Examensarbete

Analyses and Tests of Handwritten Digit Recognition Algorithms

Examensarbete utfört i Numerisk Analys
vid Tekniska Högskolan i Linköping
av

Berkant Savas

Reg nr: LiTH–MAT–EX–2003–01
Linköping 2003

Examensarbete

Analyses and Tests of Handwritten Digit Recognition Algorithms

Examensarbete utfört i Numerisk Analys
vid Tekniska Högskolan i Linköping
av

Berkant Savas

Reg nr: LiTH–MAT–EX–2003–01

Supervisor: **Lars Eldén**

Examiner: **Lars Eldén**

Linköping 4th February 2003.

Abstract

This report is a masters thesis written at the Department of Mathematics, Linköping University. Two different classification algorithms for handwritten digit recognition have been thoroughly analysed. The first algorithm uses Higher Order Singular Value Decomposition (HOSVD) of the training digits. The second algorithm relies on a specific distance measure, which is invariant to different transformations, called Tangent Distance (TD). This algorithm was modified in the implementation part by the use of numerical derivatives and an approximation of the blurring operator. Two more classification algorithms were constructed by combining the first two algorithms. All constructed algorithms have been tested with good performance for some of them. The best results were achieved by the Tangent Distance classifier with an error rate of 3 %. Finally the results of a few other classifiers are presented and compared with the test results obtained in this report.

Keywords: Handwritten digit recogintion, digit recognition, classification algorithms, tensor SVD, tangent distance, data mining.

Acknowledgments

First of all I would like to thank my supervisor, Professor Lars Eldén, at the Division of Numerical Analysis for his excellent guidance and for the many interesting and insightful discussions we have had. I am also grateful for his thorough and constructive feedback on this report. Furthermore I am thankful for the kind financial support I have received during this work.

I would also like to thank my opponent, Mattias Olsson, and Markus Johansson for their careful reading of this report and for giving me valuable comments and linguistic feedback.

Notation

This page contains the most common symbols, operators and abbreviations used in this report.

Symbols

a,b,α	Lower–case letters are used for scalars
$\mathbf{a}$,$\mathbf{b}$	Bold lower–case letters are used for vectors
A, B, $\mathbf{A}$, $\mathbf{B}$	Capitals and bold capitals denote matrices
$\mathcal{A}$, $\mathcal{B}$	Calligraphic letters denote tensors of at least third order
$\mathbb{R}$, $\mathbb{C}$	The set of reel and complex numbers

Operators and functions

$A \cdot B$	Indicates the usual matrix product
$\mathcal{A} \times_n B$	n–mode product of a tensor by a matrix
$\subset$	Subset
$\mathcal{O}(n)$	Function with the property that $\mathcal{O}(n)/n$ is bounded as $n \to \infty$

Abbreviations

SVD	Singular Value Decomposition
HOSVD	Higher Order Singular Value Decomposition
RI	Right Identifications
WI	Wrong Identifications
NBM	Number of Basis Matrices
TD	Tangent Distance
USPS	United States Postal Service (database)
MNIST	Modified National Institute of Standards and Technology (database)

Contents

Chapter 1

Introduction

The pattern recognition ability, like visually recognizing objects, understanding spoken words, discriminating things by feeling or smelling them, is very highly evolved among humans. Pattern recognition might be described as the process of making a decision based on data input. This is certainly a very huge domain with many practical applications in many fields of science. Having reliable, accurate and automated recognition devices would clearly be very useful. There are already several areas, with more to come, where pattern recognition is used, even in a commercial way. Some examples for that are: fingerprint recognition, handwritten character recognition, speech recognition, DNA sequence recognition, gas recognition, face recognition and much more.

This report is a masters thesis with the main purpose to analyse, develop and test a few classification algorithms used for identification of handwritten characters, especially digits.

1.1 The classification problem

The character classification problem is considered to be very fundamental in this context and many scientists have spent a lot of time in the past decades trying to solve the problem. No unified answer is presented. Instead there are many classification algorithms performing more or less well in specific areas.

There are several ways of dividing the problem into principally different categories. Then it is only natural that the most suitable methods to solve the problem in various cases might differ. Many algorithms are derived based on different mathematical theories. Linear algebra, functional analysis and statistics are just a few of them.

On–line and off–line classification

One way of distinguishing between the algorithms is to look at how they are used in the final application. If the algorithm is used in real–time it is often referred to be an on–line classification algorithm. If the classification objects are stored and the classification is performed at some other time the algorithm is referred to be an off–line classification algorithm. Besides these algorithms might have an incorporated time aspect attached to the data input. On–line algorithms, with attached time aspect or not, are more sensitive to the required amount of computation, but even so there are cases with acceptable results. In general, off–line classification without incorporated time aspect is a harder problem than on–line classification with time aspect [1]. An intuitive reason is that less data is used in the first case compared to the second case. Character recognition on scanned documents is most often done off–line and there are algorithms with good performance for this task. Systems in personal digital assistants (PDAs), on the other hand, are implementing on–line versions of classification algorithms with incorporated time aspect [1], [10].

Pixel images and curves

Another way of distinguishing between algorithms is to look at the way the characters are represented. For instance when we write with a pen, a shape symbol is imprinted on the paper. This shape can be represented in at least two ways - as a pixel image or as curves in a plane. Thus, the structure of the input data has a major impact on the way an algorithm is built. One might favour the curve representation in this case because the original shapes, which we want to classify, are curves and are written in a curve motion by the hand and not as dots or pixels with various intensities. Online algorithms are more suited for the curve representation.

In this report we only consider pixel images representing digits in offline classification algorithms without incorporated time aspect.

1.2 Outline of the report

The rest of the report is divided in the following chapters.

Chapter 2: The first algorithm that was analysed and tested is based on tensors and singular value decomposition of third order tensors. Initially a brief theory on tensors is presented with the main results and theorems that are required in the algorithm construction. Parts of this theory are then implemented and detailed description of this is given. Many of the conducted tests are thoroughly described in a separate section. The last part of this chapter is aimed for further analysis and useful comments to get a deeper insight of the algorithm.

Chapter 3: The second algorithm described incorporates a rather new kind of distance measure called tangent distance [14]. The theoretical background of this measure is presented. Some modifications in the implementation part are made with the purpose to get a more computationally efficient algorithm. A very simple algorithm on bases of this tangent distance is then constructed and the implementation is given in detail. The results of the conducted tests are presented and the chapter ends with some discussion.

Chapter 4: In chapter four the theory for the preceding algorithms are combined to build at least two new classification algorithms. The idea of combining these methods was suggested by my supervisor *Lars Eldén*. Advantages and disadvantages are pointed out for the different combinations. One of the new algorithms is performing rather well according to test results.

Chapter 5: The last chapter is a brief reflection on some other classification algorithms that are usual in the field. Their performance is presented with the intention to be compared with results achieved in this report. Finally a short summary of the whole report is given.

Appendix: Proofs, definitions, information about the data sets and other details of interest are gathered in the appendix.

Chapter 2

Classification by Singular Value Decomposition of Tensors

2.1 Tensor Theory

Mathematical structures or quantities as vectors and matrices are sometimes not suitable for describing the true nature of data from various fields of signal processing. Those structures can easily be extended to more general higher–order structures called multimode arrays, multidimensional matrices or, as they will be called in this report, simply as tensors. Vectors and matrices are equivalent to first and second order tensors, respectively. The constructed algorithm relies mainly on the special case of this theory for third order tensors. Therefore most of the presented theory in this section is for third order tensors. The generalization of some parts of the theory presented here is placed in Appendix A.

2.1.1 Basic definitions and properties

Different kinds of quantities will be represented in the following manner: scalars are written as lower–case letters $(a, b, \dots; \alpha, \beta, \dots)$, vectors (first order tensors) are written as bold lower–case letters $(\mathbf{a}, \mathbf{b}, \dots)$, matrices (second order tensors) are denoted as bold–face capitals $(\mathbf{A}, \mathbf{B}, \dots)$ and higher order tensors as calligraphic letters $(\mathcal{A}, \mathcal{B}, \dots)$.

Tensor representation

It is easy to define a general $(I_1 \times I_2 \times \cdots \times I_N)$–tensor but hard to visualize in its natural shape (all cases where $N \geq 3$). The tensor dimensions I_i are called modes in this field. Each Nth order tensor $\mathcal{A} \in \mathbb{C}^{I_1 \times I_2 \times \cdots \times I_N}$ can be unfolded to a matrix along each of its modes. Unfolding a tensor along its *nth* mode yields a matrix $\mathbf{A}_{(n)} \in \mathbb{C}^{I_n \times (I_{n+1} \dots I_N I_1 \dots I_{n-1})}$ where each element of $\mathcal{A}$ gets its unique place in $\mathbf{A}_{(n)}$. Folding an unfolded tensor will of course give the original tensor without changing anything.

Unfolding a third order tensor

Any third order tensor $\mathcal{A}$ can be represented in three different ways. Consider the case where $\mathcal{A}$ is an $(n_1 \times n_2 \times n_3)$ tensor with real entries. The tensor is illustrated in figure 2.1.

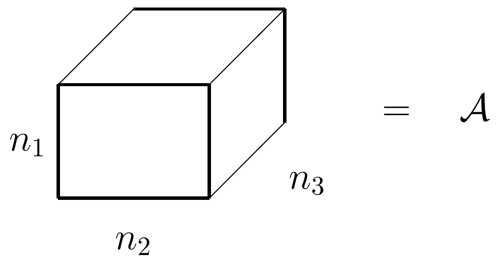

Figure 2.1. Illustration of a third order tensor.

This is obviously a block tensor in which the number of rows is given by n_1, the number of columns by n_2 and n_3 is the number of slices in the third mode. The word slice refers to the matrices, that are obtained when the index of the third mode is kept fixt. The unfolding process gives the following matrices:

$$\begin{aligned}
\mathbf{A}_{(1)} &:= (\mathcal{A}(:,1,:) \quad \mathcal{A}(:,2,:) \quad \dots \quad \mathcal{A}(:,n_2,:)) \in \mathbb{R}^{n_1 \times n_2 n_3}, \\
\mathbf{A}_{(2)} &:= (\mathcal{A}(:,:,1)^T \quad \mathcal{A}(:,:,2)^T \quad \dots \quad \mathcal{A}(:,:,n_3)^T) \in \mathbb{R}^{n_2 \times n_1 n_3}, \\
\mathbf{A}_{(3)} &:= (\mathcal{A}(1,:,:)^T \quad \mathcal{A}(2,:,:)^T \quad \dots \quad \mathcal{A}(n_1,:,:)^T) \in \mathbb{R}^{n_3 \times n_1 n_2}.
\end{aligned}$$

$\mathcal{A}(:,1,:)$ is interpreted as a part of the tensor where the first index in the second mode is kept fixt. This is a matrix consisting of the first columns in every slice. Equivalently $\mathcal{A}(:,:,2)$ is the whole second slice in the tensor and the columns in $\mathcal{A}(n,:,:)$ are the *nth* row vectors in each slice. These unfoldings are written in pseudo–Matlab notation. It should be noticed that fixing one index in any mode yields a matrix because the other indexes of the other two modes are free to vary. The unfoldings are obtained by putting the matrices next to each other as shown above.

The precise definition of the unfolding operation for tensors in general is given in A.1.

Rank properties

There is no unique way of generalising the rank concept for higher order tensors from the definitions of rank for matrices. One way is to consider the usual rank for matrices and define the n–rank of a tensor as follows.

Definition 2.1 (**n–rank**) *The n–rank R_n of a tensor $\mathcal{A}$ is the dimension of the vector space spanned by the n–mode vectors i.e.*

$$R_n = rank_n(\mathcal{A}) = rank(\mathbf{A}_{(n)}), \tag{2.1}$$

where $\mathbf{A}_{(n)}$ is the unfolding of $\mathcal{A}$ along the nth mode.

It is easy to verify that different n–ranks of a higher order tensor are not necessarily equal as is the case considering matrices.

There is a second way of defining the rank of a tensor [2], but the definition will be omitted in this context.

Scalar product, orthogonality and norm

The scalar product for vectors is generalized to include tensors of any order in a straightforward way.

Definition 2.2 *The scalar product $\langle \mathcal{A}, \mathcal{B} \rangle$ of two tensors $\mathcal{A}, \mathcal{B} \in \mathbb{C}^{I_1 \times I_2 \times \cdots \times I_N}$ is defined as*

$$\langle \mathcal{A}, \mathcal{B} \rangle = \sum_{i_1} \sum_{i_2} \cdots \sum_{i_N} b^*_{i_1 i_2 \ldots i_N} a_{i_1 i_2 \ldots i_N}, \tag{2.2}$$

in which the $*$ denotes complex conjugation.

Notice that this is completely equivalent to the scalar product for vectors. The definitions for orthogonality and norm of tensors are as follows.

Definition 2.3 *Two tensors $\mathcal{A}$ and $\mathcal{B}$ are said to be orthogonal if their scalar product is equal to zero, $\langle \mathcal{A}, \mathcal{B} \rangle = 0$.*

Definition 2.4 *The Frobenius norm of a tensor $\mathcal{A}$ is defined as*

$$\|\mathcal{A}\|_F = \sqrt{\langle \mathcal{A}, \mathcal{A} \rangle}. \tag{2.3}$$

2.1.2 Multiplication of a tensor by a matrix

When we look at the matrix case when deriving the singular value decomposition[1] for matrices one lets orthogonal coordinate transformations induce a special representation of the matrix. These transformations correspond to multiplications of the original matrix by matrices from left and from right. The same idea for the tensor case results in a product definition between a tensor and a matrix. The multiplications from left and right are generalized to multiplications along each mode of the tensor. It is easy to realize that the tensor and the matrix have to be compatible in some way. The definition for third order tensors is as follows.

Definition 2.5 *The n–mode product of a third order tensor $\mathcal{A} \in \mathbb{C}^{I_1 \times I_2 \times I_3}$ by a matrix $\mathbf{F} \in \mathbb{C}^{J \times I_n}$, denoted by $\mathcal{A} \times_n \mathbf{F}$, is a $(J \times I_2 \times I_3)$–tensor when $n = 1$, $(I_1 \times J \times I_3)$–tensor when $n = 2$ and $(I_1 \times I_2 \times J)$–tensor when $n = 3$. Letting $\mathcal{B} = \mathcal{A} \times_n \mathbf{F}$, the entries of $\mathcal{B}$ are given by folding $\mathbf{B}_n$, where $\mathbf{B}_{(n)} = \mathbf{F} \cdot \mathbf{A}_{(n)}$.*

[1]The singular value decomposition will be explained in the next section.

Some explanation of the above definition might be insightful. Consider a tensor matrix multiplication along the first mode, i.e. $\mathcal{B} = \mathcal{A} \times_1 \mathbf{F}$. Then $\mathbf{F}$ is a $(J \times I_1)$–matrix and $\mathbf{A}_{(1)}$, which is the unfolding of $\mathcal{A}$ along the first mode, is an $(I_1 \times I_2 I_3)$–matrix. The compatibility of the matrices $\mathbf{F}$ and $\mathbf{A}_{(n)}$ is clearly seen. This means that $\mathbf{B}_{(n)}$ is a $(J \times I_2 I_3)$–matrix and folding it along the first mode we get the desired $\mathcal{B}$ which is a $(J \times I_2 \times I_3)$–tensor. It should be noted that an arbitrary matrix can be folded in several ways. Knowing the final size of the n–mode product and folding the matrix to a tensor of this particular size will give the right tensor.

The definition of the n–mode product for arbitrary tensors is a straightforward generalization and is given in Appendix A.

Given the tensor $\mathcal{A} \in \mathbb{C}^{I_1 \times I_2 \times I_3}$ and the matrices $\mathbf{F} \in \mathbb{C}^{J_n \times I_n}$, $\mathbf{G} \in \mathbb{C}^{J_m \times I_m}$ and $\mathbf{H} \in \mathbb{C}^{K_n \times J_n}$ the n–mode product satisfies the following properties,

$$(\mathcal{A} \times_n \mathbf{F}) \times_m \mathbf{G} = (\mathcal{A} \times_m \mathbf{G}) \times_n \mathbf{F} = \mathcal{A} \times_n \mathbf{F} \times_m \mathbf{G}, \tag{2.4}$$

$$(\mathcal{A} \times_n \mathbf{F}) \times_n \mathbf{H} = \mathcal{A} \times_n (\mathbf{H} \cdot \mathbf{F}). \tag{2.5}$$

These equations are also valid when $\mathcal{A}$ is an arbitrary $(I_1 \times I_2 \times \cdots \times I_N)$–tensor, without any modifications [2].

2.1.3 SVD of a tensor

One of the most important results in linear algebra is the Singular Value Decomposition (SVD) theorem. The theorem is presented below for matrices and then a generalization is made for tensors.

Theorem 2.1 (Matrix SVD) *Every complex matrix $\mathbf{F} \in \mathbb{C}^{m \times n}$ can be written as the product*

$$\mathbf{F} = \mathbf{U} \cdot \Sigma \cdot \mathbf{V}^H, \tag{2.6}$$

with the following properties:

1. *$\mathbf{U}$ is an $(m \times m)$ unitary matrix.*
2. *$\mathbf{V}$ is an $(n \times n)$ unitary matrix.*
3. *Σ is an $(m \times n)$ diagonal matrix with real and non–negative entries, ordered in the followint way:*

$$\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_{min(m,n)} \geq 0. \tag{2.7}$$

The columns of $\mathbf{U}$ and $\mathbf{V}$ are called the left and right singular vectors respectively and the σ_i are the singular values.

A proof of the SVD theorem can be found in Golub [6]. The H in equation (2.6) indicates transposition and complex conjugation.

Just to mention an example of the usefulness of the decomposition, consider the minimization problem

$$\min_{\mathbf{X}} \|\mathbf{F} - \mathbf{X}\|_2,$$

with $rank(\mathbf{X}) = r$ and $r < n = rank(\mathbf{F})$. The solution to this problem gives the best[2] rank r matrix $\mathbf{X}$ that approximates a given matrix $\mathbf{F}$. SVD gives the solution, namely the product $\mathbf{X} = \mathbf{U} \cdot \widetilde{\Sigma} \cdot \mathbf{V}^H$ in which $\widetilde{\Sigma}$ has only the first r singular values along the diagonal from the original Σ.

Considering the matrix as a second order tensor it is possible to express the SVD in terms of the n–mode product.

$$\mathbf{F} = \Sigma \times_1 \mathbf{U} \times_2 \mathbf{V}. \tag{2.8}$$

For higher order tensors very similar results are obtained. Those are presented in the theorem below.

