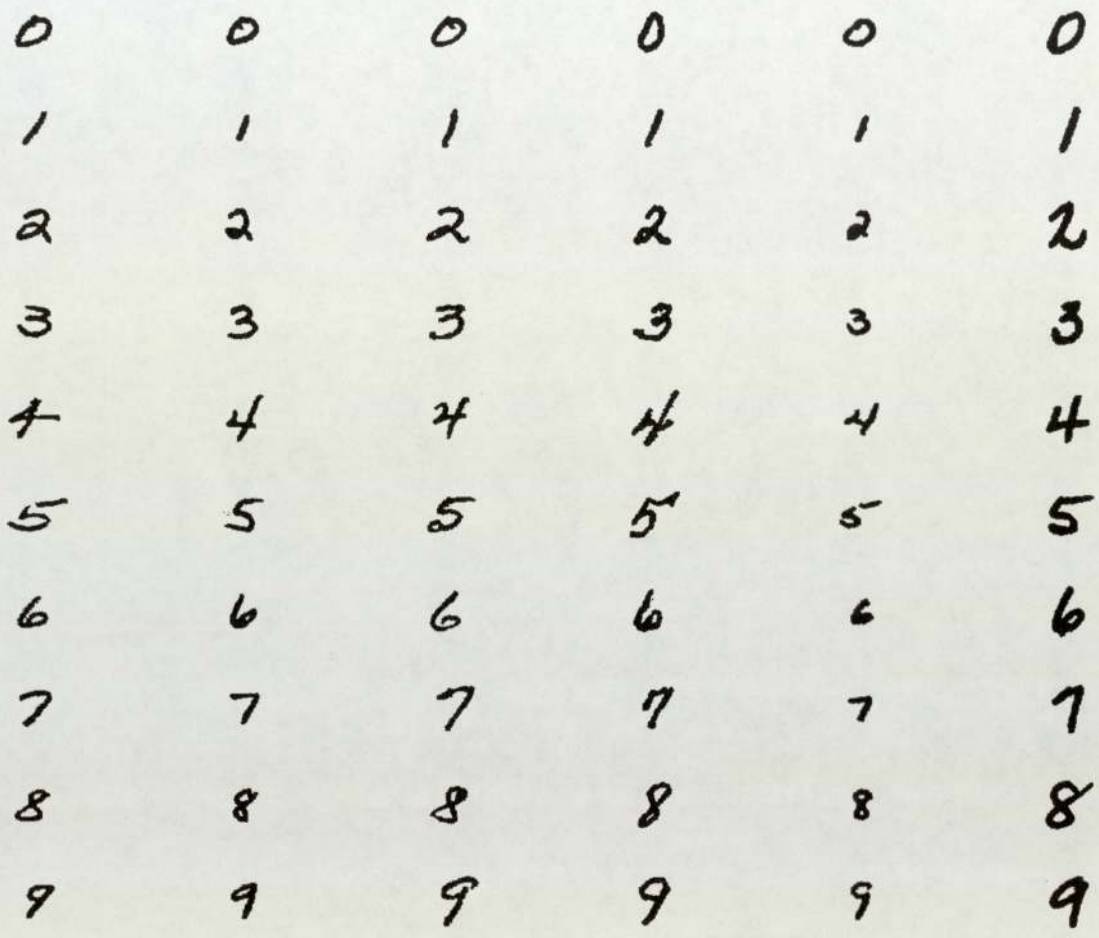

Figure 4.1: Sample of real data

Note that all tests and computations on real data have been done on a sample of about 400 examples for each digit digit. So, for the ten digits it represents about 4,000 characters. Of course, when some results are compared between real and simulated data, the number of examples considered is the same for real and simulated data.

4.2 Different estimates of the data features

As revealed in figure 4.1, the real data is heterogeneous compared to the sample of simulated data. We saw that it is important for the simulated data to be as close

as possible to the real data to improve the performance of the classification. Thus we shall see now which estimates have been used to adapt the stochastic model to the characteristics of the real data.

4.2.1 Average of the size

The size estimate

The first estimate we have used deals with the average of the size of digits. However, there is no direct means to compute this size. The solution adopted is to compute the percentage of gray-scale windows which are predominately inked. As the values of the gray-scale matrix lie between -1 and 1, and according to examples (table 3.1), it seems to be reasonable to consider the size of the whole character as the percentage of elements into the gray-scale matrix whose value is greater than 0.9.

Comparison between real and simulated data

Table 4.1 shows the comparison of this estimate between the simulated and real data.

Digit	Simulated data	Real data
0	26.4	18.8
1	15.6	9.7
2	27.0	18.8
3	29.5	19.8
4	24.6	18.7
5	26.5	20.0
6	31.1	18.6
7	16.8	16.0
8	33.7	21.4
9	25.9	19.4
average	25.7	18.1

Table 4.1: Comparison of the average of the size between simulated and real data for each digit

Table 4.1 reveals that the average of the size of the simulated data is greater than this of the real one for all digits. The modification of the average of the size on the

simulated data can be achieved by merely scaling the size of all the simulated data by a factor. Practically, this is achieved in multiplying the coordinates of the control points by a scaling factor. More accurate changes could be done after that, in considering for example the size of only one specific digit and redefining the location of the control points. Furthermore, changing the size of the characters obviously leads to consider different perfect templates, whose size is scaled as well.

4.2.2 Fluctuation of the size

The choice of the estimate

An other important factor we have to deal with is the fluctuation of the size of the characters. Figure 4.1 shows how the fluctuation of the size can be important in real data. The performance of the classifier could also be improved if this characteristic would also be taken into account. The estimate we chose in order to have an reliable value of the fluctuation is the variance of the size over a sample of examples. The different values for each digit on simulated and real data are reported in table 4.2.

Digit	Simulated data	Real data
0	0.06	0.32
1	0.02	0.08
2	0.07	0.40
3	0.07	0.41
4	0.06	0.31
5	0.08	0.41
6	0.12	0.37
7	0.05	0.21
8	0.13	0.48
9	0.10	0.30
average	0.08	0.33

Table 4.2: Comparison of the fluctuation of the size between real and simulated data

Table 4.2 reveals that the variance of the size is much greater for the real data than for the simulated data. These values could be more similar by adding a size fluctuation to the simulated data.

How to add a fluctuation of the size?

Given a uniform variable s with zero mean, the scaling factor is defined by $(1+s)$. The size of a character is then modified by multiplying the coordinates of the control points by this factor. However in this case the perfect templates are left unchanged. That means that at each pass through the neural network, this factor has to be reduced in order to finally reach the corresponding perfect template. This is achieved in considering at the step t a scaling factor whose value is $0.9^t(1+s)$.

4.2.3 The line width

We can find a good estimate of the line width of the characters in computing the percentage of gray-scale windows which are either partially or predominately inked. Thus we chose to define an estimate of the size width by calculating the percentage of elements into the gray-scale matrix whose value is greater than -0.9.

Nonetheless, this estimate depends on the average of the size of the characters since we take into consideration all values greater than -0.9. Thus we shall see in the next section the comparison between the real and the simulated data for this estimate, after having adjusted the average of the size on the simulated data.

4.2.4 Complexity of the data

The last estimate used to adapt the simulated data to the real data assesses the general complexity of the data. This value is found in computing the average quadratic error between the gray-scale image of the deformed digit and its corresponding perfect template. For example, given the gray-scale matrix of a digit $frame_{i,j}$ for $i = 1, \ldots, 14$ and $j = 1, \ldots, 16$, and the corresponding perfect template $T_{i,j}$, the complexity of the digit is calculated as following

$$Complexity = \frac{1}{2}\sqrt{\sum_{i=1}^{14}\sum_{j=1}^{16}(frame_{i,j} - T_{i,j})^2} \tag{4.1}$$

Table 4.3 shows the different values of this estimate for each digit in real and simulated data.

Digit	Simulated data	Real data
0	94.1	152.3
1	46.1	51.9
2	103.8	114.4
3	105.6	119.0
4	79.4	95.2
5	119.7	140.7
6	99.7	113.1
7	65.5	73.4
8	110.1	122.1
9	84.6	107.3
average	90.9	108.9

Table 4.3: Comparison of the average complexity of characters between real and simulated data. We can see that for all digits, the value is greater for the real data than for the simulated data.

This estimate may be useful to adjust the amplitude of the noise in the simulated data.

4.2.5 How to adjust these estimates?

The method to adjust these different parameters on simulated data would be first to reach a similar average of size by multiplying the data by a scaling factor. Then the method of computation of the line width becomes reliable and usable to adjust this second parameter. The fluctuation of the size can be so adapted according to the value of the variance of the size. Finally, the final estimate we presented, the complexity, can be utilised to find the more suitable noise amplitude to build the simulated data.

We shall see the evolution of all these parameters in the next chapter as well as their influence on the performance of the classification.

Chapter 5

Results and improvements

This section displays the results of the classification obtained throughout the evolution of the different parameters. We shall also see which pre-processing work has been done in order to make either the simulated data closer to real data, or the real data more homogeneous.

5.1 Original choice of the different parameters and first results

Some parameters presented so far in the thesis have been chosen arbitrarily and then adjusted according to the results of the classification. Here we presente the original values of most of the parameters used. We shall see then their influence and how they can be modified to improve the performance of the classifier.