Theorem 2.2 (HOSVD) *An arbitrary tensor $\mathcal{A} \in \mathbb{C}^{I_1 \times I_2 \times \cdots \times I_N}$ can be written as the product*

$$\mathcal{A} = \mathcal{S} \times_1 \mathbf{U}^{(1)} \times_2 \mathbf{U}^{(2)} \times \ldots \times_N \mathbf{U}^{(N)}, \tag{2.9}$$

with the following properties:

1. *$\mathbf{U}^{(n)}$ are unitary $(I_n \times I_n)$–matrices.*
2. *$\mathcal{S}$ is a complex tensor of the same dimensions as $\mathcal{A}$, where the subtensors $\mathcal{S}_{i_n=\alpha}$, obtained by fixing any mode–index n, fulfil two conditions:*
   - *all–orthogonality, meaning that two different subtensors fixed in the same mode are orthogonal,*

     $$\langle \mathcal{S}_{i_n=\alpha}, \mathcal{S}_{i_n=\beta} \rangle = 0 \qquad \textit{when} \qquad \alpha \neq \beta, \tag{2.10}$$

   - *and ordering: the norms of the subtensors satisfy*

     $$\|\mathcal{S}_{i_n=1}\| \geq \|\mathcal{S}_{i_n=2}\| \geq \cdots \geq \|\mathcal{S}_{i_n=I_n}\| \geq 0. \tag{2.11}$$

   The two conditions are valid for all possible n.

Proof of theorem 2.2 is given in Appendix A.

An example might be in place to give some insights of the all–orthogonality condition and the subtensors. Consider a matrix as a tensor. The subtensors are simply the columns and the row vectors. If the first mode is fixed then the corresponding subtensors are the row vectors and fixing the second mode will give the columns as subtensors. Putting $\mathbf{F} = \mathbf{U} \cdot \mathbf{S} \cdot \mathbf{V}^H$, it is clear that the columns of $\mathbf{S}$ are orthogonal. The same is true for the row vectors. $\mathbf{S}$ obviously is all–orthogonal, but we already knew that. Another important fact about the HOSVD is that there is no diagonal structure in $\mathcal{S}$ as in $\mathbf{\Sigma}$ in the matrix SVD. All the entries of $\mathcal{S}$ might have values different from zero.

The Frobenius norms in (2.11) are in fact the singular values, $\sigma_i^{(n)}$, of the $\mathbf{A}_{(n)}$. Even the $\mathbf{U}^{(n)}$ belong to the SVD of $\mathbf{A}_{(n)}$. With this knowledge it is straightforward

[2]Best in a least square and Frobenius norm sense.

to compute the whole Higher Order Singular Value Decomposition – HOSVD. Produce the unfolding of $\mathcal{A}$ along every mode, compute the $\mathbf{U}^{(n)}$ matrices for all of them by SVD and finally multiply $\mathcal{A}$ with ${\mathbf{U}^{(n)}}^H$ along the *nth* mode to get $\mathcal{S}$. An illustration is given below

$$\mathcal{S} = \mathcal{A} \times_1 {\mathbf{U}^{(1)}}^H \times_2 {\mathbf{U}^{(2)}}^H \cdots \times_N {\mathbf{U}^{(N)}}^H. \tag{2.12}$$

More detailed information about tensors is found in [2].

2.1.4 Matlab implementation of Higher Order SVD

Matlab can handle arrays with more than two indexes but unfortunately there are no functions implementing the tensor theory described above. The following functions were constructed to implement the theory. These functions work in the case of third order tensors only, but can easily be generalized.

The unfold function

According to Theorem 2.2 all three unfoldings are needed in the computation of a complete third order singular value decomposition. More specifically, the unfolding along each mode is needed when computing the unitary matrices $\mathbf{U}^{(n)}$. Once they are computed the all–orthogonal core tensor $\mathcal{S}$ can be obtained simply by tensor multiplications.

The unfolding process was implemented in a Matlab function taking two arguments

$$\mathbf{A}_{(n)} = \text{unfold}(\mathcal{A}, n) \quad , \quad n = 1,\, 2 \text{ or } 3. \tag{2.13}$$

This function does exactly the same thing as described previously in section 2.1.1.

Tensor matrix multiplication function

The second function constructed was the n–mode product for tensors. The definition gave this function with three arguments.

$$\mathcal{B} = \mathcal{A} \times_n \mathbf{U} = \text{tmul}(\mathcal{A}, \mathbf{U}, n) \quad , \quad n = 1,\, 2 \text{ or } 3. \tag{2.14}$$

The multiplication is implemented by computing $\mathbf{A}_{(n)}$ with the unfolding function, performing the matrix multiplication $\mathbf{U} \cdot \mathbf{A}_{(n)}$ and finally folding it back in the proper way into $\mathcal{B}$.

HOSVD function

Now we have all the functions needed to compute the whole decomposition. In the case of third order tensors it is given by

$$\mathcal{A} = \mathcal{S} \times_1 \mathbf{U}^{(1)} \times_2 \mathbf{U}^{(2)} \times_3 \mathbf{U}^{(3)}. \tag{2.15}$$

A straightforward implementation of Theorem 2.2 is:

1. Compute all the unfoldings of $\mathcal{A}$.
2. Compute the matrix SVD of the $\mathbf{A}_{(n)}$.
3. Multiply the unitary matrices $\mathbf{U}^{(n)^T}$ into $\mathcal{A}$ along respective mode with the newly built product function to obtain $\mathcal{S}$.

Notice that the actions in the third point are illustrated below. This is the same operation as in equation (2.12) but for the special case of a third order tensor.

$$\mathcal{S} = \mathcal{A} \times_1 \mathbf{U}^{(1)^T} \times_2 \mathbf{U}^{(2)^T} \times_3 \mathbf{U}^{(3)^T}. \tag{2.16}$$

These operations were put together in a function called svd3 to compute the whole HOSVD for third order tensors. The function is given below in pseudo–Matlab notation.

$$\begin{aligned}
[\mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \mathbf{U}^{(3)}, \mathcal{S}, \Sigma_1, \Sigma_2, \Sigma_3] &= \text{svd3}(\mathcal{A}), \qquad (2.17)\\
[\mathbf{U}^{(1)}, \Sigma_1, v] &= \text{svd}(\text{unfold}(\mathcal{A}, 1)),\\
[\mathbf{U}^{(2)}, \Sigma_2, v] &= \text{svd}(\text{unfold}(\mathcal{A}, 2)),\\
[\mathbf{U}^{(3)}, \Sigma_3, v] &= \text{svd}(\text{unfold}(\mathcal{A}, 3)),\\
\mathcal{S} &= \text{tmul}(\text{tmul}(\text{tmul}(\mathcal{A}, \mathbf{U}^{(1)^T}, 1), \mathbf{U}^{(2)^T}, 2), \mathbf{U}^{(3)^T}, 3).
\end{aligned}$$

Notice that the $\mathbf{V}$ matrices in the SVD are not required in the computation of HOSVD and therefore not needed to be computed. The general SVD algorithm can be customized in this application to speed up and reduce the amount of calculations. The n–mode singular values stored in the diagonals of Σ_1, Σ_2 and Σ_3 are not in the HOSVD theorem but it turns out that a closer investigation of them gives us some useful and interesting information.

2.1.5 Orthogonal basis matrices

A matrix $\mathbf{F}$ can be written as a sum of rank 1 matrices

$$\mathbf{F} = \mathbf{U} \cdot \Sigma \cdot \mathbf{V}^T = \sum_1^n \sigma_i \mathbf{u}_i \mathbf{v}_i^T, \tag{2.18}$$

where σ_i are the singular values and the $\mathbf{u}_i$ and $\mathbf{v}_i$ are the columns of $\mathbf{U}$ and $\mathbf{V}$ respectively. The sum is illustrated in figure 2.2. Notice that the outer product is taken between the $\mathbf{u}_i$ and $\mathbf{v}_i$.

A similar decomposition can be achieved also for arbitrary third order tensors.

$$\mathcal{A} = \sum_{i=1}^{n_3} \mathbf{A}_i \times_3 \mathbf{u}_i^{(3)} \quad \text{in which} \quad \mathbf{A}_i = \mathbf{S}(:,:,i) \times_1 \mathbf{U}^{(1)} \times_2 \mathbf{U}^{(2)} \quad \text{are orthogonal.} \tag{2.19}$$

It follows from (2.19) that every tensor $\mathcal{A}$ can be split up in a sum of tensors. An equivalent illustration for (2.19) as in figure 2.2 is given below. The squares

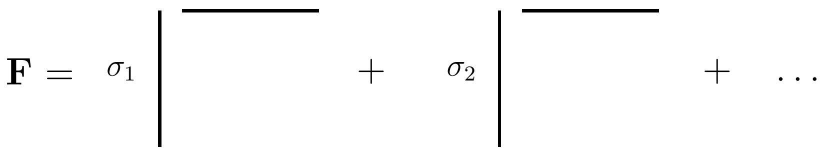

Figure 2.2. Illustration of the decomposition in (2.18).

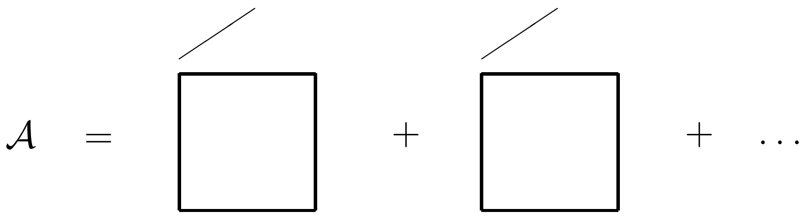

Figure 2.3. Illustration of the decomposition in (2.19).

represent the $\mathbf{A}_i$ and the tilted lines above them are the $\mathbf{u}_i^{(3)}$ vectors representing third mode multiplications yielding tensors.

An interesting fact is that each tensor in the sum consists of multiples of the same matrix $\mathbf{A}_i$. Consider the first element for example. $\mathbf{S}(:,:,1)$ is the first slice from the all–orthogonal block tensor and is a matrix. The first and second mode multiplications are the usual matrix multiplications from left and right, all according to (2.6) and (2.8), so $\mathbf{A}_i$ is also a matrix. A third mode multiplication remains to be carried out. $\mathbf{u}_1^{(3)}$ is the first column from the third unitary matrix $\mathbf{U}^{(3)}$. Consequently $\mathbf{u}_1^{(3)}$ has n_3 elements. This is the same number as the dimension along the third mode of $\mathcal{A}$ and it has to be that if the terms in the sum ought to have the same size as $\mathcal{A}$. Thus multiplying a matrix $\mathbf{A}_1$ with a vector $\mathbf{u}_1^{(3)}$ along the third mode yields a third order tensor in which the first slice is $\mathbf{A}_1$ multiplied with the first element of the vector $\mathbf{u}_1^{(3)}$ – $u_1^{(3)}(1)$, the second slice is $\mathbf{A}_1$ multiplied with the second element of $\mathbf{u}_1^{(3)}$ – $u_1^{(3)}(2)$, the third $\mathbf{A}_1$ multiplied with $u_1^{(3)}(3)$ and so on whole the way. All terms in the sum are computed in this way.

Remember that the different slices of $\mathcal{S}$ i.e. $\mathbf{S}(:,:,i)$ are orthogonal to each other, the all–orthogonality condition. A simple computation gives that the $\mathbf{A}_i$ are also orthogonal.

$$\langle \mathbf{A}_i, \mathbf{A}_j \rangle = \operatorname{tr}(\mathbf{A}_i^T \mathbf{A}_j) = \operatorname{tr}((\mathbf{U}^{(1)} \mathbf{S}(:,:,i) \mathbf{U}^{(2)^T})^T \cdot (\mathbf{U}^{(1)} \mathbf{S}(:,:,j) \mathbf{U}^{(2)^T})) =$$
$$= \operatorname{tr}(\mathbf{U}^{(2)} \mathbf{S}(:,:,i)^T \mathbf{U}^{(1)^T} \mathbf{U}^{(1)} \mathbf{S}(:,:,j) \mathbf{U}^{(2)^T}) =$$
$$= \operatorname{tr}(\mathbf{U}^{(2)} \mathbf{S}(:,:,i)^T \mathbf{S}(:,:,j) \mathbf{U}^{(2)^T}) = \operatorname{tr}(\mathbf{U}^{(2)} \cdot \mathbf{0} \cdot \mathbf{U}^{(2)^T}) = 0 \quad \text{when} \quad i \neq j.$$

These orthogonal matrices can be interpreted as a set of linearly independent basis matrices. The *tr* in the calculation above denotes *trace*.

2.2 The Data Sets

In all the tests carried out in this report two different data sets of handwritten digits were used. These sets are freely available on the Internet and are frequently used throughout the world for evaluation of classification algorithms. Thus there are extensive results and references to the data, making it easy to do an impartial comparison of the achieved performance of the algorithms.

2.2.1 U.S. Postal Service Database

The digits in this database are extracted by scanning the ZIP code on envelopes from U.S. Postal mail. The size of the images is 16×16 pixels and each pixel has an intensity range of 0 to 255. This set was downloaded from the homepage of Hastie [7].

Two sets are available, one training set containing 7291 digits and one test set containing 2007 digits. Of course the right answers to all of the pixel images are included. The digit distribution is given in table 2.1.

	0	1	2	3	4	5	6	7	8	9	Total
Train	1194	1005	731	658	652	556	664	645	542	644	7291
Test	359	264	198	166	200	160	170	147	166	177	2007

Table 2.1. The digit distribution in the U.S. Postal Service data sets.

Both of the sets were converted to Matlab data files and reshaped to third order tensors. To each tensor there is an associated vector in which the corresponding digits (the right answers) are stored. In detail we have the following data to our disposal:

1. One $(16 \times 16 \times 7291)$–tensor, which is the training set, and one $(16 \times 16 \times 2007)$–tensor, which is the test set.
2. Two vectors containing the right answers to each tensor.

Figure 2.4 gives some examples from the U.S. Postal Service database.

According to Hastie [7] this set is rather difficult from a classification point of view when compared to other data sets. For example the MNIST database, described in next section, is easier than the envelope numbers. This is also confirmed in the test results. Notice that the images in the figure above are quite well written, there are many others that are badly written.

2.2.2 Modified NIST Database

In the spring of 1992 the National Institute of Standards and Technology organized a classification competition for handwritten digits. To disposal there was a training set of 223,000 patterns and a test set of 59,000 patterns, that is the original NIST database. These two sets had different distributions and this affected the test

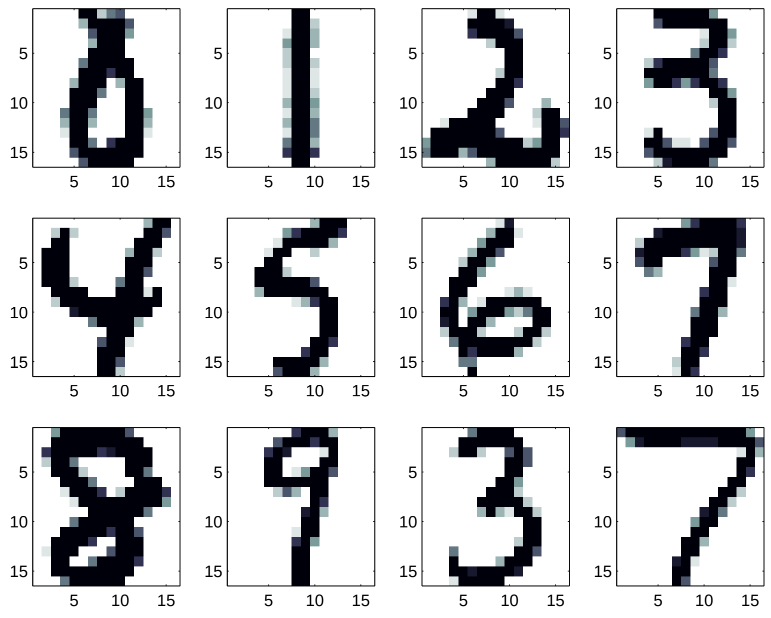

Figure 2.4. Examples from the U.S. Postal Service database.

results. A modified database was built by combining the two sets using a 50/50 ratio into a new training set with 60,000 patterns and 10,000 test patterns. The new database is called the Modified NIST or simply MNIST.

The digit distribution of the test set is given in the table below.

	0	1	2	3	4	5	6	7	8	9	Total
Test	980	1135	1032	1010	982	892	958	1028	974	1009	10000

Table 2.2. The digit distribution of the test set in the MNIST database.

The main difference between this set and the previous one is the size of the patterns. Digits in the MNIST database are slightly bigger and stored in images of 28 × 28 pixels, but the grey level in the pixels still have an intensity range of 0 to 255. Some examples from the test set are given in figure 2.5.

It should be pointed out that each pattern is embedded in an image of 28 × 28 pixels but the patterns do not fill the whole frame as the digits in the U.S. Postal Service database do.

The MNIST database was downloaded from the website located at

http://yann.lecun.com/exdb/mnist/.

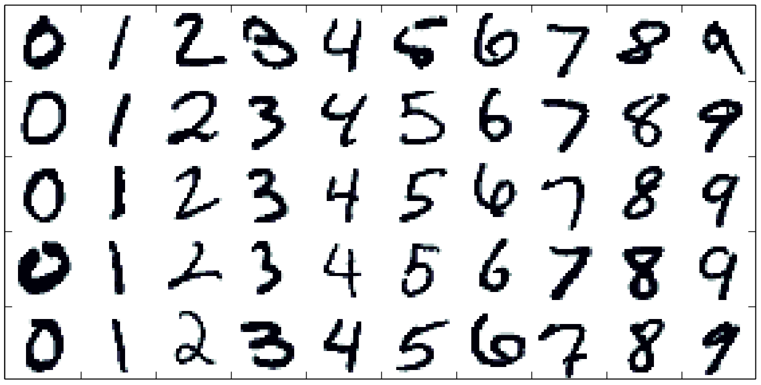

Figure 2.5. Digits from the MNIST database.

2.3 Algorithm 1

This part of the chapter is intended to describe in detail how the theory in the previous section is used to build a complete algorithm able to classify unknown handwritten digits. The main idea is to construct a set of orthogonal basis matrices for each class namely the digits 0, 1 up till 9 and then decide which of the bases sets describes the unknown digit in the best way. This is a very vague formulation but is still the basic thought behind the whole algorithm and everything will hopefully be clarified.

The database files downloaded from the Internet were converted so that every handwritten digit took the form of a matrix with pixel values as elements. These matrices were then put together to build a third order tensor. The different digit images are the different slices in this block tensor. Doing this we end up with two different third order tensors, one training set and one test set. The training set is used when constructing the algorithm, which is the training phase, and the test set is used in the evaluation phase.

2.3.1 Constructing a basis

Building an algorithm with the objective to discriminate an unknown digit to one of predefined classes, some kind of information must be incorporated in the algorithm about each class. This information is then used in the classification process. In our special case the algorithm must contain information about each kind of digit and this information is retrieved from the training set as mentioned earlier.

Computing the basis matrices

A set of basis matrices, as described in section 2.1.5, is constructed for each class, i.e. an orthogonal basis for the ones, one for the twos, for the threes and so on. The source for the bases is a tensor with all the digits of the same kind from the training set. In other words, to construct a set of bases for the twos, all of the twos in the training set have to be gathered in a separate tensor and this new tensor is the data for the HOSVD function. This will give the decomposition from which the basis matrices $\mathbf{A}_i^j$ are computed. The j indicates the class belonging of the basis matrix. There are ten classes (ten different kinds of digits), and therefore j varies from one to ten. The i indicates the number of the actual basis matrix for a given class.

Let us look more closely what this means. Let $\mathcal{A}_{twos} \in \mathbb{R}^{16\times 16\times n_3}$ be the tensor with the twos and assume that the HOSVD is already computed. Then according to equation (2.19)

$$\mathcal{A}_{twos} = \sum_{i=1}^{n_3} \mathbf{A}_i^{two} \times_3 \mathbf{u}_i^{two,(3)},$$

where the $\mathbf{A}_i^{two}$ are orthogonal basis matrices. This also means that every 2 in $\mathcal{A}_{twos}$ is a unique linear combination of the same basis matrices $\mathbf{A}_i^{two}$. The coefficients in the linear combination are given by the elements of the $\mathbf{u}_i^{two,(3)}$ vectors. For example, the first 2 in $\mathcal{A}_{twos}$, which also is the first slice, can be written like this

$$\mathbf{A}(:,:,1) = u_1^{(3)}(1)\mathbf{A}_1 + u_2^{(3)}(1)\mathbf{A}_2 + u_3^{(3)}(1)\mathbf{A}_3 + \cdots + u_{n_3}^{(3)}(1)\mathbf{A}_{n_3}. \tag{2.20}$$

The j index, which is *two*, is omitted here for simplicity. In this case the coefficients in front of the bases are the first elements of $\mathbf{u}_i^{(3)}$. The coefficients for the second 2 in the source tensor are the second elements of $\mathbf{u}_i^{(3)}$ and similar for all the others. The linear combination in (2.20) is illustrated in figure 2.6 for an arbitrary 2 in $\mathcal{A}_{twos}$.

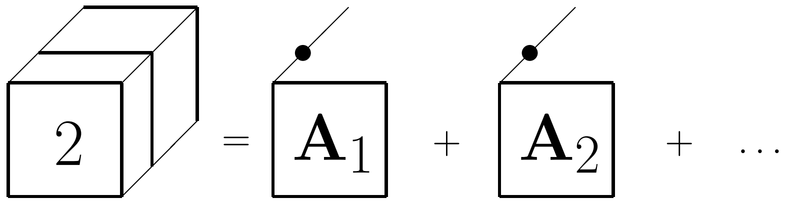

Figure 2.6. Illustration of a linear combination as in (2.20).

The block in figure 2.6 represents all the twos. A particular two is chosen from this tensor. This two is a linear combination of the $\mathbf{A}_i$ with coefficients from the vectors marked with dots.

2.3.2 Least squares problem

Suppose that the basis matrices are constructed and available for all the classes. Which set of bases is then describing an unknown digit $\mathbf{X}$ in the best way? Consider the minimization problem

$$\min_{\alpha_i^j} \|\mathbf{X} - \sum_{i=1}^{k} \alpha_i^j \mathbf{A}_i^j \|_F, \qquad j \quad \text{fixed}, \tag{2.21}$$

in which the α_i^j are unknown scalars to be determined and k is the number of basis matrices (vectors) used. This is a least squares problem and is particularly easy to solve when the $\mathbf{A}_i^j$ are orthogonal and they are orthogonal when j is kept fixed. In terms of the minimization the answer to the question above is: The set of bases, which gives the smallest norm. Notice that the $\mathbf{A}_i^j$ matrices in (2.21) actually belong to one specific class given by index j. This means that the minimization problem is solved with the basis matrices for each class, totally ten times. The unknown is most probably belonging to the class with the set of bases that gives the smallest norm.

It should also be noticed that the whole theory is built on the assumption that unknown digits from one class are described well as a linear combination of the basis matrices for that specific class. Particularly, the algorithm relying on this theory is useful if the unknown digits in a specific class are described well with few of the basis matrices for that class.

Cosine measure

In the special case when $k = 1$ and the matrices $\mathbf{X}$ and $\mathbf{A}_1$ are normed, $\|\mathbf{X}\|_F = \|\mathbf{A}_1\|_F = 1$, α_1 is given by the cosine of the angle between two matrices. The definition of the cosine angle of the matrices is in complete analogy with the definition for vectors.

$$k = 1 \qquad \text{gives} \qquad \alpha_1 = cos(\theta) = \langle \mathbf{X}, \mathbf{A}_1 \rangle. \tag{2.22}$$

This special case only uses the first basis matrix in each set and may not be suitable in all applications, not even this one. Good performance for this algorithm would mean that the test digits are very well written and the variation in a given class is not so big. These are of course not realistic assumptions in practical applications.

General solution

Even in the general case the particular least squares problem (2.21) is easy to solve due to the fact that the basis matrices $\mathbf{A}_i$ are orthogonal. The class index j is omitted for simplicity. If the basis matrices are also normalized, i.e. $\langle \mathbf{A}_i, \mathbf{A}_i \rangle = 1$, the solution is given by $\alpha_i = \langle \mathbf{X}, \mathbf{A}_i \rangle$. Proof of this statement is given in Appendix A. Inserting the solution into equation (2.21) gives the following expression of the squared norm

$$\min_{\alpha_i} \|\mathbf{X} - \sum_{i=1}^{k} \alpha_i \mathbf{A}_i\|_F^2 = \mathrm{tr}[(\mathbf{X} - \sum_{i=1}^{k} \alpha_i \mathbf{A}_i)^T (\mathbf{X} - \sum_{i=1}^{k} \alpha_i \mathbf{A}_i)] =$$

$$= \langle \mathbf{X} - \sum_{i=1}^{k} \alpha_i \mathbf{A}_i, \mathbf{X} - \sum_{i=1}^{k} \alpha_i \mathbf{A}_i \rangle =$$

$$= \langle \mathbf{X}, \mathbf{X} \rangle - 2\sum_{i=1}^{k} \langle \mathbf{X}, \alpha_i \mathbf{A}_i \rangle + \sum_{i=1}^{k} \alpha_i^2 = \langle \mathbf{X}, \mathbf{X} \rangle - \sum_{1}^{k} \alpha_i^2,$$

where we have used the fact $\langle \mathbf{A}_i, \mathbf{A}_j \rangle = \delta_{ij}$ (Kronecker delta). If also the unknown digits are normalized to unity, the only computations in the classification phase are the squares of the scalar products $\langle \mathbf{X}, \mathbf{A}_i \rangle$, summations and a comparison test. The basis sets are available at all times and are therefore a part of the algorithm.

The reduction of the data from the training set to a set of basis matrices is actually a data compression. From approximately 1000 digits belonging to one class a basis is constructed by HOSVD consisting of at most 20 matrices of the same size as the original digits. This is a data compression by approximately 98%.

The complete classification algorithm is as follows.

- Training phase:
  1. Collect the digits from the training set into tensors with digits of the same type.
  2. Compute the HOSVD of these tensors.
  3. Compute and store the basis matrices. This is a data compression.
- Test phase:
  1. Solve the least squares problem for each set of bases, i.e. compute the scalar products $\langle X, \mathbf{A}_i^j \rangle$ for all i and j.
  2. Take the label of the set that gives the lowest minimum as a guess.

2.4 Tests and results

The process described this far is very general and there are several parameters that can be varied to customize an algorithm. One such parameter is the number of basis matrices to be used in the classification. In this section most of the tests conducted to validate the algorithms are presented.

2.4.1 Test 1

The first step is a test of the first basis matrix against the source tensor from the training set that it was constructed from. Thus we are looking for the answer of how well the digits of a specific kind are represented by their own first basis matrix. Figure 2.7 shows the look of the first basis matrix for few digits. It is seen clearly which digit the basis matrices represent.

There is a good agreement between digits if the cosine measure is as close to one as possible. The smaller the cosine angle is the closer and more alike two matrices

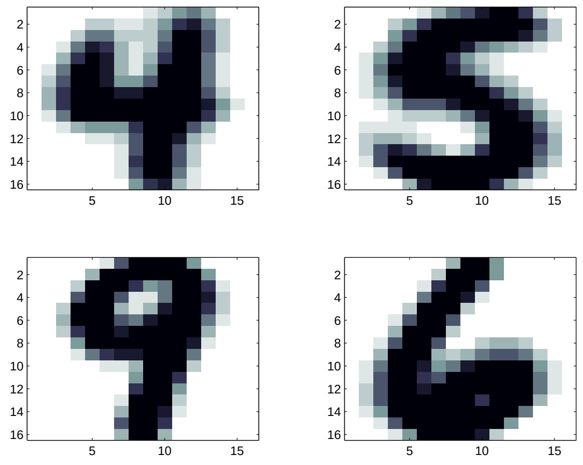

Figure 2.7. The first basis mtrix for few digits from the U.S. Postal Service database.

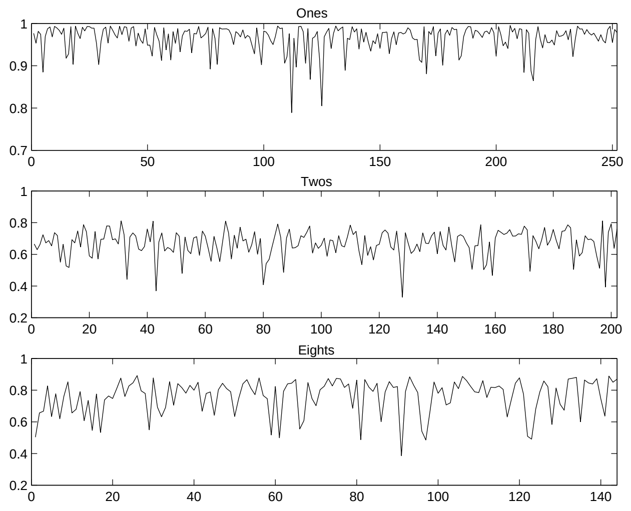

Figure 2.8. The cosine between the first basis matrix for three different classes and their respective source tensor, which include all digits of the same kind from the training set.

are. During the test it was established that not all the digits are described well by their first basis matrix. For example the ones are described very well by their first basis matrix while the cosine between the twos and their first basis matrix are considerably lower. See figure 2.8. Notice that the spikes indicate that there are less well written digits in the training set. The average of the cosines for all the digits are given in the table below. Here we get a first indication about which digits are hard to identify. According to the table twos and fives are not characterized

Digit	0	1	2	3	4	5	6	7	8	9
Average cosine angle	0,71	0,96	0,67	0,76	0,75	0,69	0,79	0,83	0,76	0,83

Table 2.3. The average of the cosine angles for all the digits.

well by the first basis matrix. One would then expect that these digits are harder to classify than others. This is verified in the next tests.

2.4.2 Test 2

The second series of tests were a continuation of the previous one, but this time the whole test set was tested against the first basis matrices at a time. In the tests the cosines were measured between the first matrices and all the test digits. An unknown digit was decided to belong to a certain class if the cosine was close enough to one. In other words if the cosine was above a given threshold value , the digit was assumed to belong to the same class as the basis matrix. The results are summarized in table 2.4, where RI and WI mean right identification and wrong identification respectively.

Digit (Total)	**0** (356)			**1** (264)			**2** (198)		
RI	181	222	265	246	255	258	25	72	118
WI	3	34	84	6	25	61	12	262	518
Border value	0.70	0.65	0.60	0.85	0.80	0.75	0.75	0.70	0.65
Digit (Total)	**3** (166)			**4** (200)			**5** (160)		
RI	82	113	135	24	73	110	41	54	70
WI	14	72	291	1	37	153	14	38	126
Border value	0.75	0.70	0.65	0.85	0.80	0.75	0.75	0.73	0.70

Digit (Total)	**6** (170)				**7** (147)				
RI	109	117	121	128	66	76	93	97	107
WI	6	22	42	74	3	21	38	52	76
Border value	0.78	0.76	0.74	0.72	0.85	0.83	0.81	0.80	0.78
Digit (Total)	**8** (166)				**9** (177)				
RI	35	47	69	81	63	86	103	116	129
WR	3	9	26	74	5	15	40	69	99
Border value	0.82	0.80	0.78	0.76	0.88	0.86	0.84	0.82	0.80

Table 2.4. Test results for the cosine angles for all digits in the test set.

The following conclusions are drawn from the table. Lowering the threshold value implies that more digits of the same class can be identified. But this also implies that more wrong identifications are made. In almost all of the cases the proportions between RI and WI are unacceptable, besides RI is not even close to identify all the digits of the same kind. Also these tests confirm that especially twos and fives are hard to recognize and other digits can easily be mistaken as a two or a five due to the high WI.

2.4.3 Test 3

The third test round is conducted precisely as the algorithm formulation in section 2.3. Up to 20 bases for each class were used in the classification and 10 least squares problems are solved for every unknown. A digit is said to belong the class of the basis set that gives the lowest minimum. In each test the number of basis matrices was the same for all the classes. This is the first and most natural way to use the algorithm in real classification applications. Both data sets were tested with this method.

NBM	1	2	3	4	5	6	7	8	9	10
WI 0	64	24	17	11	8	10	8	8	5	6
WI 1	4	12	8	8	10	6	4	4	3	3
WI 2	60	46	36	26	23	24	19	22	20	19
WI 3	35	23	28	27	22	23	22	18	19	20
WI 4	53	49	38	31	33	31	28	21	18	17
WI 5	49	35	33	29	29	28	26	25	23	21
WI 6	25	14	13	9	10	11	10	11	10	10
WI 7	25	16	17	13	10	9	9	10	8	9
WI 8	41	33	34	32	34	32	26	25	23	21
WI 9	42	23	19	15	16	16	15	16	19	11
WI total	398	275	243	201	195	190	167	160	148	137
WI in %	19.8	13.7	12.1	10.0	9.72	9.47	8.32	7.97	7.37	6.83

Table 2.5. The total number of wrong identifications for the U.S. Postal Service test set.

NBM stands for number of basis matrices and WI stands, as earlier, for wrong identifications. The middle part of the table gives the distribution of the digits that are identified wrong. It is clear that the more basis matrices that are used, the better the algorithm performs. But this does not mean that the incorrect classifications for all of the different kinds of digits decrease. The overall result gets better. As an example of this examine the WI 3 row or WI 6 row in the table.

One interesting thought is to use different number of basis matrices for different kinds of digits. This subject was not investigated in detail but the few tests made indicate that such a construction is not beneficial for the overall result. The test results are omitted.

Table 2.5 gives detailed information about the test results for the U.S. Postal Service digits with ten basis matrices used in the algorithm, at most. Figure 2.9 gives the total error rate for both sets with up to 20 basis matrices.

It is obvious that the MNIST database is easier to classify. An equivalent conclusion would be that the amount less well written digits in USPS database is

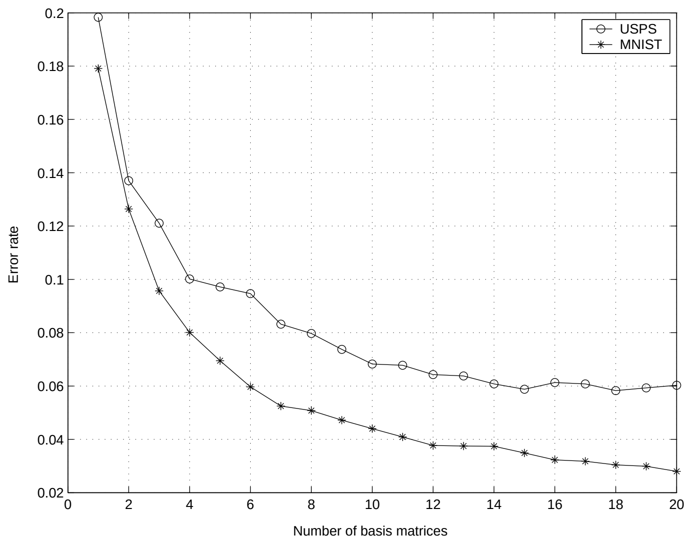

Figure 2.9. Classification results with 20 basis matrices for both data sets.

greater than in the MNIST set. This classification algorithm gives approximately an error rate of 6 % for the USPS test set and 3 % for the MNIST test set for the optimal number of basis matrices.

2.4.4 Test 4

It is not always desired to classify as many objects as possible in all applications. Sometimes it is more important to be confident that an algorithm makes the right classification in every case. If there is a high risk that a wrong classification is to be made then the particular object should be sorted out as unidentified. This is easily taken care of if an extra condition is added to the second step in the test phase of the algorithm. The condition is: If the lowest minimum is significantly lower then all the other minima then take the label of that basis set.

This idea was implemented in a fourth series of tests simply by comparing the lowest minimum for each number to the second lowest minimum. If the difference is bigger than a given value, let us call it buffer value, then classify as before. But if the difference is smaller then sort out the digit as unidentified. The latter would mean that there is at least one other minimum in the buffer region. The results for six tests with increasing buffer value from top to bottom are given in table 2.6. SO stands for sorted out, which is the same as the number of unidentified digits.

Increasing the buffer value makes the algorithm become more reliable. The price for this confidence is the large number of unidentified digits. Increasing the number of basis matrices also gives less wrong identifications. But the number of correctly identified digits does not necessarily increase. This is clearly seen in the

NBM	1	2	3	4	5	6	7	8	9	10
RI	1528	1658	1690	1751	1743	1746	1761	1786	1806	1816
WI	260	186	160	119	125	112	92	95	86	83
SO	219	163	157	137	139	149	154	126	115	108
RI	1440	1571	1619	1660	1667	1666	1681	1696	1712	1723
WI	167	116	95	69	66	68	52	57	51	51
SO	400	320	293	278	274	273	274	254	244	233
RI	1332	1467	1503	1547	1560	1576	1591	1595	1606	1608
WI	102	71	52	34	40	34	33	31	27	29
SO	573	469	452	426	407	397	383	381	374	370
RI	1271	1372	1389	1433	1462	1469	1471	1485	1490	1489
WI	59	39	25	20	18	19	20	17	17	16
SO	731	596	593	554	527	519	516	505	500	502
RI	985	1139	1131	1181	1198	1207	1195	1191	1157	1137
WI	17	10	9	7	7	4	6	6	5	5
SO	1005	858	867	819	802	796	806	810	845	865
RI	757	883	898	944	940	896	874	857	729	701
WI	6	4	5	4	5	4	4	4	1	1
SO	1244	1120	1104	1059	1062	1107	1129	1146	1277	1305

Table 2.6. Results from six series of tests with test digits from the U.S. Postal Service set. The number of basis matrices is varied from 1 to 10 in each series. The buffer values are in the order 1, 5, 10, 15, 20 and 25 in the different series.

last test in table 2.6.

One might also suspect that correctly identified digits are shovelled over to the unidentified digits when increasing the number of basis matrices. And this is exactly what happens. Increasing the number of basis matrices for a class gives a lower minimum for the least squares problem. The consequence of this is a higher probability that another minimum gets into the buffer region with the result of sorting out the digit.