5.1.1 Parameters of the network

The input to the network

We saw that the input to the network is defined at each step as the sum of a fraction of the previous output and a fraction of the previous target. In other words, the input

at the step t is

$$I^{\mu,t} = (1 - \lambda)\hat{O}^{t-1} + \lambda\hat{I}^{\mu,t} \tag{5.1}$$

where $\hat{O}$ is defined in 3.5 by

$$\hat{O}^t = (1 - \tilde{\lambda})I^t + \tilde{\lambda}O^t \tag{5.2}$$

We aim at finding the most suitable values for both parameters λ and $\tilde{\lambda}$. Since those parameters can widely influence how well the network learn, the most suitable values have been found empirically, testing the performance of the network for a range of values lying between 0 and 1. Note that some experiments showed that both parameters λ and $\tilde{\lambda}$ are independent. For that reason, we do not need to find a couple of values. Consequently, some experiments have been carried out to find λ with a constant value of $\tilde{\lambda}$, and some others to find $\tilde{\lambda}$ with a constant value of λ. Figure 5.1 shows the error rate according to the value of λ. The minimum of the error rate is reached for $\lambda = 0.175$.

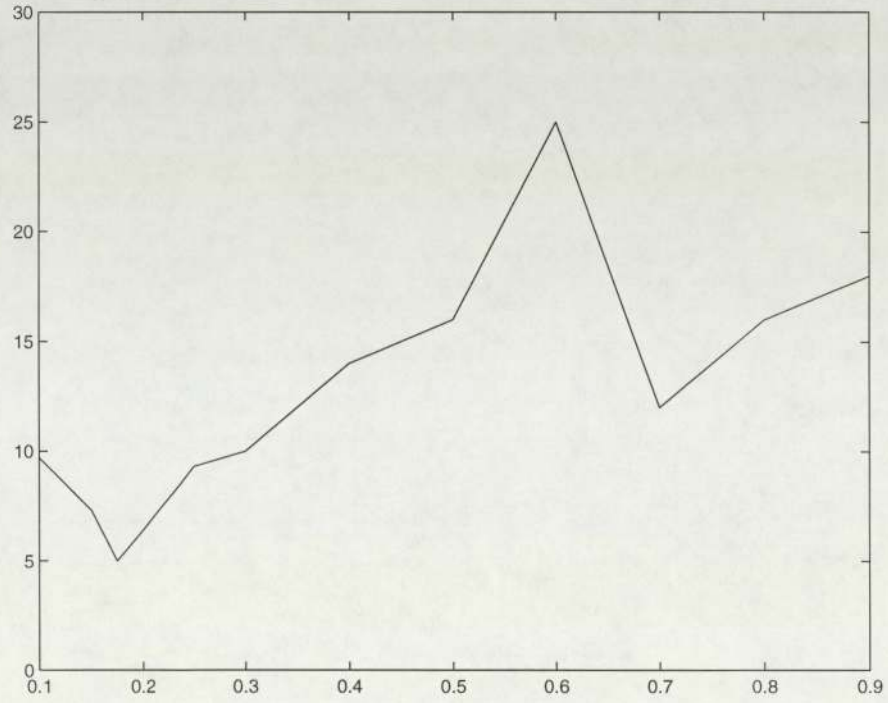

Figure 5.1: The misclassification error rate according to the value of the parameter λ. The classification has been taken over a sample of 2000 simulated patterns. The minimum of the error rate occurs for $\lambda = 0.175$.

The similar figure 5.2 gives the performance of the classification on simulated data according to the value of the parameter $\tilde{\lambda}$. The minimum of the error rate is reached for $\tilde{\lambda} = 0.1$.

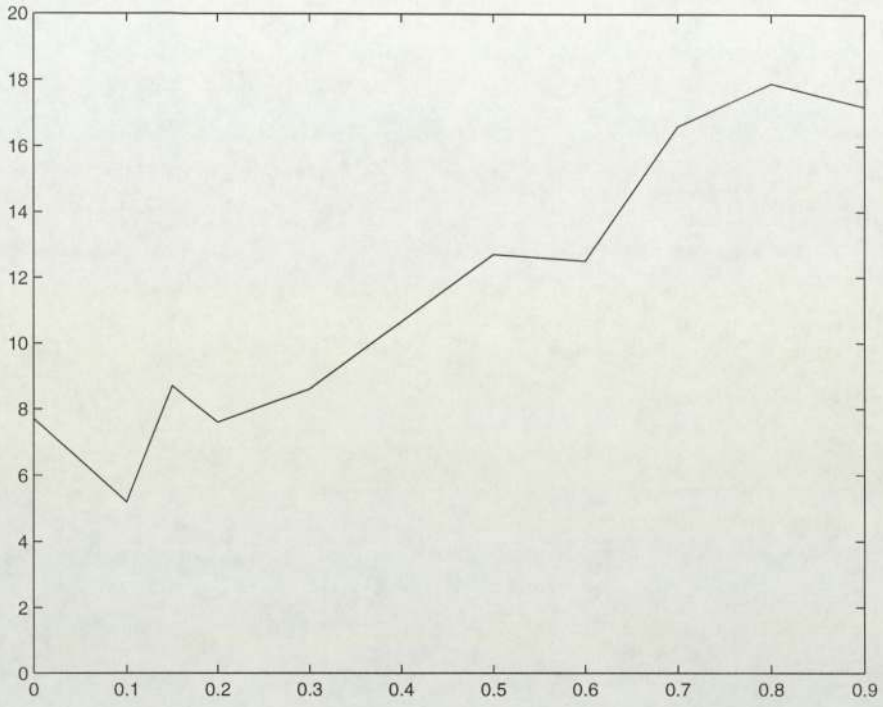

Figure 5.2: The misclassification error rate according to the value of the parameter $\tilde{\lambda}$. The minimum of the error rate occurs for $\tilde{\lambda} = 0.1$.

These values of λ and $\tilde{\lambda}$ shall stay unchanged for all the future tests.

Number of hidden units

There is no algorithm to define the ideal number of hidden units. However the first tests have been done with 20 hidden units whereas only the first five digits have been considered. Indeed, the first experiments only aim at adjusting different parameters, and for that reason, we are more interested in the comparison between the different results than in the results themselves.

Learning rate and number of input patterns

The value of the learning rate has been initially set equal to 0.0001 and the first experiments have been done in training the network with 10000 different input patterns.

The evolution of the quadratic error during the learning period is an important estimate to know if it is worth to modify these values. Indeed, increasing the number of input patterns is relevant only if the error is decreasing during the whole learning period.

Methodology of the experiments

As explained, the training set is constitued of simulated digits built from perfect templates. The performance of the network is then evaluated using a validation set of real data constitued of a large sample of 4000 real handwritten digits. The performance of the network has finally been confirmed bu measuring its performance on an other set of 4000 real digits. Note that the classification rate has been the same using either validation set or test set. This shows that the parameters have never been adapted to a specific set of data.

Results

As explained just above, the network has been trained only with digits 0 to 4. Indeed, the first aim was to adjust the various parameters of the network and simulated data, which are the same for all digits. It is so sufficient and quicker to only consider several digits. Figures 5.1 and 5.2 show that the classification is performed on the simulated data with an error rate of about 5%. But though results are satisfactory on simulated data, the error rate on real data is about 70% on a sample of 2106 examples.

Explanations

The results obtained on the simulated data prove that the dynamic model of the network which has been defined is efficient. The reason of the bad classification on real data is due to the too wide differences between real and simulated data. Thus, the first improvements to make aim at adapting the simulated data to the real data.

5.1.2 Training and recognition time

Classification speed is also of prime importance when testing the performance of a classifier. Using this method, the time required on a Sparc 5 to recognise a test pattern starting with 140 by 160 pixel map image is about 0.1 second. The training period over 10000 input patterns takes about 4 hours which represents less than 1.5 seconds for one input pattern.

5.2 First improvements

The first improvements concern the stochastic model defining the forward model of the distribution. The goal is actually to make the simulated data closer to the real data, using the estimates presented in the previous chapter.

5.2.1 Average of the size

The previous chapter showed that the average of the size is greater for the simulated data than for the real data. The idea is so to determine a scaling factor such that the average of the size of the simulated data is as close as possible to this of the real data. The average of the size of each digit has been computed using several different values of the scaling factor. The results are displayed in table 5.1.

The table reveals that the more adapted value for this factor of attenuation is 0.72. This value shall stay unchanged from now and for all the future experiments.

5.2.2 Fluctuation of the size

We also saw in the previous chapter that the fluctuation of the size is greater in the real data than in the simulated data. It may be useful to make the size of the simulated data fluctuate in order to make them closer to reality and to obtain more uniform results between simulated and real data. The idea is to define a uniform variable s with zero mean and variance σ_s to be determined. We can then scale the

	0.6	0.7	0.72	real data
0	14.5	18.9	19.4	18.8
1	7.4	9.1	9.3	9.7
2	14.4	18.8	19.2	18.8
3	14.0	19.2	19.3	19.8
4	12.9	16.2	16.8	18.7
5	16.2	19.8	20.2	20.0
6	16.1	19.6	20.0	18.6
7	12.2	15.5	15.7	16.0
8	18.5	22.2	22.7	21.4
9	15.1	19.1	19.5	19.4
average	14.1	17.8	18.2	18.1

Table 5.1: Average of the size of the simulated data for various values of the scaling factor. The size of a character is computed in counting the number of elements of the gray-scale matrix whose value is greater than 0.9.

size of the digit by the factor $(1 + s)$. Table 5.2 shows the variance of the size for each digit and for various values of the variance σ_s of the uniform variable s.