2.5 Analysis and discussions

In this last part of the chapter the algorithm will be analysed and discussed in several sections. The point is to better understand the theory and get some insights of why and when the classification algorithm will perform acceptably well.

2.5.1 Vector space interpretation

The entire theory can be viewed in a second way in very simple linear algebra terms. The digit images are represented as matrices but they could also be presented as vectors. Then one might look at these vectors as objects of a high dimensional vector space. If the size of the images are $(m \times n)$ then the vectors would have mn

elements and be a vector in $\mathbb{R}^{mn}$. It is natural to assume that digits of the same kind are close to each other in this space, at least closer to digits from its own class than to digits from any other class. This would imply that the different classes are somewhat separated in space. I.e. all the ones are gathered in a certain region, the twos are gathered in another region and the same for the other classes.

The process of computing the basis matrices with help of the SVD on the training set would correspond to building a subspace for a given class. The SVD gives automatically basis vectors that span the subspace in a way that most of the variation in the training set is taken care of. A subspace is computed for each class.

Then there is the least squares minimization problem. An unknown digit is somewhere in $\mathbb{R}^{mn}$. Now we have to decide which class it belongs to. It is most likely that the unknown digit belongs to the closest located class. One way to measure the closeness of a vector in space to a given subspace is to take the orthogonal projection of the vector into this subspace. Then the vector is split in two parts. One of these lies in the subspace and the other, called the residual, is orthogonal to the subspace. The closer the vector is to the subspace, the smaller is the norm of the residual vector. The classification algorithm makes 10 orthogonal projections into different subspaces. Each subspace symbolizes different digits, and the label of the unknown is given by the subspace that gives the smallest norm of the residual vectors. This is the interpretation of the least squares solution.

2.5.2 Angles between subspaces

In the previous section a subspace interpretation of the basis sets was discussed. In this section the relations between different classes will be discussed. Once again the basis matrices for a given class are considered as basis vectors spanning a subspace. And the question is how close a given subspace is to another given subspace. The question is very relevant because the answer will give information about the closeness of the different regions in the space, in which the digits are grouped.

For example the threes, the fives and the twos have shapes pretty similar to each other. One would expect that their regions in this high dimensional space are close. Then comes other questions: is this good and if not how bad is it? Can the regions be separated? Knowing the answers to these questions will give a further understanding why the algorithm works as it does or perhaps why it does not work well as one might expect.

Let $\mathbf{F}$ and $\mathbf{G}$ represent subspaces in $\mathbb{R}^m$. Then a set of angles, called the principal angles, is derived measuring how much these subspaces are separated. A method of computing the principal angles between two subspaces is described in Golub [6]. The basis vectors spanning the subspace for one class are the columns of $\mathbf{F}$ and the basis vectors for the other class are the columns of $\mathbf{G}$. Table 2.7 gives a few sets of angles between subspaces.

The principal angles are for the classes given in the first two columns. The dimension of the subspaces was chosen to be eight. This means that the first eight basis matrices for each class were used. First thing to notice is that the angles

Class of **F**	Class of **G**	The principal angles							
0	1	0.30	0.91	1.22	1.32	1.45	1.51	1.54	1.56
1	3	0.37	0.95	1.38	1.43	1.45	1.50	1.53	1.56
1	5	0.30	1.01	1.29	1.33	1.43	1.47	1.54	1.55
2	5	0.30	0.87	0.99	1.08	1.27	1.32	1.43	1.56
3	5	0.30	0.38	0.67	0.74	0.90	1.14	1.37	1.50
3	8	0.30	0.56	0.77	0.80	1.14	1.31	1.42	1.54

Table 2.7. Principal angles between some of the classes given by **F** and **G**.

are between 0 and $\pi/2$ as they should be. If two vectors are separated as much as possible, then they are orthogonal and the angle between orthogonal vectors is $\pi/2$. Second thing to notice is that the first three sets of angles are typically a bit higher than the last three sets. But then again the last three sets of angles are taken between similar in shape digits. The first three comparisons - zero to one, one to three and one to five are not so similar and the angular distribution is rather much the same.

This is a confirmation that some digits are closer to each other in the high dimensional space. This will result in difficulties when trying to identify an unknown digit that belongs to classes who are close to each other, especially if it is less well written.

Increasing the number of basis matrices when describing a class gives better performance of the algorithm, but increasing the number of basis matrises also makes the risk for intersection between subspaces bigger. This would result in that an unknown digit can be written arbitrarily well as a linear combination of the basis for some other class than its own. This is not a desirable situation! If the number of bases is kept low, the algorithm will also need less memory and run faster, which is a positive effect.

2.5.3 Singular values

The question of how many basis matrices that are needed in the algorithm is of crucial importance. The algorithm performs better if the number of basis matricis is increased, but then the memory requirements and the computation time will increase. One way, besides tests, to decide the number of basis matrices for each class is to examine the singular values for the training tensors in the third mode. This is the same as examining the norms for the basis matrices given that they are not normalized.

Figure 2.10 gives the singular values along the third mode for the tensors containing the ones, the twos and the threes. These third mode singular values reflect the nature of the source tensors from the training set. Ideally it is desired that the singular values decrease towards zero as fast as possible. If this was the case the variation of the digits in the same class would be very small. I.e. the variation of the digits of the same class from the training set is given of the third mode singular

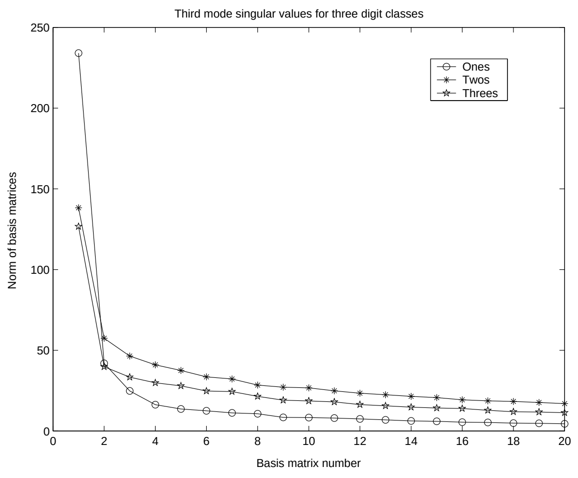

Figure 2.10. Singular values for three digit classes.

values. Examine the three curves in figure 2.10. The singular values for the ones decrease with the highest rate. The singular values for the twos are considerably higher and the singular values for the threes are placed somewhere in between. This is in total agreement with the presented test results. The ones are easy to find and the twos are relatively hard to identify. The classification of the threes is harder than for the ones but not quite as difficult as for the twos. This reasoning implies that the natural variation of the threes is greater than for ones. The twos have even higher variation.

This is not as strange as it might sound. On the contrary it is very logical if the actual look of the different digits is considered. It is quite clear that a one is a simpler digit than a two or a three. Analysis of the singular values can be very useful to get an idea of the variation of the objects in a given class.

2.5.4 Residual curves

In general increasing the number of basis matrices gives overall better results but the error for different classes do not necessarily decrease. This was already mentioned earlier and test results confirming this is found in table 2.5. In this section the minima of the least squares problem will be examined. Especially we will see the progress of the smallest minimum of a badly written digit. Consider the digit in figure 2.11, it is no doubt that the five is badly written.

The curves in next figure are all the minima for a given number of basis matrices and the $10th$ values are the values for the zero class. The rings indicate the minimum of a curve. Therefore the rings also indicate the output of the algorithm. For example the smallest minimum in the curve of one basis matrix occurs at four. This is not the right answer of course, because the digit is a five. Increasing the number of basis matrices to three gives a new minimum, namely at eight, but still

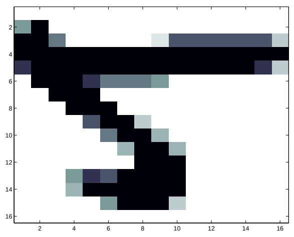

Figure 2.11. A badly written five from the U.S. Postal Service set.

not the right one. Adding another basis matrix gives the right answer - at last the minimum is at five. These curves are in the first part of figure 2.12.

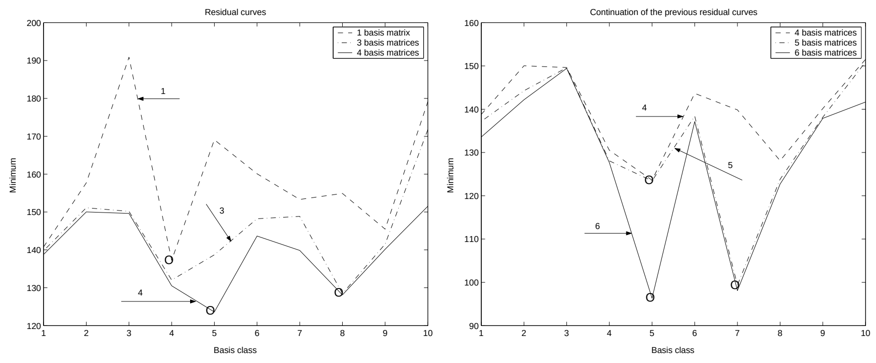

Figure 2.12. Residual curves for the five above.

Continuing this procedure, increasing the number of bases, shows that the minimum does not remain at the right place just because it was there previously. The second part of the figure is a continuation of the residual curves. The curve for four basis matrices is the same as in the first part. The other curves are for five and six basis matrices. The minimum jumps to seven and then backs to five again. In five cases the algorithm gives four different outputs. This is not a desirable characteristic of the ugly digits or of the algorithm.

Notice that the curves are completely under each other, no intersections. This is natural because increasing the number of basis matrices gives a smaller minimum, for all the classes.

Notice also that the minimum for some classes decrease a lot and others almost do not decrease at all. In terms of the subspaces spanned by the basis matrices it can be interpreted in the following way. When increasing the number of basis the subspaces for the classes are enlarged. All digits have their specific places in the high dimensional space and adding a new basis matrix enlarges the different subspaces in different directions in space. Some of these directions are towards a given digit, others not. If the subspace is enlarged in the wrong way the minimum will not be affected, but if the subspace is enlarged toward a given digit the minimum will decrease considerably.

2.5.5 Number of operations

How fast an algorithm performs a classification is of great importance in real applications. The training phase of the algorithm is also important even though it is not relevant at all in an actual application. An overview of the amount of computation needed to construct this algorithm is given below.

Training phase

In this algorithm the only thing done in the training phase is the computation of the basis matrices, which are stored and available at any time in the test phase. Inserting equation (2.12) in (2.19) gives

$$\mathcal{B} = \mathcal{A} \times_3 \mathbf{U}^{(3)}, \tag{2.23}$$

which is a consequence of property (2.5). The slices of $\mathcal{B}$ are the basis matrices for the class in question. Thus the only computations needed for computing the basis matrices for one specific class is the usual SVD of the third mode unfolding $\mathbf{A}_{(3)}$ and a third mode multiplication equivalent to a usual matrix product.

There are several algorithms for the SVD computation and Golub [6] gives the number of operation or flop counts for a specific algorithm to be approximately $4m^2n + 22n^3$, assuming an $(m \times n)$–matrix. The matrix product between two matrices $\mathbf{A} \sim (m \times n)$ and $\mathbf{B} \sim (n \times p)$ takes $2mnp$ flop counts.

Having a $(I_1 \times I_2 \times I_3)$–tensor would give SVD of $(I_3 \times I_1 I_2)$–matrix, which is approximately (1000×800) for digits from the MNIST database, and a product with a $(I_3 \times I_3)$–matrix from left. In these terms the total amount of flop counts for one class would be

$$6I_3^2 I_2 I_1 + 22 I_2^2 I_1^2.$$

In our case there are ten classes, but the digits in the training sets are not equally distributed. Therefore the total number of operations will be approximately ten times the expression above.

Test phase

In the test phase a series of scalar products between $(I_1 \times I_2)$–matrices are computed. Each scalar product is equivalent to a $(I_1 I_2)$ sized vector scalar product. Assuming that the algorithm uses m basis matrices for each class and there are ten classes the total number of operations needed to be calculated in an actual application is approximately

$$20mI_1I_2 + 10m.$$

Besides these calculations a comparison test between ten numbers to get the minimum is also done in each classifications. Notice that the operations above are for classification of only one digit.

One of the advantages of this algorithm is that all the computation of the bases, i.e. the training phase, is done off–line, before the classification device is put in use. The only computations that are needed in real–time, when the device is in use, are the operations given in the test phase.

2.5.6 Last comments

Firstly, the theory is so basic that researchers proposed a pattern recognition algorithm using principal components, which is very close to SVD, as early as in the seventies. In an article written by Wold [15] the algorithm was applied in a small recognition problem involving three different classes. In a more recent article from 1997 [8], the classes for handwritten digits were modelled using SVD. The subspaces spanned by the basis matrices are models of the different classes for the digits. The main difference between the theory in those articles compared to the theory in this report is that they worked from a matrix and a statistical point of view, while the theory here was presented in the framework of tensor theory. The HOSVD and the basis matrices make this theory very easy to overview and is intuitively attractive in the terms presented. Of course all the results could and can be achieved only in terms of matrices but then again it would be a bit messier.

Secondly, the $\mathbf{U}^{(1)}$ and $\mathbf{U}^{(2)}$ matrices are not used in any way. These matrices can be used for compressing the data, suggested in [3] and [4]. The method is applied to electronic nose sensor data with promising results. It would not be beneficial to compress the source data along the first and second mode in our case because the digits are already quite small, but in other applications with bigger objects to classify the method might have significant role in reducing the amount of computation and memory requirements.

Another analysis on this algorithm, that might give further insights and perhaps improve the performance, would be to examine the distributions of the α_i coefficients in equation (2.21). A quick check of this revealed that different digits have different coefficient distributions. How this fact can be useful is an open question. Perhaps it might be useful for detecting less well written digits.

Chapter 3

Tangent Distance Classification

The main parts of this chapter are the introduction of the tangent distance, tangent vectors and implementing these ideas in a classification algorithm. Finally some tests and results will be presented.

Usually there is a large variation of the digits in a given class. The digits might have different styles, they can be rotated, written big or small, and the lines might be thin or thick. All these and other factors contribute to the variation of the digits in the class. At the same time digits from different classes might be very similar to each other. Clearly a good algorithm has to distinguish between the differences within a class and the differences between classes. The algorithm described in the previous chapter takes only the greatest variation into account when classifying. This is not an optimal way of identification if the aim is to minimize the error rate. The thought behind the tangent distance is to address and handle different kinds of invariant transformations.

3.1 Transformation invariance

The closeness between different objects in a number of classification algorithms can be measured through the Euclidean distance. Nearest neighbour or the k–nearest neighbours classifiers in [13], [7] are examples of such algorithms. The Euclidean distance between two vectors $\mathbf{x}$ and $\mathbf{y}$ in $\mathbb{R}^n$, written $d_E(\mathbf{x}, \mathbf{y})$, is given by

$$d_E(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}. \tag{3.1}$$

The nearest neighbour classifiers calculate a distance, for example the Euclidean distance, between an unknown and all the prototypes. The digits in the training set would be the prototypes and the unknown is labelled as the label of the closest digit. The label of the majority of the k closest digits gives the decision in the k–nearest neighbours classifiers.

In this context the digits are represented as vectors as seen and commented on earlier. For the envelope digts (USPS), which are stored in 16×16 grey level pixel images, are seen as vectors of a 256 dimensional space called pixel space [14].

The problem with the Euclidean distance is its high sensitivity to different kinds of transformations. Consider a translation of some digit along the x– or y–axis. The Euclidean distance between a digit and a translated version of the same digit is clearly not zero and it might increase if the translation is made bigger. Such a translation should not affect the distance measure at all. But the Euclidean distance cannot take this property into account and often causes misclassifications. There is a need for a distance measure that is invariant to translations in the pixel space.

Translation is only one kind of transformation that the distance measure should be invariant to. Other transformations such as rotations, line thickening or thinning, scaling and others should also be invariant in the measured distance.

In the case of character recognition these transformations are familiar [14]. It is not obvious to what kind of transformations the distance measure should be invariant to in other applications. Therefore the transformations are application specific and have to be predefined.

In many cases even small differences do cause wrong classification in nearest neighbour algorithms. Consider the digits in figure 3.1. The first digit is an un-

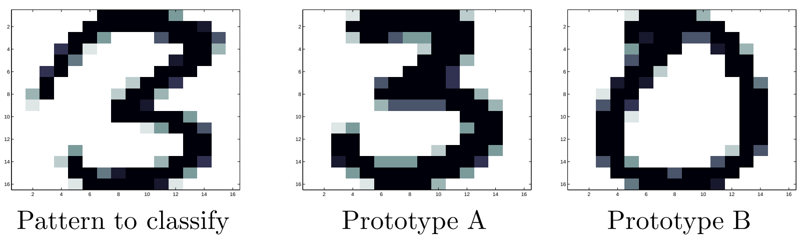

Figure 3.1. According to the Euclidean distance the unknown digit is closer to prototype B even in this simple case.

known to be classified. The other two are prototypes with known labels. Calculation of the Euclidean distance between the unknown and the two prototypes A and B gives 13.9 and 13.6 respectively. Thus a smaller distance for prototype B resulting in a wrong classification even in this very obvious case. A small clockwise rotation, a bit line thickening and resizing the pattern are the transformations in question that after transforming the unknown might give the right answer.

To sum up the discussion so far, the desired distance measure has to filter out the variations inside a class and at the same time be able to detect the differences between separate classes in the classification process.

3.2 Tangent distance

This section is intended to describe the tangent distance [14]. First some assumptions have to be made. All the digits in both the training and test set are considered as vectors in pixel space. The dimension of the pixel space is 256 for the envelope digits and 784 ($= 28 \times 28$) for the MNIST digits. The second assumption is that the invariant transformations are already defined and continuous. These transformations will be described closely further ahead in this chapter.

3.2.1 Set of transformations and manifolds

Consider any given pattern $\mathbf{p}$ and its translated version along the x–axis. The transformation or the translated version can be written $s(\mathbf{p}, \alpha_x)$ where α_x is the amount of translation. α_x is a variable in this case and letting it vary continuously a whole set of translated versions of $\mathbf{p}$ is produced. This set constitutes a one–dimensional curve in pixel space. The curve is continuous because α_x is varying continuously and the transformation function $s(\mathbf{p}, \alpha_x)$ is continuous. Remember that all digits are presented as vectors therefore the pattern is denoted with bold lower–case letter.

Introducing other kinds of transformations such as translation along the y–axis, rotation and scaling would give four parameters to vary, appropriately denoted α_x, α_y, α_θ and α_{sca}. The set of transformed patterns that are obtained by varying the α parameters in $s(\mathbf{p}, (\alpha_x, \alpha_y, \alpha_\theta, \alpha_{sca})) = s(\mathbf{p}, \boldsymbol{\alpha})$ gives a four–dimensional surface in pixel space. In general n transform parameters in $\boldsymbol{\alpha}$ gives a manifold for the set of all transformed patterns. The word manifold should be interpreted as a higher dimensional surface embedded in pixel space. The dimension of this manifold is at most n. Denoting the manifold $S_{\mathbf{p}}$, we can write

$$S_{\mathbf{p}} = \{\mathbf{x} : \mathbf{x} = s(\mathbf{p}, \boldsymbol{\alpha})\}, \tag{3.2}$$

where $\boldsymbol{\alpha}$ is a vector with allowable transform parameters. The general transform functions $s(\mathbf{p}, \boldsymbol{\alpha})$ is assumed to be differentiable with respect to $\mathbf{p}$ and all the transform parameters α_i [14]. Furthermore it is required that $s(\mathbf{p}, \mathbf{0}) = \mathbf{p}$.

This manifold can in principle be used in the following way. For every unknown digit to be classified the manifold could be calculated. We could also calculate the respective manifolds for the prototype digits. Doing this we would have two manifolds in pixel space. The true invariant distance between an unknown and a prototype digit would be the minimal distance between these two manifolds or surfaces.

Practically there are problems with this approach. Consider the rotation parameter. Small rotations are no problem, but rotating a given number, for example a "6", would eventually give a "9" and vice versa. This is not good because all the sixes and nines would melt down to one class and separating them through this invariant distance would be impossible. These manifolds give global invariance but only local invariance is desired. Another serious problem is that in most applications there is no simple analytical expression for the manifold $S_{\mathbf{p}}$ in equation (3.2).

Even if the expression for the manifolds were given, finding this minimal distance is a non–linear problem with multiple minima. These kinds of minimzation problems are very difficult to solve.

3.2.2 Illustration of the tangent distance

Consider $s(\mathbf{p}, \alpha_x)$ again, the non–linear one dimensional curve in pixel space representing all the translated versions of pattern $\mathbf{p}$. A local approximation of this manifold (curve) can be obtained by a first order Taylor expansion of $s(\mathbf{p}, \alpha_x)$ around $\alpha_x = 0$. The linear approximation is given by

$$s(\mathbf{p}, \alpha_x) = s(\mathbf{p}, 0) + \alpha_x \frac{\partial s(\mathbf{p}, \alpha_x)}{\partial \alpha_x} + O(\alpha_x^2) \approx \mathbf{p} + \alpha_x \mathbf{t}, \qquad (3.3)$$

where $\mathbf{t} = \frac{\partial s(\mathbf{p}, \alpha_x)}{\partial \alpha_x}$, evaluated at $\alpha_x = 0$, is called the tangent vector. Obviously the local Taylor approximation is given by the pattern $\mathbf{p}$ and the tangent vector $\mathbf{t}$.

This is illustrated in figure 3.2. The elements in the upper row are x–translated variants of the pattern in the transformation function, starting with two pixels to the left, one to the left, no translation i.e. the original pattern, one pixel to the right and finally two pixels to the right. The linear approximation of the non–linear manifold is presented in the row below. It is the same as the right hand side of (3.3). The pattern $\mathbf{p}$ and the tangent vector $\mathbf{t}$ are reshaped to matrices and five different approximations with the corresponding α_x are given. The approximations are good

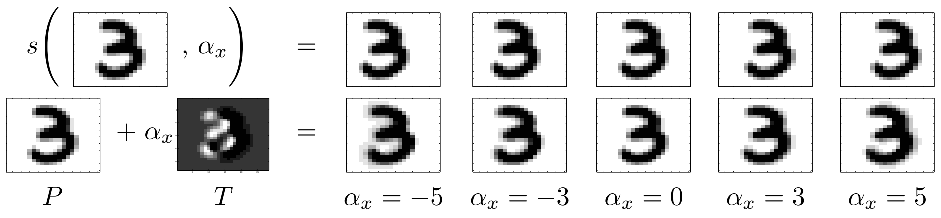

Figure 3.2. First row is exact x–translations of the pattern $\mathbf{P}$, which are elements of the manifold $S_{\mathbf{P}}$. The elements in the second row are approximations all lying on the tangent plane, in this case it is a tangent line.

for the inner two digits with $\alpha_x = \pm 3$ but less good for the outer digits. This is a clear consequence of omitting the higher order terms in the Taylor expansion in (3.3). It should be noticed that α_x is not the wanted translations in pixels. Notice also that nothing is said about how the tangent vector is computed. The objective is only to present the structure of the manifolds and the vectors on them. The tangent vectors will be the subjects of the next section where they will be analysed in detail.

Approximation of both manifolds, the one belonging to the unknown and the other belonging to the prototype, will give two lines. The tangent distance is then defined to be the smallest distance between these two lines. This problem, on the other hand, is a linear one and the solution is easily computed with effective algorithms.

In general there are n transform variables. Taylor expansion of the manifold $S_{\mathbf{p}}$ gives n tangent vectors. Each tangent vector is a partial derivative of the transform function $s(\mathbf{p}, \boldsymbol{\alpha})$. The corresponding expression for equation (3.3) is

$$\begin{aligned} s(\mathbf{p}, \boldsymbol{\alpha}) = s(\mathbf{p}, \mathbf{0}) + \alpha_1 \frac{\partial s(\mathbf{p}, \boldsymbol{\alpha})}{\partial \alpha_1} + \alpha_2 \frac{\partial s(\mathbf{p}, \boldsymbol{\alpha})}{\partial \alpha_2} + \cdots + \alpha_n \frac{\partial s(\mathbf{p}, \boldsymbol{\alpha})}{\partial \alpha_n} + O(|\boldsymbol{\alpha}|^2) \approx \\ \approx \mathbf{p} + \alpha_1 \mathbf{t}_1 + \alpha_2 \mathbf{t}_2 + \cdots + \alpha_n \mathbf{t}_n = \\ = \mathbf{p} + \begin{pmatrix} \mathbf{t}_1 & \mathbf{t}_2 & \ldots & \mathbf{t}_n \end{pmatrix} \boldsymbol{\alpha} = \mathbf{p} + \mathbf{T}\boldsymbol{\alpha}. \end{aligned} \tag{3.4}$$

The columns of matrix $\mathbf{T}$ are the different tangent vectors and right hand side of equation (3.4) is interpreted as a hyperplane, also called tangent plane, in pixel space. Planes with higher dimensions embedded in a high dimensional space are called hyperplanes.

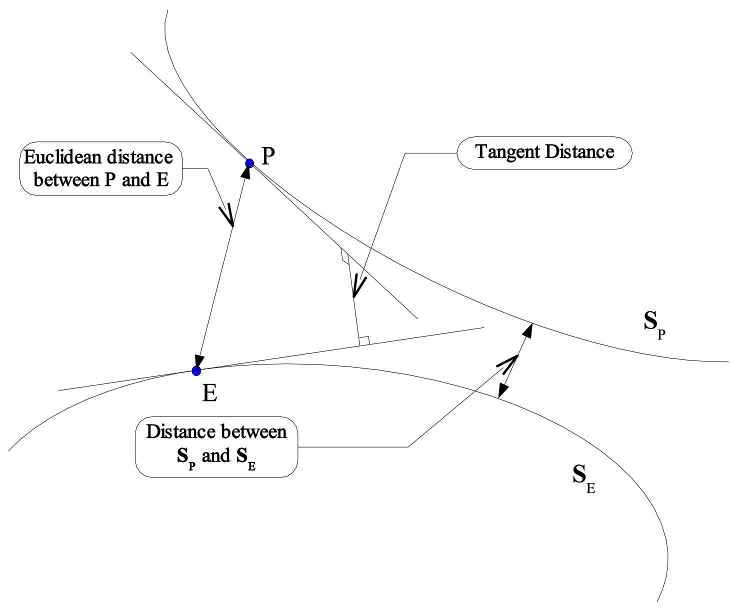
Euclidean distance between P and E
P
Tangent Distance
$\mathbf{S}_P$
E
Distance between $\mathbf{S}_P$ and $\mathbf{S}_E$
$\mathbf{S}_E$

Figure 3.3. Illustration of the manifolds $\mathbf{S}_E$ and $\mathbf{S}_P$ and the distance between them, the Euclidean distance between the unknown P and the prototype E, the linear approximations and most importantly the Tangent Distance.

The whole concept is illustrated in figure 3.3. There are two patterns, P and E. These are still vectors but are denoted with capitals for aesthetic reasons. The line between P and E is the Euclidean distance. Varying the respective transform variables in $s(P, \boldsymbol{\alpha}_P)$ and $s(E, \boldsymbol{\alpha}_E)$ two manifolds are produced, $\mathbf{S}_P$ and $\mathbf{S}_E$. These manifolds are not linear and therefore presented as curves. The true invariant distance between P and E is the minimal distance between the two manifolds, also drawn in the figure. The Taylor approximations are the tangent lines of these manifolds at P and E. These tangent lines do not intersect in pixel space and the minimal distance between them is the tangent distance [14]. The dimension of the hyperplanes (the tangent lines in the figure) is much smaller than the dimension of the pixel space and the probability that two hyperplanes belonging to different patterns will intersect is extremely, vanishing small.

3.2.3 Implementation of the tangent distance

In this section the computation of the tangent distance will be described. It is assumed that all the tangent vectors are computed. Suppose that there are n transform variables and the pattern $\mathbf{p}$ is given. A first order hyperplane approximates the non–linear manifold $S_{\mathbf{p}}$. This hyperplane is tangent to $S_{\mathbf{p}}$ at $\mathbf{p}$. The tangent vectors, which are the partial derivatives of the transform function evaluated at $\boldsymbol{\alpha} = \mathbf{0}$, together build a matrix giving the tangent plane. The expression for this matrix $T_{\mathbf{p}}$, as in (3.4), is

$$T_{\mathbf{p}} = \left. \frac{\partial s(\mathbf{p}, \boldsymbol{\alpha})}{\partial \boldsymbol{\alpha}} \right|_{\boldsymbol{\alpha}=\mathbf{0}} = \left[\frac{\partial s(\mathbf{p}, \boldsymbol{\alpha})}{\partial \alpha_1}, \frac{\partial s(\mathbf{p}, \boldsymbol{\alpha})}{\partial \alpha_2}, \ldots, \frac{\partial s(\mathbf{p}, \boldsymbol{\alpha})}{\partial \alpha_n} \right]_{\boldsymbol{\alpha}=\mathbf{0}}. \tag{3.5}$$

Letting $TP_{\mathbf{p}}$ symbolize the entire tangent plane, the tangent distance between the patterns $\mathbf{p}$ and $\mathbf{e}$ can formally be defined by

$$d_{TD}(\mathbf{p}, \mathbf{e}) = \min_{\mathbf{x} \in TP_{\mathbf{p}}, \mathbf{y} \in TP_{\mathbf{e}}} ||\mathbf{x} - \mathbf{y}||, \tag{3.6}$$

where $TP_{\mathbf{p}}$ and $TP_{\mathbf{e}}$ are the respective tangent planes. The equations for the tangent planes $TP_{\mathbf{p}}$ and $TP_{\mathbf{e}}$ are explicitly given by

$$\begin{aligned} \mathbf{p}'(\boldsymbol{\alpha}_{\mathbf{p}}) &= \mathbf{p} + T_{\mathbf{p}} \boldsymbol{\alpha}_{\mathbf{p}}, && (3.7) \\ \mathbf{e}'(\boldsymbol{\alpha}_{\mathbf{e}}) &= \mathbf{e} + T_{\mathbf{e}} \boldsymbol{\alpha}_{\mathbf{e}}. && (3.8) \end{aligned}$$

Now $\mathbf{x}$ and $\mathbf{y}$ are on the tangent planes given by (3.7) and (3.8) with certain values of $\boldsymbol{\alpha}_{\mathbf{p}}$ and $\boldsymbol{\alpha}_{\mathbf{e}}$ respectively. Inserting the equations into (3.6) we get a minimization problem to solve. The terms are either vectors or matrices. Restructuring the elements the whole minimization problem boils down to the least squares problem given below.

$$d_{TD}(\mathbf{p}, \mathbf{e}) = \min_{\boldsymbol{\alpha}_p, \boldsymbol{\alpha}_e} ||\mathbf{p} + \mathbf{T}_p \boldsymbol{\alpha}_p - (\mathbf{e} + \mathbf{T}_e \boldsymbol{\alpha}_e)|| =$$

$$= \min_{\boldsymbol{\alpha}_p, \boldsymbol{\alpha}_e} \left\| \begin{pmatrix} \mathbf{T}_p & -\mathbf{T}_e \end{pmatrix} \begin{pmatrix} \boldsymbol{\alpha}_p \\ \boldsymbol{\alpha}_e \end{pmatrix} - (\mathbf{e} - \mathbf{p}) \right\| = \min_{\mathbf{x}} ||A\mathbf{x} - \mathbf{b}||.$$

All the presented thoughts until now are found in [14], but some of the problems we have chosen to solve in different, maybe more efficient, way. The first departure from [14] is made when solving the least squares problem. Notice that the primary interest is to get the residual of the least squares problem, not the specific vector $\mathbf{x}$ that gives the solution for the minimum.

Knowing this, the minimization problem can be split up into two separate problems, the first one where the matrix A is non–singular and the second one where A is singular. In almost all cases A will be non–singular but to guarantee that the algorithm works every time the case of A being singular must also be considered. The meaning of a singular A is that the two tangent planes have parallel tangent vectors.

Case 1: A non–singular

Assuming that A is non–singular and knowing that its dimensions are approximately 250×15 the minimization problem is solved conveniently using the QR decomposition of A. Let $A = QR$, where Q is an orthogonal matrix and R triangular matrix. Then the least squares problem can be written as

$$\|\mathbf{b} - A\mathbf{x}\|^2 = \left\| Q^T\mathbf{b} - \begin{pmatrix} R \\ \mathbf{0} \end{pmatrix} \mathbf{x} \right\|^2 = \left\| \begin{pmatrix} \mathbf{c} \\ \mathbf{d} \end{pmatrix} - \begin{pmatrix} R\mathbf{x} \\ \mathbf{0} \end{pmatrix} \right\|^2 =$$

$$= \|\mathbf{c} - R\mathbf{x}\|^2 + \|\mathbf{d}\|^2.$$

Minimum is obtained by choosing $\mathbf{x} = R^{-1}\mathbf{c}$. Then the first term in the right hand side vanishes and the norm of the residual and the minimum is given by $\|\mathbf{d}\|$.

Case 2: A singular

In the case when A is singular a similar solution method is used but this time with SVD instead of QR. Let $A = U\Sigma V^T$, $\mathbf{z} = V^T\mathbf{x}$ and $\mathbf{c} = U^T\mathbf{b}$. Then

$$\|\mathbf{b} - A\mathbf{x}\|^2 = \|U^T\mathbf{b} - \Sigma V^T\mathbf{x}\|^2 = \|\mathbf{c} - \Sigma\mathbf{z}\|^2 =$$

$$\left\| \begin{pmatrix} \mathbf{c}_1 \\ \mathbf{c}_2 \end{pmatrix} - \begin{pmatrix} \Sigma_r & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{pmatrix} \begin{pmatrix} \mathbf{z}_1 \\ \mathbf{z}_2 \end{pmatrix} \right\|^2 = \|\mathbf{c}_1 - \Sigma_r\mathbf{z}_1\|^2 + \|\mathbf{c}_2\|^2.$$

Choosing $\mathbf{z}_1 = \Sigma_r^{-1}\mathbf{c}_1$ causes, once again, the first term to vanish and the minimum is $\|\mathbf{c}_2\|$. More detailed information about these solutions is found in [6] and [16].

3.2.4 Illustration

In this section some illustrative results are presented. The difference between the Euclidean and the tangent distance will be illustrated when those are used in simple classification tests. We will have 10 known digits, one of each kind arbitrarily chosen from the test set. Those digits are presented in figure 3.4. Both distances, the Euclidean and the tangent distance, will be computed between a specific digit, namely the 3 in the figure, and all the other numbers in the figure including itself. This will be done for a whole series of x–translated versions of the 3. The amount

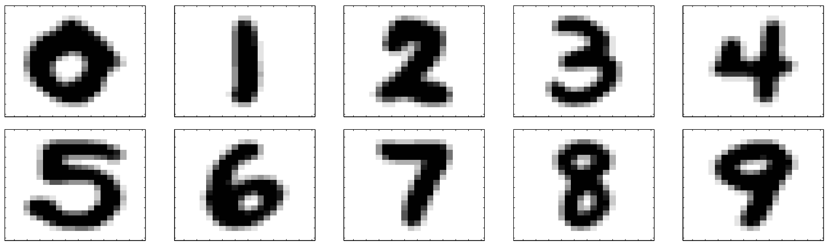

Figure 3.4. These 10 digits were used in the distance tests.

of translation is given in the x–axis and is ranging from four pixels to the left and four pixels to the right. The results are given in figure 3.5. The Euclidean distances are presented in the first graph and the tangent distances in the second. Notice first of all that there are 10 curves in each graph, one for each digit. On each curve there are 9 markers at the integer values of the x–axis. Those are the computed distances.

Examine the graph labelled with 3. The distance between the original 3 and itself is 0, for both distances. This is seen in the both graphs and is nothing strange, on the contrary it is trivial. But when the 3 is translated and both distances are computed to the original 3 the tangent distance remains low for a bigger interval than for the Euclidean distance. The local invariance property is clearly seen for

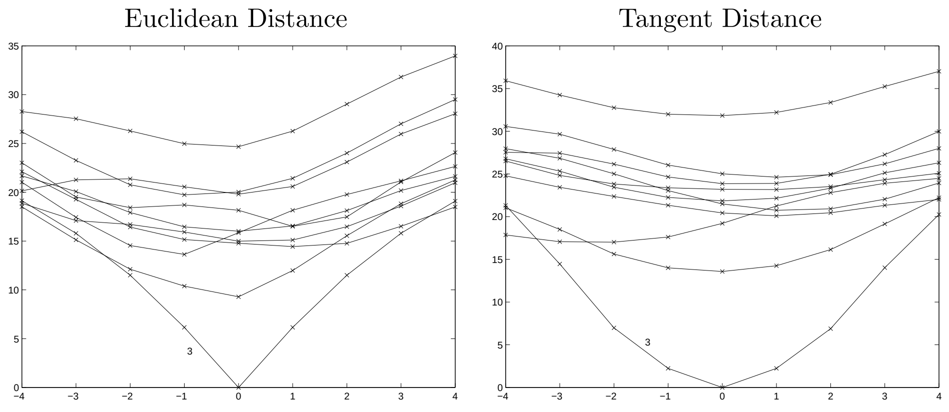

Figure 3.5. Test results between the digits from figure 3.4. Both the Euclidean and the tangent distance is computed between the translated versions of the 3 and all the other digits.

the tangent distance. If this test is considered as a nearest neighbour classification the tangent distance gives a correct answer for a bigger translation interval. Look at the Euclidean distance. After a translation of the 3 two pixels to the left and three to the right the smallest minimum or the nearest digit is no longer the 3 as it should be. This interval is enlarged in the case with the tangent distance and the classifier makes a wrong decision only when the 3 is translated four pixels to the left. Similar results are obtained for other digits and transformations. Analogous results with figure 3.5 are also presented in [14] and [9] for the same dataset.