	0.6	0.8	0.9	real data
0	0.13	0.14	0.16	0.32
1	0.02	0.03	0.04	0.08
2	0.11	0.15	0.18	0.40
3	0.12	0.17	0.20	0.41
4	0.10	0.13	0.15	0.31
5	0.14	0.15	0.17	0.41
6	0.21	0.23	0.28	0.37
7	0.07	0.08	0.09	0.21
8	0.21	0.25	0.30	0.48
9	0.14	0.17	0.18	0.30
average	0.13	0.15	0.18	0.33

Table 5.2: Variance of the size of simulated according to the variance σ_s of the uniform variable s

Whatever the value of the variance σ_s is, the variance of the size is always lower in the simulated data than in the real data. The problem is that it would not be efficient to set the variance σ_s equal to a value greater than 0.9 since many digits would then be either larger than the window or on the other hand too small. What is surprising is that we cannot find a suitable value for the simulated data which would allow to obtain similar results to the real data for the variance of the size. That would actually

mean that the real data are either small or large.

5.2.3 Complexity of the data

We shall try now to adjust the amplitude of the noise in computing the complexity of the simulated data using various values. The amplitude of the noise actually corresponds to the variance of the Gaussian noise added to the coordinates of the control points. This Gaussian noise appears in equation 2.19 and is called η_j. The complexity of the data is given in table 5.3 for different values of the variance.

	0.2	0.25	real data
0	129.1	131.0	152.3
1	37.1	39.7	51.9
2	103.9	106.3	114.4
3	111.0	114.8	119.0
4	72.6	79.9	95.2
5	113.7	117.2	140.7
6	102.6	106.1	113.1
7	65.3	70.5	73.4
8	112.9	117.7	122.1
9	87.4	90.5	107.3
average	93.5	97.4	108.9

Table 5.3: Complexity of the simulated data according to the value of the variance of the Gaussian noise η_j

This table reveals that the computation of this estimate gives greater values for the real data than for the simulated data. As for the fluctuation of the size, it would not be efficient to construct a model with too important noise since the characters would be indecipherable and less close to the real data.

5.2.4 The line width

The line width has also been adjusted. However this parameter is not constant in the real data. That is why it has been defined as a uniform variable in order to make the line width fluctuate in the simulated data as well. We want the network to perform the classification whatever the line width is, and to reach the corresponding templates whose line has an average width.

5.2.5 Performance of the classifier

Two different neural networks have been trained, one for digits 0 to 4, and another one for digits 5 to 9. Besides, networks have been trained with both values of the variance of the noise η_j 2.0 and 2.5.

Amplitude of the noise of 2.0

The classification on the simulated data has been performed with an error rate of 14.1% for a variance of the noise η_j equal to 2.5. Those results are of course worse than previously. Indeed the simulated data is now more complex since it has been built with a fluctuation of the size and of the line width. The results of the classification on real data are reported for each digit in table 5.4.

0	37.5	5	65.3
1	8.2	6	24.4
2	47.3	7	38.6
3	18.1	8	69.9
4	58.1	9	36.1
average	33.8	average	46.9

Table 5.4: The misclassification performance on real data after the first modifications. The values represent the error rate of the classification which has been taken over 4081 examples for the 10 digits. The global average of the error rate for the 10 digits is 39.55%.

We can see that the results are still not satisfactory on real data and that the performance of the classifier widely depends on the digit. This last point may mean that some templates are not adapted to the real data.

Amplitude of the noise of 2.5

We shall see now the influence of the amplitude of the noise. Table 5.5 shows the results on real data for digits 0 to 4 only.

If the results are worse on real data with a greater value for the amplitude of the noise, it means that the problem does not come from the ability of the network

0	36.1
1	15.5
2	49.9
3	21.4
4	48.4
average	34.3

Table 5.5: Performance of the misclassification on real data for digits 0 to 4 with an amplitude of the noise equal to 2.5. The classification has been taken over a sample of 2106 examples.

network to learn, but rather from the templates which are not close enough to reality. This idea is emphasised by the fact that training the network with more hidden units gives results which are even worse on real data. Indeed, the average of the error rate is 42% on the same sample of real data when using a network with 25 hidden units.

We shall now see how to increase again the complexity of the simulated data with a view to translating various ways real characters may be written.

5.3 Increasing the complexity of the simulated data

The goal is now to increase the complexity of the simulated data in order to reach similar performance between real and simulated data. We can notice for example that we did not consider any orientation in our model, whereas many characters from the real data are skewed. Thus a random orientation shall be added to the simulated data.

5.3.1 Orientation of the characters

How to construct skewed characters?

The idea here is to define a random angle of rotation ψ, whose value is determined according to a uniform variable with zero mean and variance σ_ψ. The figure is then rotated with the angle ψ around the lower-left corner of the figure. In other words, the

new coordinates (x', y') after rotation of the point (x, y) are

$$x' = x\cos\psi + y\sin\psi \tag{5.3}$$

$$y' = -x\sin\psi + y\cos\psi \tag{5.4}$$

Of course, we aim at training the neural network in order to reach the same perfect template which means that during training, the initial value of ψ computed for each character shall actually be at the step t: $0.9^t\psi$. The problem is now to find the most suitable value of σ_ψ.

Influence of the rotation on the complexity

The complexity of the simulated data is now expected to be increased after this new modification. Table 5.6 shows the results for both values $\frac{\pi}{3}$ and $\frac{\pi}{4}$ of σ_ψ.

	$\frac{\pi}{4}$	$\frac{\pi}{3}$	real data
0	141.8	143.4	152.3
1	44.7	46.9	51.9
2	116.5	117.9	114.4
3	118.2	120.9	119.0
4	90.7	90.3	95.2
5	120.7	121.8	140.7
6	107.8	108.9	113.1
7	83.3	85.3	73.4
8	118.1	118.7	122.1
9	92.4	93.5	107.3
average	103.4	104.8	108.9

Table 5.6: Complexity of the simulated data according to the value of σ_ψ, the variance of the uniform variable ψ which defines the angle of rotation of the characters

Table 5.6 shows that the complexity is now almost similar between simulated and real data thanks to this new modification. We shall see now the results of the classification if the network is trained with this new model.

5.3.2 Results of the classifier

Again, two different networks have been trained, one for digits 0 to 4, and one for digits 5 to 9. We shall see now the results in detail, for each digit of the simulated and

real data.

The simulated data

The classification has been taken over a sample of 2000 examples for each network is given in table 5.7 for $\sigma_\psi = \frac{\pi}{4}$.

0	12.2	5	47.0
1	0.4	6	21.2
2	21.2	7	4.2
3	15.8	8	50.6
4	16.4	9	45.6
average	13.2	average	31.9

Table 5.7: The misclassification error rate on simulated data with $\sigma_\psi = \frac{\pi}{4}$. The global average of the error rate for the ten digits is 22.6%.

The real data

The results obtained on real data are given in table 5.8 for the same value of $\sigma_\psi = \frac{\pi}{4}$.

0	30.3	5	91.76
1	11.6	6	50.25
2	65.1	7	19.06
3	14.1	8	62.25
4	62.21	9	62.66
average	35.5	average	55.8

Table 5.8: Performance of the misclassification on real data with $\sigma_\psi = \frac{\pi}{4}$. The global average of the error rate for the ten digits is 45.3%.

Besides, the same networks have been trained with a value of σ_ψ equal to $\frac{\pi}{3}$. The error rate on simulated data is greater than for $\sigma_\psi = \frac{\pi}{4}$ as it could be expected since the data are more complex. However, the error rate on real data also is greater whereas table 5.6 shows that the difference of complexity between real and simulated data is less important for this value of σ_ψ. The global error rate obtained for the ten digits is indeed about 50%.

Explanations of the results

As it has been pointed out, increasing the complexity, by for example adding a rotation of the characters, really improves the performance of the classifier. However, this complexity cannot be increased too much since the digits would be then even less closer to the reality. That is why we shall try rather now to decrease the complexity of the real data in achieving pre-processing work on characters before classifying them.

5.4 Decreasing the complexity of the data

We saw that the complexity of the characters in real data is very important. The key idea is now to decrease this complexity in first normalising the size of the characters. Besides we do not have much information about the orientation of the characters since a reliable estimate of this parameter is difficult to find. Thus we decided to modify the procedure which transforms the initial image into its gray-scale format, in order to normalise the size of the characters and suppress the skew angle.

5.4.1 Method adopted

The coordinates of the center of mass of the image are (G_X, G_Y). The computation of those coordinates has been explained in equation 3.1.