3.2.5 Some comments

Hierarchy of distances

This method is very computationally heavy. There are several ways to reduce the amount of computations. One way is to use a hierarchy of distances as described in [14]. Such a hierarchy is for example the Euclidean distance with increasing resolutions and the tangent distance with increasing number of tangent vectors. The idea is first to use simple and fast algorithms and, as the classification gets more difficult change to a distance measure higher in the hierarchy and eventually use the full tangent distance only on the hardest classifications. According to the article [14] the overall computational cost is thereby greatly reduced.

One–sided tangent distance

Another way of reducing the amount of computations is to use so called one–sided tangent distance. Everything is the same as before but the tangent vectors for one of the comparing patterns is completely omitted. This gives fewer tangent vector computations and smaller minimization problems [14], [9].

3.3 Tangent vectors

In this section the computation of the different tangent vectors will be described. The mathematical background for the tangent vectors is rather complex. It involves several mathematical fields. Most of the theory will be presented with details and the results of other parts of the theory will be used without the proper derivation.

3.3.1 Blurring

The classification data can be preprocessed in several ways [12], [13]. Preprocessing means that the digits we want to classify are changed with the purpose to get a better result in the classification. Blurring and normalization are examples of preprocessing.

According to Simard [14] blurring is of great importance for the identification process, at least when classifying handwritten digits. The blurring can be described as smoothing the pattern or making sharp edges and corners softer.

Different kinds of blurring can be obtained depending on the function used in the operation. One usual function that is often used is the Gaussian function

$$g(x, y) = e^{-\frac{x^2+y^2}{2\sigma^2}}. \tag{3.9}$$

The standard deviation σ is used to control the amount of blurring. If σ is small the blurring is small and if σ is big the blurring is large. This thought can be illustrated by using the Gaussian function with one variable and plotting the function for few σ. See figure 3.6 below. These functions can be interpreted as if we had an element

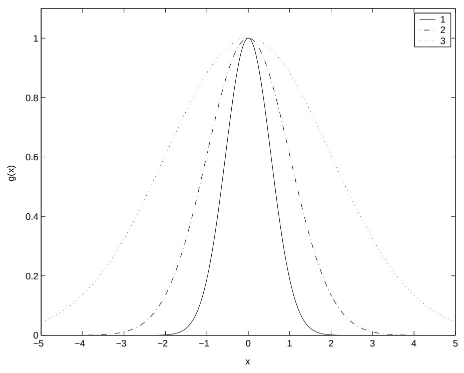

Figure 3.6. Three plots of the Gaussian function $g(x)$ with $\sigma_1 = 0.3$, $\sigma_2 = 1$ and $\sigma_3 = 4$.

with only one pixel located at zero with amplitude one. The corresponding blurred image would be one of the plotted functions depending on the blurring factor σ. Usually only the values of the Gaussian function at the integers on the x–axis are used. Figure 3.7 gives two examples with different σ together with the original pattern. It is clear that we get a smother and softer image but we also see a danger

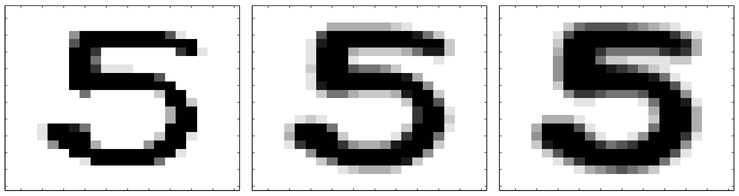

Figure 3.7. First image is the original pattern. This patterns is blurred with $\sigma = 0.7$ (the second image) and with $\sigma = 1$ (the third image).

in blurring an image too much. The gap in the lower part of the five is getting smaller. Increasing σ will eventually close the gap in the lower part and the image can be mistaken to be a six or even an eight because the gap in the upper right side of the image will also close. The blurring is good and the more the better but without causing any damage to the pattern shape that is vital for the digit.

3.3.2 Differential geometry concepts

Until now a very important stage in the derivation of the tangent distance has been left out, namely the actual computation of the tangent vectors. The theory of differential geometry gives us the means to compute these tangent vectors. These aspects of the theory will not be described in detail, but only the parts that are necessary to continue this presentation.

Let $f(x, y)$ be a continuous function with two variables that has the shape of some digit. Imagine $f(x, y)$ as a continuous equivalent of a discrete digit. Remember that all the digits in the data sets are discrete images. Assume that we want to compute the tangent vector corresponding the transform variable α_x, i.e. the x–translation. The appropriate transformation is given by

$$t_{\alpha_x} : \begin{pmatrix} x \\ y \end{pmatrix} \longmapsto \begin{pmatrix} x + \alpha_x \\ y \end{pmatrix}. \tag{3.10}$$

Applying the action of this transformation to the function f we get the set of all x–translated versions of the function f. This is written $s(f, \alpha_x)$ and the non–linear manifold is given by

$$S_f = \{f' : f' = s(f, \alpha_x)\}. \tag{3.11}$$

Notice that this is in complete analogy with thoughts presented in section 3.2. The only difference is that now we have a continuous function f instead of the pattern $\mathbf{p}$. s is seen as a functional that takes the function f and a transform variable α_x as arguments. Using the transformation t_{α_x} and the function f we can write s as

$$s(f, \alpha_x)(x, y) = f(x + \alpha_x, y). \tag{3.12}$$

Differentiating this around $\alpha_x = 0$ we get

$$\left.\frac{\partial s(f, \alpha_x)(x, y)}{\partial \alpha_x}\right|_{\alpha_x=0} = \left[\frac{\partial f(x, y)}{\partial x}\frac{\mathrm{d}(x + \alpha_x)}{\mathrm{d}\alpha_x} + \frac{\partial f(x, y)}{\partial y}\frac{\mathrm{d}y}{\mathrm{d}\alpha_x}\right]_{\alpha_x=0} = \frac{\partial f}{\partial x}. \tag{3.13}$$

This approach gives an operator

$$L_{\alpha_x} = \frac{\partial}{\partial x}, \tag{3.14}$$

corresponding to the actual transformation and a way to compute the tangent vector. In this case the tangent vector (here a more appropriate name would be tangent function) is simply the x–derivative of the function f i.e. $L_{\alpha_x}(f) = \frac{\partial f}{\partial x}$. The linear approximation (the tangent plane) of the non–linear manifold is $f + \alpha_x \frac{\partial f}{\partial x}$, compare with (3.3).

3.3.3 Computation of the tangent vectors

There are two parts left in the presentation of the tangent distance. One of them is to connect the continuous function $f(x, y)$ with the discrete images of the digits. This is the subject of this section. The other part is to present all the transformations that are used in the classification. These transformations are given in the next section.

Thus we need a way of transforming the discrete vector images to continuous functions. Let the pattern $\mathbf{p}$ be given. Reshape this vector to a matrix, as it would be displayed on a screen. Then the pattern could be written as $P[i,j]$ where i and j are integers covering the domain in the xy–plane. For every i and j the value of $P[i,j]$ is given by the corresponding pixel value. This can be written as a single discontinuous function using the delta function $\delta(x)$. The result is

$$P'(x,y) = \sum_{i,j} P[i,j]\delta(x-i)\delta(y-j). \tag{3.15}$$

Convolution of P' with a Gaussian function yields a continuous function f. The Gaussian is the same as in equation (3.9) and the mapping is given by

$$C : P \longmapsto f = P' * g_\sigma. \tag{3.16}$$

This convolution gives the two dimensional continuous function representing the pattern $\mathbf{p}$ as follows

$$\begin{aligned} f(x,y) = P' * g_\sigma(x,y) = \iint P'(\xi,\eta) g_\sigma(x-\xi, y-\eta)\mathrm{d}\xi\mathrm{d}\eta = \\ = \sum_{i,j} P[i,j] g_\sigma(x-i, y-j). \end{aligned} \tag{3.17}$$

This is clearly a continuous function because it is a sum of two–dimensional Gaussians.

The patterns can be made continuous in several ways by doing the mapping C differently or by using some other function instead of the Gaussian function. Simard [14] gives two reasons of why this mapping is favoured: The σ parameter in the Gaussian can be used to control the smoothing factor, very similar to the blurring process. As before blurred and smoothed patterns are preferred because if the pattern is smooth the local approximation (the tangent plane) by the tangent vectors will be valid in a greater interval around the pattern. The second reason for the Gaussian is that when applying the transform operators L_α on the function f the computation of the tangent vector becomes particularly easy. For the case of x–translation the computation is

$$L_{\alpha_x}(f) = \frac{\partial}{\partial x}(P' * g_\sigma) = P' * \frac{\partial g_\sigma}{\partial x} = \sum_{i,j} P[i,j] \frac{\partial g_\sigma(x-i, y-j)}{\partial x}. \tag{3.18}$$

Evaluating this two dimensional function at $x = i$ and $y = j$ over the domain of the original image gives the corresponding tangent vector. This is in matrix form at the moment, but reshaping it gives the tangent vector, and this time it is a vector with the same size as the pattern. The whole computation is a series of multiplications of the pixels from the original pattern and values from the two–dimensional Gaussian taken from appropriate places and a summation.

Now it is time to present the different kinds of transformations.

3.3.4 Important transformations

In total seven different transformations were used. All of them are given in this section together with the transform function, the corresponding operator and an illustration of the transformation. Remember that the aim of these transformations is to make the classification algorithm invariant with respect to the variation inside a given class.

$X-$translation

The transform function that makes the classification algorithm locally invariant with respect to translation along the x–axis is given by

$$t_\alpha : \begin{pmatrix} x \\ y \end{pmatrix} \longmapsto \begin{pmatrix} x+\alpha \\ y \end{pmatrix}. \tag{3.19}$$

The corresponding operator is given by

$$L_X = \frac{\partial}{\partial x}. \tag{3.20}$$

This is the same operator as examplified in section 3.3.2. The action of the operator is illustrated in figure 3.8. The lines drawn in the graphs are meant as support

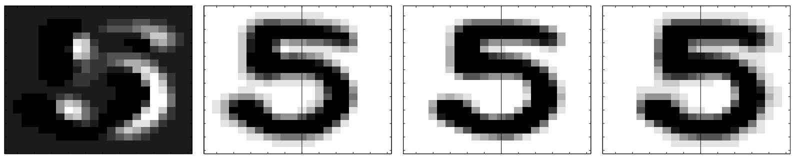

Figure 3.8. The first graph is the image of the x–translation tangent vector. The second and the fourth are translated images of the third one according to $P_{x-tran} = P + \alpha T$ with one negative and one positive α value. P is the third image and T is the tangent vector.

lines to better see the differences between the images.

$Y-$translation

The transform function that makes the classification algorithm locally invariant with respect to translation along the y–axis is given by

$$t_\alpha : \begin{pmatrix} x \\ y \end{pmatrix} \longmapsto \begin{pmatrix} x \\ y+\alpha \end{pmatrix}. \tag{3.21}$$

The corresponding operator is given by

$$L_Y = \frac{\partial}{\partial y}. \tag{3.22}$$

The action of the operator is illustrated in figure 3.9. Support lines are also includet in these graphs.

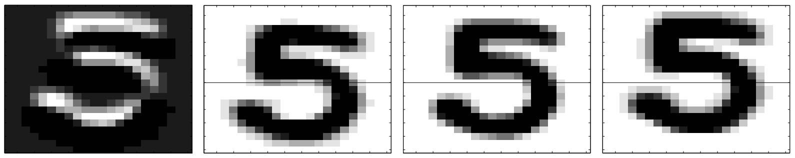

Figure 3.9. The first graph is the image of the y–translation tangent vector. The second and the fourth are translated images of the third one according to $P_{y-tran} = P + \alpha T$ with one negative and one positive α value. P is the third image and T is the tangent vector.

Rotation

The transform function that makes the classification algorithm locally invariant with respect to rotation is given by

$$t_\alpha : \begin{pmatrix} x \\ y \end{pmatrix} \longmapsto \begin{pmatrix} x\cos\alpha - y\sin\alpha \\ x\sin\alpha + y\cos\alpha \end{pmatrix}. \tag{3.23}$$

The corresponding operator is given by

$$L_R = y\frac{\partial}{\partial x} - x\frac{\partial}{\partial y}. \tag{3.24}$$

The action of the operator is illustrated in figure 3.10. Notice that one of the

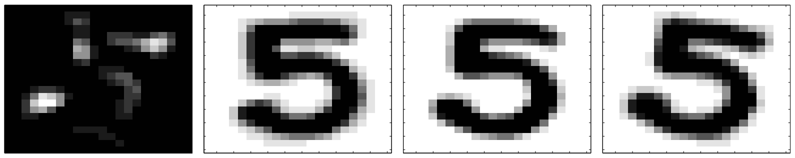

Figure 3.10. The first graph is the image of the rotation tangent vector. The second and the fourth are rotated images of the third one according to $P_{rot} = P + \alpha T$ with one negative and one positive α value. P is the third image and T is the tangent vector.

rotations is clockwise and the other is counter clockwise.

Scaling

The transform function that makes the classification algorithm locally invariant with respect to scaling is given by

$$t_\alpha : \begin{pmatrix} x \\ y \end{pmatrix} \longmapsto \begin{pmatrix} x + \alpha x \\ y + \alpha y \end{pmatrix}. \tag{3.25}$$

The corresponding operator is given by

$$L_S = x\frac{\partial}{\partial x} + y\frac{\partial}{\partial y}. \tag{3.26}$$

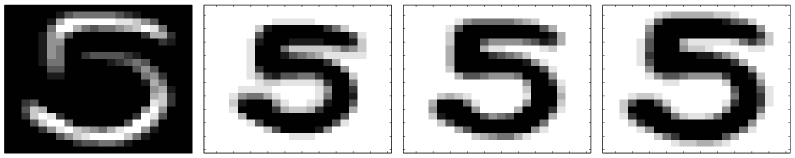

Figure 3.11. The first graph is the image of the tangent vector. The second and the fourth are scaled images of the third one according to $P_{sca} = P + \alpha T$ with one negative and one positive α value. P is the third image and T is the tangent vector.

The action of the operator is illustrated in figure 3.11. We see that the same tangent vectors makes the image both smaller and bigger depending on the sign of α. The same is true for all the other tangent vectors.

Parallel hyperbolic transformation

The transform function that makes the classification algorithm locally invariant with respect to streching is given by

$$t_\alpha : \begin{pmatrix} x \\ y \end{pmatrix} \longmapsto \begin{pmatrix} x + \alpha x \\ y - \alpha y \end{pmatrix}. \tag{3.27}$$

The corresponding operator is given by

$$L_{PH} = x\frac{\partial}{\partial x} - y\frac{\partial}{\partial y}. \tag{3.28}$$

The action of the operator is illustrated in figure 3.12. This operation can be seen

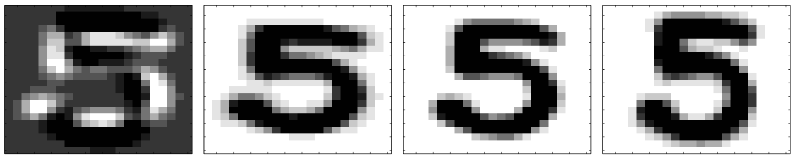

Figure 3.12. The first graph is the image of the tangent vector. The second and the fourth are stretched images of the third one according to $P_{par-hyp} = P + \alpha T$ with one negative and one positive α value. Here P is the third image and T is the tangent vector.

as a stretching along one axis and compression along the other. Changing the sign of α makes the operation in the opposite direction.

Diagonal hyperbolic transformation

The transform function that makes the classification algorithm locally invariant with respect to a second type of streching is given by

$$t_\alpha : \begin{pmatrix} x \\ y \end{pmatrix} \longmapsto \begin{pmatrix} x + \alpha y \\ y + \alpha x \end{pmatrix}. \tag{3.29}$$

The corresponding operator is given by

$$L_{DH} = y\frac{\partial}{\partial x} + x\frac{\partial}{\partial y}. \tag{3.30}$$

The action of the operator is illustrated in figure 3.13. The difference here is that

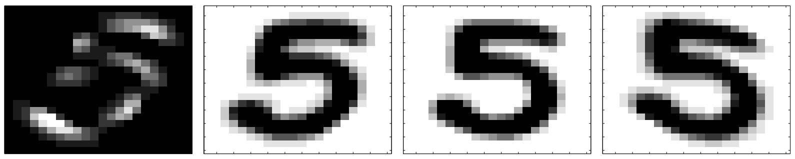

Figure 3.13. The first graph is the image of the tangent vector. The second and the fourth are stretched images of the third one according to $P_{dia-hyp} = P + \alpha T$ with one negative and one positive α value. Here P is the third image and T is the tangent vector.

the stretching and the compression is done along the diagonals.

Thickening

The last transformation is the thickening and thinning transformation. The derivation is a bit different in this case. There is no simple form of the transformation function so it will be omitted. The operator that does the job is presented with some intuitive reasoning. Complete derivation is found in [14]. The thickening and thinning operator is given by

$$L_{TT} = \left(\frac{\partial}{\partial x}\right)^2 + \left(\frac{\partial}{\partial y}\right)^2. \tag{3.31}$$

The action of the operator is illustrated in figure 3.14. Here is the reasoning of why

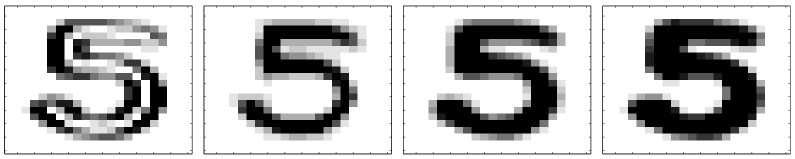

Figure 3.14. The first graph is the image of the tangent vector. The second and the fourth are thinned and thickened images of the third one according to $P_{thick-thin} = P+\alpha T$ with one negative and one positive α value. P is the third image and T is the tangent vector.

this works. We want to build a matrix for a given image that when adding this matrix to the image the sum image gets thicker and when subtracting the matrix from the image we want the image to get thinner. This means that we either increase or decrease the grey levels in the pixels near the boundary of the shape in the image. Using the square of the x–derivative operator we get a matrix that increases the grey levels on the left and right side of the shape. The corresponding y–derivative operator changes the regions above and below the shape. Taking the sum of both operators we get the desired tangent vector, see figure 3.14 again.

3.4 Algorithm 2

The identification process with the tangent distance is actually a nearest neighbour classifier and the whole concept is summarized in the following algorithm.