Normalisation of the size

The size of the characters is normalised in constructing the gray-scale image such that the average distance between the center of mass and the different points of the image is constant for all digits. The former value of this average distance D_G is computed as follows

$$D_G = \frac{\sum_{i=1}^{140} \sum_{j=1}^{160} frame(i,j) \sqrt{(i - G_X)^2 + (j - G_Y)^2}}{N} \tag{5.5}$$

where N is the total weight of the image. Every point (x, y) is moved to the new one (x', y') using

$$x' = \frac{(x - G_X)C}{D_G} \tag{5.6}$$

$$y' = \frac{(y - G_Y)C}{D_G} \tag{5.7}$$

where C is a constant which determines the average of the size of the digits.

Suppression of the skew angle

The skew angle of the characters is computed as the average angle of every point of the image with respect to the X-axis. Obviously, this computation does not give the real skew angle of the figure as it is usually defined, but what is important here is to apply the same method to all characters in order to reach more homogeneity in the data. The average angle Ψ is calculated as

$$\Psi = \frac{\sum_{i=1}^{140} \sum_{j=1}^{160} frame(i, j) \arctan \frac{j - G_Y}{i - G_X}}{N} \tag{5.8}$$

The image is then rotated with an angle $(-\Psi)$ using the method explained in the above paragraph (equations 5.3 and 5.4).

5.4.2 Visualisation on examples

Figure 5.3 shows several examples of transformation of characters to their gray-scale format. Compared to figure 3.1, the size of the digits is now normalised and the skew angle is suppressed.

5.4.3 New results

New experiments have been carried out. Several networks have been trained with different values of the amplitude of the noise. The best results on real data have been reached for a noise factor of 0.25. Those results are displayed in table 5.9.

Table 5.9 reveals that if this last modification improves the performance of the classifier, we still obtain bad results for specific digits such that digit 5 or digit 2,

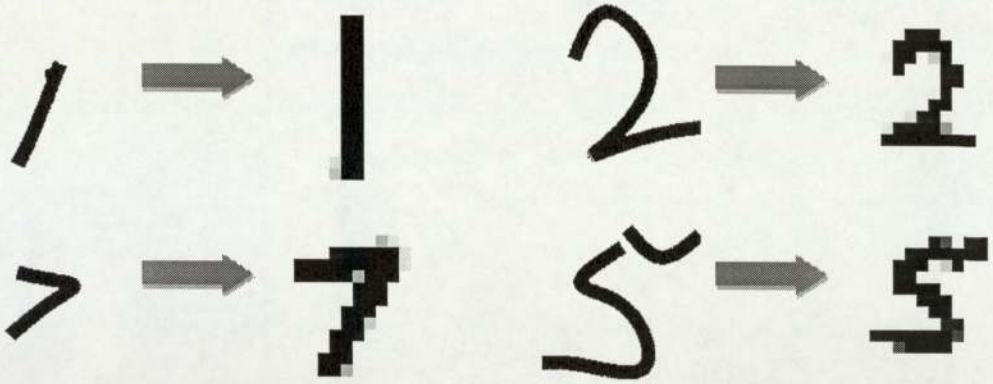

Figure 5.3: Examples of the transformation of a character into its gray-scale format after normalisation of the size and suppression of the skew

0	19.8		5	74.7
1	0.0		6	25.1
2	59.0		7	20.5
3	10.5		8	52.7
4	8.6		9	16.9
average	18.8		average	37.0

Table 5.9: The misclassification performance on real data after normalisation of the size and suppression of the skew. The global average of the error rate for the ten digits is 27.6%.

whereas simulated digits are well recognised. Indeed, an error rate of about 5% is now obtained on simulated for digits 0 to 4, and about 20% for digits 5 to 9. In this case, the problem is definitely due to the non-adapted perfect templates used to define those digits. We shall see now which modifications have been made to the templates.

5.5 Last improvements and final results

5.5.1 The new perfect templates

The main point here is to find perfect templates whose shapes are as close as possible to the real data. In other words, we aimed at finding control points whose location is the average position of the corresponding control points of the real data. To find the most suitable location, several experiments have been carried out with different templates. Obviously, the main modifications have been achieved to digits for which

the classifier gave the worst results. The best performance has been obtained using the templates shown in figure 5.4. Table A.2 shows the corresponding coordinates of the new control points.

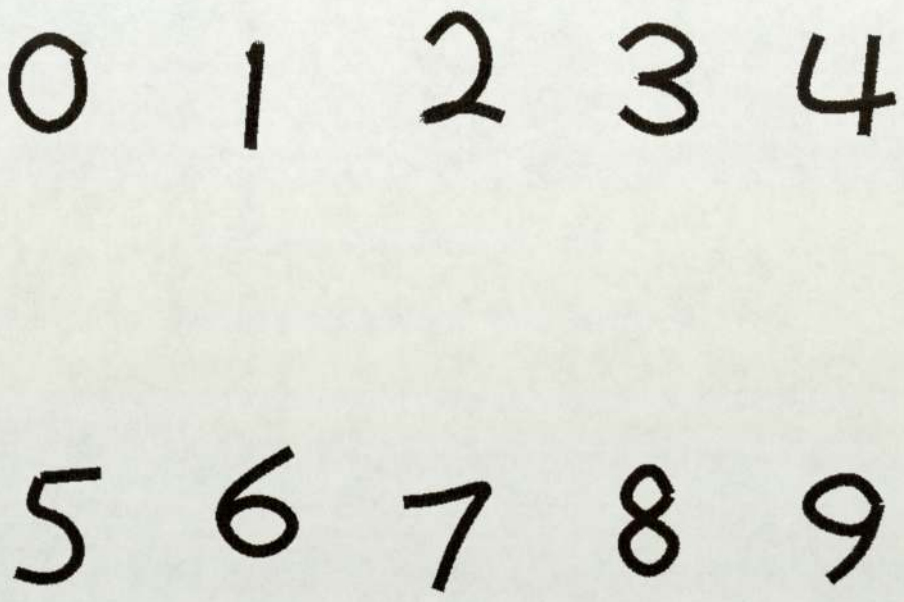

Figure 5.4: New perfect templates

Comparing figure 5.4 to figure 2.2 which shows the original perfect templates, we can point out several modifications:

- Digits 0 and 8 are less wide
- Digit 1 is now drawn with only one straight line
- The location of the control points of digit 2 are less distant to each other on the lower-left corner in order to construct more digits whose shape possesses a loop as in the real data
- The second segment of digit 4 has been moved
- Digit 5 is more skewed

Besides, it has been useful to increase the amplitude of the noise with those new templates since they are closer to the real data in order to construct as many different shapes of characters as possible which are likely to be met in real data.

5.5.2 Performance using those new templates

The problem to be solved now is quite different since the performance of the classifier is better on real data than on simulated data. Indeed the error rate is now about 16% on real data but it has increased up to 24% on simulated data. Those results would lead to think that the classification performance could be improved in modifying the parameters of the network, and that the templates are now adapted to the real data. To understand how the network is learning, we computed the error function such as it has been defined in equation 3.6 after each input pattern. Figure 5.5 shows the evolution of this error over the 10000 examples of the learning period.

We can see that the error is decreasing all over the learning period. That obviously means that better results could be reached in increasing the number of input patterns. Now that the templates are more adapted to the real data, it is worth increasing the number of hidden units as well. Several new experiments have been done with different values of hidden units. The actual best results are displayed in table 5.10. A network with 50 hidden units has been trained over 50000 input examples.

0	4.7	5	13.6
1	0.2	6	11.9
2	18.2	7	9.5
3	23.6	8	19.6
4	5.3	9	23.3
average	10.1	average	15.5

Table 5.10: The misclassification performance on real data using a network with 50 hidden units and trained over 50000 examples. The global error rate for the ten digits is 12.7%.

With those same parameters, the performance is still worse on simulated data as shown in table 5.11.

0	12.0	5	14.7
1	3.4	6	17.4
2	14.8	7	13.0
3	26.6	8	18.5
4	19.2	9	26.3
average	15.2	average	18.0

Table 5.11: The misclassification performance on simulated data using a network with 50 hidden units and trained over 50000 examples. The global error rate for the ten digits is 16.6%.

5.6 Present conclusions

The problem after the first experiments was to reach the same performance on real data as on simulated data. In this case, the problem came mainly from the perfect templates which were not adapted to the real data. However the problem finally changed since better performance is achieved on real data than on simulated data. We saw that different modifications to the parameters of the network such as increasing the number of input patterns or the number of hidden units improve that performance. Nevertheless new experiments show that longer learning period would not help anymore to reach better results. Indeed, by considering the misclassified digits from the real data, and on the other hand from the simulated data, we can understand why such results are obtained on simulated data. Actually figure 5.6 shows that misclassified simulated digits are especially constituted of indecipherable digits which are generated by our model whereas misclassified real digits seem to be quite easy to be recognised (figure 5.7). This idea leads to think that the high error rate we obtained on simulated data is due to these junk characters. In addition, if real digits such these shown in figure 5.7 are not recognised, that means that the location of the control points is still not adapted to the real world data and that better performance on real data would be achieved by modifying the perfect templates in order to make them again closer to reality.

Finally, a single network has been trained to classify all ten digits. Obviously, the performance is lower than before since the network is more likely now to confuse digits.

The results achieved on real data are displayed in table 5.12.

0	25.2
1	1.5
2	29.9
3	52.6
4	47.1
5	15.9
6	24.9
7	20.0
8	27.7
9	33.8
average	27.6

Table 5.12: The misclassification performance on real data training a network for the ten digits

We can see that the performance of the classifier is still not reliable enough. Improvements on simulated data are still to be done in order to attain more acceptable results.

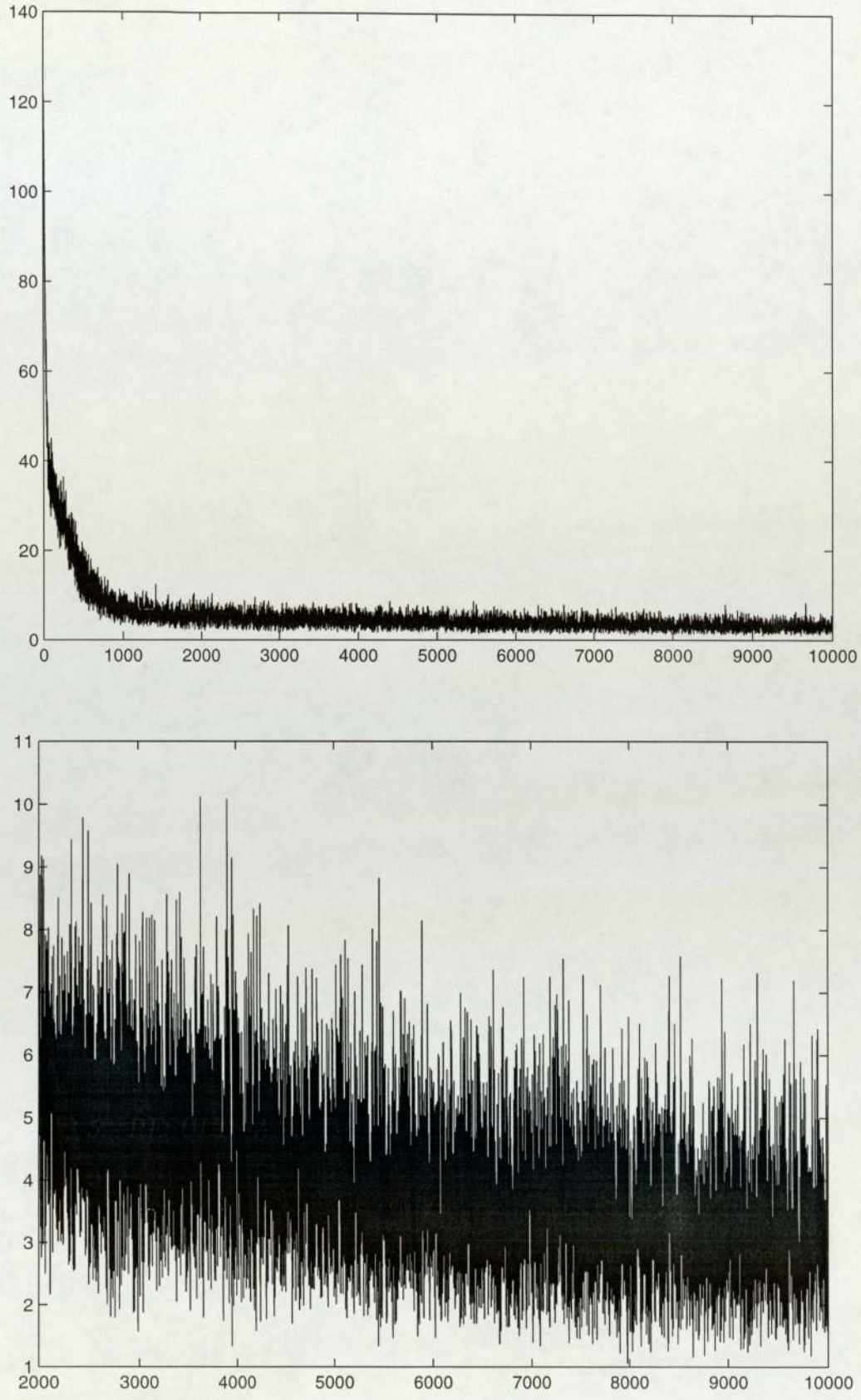

Figure 5.5: Evolution of the error over the 10000 examples of the learning period

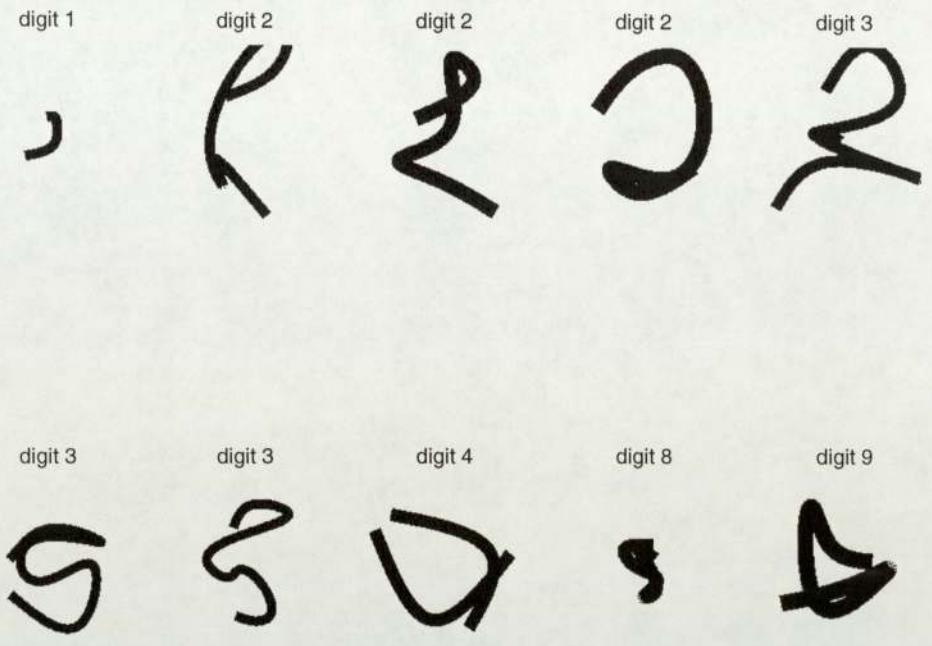

Figure 5.6: Sample of misclassified simulated data

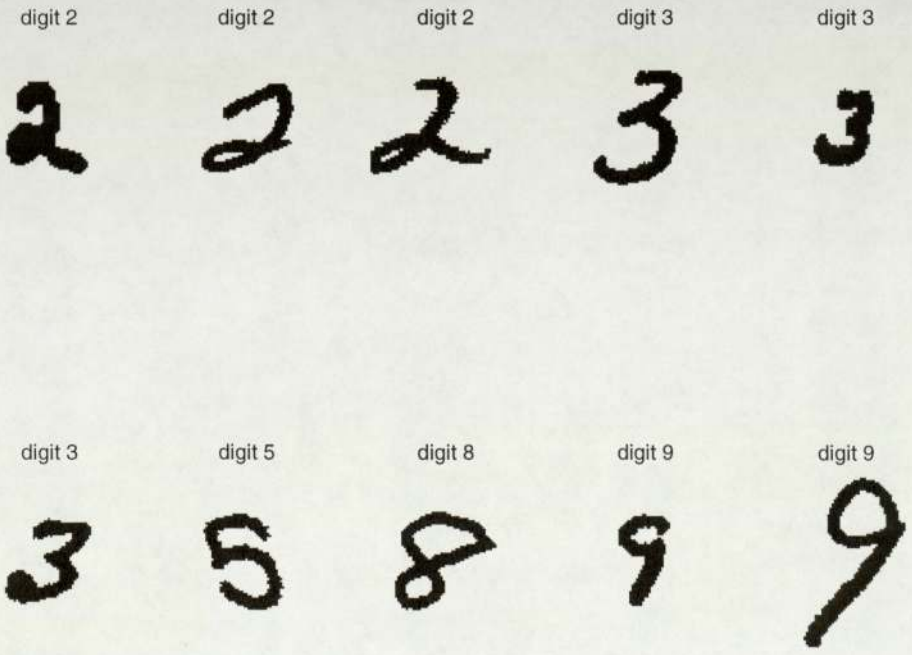

Figure 5.7: Sample of misclassified real world data

Chapter 6

Conclusions and future work

6.1 Conclusions

A method combining deformable templates and neural networks for handwritten characters recognition has been presented. As in the deformable templates approach to HCR, a character is described by a set of control points and spline segments are drawn through these points. A forward model of the distribution of characters is then obtained by adding a noise process at the location of those control points. We saw how this noise process has been chosen in order to translate the way characters are actually written. For that reason, the noise at each control point has been computed recursively as a weighted average of the noises defined at the previous points.

This forward model of the distribution has been built in order to train a recurrent neural network whose aim was to incrementally undeform the characters back to their corresponding perfect templates. It was required that the full inversion be achieved in 20 steps. Due to the large amount of data which was required to train the network, on-line learning was utilised. The weights of the two layers of the network were updated using gradient descent optimisation in conjunction with back-propagation.