1. Compute the tangent vectors for the unknown digit.
2. Compute the tangent vectors for all the prototypes.
3. Solve the least squares problems to get the tangent distance between the unknown and the prototypes.
4. Take the label of the prototype that gives the smallest tangent distance as the output of the algorithm.

Notice that this time there is neither training nor test phase in the algorithm. All the digits in the test set are used as prototypes. This means that the whole test set is a part of the algorithm and all the test digits have to be stored and available when using the algorithm. This is one of the major drawbacks of this algorithm. It is very memory demanding and computationally heavy. Performing one–sided tangent distance decreases the amount of computation but makes the performance of the algorithm worse.

Extending this algorithm to be a k–nearest neighbour classifier is straightforward. The only difference is that the label of the majority of the k–smallest tangent distances is taken as the output of the algorithm.

3.5 Approximations and simplifications

The theory behind the tangent distance, described in the previous sections, is very mathematical and not always suited for applications. At least not if it is to be applied strictly. The algorithmic implementation of the tangent distance is very slow due to the extensive amount of computations. Two different methods are presented in the section which highly reduce the amount of computations performed by the algorithm. First, the blurring operator will be approximated and second, numerical derivatives on the patterns will be used instead of the derived operators.

3.5.1 Approximation of the blurring operator

Once again, the blurring of the patterns is very important. The error rate is greatly reduced if the images are blurred before they are classified. The Gaussian function, used in the blurring, is decreasing rather fast. One obvious way to approximate it is to cut the tail off when it is low enough.

The blurring factor is set to $\sigma = 0.9$, the same as Simard used in their experiments. This σ value is considered to give high enough blurring without making any damage to the information in the patterns. It will also be easy to compare the achieved results.

Let us consider the one–dimensional equivalence of the two dimensional Gaussian function in 3.9. This function is plotted in the figure below with $\sigma = 0.9$. Taking the values of the three closest pixels we get $g(1) = 0.54$, $g(2) = 0.085$ and

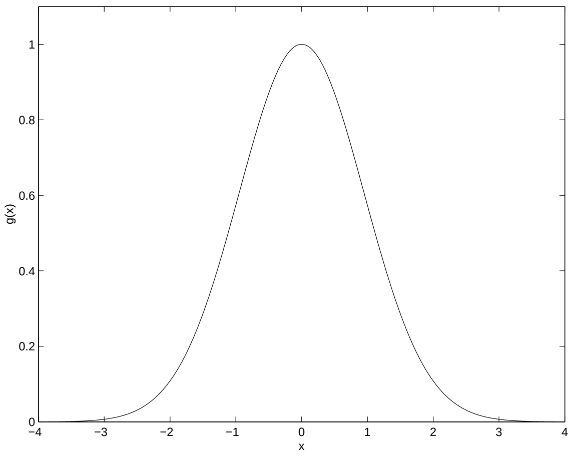

Figure 3.15. Plot of the one dimensional Gaussian $g(x) = e^{-\frac{x^2}{2\sigma^2}}$ with $\sigma = 0.9$.

$g(3) = 0.0039$. We see that already three pixels away the function has decreased considerably.

Consider the Gaussian with two variables. If the complete blurring is done then we know that the Gaussian at every pixel has influence on every other pixel of the image. This is because there is a Gaussian on every pixel and every Gaussian covers the whole xy–plane. A pixel value of the blurred image is the sum of the contributions from every Gaussian. For example if the images are 16×16 pixels the computation of every pixel value (totally 256) involves a summation of 256 terms.

Applying the thought of cutting off the tail to the two–dimensional Gaussian function is the same as letting the Gaussian on a pixel influence only the immediate neighbourhood of the same pixel. Hopefully the error introduced by this approximation is small enough. The fast reduction of the Gaussian function argues that this will work.

Look at figure 3.16. The squares are representing pixels on an image. The coordinates of the square centres are integer valued. Suppose that there is a two-dimensional Gaussian at the origin. The area with the darker squares is the neighbourhood the Gaussian is allowed to influence. The maximum error from the surrounding brighter area is calculated to be 0.0674. This is under the condition that the images are normalized to a maximum pixel value of one. Zero is white

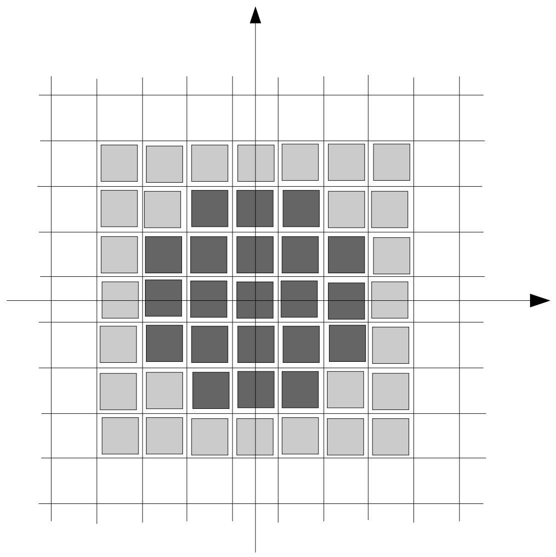

Figure 3.16. The Gaussian influences only the immediate neighbourhood, the area with darker squares, of the pixel at the origin. The maximum error originating from the surrounding brighter area is calculated to be 0.0674.

and one is black. It should be observed that very few pixels are black. Most of the pixels in the image are white and the probability that the whole surrounding area has high pixel values is very low. Remember that the height of the Gaussian over a pixel is given by the pixel value. The error from the surrounding brighter area is probably much lower than the calculated maximum.

Doing this approximation reduces the summation terms to only 21, originally 256. The gain in computation time is enormous. But exactly how good is the approximation? Figure 3.17 gives an example. We can notice a slight difference between the two images. The first one is a bit darker than the second. How good the performance is with the approximated blurring is found in the test section.

3.5.2 Numerical derivatives

The operators for the different tangent vectors are almost in every case a combination of $x-$ and $y-$derivatives. These are theoretically applied to continuous functions corresponding to specific digits. The tangent vectors are calculated through convolving the digits with the derivatives of the Gaussian (3.18). Applying this in an algorithm would be hard and not very effective. The question we asked ourselves was: Why not use numerical derivatives on the patterns as they are? An investigation with a few experiments showed that the operators can be approximated

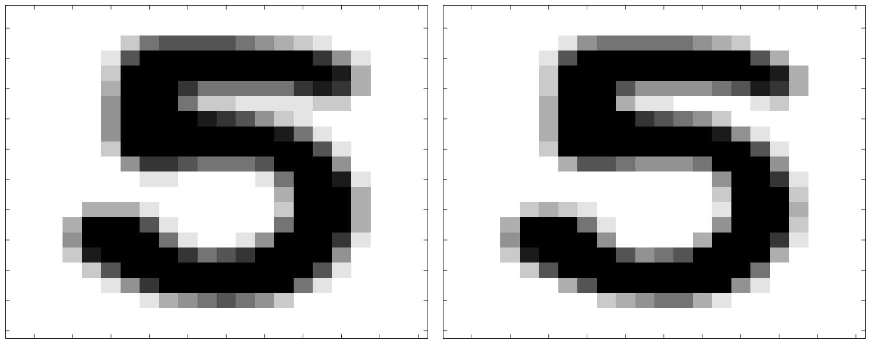

Figure 3.17. The first image is a complete blurring and the second is an approximation of the blurring.

rather well with numerical derivatives.

This idea was implemented with central difference approximation of the derivative [5]. The central difference approximations of the x– and y–derivatives is given by

$$\frac{\partial f(x,y)}{\partial x} \approx \frac{f(x+h,y)-f(x-h,y)}{2h}, \tag{3.32}$$

$$\frac{\partial f(x,y)}{\partial y} \approx \frac{f(x,y+h)-f(x,y-h)}{2h}. \tag{3.33}$$

The h in the equations is the distance between two pixels, which is one, and the function $f(x,y)$ symbolizes the image stored as a matrix. Using the $P[i,j]$ instead of $f(x,y)$ and inserting $h=1$ we get the following numerical derivatives,

$$P_x[i,j] = \frac{P[i+1,j]-P[i-1,j]}{2}, \tag{3.34}$$

$$P_y[i,j] = \frac{P[i,j+1]-P[i,j-1]}{2}. \tag{3.35}$$

The central differences are written as P_x and P_y where P is the image in matrix form as displayed on the screen. We observe that the central difference approximation takes pixel values outside the actual image. Extending the size of the image with one pixel in each direction solves this problem.

The amount of computation for the tangent vectors is the same as for the computation of the blurring of an image (if the original derivation is used). This is because the derivative operation (x– or y–derivatives), applied on the convolved image, can be moved in to act only on the Gaussian, see equation (3.18). In this way the convolution is done with the derivatives of the Gaussian function. The idea of approximating the blurring operator by cutting off the tail of the Gaussian cannot be applied when computing the tangent vectors. The reason for this is the look of the x– and y–derivatives of the Gaussian, they have high function values away from the origin and cutting off the tail of the derivatives is simply not an approximation. This means that the complete convolution has to be performed, at least when computing the tangent vectors. Furthermore, the use of the Gaussian in the tangent vectors incorporates automatically the blurring of an image. The image does not have to be blurred separately before its tangent vectors are computed. If this was done the image would have been blurred twice. Using numerical derivatives is much cheaper in computation compared with the original derivative operations trough convlution. One and important difference is that the blurring is not present in the numerical derivatives and therefore it has to be applied on already blurred images. The whole phase of blurring the images and computing the tangent vectors can be summed as: Blur the image with the approximated blurring operator, compute the numerical $x-$ and y–derivatives of the blurred image and finally form the tangent vectors with the appropriate operations.

3.6 Tests and results

Only two tests were conducted using the tangent distance. In both tests the derived approximation of the blurring and the numerical derivatives were implemented. The algorithm was designed as a nearest neighbour classifier and all the tangent vectors were used with a double–sided tangent distance calculation. The two tests are actually the same but with a different approach.

3.6.1 Test 1

The first test was conducted in two phases. In the first part of the test only ten prototypes were used in the classification. These prototypes were chosen as somewhat ideal from each class in the training set. Reducing the amount of prototypes to one pattern of each class decreases the amount of computations enormously. This can be compared with ten tangent distance computations instead of over 7000 (the total amount of prototypes). The output of the algorithm is the label of the digit that gives the lowest tangent distance. It should be noticed that this is not a realistic approach in reality because one digit together with the invariance properties of the tangent vectors can not cover all the variation in a given class. The only reason for this was to keep down the amount of computations. The performance is not acceptable, as was expected and is given in the table below. The WI in the table

	0	1	2	3	4	5	6	7	8	9	Total
WI	84	3	102	11	117	84	18	13	76	22	530

Table 3.1. Distribution of the incorrectly classified digits in the first phase of the test.

still means wrong identifications. Even though the result is poor it is surprisingly good when taking into account that only 10 prototypes were used. This algorithm is obviously good enough to classify correctly about 75 % of all the images with so few prototypes. Once again twos, fours, fives and zeros are harder to classify correctly. Totally there are 530 wrong identifications in a test with 2007 unknowns.

The second part of this test is only on the 530 digits that were not classified correctly above. This time all the 7291 digits in training set were used as prototypes. The test result is given in table 3.2. The achieved result is very good. The error

	0	1	2	3	4	5	6	7	8	9	Total
WI	4	1	8	7	8	6	1	4	8	1	50

Table 3.2. Incorrectly classified digits in the second phase of the test.

rate is down at 2.5 % if we combine this with the first part of the test. Remember that this approach of dividing the classification in two parts is not possible in reality because there is no way of knowing witch digit is classified correctly and which is not. Thus using this result to compare with presented results in other articles is misguiding.

3.6.2 Test 2

The second test conducted was a complete one. All the digits in the training set were used as prototypes in the classification of all the unknowns from the test set. The result, given in table 3.3, is much more suited for comparison with results from other articles. Totally there are 63 wrongly classified digits. This would mean

	0	1	2	3	4	5	6	7	8	9	Total
WI	10	8	2	3	11	12	2	2	5	8	63

Table 3.3. Distribution of the incorrectly classified digits in the complete test.

that besides the 50 digits from the second part of the first test there are 13 more digits that were incorrectly classified. 63 digits of 2007 is about 3.1 % and still a very good achievement by the algorithm. We can compare this result with 2.5 % error rate in [14] for the same set of digits. The difference can be explained by the blurring approximation and the numerical derivatives implemented in our experiments. This loss in performance can be justified by the gain of computation time due to the made simplifications.

3.7 Analysis and discussions

Some analysis and few comments are presented in this last section of chapter 3. The analysis is a computation of the number of operations needed to perform classification of one unknown digit. Finally an example is given of how this theory is used with other purpose.

3.7.1 Number of operations

The analysis presented here is valid only for the approximated version of the blurring operator and when computing the tangent vectors with numerical derivatives.

Blurring operator

The approximation of the blurring is still used to imitate the blurring with the two dimensional Gaussian function. This can be done as follows. For every pixel compute all the contributions from the surrounding pixels with a weighted sum. The weights in the sum are given by Gaussian function and the other terms are the grey level values of the corresponding pixel. The value for one pixel, for the blurred image, can be calculated with 24 arithmetic operations under the conditions that all the weights are stored. If the size of the image is $I_1 \times I_2$ pixels the complete blurred image requires $24I_1I_2$ arithmetic operations.

Tangent vectors

If the blurred image is computed the following operator computations are required to get all of the tangent vectors:

$$\frac{\partial}{\partial x}, \quad \frac{\partial}{\partial y}, \quad x\frac{\partial}{\partial x}, \quad y\frac{\partial}{\partial x}, \quad x\frac{\partial}{\partial y}, \quad y\frac{\partial}{\partial y}, \quad \left(\frac{\partial}{\partial x}\right)^2, \quad \left(\frac{\partial}{\partial y}\right)^2.$$

The images corresponding to these operators can be computed with $10I_1I_2$ flops. To get the appropriate tangent vectors additional $5I_1I_2$ flops are needed. This gives totally $15I_1I_2$ operations to get all the tangent vectors.

QR solution of least squares problem

Having all the tangent vectors the only thing remaining, to get the tangent distance, is to solve the least squares problem in section 3.2.3. A least squares problem is solved through QR factorisation with $2n^2(m - n/3)$ flops [6], where $m = I_1I_2$ and n the number of tangent vectors (=7) for the single–sided and double the numbers of tangent vectors (=14) for the double–sided tangent distance.

Gathering all the phases we get about $430I_1I_2$ flops for the computation of the tangent distance between two digits.

3.7.2 Enlarging data sets

This theory is also useful in other purposes. Other classification algorithms as neural network classifiers (not discussed previously) are strongly dependent on having sufficient amount of patterns in the training set to get good performing algorithms. If the training set is poor, and there is no way to get labelled patterns, some kind of invariant transformation could be applied on the available patterns. In this way artificial patterns can be produced from the existing patterns to enlarge the training set. It should be noticed that the invariant transformations inside a given class have to be identified and implemented.

Enlarging the training set is not useful in our case because the tangent distance between a given image and all of the transformed images computed with the tangent vectors is zero. The new artificial images do not give any new information to the training set and therefore do not contribute at all to the classification algorithm. This whole theory can be seen in these terms as an algorithm that enlarges a given training set and takes the digit from the enlarged set, which is theoretically infinitely large, that is closest to a given unknown image in Euclidean norm.

Chapter 4

Combinations of TD and HOSVD

The theories behind the algorithms, described in the previous two chapters, are quite different. The memory usage in the HOSVD–algorithm is not big. The set of the training digits is reduced to the orthogonal basis matrices for each class. If ten basis matrices are used for each class and there are ten classes, the total number of image–sized matrices, needed to be stored in the algorithm, is 100. This can be compared with approximately 7000 digits for the USPS database and 10 000 digits for the MNIST database in the tangent distance classifier, where all of the digits in the training set are stored in the algorithm. Taking into account that each digit consists of 256 floating–point numbers (or digits) one concludes that the reduction of the memory usage in the HOSVD–algorithm is enormous. Considering the performance of both algorithms it is clear that the TD classifier does a better job.

In this chapter we present two different methods of combining the theory of the two algorithms. The first one incorporates the tangent vectors in the HOSVD algorithm. The second incorporates the HOSVD in the tangent distance classifier. Both methods are tested with the USPS database and finally some analysis and comments are given.

4.1 HOSVD with tangent vectors

The idea for the first combination is the following: For a given digit, compute its tangent vectors and consider the original digit together with its tangent vectors as one object in a high dimensional vector space. Proceed with the classification in the same way as described in chapter 2. This time the objects are not matrices but third order tensors (or block tensors). The theory is the same as before but there are obvious modifications in the implementation. Some of the details are given in the subsequent section.

4.1.1 Implenting HOSVD with tangent vectors

There are principally two different ways of computing the set of a basis matrices, which are actually not equal but both are orthogonal and that is the main requirement. Both methods are given below.

Method 1

One straightforward way of implementing the idea given above is to form the third order tensors. Consider the four together with its tangent vectors reshaped to matrices in figure 4.1. Those matrices are used to form a third order tensor in

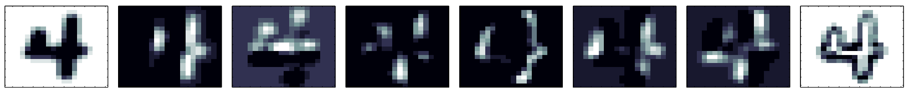

Figure 4.1. The images of a four and its tangent matrices.

which the tangent matrices are the slices of the tensor. This is illustrated in figure 4.2. The whole third order tensor in the figure is considered as one object. If the

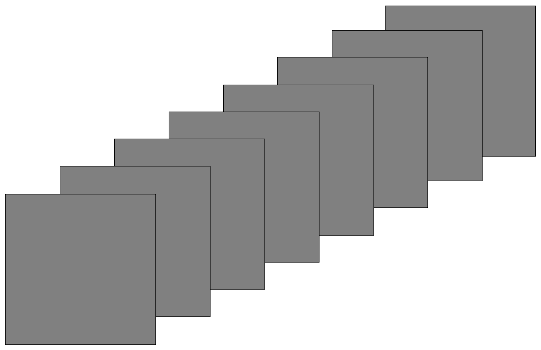

Figure 4.2. Illustration of the third order tensor formed by the tangent vectors and the original image.

tangent vectors for all the digits in the training set are computed and reshaped to third order tensors then we can build a forth order tensor. Assuming the size of the matrices to be 16×16 and there are seven tangent matrices we get a third order tensor with the dimensions $16 \times 16 \times 8$. Each of these tensors corresponds to one digit. If we have 100 digits then the forth order tensor would have the dimensions $16 \times 16 \times 8 \times 100$.