The main goal of this project was to test this method against real world data. Unfortunately, the model was not sufficiently adaptive to be able to handle the large diversity

of characters which are likely to be met in practice. Indeed, the first experiments gave a misclassification rate of about 70% on real data whereas this rate was only of about 5% on simulated data. Therefore techniques to improve the model were investigated in the form of adapting the simulated data to the real data by either adjusting the location of the control points, or increasing the complexity (fluctuation of the size, rotation of the whole characters). We also saw how the complexity of the data was reduced by normalising its size and suppressing its skew. The best performance provided by the classifier was a misclassification rate of approximatively 10% on real data in considering only 5 digits. In addition, the last experiments gave worse results on simulated data than on real data. We pointed out that this fact was due to the junk characters which were built by the stochastic model and consequently, the classification performance on real data can be improved further by making the simulated data closer to the real data.

At this present time, the classifier is not reliable enough to confidently build the vector field associated with the trajectory of the network. The next paragraph explains how to construct this vector field.

6.2 Future work

So far the model we built aimed at classifying real data with a reasonable error rate. We wanted this classification to be achieved by progressively making unidentified characters less deformed in order to be able to analyse the entire trajectory of the network. This trajectory can be associated with a vector field, which then allows to find the deformation of the underlying plane corresponding to the deformation of the character and to provide structural identification of it.

6.2.1 How to find the vector field?

The aim of this paragraph is to determine how to find the vector corresponding to the trajectory of the network at a point x. This vector is denoted by the function h.

The function f corresponds to the output to the network at the step t, and g at the step $t+1$.

The one dimensional problem

We shall first see how this problem can be solved in one dimension. Given both functions f and g, the problem consists of finding the function h such that $g = f(h(x))$.

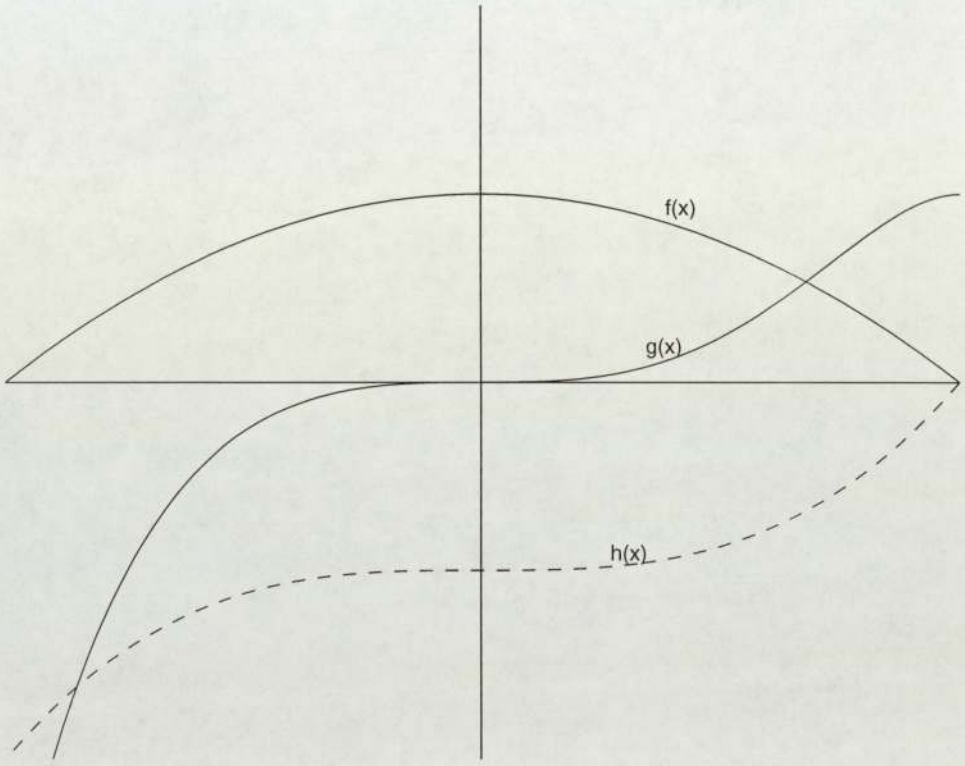

Figure 6.1: Given functions f and g, we aim at finding the function h such that $g = f(h(x))$

We note φ the function defined as $\varphi(x) = h(x) - x$. If we assume that $\varphi(x)$ is small, we can compute $g(x)$ as

$$g(x) = f(h(x)) = f(x + \varphi(x)) \simeq f(x) + f'(x)\varphi(x) \tag{6.1}$$

We can so deduce

$$\varphi(x) = \frac{g(x) - f(x)}{f'(x)} \tag{6.2}$$

Equation 6.2 might also have been obtained from requiring that $\varphi(x)$ minimises

$$(f(x) + f'(x)\varphi(x) - g(x))^2 \simeq (f(x + \varphi(x)) - g(x))^2 \tag{6.3}$$

The two dimensional problem

We conserve the same notations than in the above paragraph, but now x and $\varphi \in \mathbb{R}^2$. The expression of $g(x)$ becomes

$$g(x) = f(h(x)) = f(x + \varphi(x)) \simeq f(x) + grad(f(x))^T \varphi(x) \tag{6.4}$$

and for each point x, we choose $\varphi(x)$ to minimise

$$(f(x) + grad(f(x))^T \varphi(x) - g(x))^2 \tag{6.5}$$

The problem we have to deal with now is that we have two unknowns for the only equation 6.4. The idea is then to choose the function $\varphi(x)$ such that if a point x is close to another point $\hat{x}$, then $\varphi(x)$ is close to $\varphi(\hat{x})$.

To solve the problem numerically, assume points $X_{i,j}$ form a grid in the real plane and denote by $\varphi_{i,j}$ the value $\varphi(X_{i,j})$. Since $X_{i+1,j}$ is a neighbouring point of $X_{i,j}$, we want $\varphi_{i,j}$ to be close to $\varphi_{i+1,j}$. Setting

$$\bar{\varphi}_{i,j} = \frac{1}{4}(\varphi_{i+1,j} + \varphi_{i-1,j} + \varphi_{i,j+1} + \varphi_{i,j-1}) \tag{6.6}$$

we thus require

$$\varphi_{i,j} \simeq \bar{\varphi}_{i,j} \tag{6.7}$$

Combining constraint 6.7 with equation 6.5, we thus choose $\varphi_{i,j}$ to minimise

$$E(\varphi_{i,j}) = \sum_{i,j} (f(X_{i,j}) + gradf(X_{i,j})\varphi_{i,j} - g(X_{i,j}))^2 + \mu \sum_{i,j} (\varphi_{i,j} - \bar{\varphi}_{i,j})^2 \tag{6.8}$$

where the parameter μ controls the smoothness of the vector field φ. Note 6.8 is quadratic in $\varphi_{i,j}$, and so it is straightforward to solve.

Figure 6.2 shows an example of vector field translating the deformation of one character. The character is the deformed one, and the vector field corresponds to the trajectory of the network.

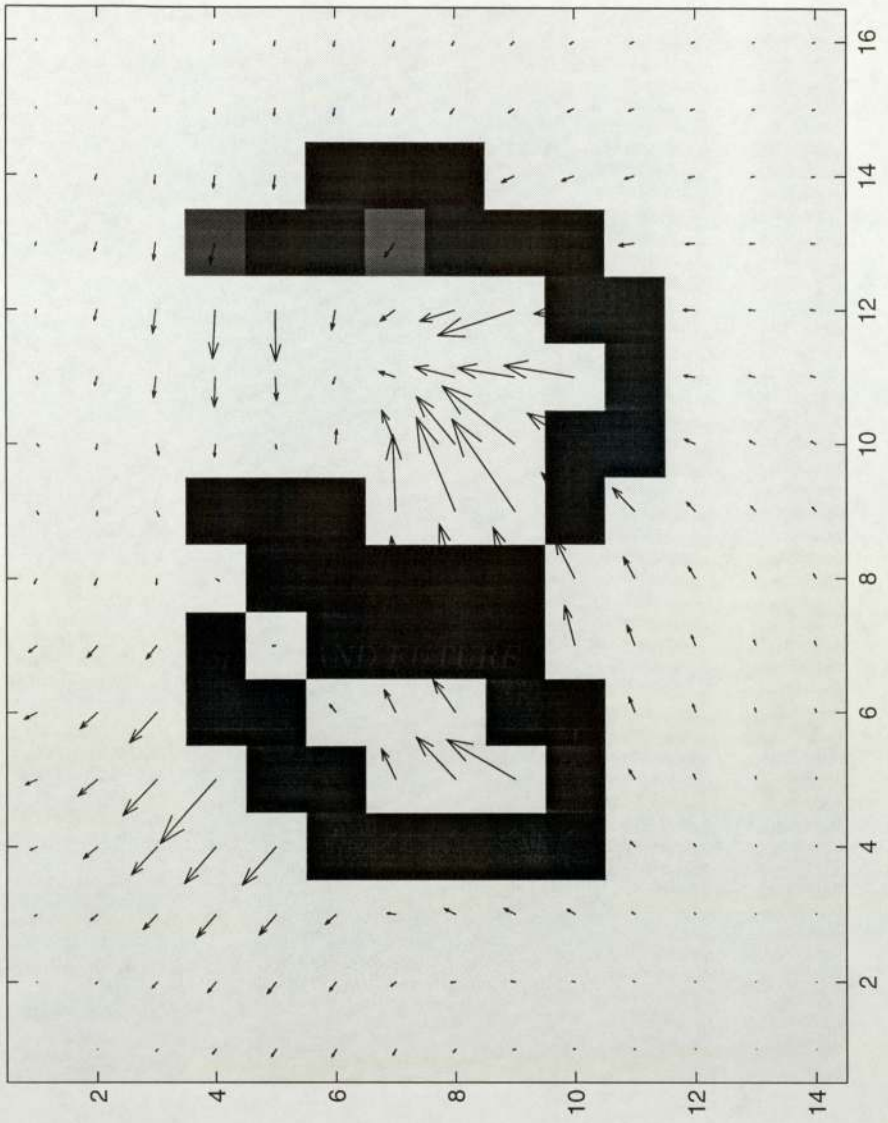

Figure 6.2: Example of vector field associated with the deformation of a character

We found how to compute the trajectory corresponding to one pass through the network. Nevertheless, the total deformation of a character is computed in integrating φ over all time steps.

6.2.2 The control points of the deformed data

Once this vector field is found, it becomes possible to locate the position of the control points on deformed real data. Indeed, from the deformed template, a vector field associated with the deformation of the character can be constructed. If this vector field is applied to the original deformed digit, the corresponding perfect template should

be reached for which the location of the control points is perfectly known. Therefore, it is also possible to find the location of the control points for real data by applying the opposite vector field to the perfect template.

6.2.3 Reliability of the match

Assuming the vector field corresponding to the trajectory of the network is found, the reliability of the match can be assessed by applying the vector field to the undeformed character and computing the average quadratic error between the result and the corresponding perfect template. The value of this error is then a good estimate of the reliability of the classification.

6.2.4 New location of the control points

The idea is that once the control points of the real data are known, we can define the new locations of the control points for the stochastic model by averaging the known locations over the real data. In addition, it would also be possible to define the noise process at these points by computing the deformation at the different points. In other words, this first approach allows us to define a new forward model of the distribution which will be well adapted to the real data since it is constructed from it. Furthermore, it becomes possible to adapt the classifier to the characteristics of a single writer in computing the average location of those specific control points and defining the main features of the characters.

Finally, this new forward model can be used to retrain the network. What is interesting about this method is that it is possible to repeat this process as many times as we want in order to improve the performance of the classifier since each time the resulting forward model is closer to reality.

Appendix A

Location of the control points

A.1 Original values

Digit	Coordinates of the control points									
0	650 700	400 850	150 700	150 300	400 150	650 300	650 700	-	-	-
1	250 650	400 850	450 900	435 580	400 100	-	-	-	-	-
2	170 700	400 900	600 750	300 315	150 100	200 100	650 100	-	-	-
3	150 750	400 900	650 700	300 505	250 500	300 495	650 200	400 100	150 250	-
4	300 900	100 500	700 500	eos	400 800	400 500	400 100	-	-	-
5	100 890	100 700	100 650	400 550	600 250	100 150	eos	100 900	350 900	600 900
6	600 900	100 300	350 100	600 300	350 500	100 310	100 300	-	-	-
7	100 900	500 900	700 900	600 800	200 100	-	-	-	-	-
8	600 700	400 900	200 700	650 300	400 100	150 300	550 620	600 700	-	-
9	600 700	400 900	150 700	400 500	640 650	650 700	640 650	300 100	-	-

Table A.1: Coordinates of the original control points. In the work of Dr R. Urbanczik the size of the frame used was 800 by 1000. The coordinates of the control points had to be converted to the new smaller frame size. This has been achieved in merely multiplying the X-coordinates by 140/800 and the Y-coordinates by 160/1000.

A.2 Final values

We saw that it was useful to modify the coordinates of the control points used to define the perfect templates in order to make them closer to real world data. The final values which have been used for the last experiments are given in table A.2.

Digit	Coordinates of the control points									
0	650 700	400 850	200 700	200 300	400 150	600 300	600 700	-	-	-
1	400 820	450 850	470 900	455 580	420 100	-	-	-	-	-
2	170 700	400 900	550 750	220 110	150 100	200 130	720 100	-	-	-
3	150 750	400 900	650 700	320 505	280 500	320 495	650 200	400 100	150 250	-
4	200 900	100 400	750 400	eos	600 900	560 500	520 100	-	-	-
5	180 890	140 650	290 580	450 250	290 100	0 140	eos	180 890	390 910	620 930
6	600 900	100 300	350 100	600 300	350 500	100 310	100 300	-	-	-
7	100 800	500 870	700 900	650 800	350 100	-	-	-	-	-
8	550 700	400 900	240 700	580 300	400 100	220 300	500 620	650 700	-	-
9	650 700	450 900	150 700	400 500	640 650	650 700	640 650	300 100	-	-

Table A.2: Coordinates of the control points we finally used

Appendix B

Code

B.1 Generating deformed templates

The goal of this procedure is, given a value d of digit between 0 and 9, and a value t corresponding to the step, to construct a 140 by 160 element matrix corresponding to the frame which contains the image of a digit d to which the noise $0.9^t\mu$ is added. The noise is initialised at the first execution of this procedure which corresponds to a value of t equal to 0. This procedure has been implemented in C and is called from a Matlab routine ([Mokhtari and Mesbah 1997]).

```
/* Input : 2 arguments, digit 'd' and step 'time' */
/*     if 'time' = 0, it generates noise 'mu' for the desired digit */
/*     if 'time' <>0, it gives a frame of the digit to which it
           has been added the noise 'time' times attenuated */


#include "include_file.h"

#include "declaration.h"


/* Initialisation of the static array noise */
void init_noise()
{
  int i;
  static long idum = -1;
  scale = (ran1(&idum)-0.5)*fluctuation_factor;
```

```
  angle = (ran1(&idum)-0.5)*angle_fluctuation_factor;
  width_factor = (ran1(&idum)-0.5)*width_fluctuation;
  for (i=0;i<2*nb_max_points;i++)
    noise[i] = gasdev(&idum)*noise_level;
}


void init_x_y(int d, double x[], double y[])
{
  int i;
  for (i=0;i<number_point[d];i++) {
    if (digit[d][0][i]==eos)
      x[i+1] = eos;
    else {
      x[i+1] =
digit[d][0][i]*((double)framex/(double)framex_init)*size_attenuation +
20.0;
      y[i+1] =
digit[d][1][i]*((double)framey/(double)framey_init)*size_attenuation +
20.0;
    }
  }
}


void add_noise(int time, int d, double x[], double y[], double
x_mod[], double y_mod[])
{
  int count = 0;
  int i, j;
  double denom, numx, numy, l, xm, ym, num;
  double fluctuation;
  double phi;
  x_mod[1] = x[1];
  y_mod[1] = y[1];
  for (i=2;i<=number_point[d];i++) {
    if (x[i]==eos)
      x_mod[i] = eos;
    else {
      denom = 0;
      numx = 0;
      numy = 0;
      for(j=1;j<=i-1;j++) {
        if (x[j]!=eos) {
          l = 1/(dist(x[i],y[i],x[j],y[j])+epsilon);
          numx += l * (x_mod[j]-x[j]);
          numy += l * (y_mod[j]-y[j]);
```

```
          denom += 1;
        }
      }
      if (denom!=0) {
        numx = numx/denom;
        numy = numy/denom;
      }
      x_mod[i] = x[i] + numx + noise[count] *
pow(attenuation_factor,time)*sqrt(1/l-epsilon);
      y_mod[i] = y[i] + numy + noise[count+1] *
pow(attenuation_factor,time)*sqrt(1/l-epsilon);
      count += 2;
    }
  }
  fluctuation = 1.0 + scale * pow(attenuation_factor,time);
  phi = angle * pow(attenuation_factor,time);
  for (i=1;i<=number_point[d];i++) {
    if (x_mod[i]!=eos) {
      x_mod[i] = fluctuation * x_mod[i];
      y_mod[i] = fluctuation * y_mod[i];
      x_mod[i] = x_mod[i] * cos(phi) + y_mod[i] * sin(phi);
      y_mod[i] = -x_mod[i] * sin(phi) + y_mod[i] * cos(phi);
    }
  }
  num = 0;
  xm = 0;
  ym = 0;
  for(i=1;i<=number_point[d];i++) {
    if (x_mod[i] != eos) {
      num +=1;
      xm += x_mod[i];
      ym += y_mod[i];
    }
  }
  xm /= num;
  ym /= num;
  for(i=1;i<=number_point[d];i++) {
    if (x_mod[i] != eos) {
      x_mod[i] = x_mod[i] -xm + framex/2;
      y_mod[i] = y_mod[i] -ym + framey/2;
    }
  }
}


/* Returns the number of points 'n' of the following segment */
int nb_points(int break_point, double x_mod[])
```

```
{
  int n = 0;
  while (x_mod[n+1+break_point] != eos)
    n++;
  return(n);
}


/* Initialisation of the variables x_seg[], y_seg[] and t[] */
void init_segment(int n, int break_point, double x_mod[], double
y_mod[], double x_seg[], double y_seg[], double t[])
{
  int i;
  for (i=1;i<=n;i++) {
    x_seg[i] = x_mod[i+break_point];
    y_seg[i] = y_mod[i+break_point];
  }
  /* Initialisation of the parameter t */
  t[1] = 0;
  for (i=1;i<n;i++)
    t[i+1] = t[i] + dist(x_seg[i],y_seg[i],x_seg[i+1],y_seg[i+1]);
}


/* plot the point (x,y) into the frame */
void plot_point(int x, int y, double frame[])
{
  if ((x>=0)&&(x<framex)&&(y>=0)&&(y<framey))
    frame[y*framex+x] = 1;
}


/* plot the line between (x1,y1) and (x2,y2) into the frame */
void plot_line(double x1, double y1, double x2, double y2, double frame[])
{
  double p;
  double length;
  double x_direct,y_direct,norm;
  length = dist(x1,y1,x2,y2);
  x_direct = x2-x1;
  y_direct = y2-y1;
  norm = sqrt(pow(x_direct,2)+pow(y_direct,2));
  x_direct /= norm;
  y_direct /= norm;
  for (p=0;p<=length;p+=step)
    plot_point(floor(x1+p*x_direct),floor(y1+p*y_direct),frame);
}
```

```
void generate_curve(int time, double t[], double x[], double y[],
double y2_0[], double y2_1[], int n, double frame[])
{
  double p;
  double x1,y1,x_old,y_old,x_norm,y_norm,norm;
  int x2,y2;
  double linewidth;
  linewidth = width + width_factor * pow(attenuation_factor,time);
  for (p=0;p<=t[n];p+=step) {
    splint(t,x,y2_0,n,p,&x1);
    splint(t,y,y2_1,n,p,&y1);
    if (p>0) {
      x_norm = -(y1 - y_old);
      y_norm = x1 - x_old;
      norm = sqrt(pow(x_norm,2)+pow(y_norm,2));
      x_norm /= norm;
      y_norm /= norm;
      plot_line(x1-(linewidth/2)*x_norm,y1-(linewidth/2)*y_norm,
x1+(linewidth/2)*x_norm,y1+(linewidth/2)*y_norm,frame);
    }
    x_old = x1;
    y_old = y1;
  }
}


main(int time, double frame[])
{
  int break_point = 0;
  int n;
  int i;
  double t[nb_max_points+1], x[nb_max_points+1], y[nb_max_points+1];
  double x_mod[nb_max_points+1], y_mod[nb_max_points+1];
  double x_seg[nb_max_points+1], y_seg[nb_max_points+1];
  double y2_0[nb_max_points+1], y2_1[nb_max_points+1];
  double yp1 = 1e30;
  double yp2 = 1e30;

/* Initialisation of the frame and array digit */
  init_points(frame);
/* Initialisation of x[] and y[] */
  init_x_y(d, x, y);
/* Addition of the noise */
  add_noise(time, d, x, y, x_mod, y_mod);
```

```
/* At each pass through this loop, one segment of the digit is drawn */
  do {
/* Gives the number of control points of the following segment */
    n = nb_points(break_point, x_mod);
/* Initialisation of the points for the current segment */
    init_segment(n, break_point, x_mod, y_mod, x_seg, y_seg, t);
/* Generation of the two splines for the segment */
    spline(t,x_seg,n,yp1,yp2,y2_0);
    spline(t,y_seg,n,yp1,yp2,y2_1);
/* Generation of the digit */
    generate_curve(time,t,x_seg,y_seg,y2_0,y2_1,n,frame);
    break_point += n+1;
  }
  while (break_point != number_point[d]);
}


void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray
*prhs[])
{
  int i,j;
  double *frame;
  int time;

  init_rand();
  time = (int) mxGetScalar(prhs[1]);
  if (time==0) {
    d = (int) mxGetScalar(prhs[0]);
    if ((d<0)||(d>=10))
/* We generate a random integer in the range 0-9 if the value of the
   input 'd' is not an integer lying between 0 and 9 */
      d = rand()%(10);
/* Initialisation of the vector noise[] */
    init_noise();
  }
  else {
    frame = mxCalloc(framex*framey,sizeof(double));
    plhs[0] = mxCreateDoubleMatrix(framex,framey,0);
    frame = mxGetPr(plhs[0]);
    main(time,frame);
  }
}
```

B.2 Creation of the gray-scale

This procedure computes the elements of the gray-scale matrix, given the 140 by 160 element matrix calculated thanks to the procedure presented in the above paragraph.

```
/* Input : 140 by 160 matrix containing the image of the digit */
/* Output : 14 by 16 element matrix containing the gray-scale of the
            input frame */


#include "include_file.h"

/* Returns the value of the element (i,j) of the frame */
double frame_value(double frame[], int i, int j)
{
  double value = 0;
  if ((i>=0)&&(i<framex)&&(j>=0)&&(j<framey))
    value = frame[j*framex+i];
  return(value);
}


main(double frame2[], double frame[])
{
  double N = 0;
  double M;
  double XS = 0;
  double YS = 0;
  double phis = 0;
  double ds   = 0;
  int i,j,k,l;
  double gX,gY,X,Y,phi,lambda;

/* Computation of the center of mass (XS,XY) */
  for (i=0;i<framex;i++) {
    for (j=0;j<framey;j++) if (frame[j*framex+i] != 0) {
      N++;
      XS += (i+1);
      YS += (j+1);
    }
  }
  XS /= N;
  YS /= N;

/* Computation of the average skew angle phi */
```

```
  for (i=0;i<framex;i++) {
    for (j=0;j<framey;j++) if (frame[j*framex+i] != 0) {
      gX = (i+1)-XS;
      gY = (j+1)-YS;
      ds += sqrt(gX*gX+gY*gY);
      phi = atan2(gY,gX);
      if (phi < 0) phi += 2*3.14159;
      phis += phi;
    }
  }
  ds /= N;
  phis /= N;
  phi = phis - 3.14159;

/* Suppression of the skew and normalisation of the size */
  lambda = 35/ds;
  for (i=0;i<framex_scale;i++)
    for (j=0;j<framey_scale;j++)
      frame2[j*framex_scale+i] = 0;
  for (i=0;i<framex;i++) {
    for (j=0;j<framey;j++) if (frame[j*framex+i] != 0) {
      X =  ((i+1)-XS)* lambda;
      Y =  ((j+1)-YS)* lambda;
      gX =  X*cos(phi) + Y*sin(phi);
      gY = -X*sin(phi) + Y*cos(phi);
      gX = (gX*framex_scale)/framex;
      gY = (gY*framey_scale)/framey;
      k  = floor( gX + 0.5* framex_scale +0.0);
      l  = floor( gY + 0.5* framey_scale +0.0);
      if ( (0 <= k) && (k < framex_scale) &&
           (0 <= l) && (l < framey_scale)    )
        frame2[l*framex_scale+k] += lambda;
    }
  }
  for (i=0;i<framex_scale;i++)
    for (j=0;j<framey_scale;j++)
      frame2[j*framex_scale+i] = tanh( frame2[j*framex_scale+i]/3 -6);
}


void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray
*prhs[])
{
  plhs[0] = mxCreateDoubleMatrix(framex_scale,framey_scale,0);
  main(mxGetPr(prhs[0]),mxGetPr(plhs[0]));
}
```

Bibliography

Bishop, C. M. (1995). *Neural Networks for Pattern Recognition.* Oxford University Press.

Burr, D. (1981, November). Elastic matching of line drawings. *IEEE Transactions on Pattern Analysis and Machine Intelligence 3*, 708–713.

Hinton, G., C. Williams, and M. Revow (1992). *Combining two methods of recognizing hand-printed digits*, pp. 53–60. Elsevier.

Impedovo, S. (1994). *Fundamentals in handwriting recognition.* Springler-Verlag.

Lam, L. and C. Y. Suen (1988). Structural classification and relaxation matching of totally unconstrained handwritten zip-code numbers. *Pattern Recognition* ***21*** (1), 19–31.

Lee, Y. (1991). Handwritten digit recognition using k nearest-neighbor, radial-basis function, and backpropagation neural networks. *Neural Computation* ***3***(6), 440–449.

Mokhtari, M. and A. Mesbah (1997). *Apprendre et maitriser MATLAB*, pp. 286–298. Springer-Verlag.

Press, W. H., S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery (1992). *Numerical Recipes in C* (Second ed.). Cambridge University Press.

Revow, M., C. K. I. Williams, and G. E. Hinton (1996). Using generative models for handwritten digit recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence* ***18***(6), 592–606.

Suen, C. Y., C. Nadal, R. Legault, T. A. Mai, and L. Lam (1992, July). Computer recognition of unconstrained handwritten numerals. *Proceedings of the IEEE* ***80***(7), 1162–1180.

Urbanczik, R. (1991). Learning temporal structure by continuous backpropagation. *Second International Conference on Artificial Neural Networks*, 124–128.