The first step in the HOSVD algorithm was to gather all the digits from a given class and build a third order tensor. Applying this first step here gives a forth order tensor. The dimension along the forth mode is the number of digits from the same class. This forth order tensor is used as a source tensor for the HOSVD decomposition. From this decomposition the basis matrices are calculated but here they are not matrices, they are third order tensors as the actual objects. A set of basis, consisting of third order tensors, is calculated for every class. The next step is to solve the least squares problems with the bases for each class. Also here is a need for minor modifications to handle the tensor structure in the minimization problem. The solution to the least squares problem is the same as before but the scalar product is taken between tensors and not matrices. The output of the algorithm is given as earlier.

To implement this algorithm, as described above, it would be necessary to redesign almost all of the Matlab functions constructed in chapter 2. This was not carried out. Instead the third order tensors were reshaped to matrices and those matrices were used in the classification process precisely in the same way as in chapter 2. The difference is that the matrices now are not representing a single digit but all the eight matrices given in figure 4.1. By calculating the basis in this way no new functions were needed. Observe that reshaping the basis matrices to third order tensors gives the same end result as one would have obtained if the algorithm was implemented to forth order tensors.

Method 2

Consider the digits as matrices and assume that HOSVD is performed as in chapter 2. Calculate one of the tangent vectors for the digits in the source tensor used in the HOSVD. Form a new third order tensor, which consists of one of the tangent vectors of the corresponding digits from the source tensor. Compute the HOSVD decomposition and a set of orthogonal basis matrices of the tensor with the tangent vectors. For every kind of tangent vector a third order tensor is build corresponding to the digits from the original source tensor. Computing the HOSVD decomposition for these tensors yields a set of orthogonal basis matrices for each kind of tangent vector.

Now the basis for the different tangent vectors and the digits are orthogonal. If we build a third order tensor with the first basis matrix for every kind of tangent vector and the first basis matrix for the digits we obtain a tensor of the same size as in figure 4.2. This tensor is set to be the first basis tensor. We continue building basis tensors by taking the second basis matrices for the different kinds of tangent vectors and the second basis matrix for the digits, the third basis matrices for the tangent vectors and the digits, the fourth basis matrices, the fifth basis matrices and so on.

The orthogonal property for tensors is given by the scalar product in definition 2.3. It is clear that each slice in the basis tensors is orthogonal to the slices in the same positions from all the other basis tensors and thereby the whole basis tensors are orthogonal. This is not true for the basis tensors calculated as in method 1, the slices in the same positions in those basis tensors are not orthogonal but the whole tensors are orthogonal. This property can be described in terms of matrices in this manner: Consider two matrices $A = [\mathbf{a}_1\ \mathbf{a}_2\ \dots\ \mathbf{a}_n]$ and $B = [\mathbf{b}_1\ \mathbf{b}_2\ \dots\ \mathbf{b}_n]$

in which the $\mathbf{a}_i$ and $\mathbf{b}_i$ are the respective column vectors. Assume that A and B are orthogonal, i.e. $\langle A, B \rangle = 0$. This corresponds to the relationship in method 1. The difference to method 2 is that not only the matrices are orthogonal but also the column vectors from A and B with the same index are orthogonal, i.e. $\mathbf{a}_i \cdot \mathbf{b}_i^T = 0$ for all i.

Having orthogonal basis matrices one can continue with the least squares problems in the same way as in method 1. But this is not the only way. The least squares problems can be solved separately for every digit and every tangent vector in exactly the same way as in chapter 2. Proceeding in this way does not necessarily give better performance of the algorithm. The minimum of the least squares problem for the digits already gives a first output alternative. If we suppose that this output is incorrect, it would be required of the algorithm that the least squares problems, belonging to the tangent vectors, give a different output. The contribution of these least squares problems has to be in the right direction so that the complete minimum accurse for some other class, that is potentially the right one. It is not likely that solving the same problem for all the different tangent vectors gives different answers because all of the tangent vectors are computed from the same digits. This whole process can be seen as different preprocessings of the digits before the classification is made.

Test results show that this method of incorporating the tangent vectors into the HOSVD algorithm barely has any effect on the algorithm performance. In other words the extra effort made in this section is redundant. Because of this the algorithmic formulation will be omitted.

4.2 Tangent distance with HOSVD

The combination described in the previous section incorporates the tangent vectors in the HOSVD algorithm without measuring the tangent distance. The combination can be done the other way around. In this section we incorporate the HOSVD in the tangent distance classifier. By this we mean that instead of storing all the digits in the training set we compute and store the basis matrices of each class as in chapter 2, solve the least squares problem for every class. This gives ten virtual digits, each one of them is a linear combination of the basis matrices for the different classes. Finally the tangent distance is calculated between an unknown digit and all ten virtual digits. The class of the digit that gives the smallest tangent distance is the output of the algorithm. It turns out that this method can be an alternative to the algorithms described in chapter 2 and 3.

4.2.1 Implementing TD with HOSVD

The implementation is straightforward since no new theory is required and all the necessary functions are already constructed. Let $\mathbf{A}_i^j$ with $j = 1, \ldots, 10$ and $i = 1, \ldots, k$ be the same basis matrices as in chapter 2. Remember that $\mathbf{A}_i^j$ is the *ith* basis matrix for the *jth* class. k is the number of basis matrices for each class and is a design variable needed to be determined.

Next step is to solve the least squares problems:

$$\min_{\alpha_i^j} \|\mathbf{X} - \sum_{i=1}^{k} \alpha_i^j \mathbf{A}_i^j\|_F, \qquad j = 1, \ldots, 10. \tag{4.1}$$

Remember that $\mathbf{X}$ is the unknown digit to be classified. The solutions to the minimization problems above give the coefficients α_i^j in the linear combination with the basis matrices. The virtual digits, denoted $\mathbf{Z}^j$, are given by

$$\mathbf{Z}^j = \sum_{i=1}^{k} \alpha_i^j \mathbf{A}_i^j, \qquad j = 1, \ldots, 10. \tag{4.2}$$

All the tangent vectors are computed for every $\mathbf{Z}^j$ and the last step is to calculate the tangent distances, $d_{TD}(\mathbf{X}, \mathbf{Z}^j)$, between $\mathbf{X}$ and the $\mathbf{Z}^j$. The output is determined from the smallest tangent distance.

The details for all of these operations are found in the previous two sections.

4.3 Algorithm 3

The process described in the previous section can be formulated as an algorithm in the following way.

- Training phase:

  1. Collect the digits from the training set into tensors with digits of the same type.
  2. Compute the HOSVD of these tensors.
  3. Compute and store the basis matrices. This is a data compression.

- Test phase:

  1. Solve the least squares problem for each set of basis matrices, i.e. compute the scalar products $\langle X, \mathbf{A}_i^j \rangle = \alpha_i^j$ for all i and j.
  2. Compute the virtual digits $\mathbf{Z}^j$.
  3. Compute the tangent vectors for the unknown and for the virtual digits.
  4. Solve the least squares problems to get the tangent distances between the unknown and the virtual digits.
  5. Take the label of the virtual digit that gives the smallest tangent distance as the output of the algorithm.

Notice that the training phase is the same as in chapter 2. Also the test phase can be distinguished from the algorithmic formulations in the previous chapters.

4.4 Tests and results

The conducted tests in this chapter were not so extensive as in the previous chapters. But they are sufficient enough to get an idea of how well the algorithm performs and draw some conclusions.

4.4.1 Test 1

The first tests were an implementation, described in method 2 previously, of the combination of the tangent vectors with the HOSVD decomposition. In a first test ten basis tensors for each class were used. The original digits were also blurred before the tangent vector computations. The basis tensors were computed from third order tensor objects consisting of the blurred images (not the original) together with their tangent vectors.

The algorithm gave 118 (error rate of 5.88 %) incorrect classifications. This can be compared to 137 (error rate of 6.83 %) incorrect classifications in chapter 2 when ten basis matrices were used. There is a small improvement in the algorithm with incorporated tangent vectors.

A closer investigation showed that this improvement in performance is not originating from the tangent vectors. It originated from the use of blurred digits instead of the original digits. A second test was conducted only using the blurred images, i.e. without the tangent vectors. The test is the same as the tests in chapter 2. This time there were 119 (error rate of 5.93 %) incorrect classifications. The contribution of the tangent vectors is barely noticeable.

4.4.2 Test 2

The conducted tests when HOSVD is incorporated in the TD are presented in this section. The tests are implemented according to algorithm 3 in section 4.3. Ten tests were carried out. The number of basis matrices, when computing the virtual digits, was varied in the tests. The more basis matrices that are used the better approximations are the virtual digits of the unknown digit. The test results are given in table 4.1.

NBM	2	3	5	7	8	9	10	12	15	20
WI total	155	153	138	115	116	112	109	112	111	125
WI in %	7.72	7.62	6.88	5.73	5.78	5.58	5.43	5.58	5.53	6.64

Table 4.1. The total number of wrong identifications in the different tests.

NMB stands for number of basis matrices and WI stands for wrong identifications. First thing to notice in the results is that the performance is not as good as in chapter 3. This is not strange, because here we skip the whole training set and instead use virtual digits. The basis matrices, forming the virtual digits, obviously cannot replace the whole training set without loss of quality. But compared to the results in chapter 2 we have an improvement. Another positive effect is that good result is obtained with very few basis matrices. The results show that the performance gets better when increasing the number of basis matrices up till ten. Further increase of the number of basis matrices does not seem to give better performance.

The result 109 WI, when using ten basis matrices, can be compared to the second test in the preceding section with 119 WI, in which also ten basis matrices were used besides that the digits were blurred. This comparison argues that the latter method with tangent distance is a bit better.

4.5 Analysis and comments

This part of the chapter is intended to give further insight and some comments to the algorithms treated here.

4.5.1 Investigation of the virtual digits

We start with investigating how good the virtual digits are. They are not real digits, they can be seen as vectors of a subspace, for some class, spanned by the respective basis matrices. The virtual digits are the best approximations, in terms of the bases of the different classes, to a given unknown. So in a way, the one virtual digit that best describes the unknown is an alternative output. This is done in the HOSVD algorithm. But we proceed with the tangent distance because it has better performance. If we get the right answer at once there is no need to compute the tangent distances between the unknown and the virtual digits. The tangent distance is needed in approximately 10 % of all classifications. These are the incorrectly classified digits and the tangent distance may correct the answer for half of them. In other words we end at approximately 5 % incorrect classifications. So much extra work for so little feedback one might think. The tangent distance has to be calculated in all cases because there is no way of knowing which digits are wrongly classified.

We will investigate two digits, one written rather well, the other written badly. The digits are given in figure 4.3. The blurred versions are also given because the

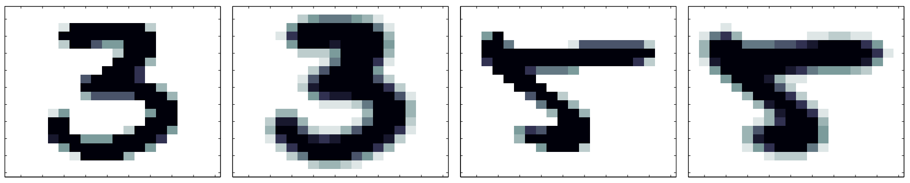
Figure 4.3. Two digits together with their blurred versions. The three is rather well written and the five is not.

virtual digits are actually approximating them and not the original digits.

The solution of equation (4.1) gives the α_i^j constants and thereby enables the computation of the virtual digits, $\mathbf{Z}^j$. A whole series of virtual digits was calculated for both of the digits. All virtual digits are found in figure 4.4 and figure 4.5. The first figure belongs to the three and the second figure belongs to the five. The virtual digits were calculated with three different number of basis matrices, namely five,

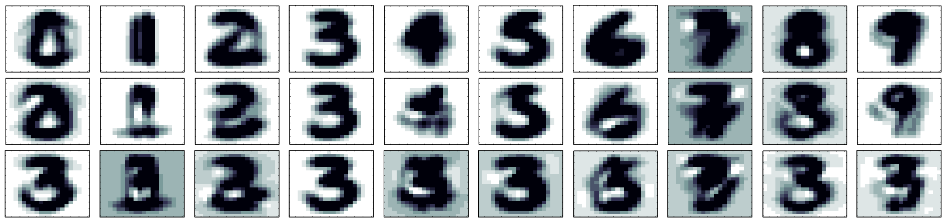

Figure 4.4. Virtual digits belonging to the well written three. The first row is calculated with five basis matrices, the second row with ten and the third row with 20.

ten and 20. There are ten digits in each row, which were computed with the basis matrices for the respective classes. This means that the digits in the same column are calculated with the basis matrices for the same class. The basis matrices for

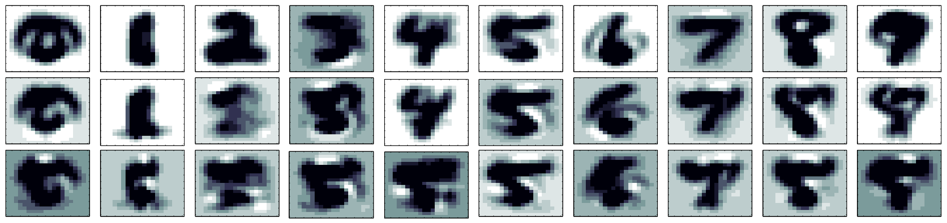

Figure 4.5. Virtual digits belonging to the badly written five. The first row is calculated with five basis matrices, the second row with ten and the third row with 20.

the zeros are used for the first column, the basis matrices for the ones are used for the second column and so on until the last, tenth column in which the basis matrices for the nines are used. This is clearly seen in the first rows of both figures.

First of all we see that increasing the number of basis matrices the virtual digits approximate the unknown digits better and better for all the classes. Examine the third row in figure 4.4. There are several images that are approximating the three very well, besides the one that is built with the basis matrices for the threes. The images in the middle row are starting to get the shape of the three but one can still observe what kind of basis matrices that are used. It is most obvious in the first row which basis matrices are used, but there is no resemblance with the three. In this case there is no need to use many basis matrices because even with few basis the three is described well.

The same thing is not true for the badly written five. Here we need many basis matrices to get good resemblance with the five. The reason for this is that the first basis matrices are not suited to describe non–ideal fives. None of the first basis are suited to describes non–ideal digits. We can observe that the resemblance is increasing for each row in figure 4.5. Not only for the image built with the basis for the fives but also for the images built with the basis for the fours and eights.

4.5.2 Number of operations

Algorithm 3 described in section 4.3 is obviously more expensive, in computations that is, than the pure HOSVD algorithm. But on the other hand it performs better. The question is if the better performance is outweighing the extra computations. We can also conclude that the pure tangent distance classifier is much more expensive compared to all other algorithms. The conclusion is drawn from the fact that in a pure TD classifier thousands (all the digits in the training set) tangent distances are calculated to be compared with only ten (the virtual digits) tangent distances each one belonging to a different class.

Assuming the tangent vectors are calculated as in chapter 3 the only design variable is the number of basis matrices. Since there are no new phases in the algorithm all the necessary operations are given in the previous chapters.

Training phase

First of all the basis matrices need to be computed. This part is exactly the same as in section 2.5.5 and will be omitted here.

Test phase

Assume that the algorithm uses k basis matrices when calculating the virtual images. The test phase can be split up in two parts: computation of the virtual digits and computation of the tangent distances. Assume also that the size of the images is $(m \times m)$ and that there are 10 classes. The calculation of the virtual digits is done in approximately

$$10(2km^3 + 2km^2) = 20km^2(m+1)$$

flop counts.

The calculation of the tangent distance can be done with approximately $430m^2$ flops. This is derived in section 3.7.1 and is for the computation of one tangent distance. Remembering that there are ten virtual digits and therefore ten tangent distances are computed we get the total number of operations for the whole test phase to be

$$20km^2(m+1) + 4300m^2.$$

Chapter 5

Comparison with Other Algorithms, Summary and Conclusions

Before the summary and the conclusions we will present, very shortly, few other classification algorithms. They will not be analysed at all, only an intuitive picture of how they work will be given.

5.1 Comparison with other algorithms

First of all we point out that there is an abundance of classification algorithms. Most of them have the same starting point but differ in the way the implementation is solved. The most usual starting point is the following. It is assumed that all objects belong to some kind of space. The objects that belong to the same class are gathered in that space. They are closer to each other than to objects from other classes, in some sense that is. This point of view has already been presented in this report, but where we go from here differs.

Furthermore, it is assumed that there are, at least theoretically, clear boundaries between the different classes. The problem is to find functions that separate the classes. Those functions will then give the boundaries in question and classification can be done. To find these functions is a very hard problem. A usual way of continuing is to form the boundaries with the help of elements that we already know to which class they belong. This would be the training set.

Statistical methods

Statistical methods can be used to approximate the boundaries. But to get good approximations the number of objects in the training set have to be very high. The more objects there are the better the approximation becomes and also the performance. But there are certain questions that need to be answered such as: How big training set is needed to get a given accuracy? Do I have training set of that size? If not how can we generate more training samples? These questions are not trivial. Detailed information about this method is found in [13], [7].

Neural networks

Another way to get an approximation of the boundaries is to use neural networks. Neural networks is a very big field and can be used for many purposes. Classifications is just one of them. The idea of neural networks is to imitate the way nervecells in the brain are working. It can be seen as a black box with many input and output channels. Different output channels or a combination of them is activated depending on the activation on the input channels. There are a number of variables or trainable parameters (hundreds, thousands or even ten thousands depending on the complicity of the problem) in that black box that can be changed. Changing the parameters also changes the output activations. Through those parameters any given boundary, surface or function can be expressed. The neural network can thereby also be seen as a function $f_{\boldsymbol{\alpha}}$ given by

$$f_{\boldsymbol{\alpha}} : \mathbb{R}^m \longrightarrow \mathbb{R}^n, \tag{5.1}$$

were m is the number of inputs and n is the number of outputs. The function itself is characterized by the parameters, which are contained in $\boldsymbol{\alpha}$.

The trainable parameters in neural networks are initially set to some default values. These are eventually changed in an iterative process by using the training set, often referred to as a learning process. This process is mainly characterized by letting the samples in the training set pass through the network. Knowing the wanted output for a given sample the actual output is observed and if necessary the parameters are updated to get another, satisfying, output for the same sample. The whole training set is passed through the network in each iteration. Because of this neural networks are highly dependant on the training set. All possible variations for all the classes have to be contained in the training set, which is not realistic. Therefore to get good performance very large training sets are required.

The flexibility of neural networks is a great advantage. The former black box can be customized or be given a special architecture for a particular problem. The architecture is the actual structure of the network. There is much more to say about neural networks but the line must be drawn at some point. For detailed description of the wide possibilities for neural networks the reader is referred to books on this topic [13], [7], [11].

Test results

There are many neural network classifiers with different architectures for handwritten digit recognition. These algorithms are most often tested on the digits from the MNIST database. The best neural networks have an error rate of approximately 1 %, some even below. Those networks have high complexity and custom made architecture. Other standard neural network classifiers have an error rate of up to 4.5 % – 4.7 %. These networks are rather simple and do not have so many trainable parameters. Other algorithms with low error rate are the tangent distance classifier ≈ 1.1 % and support vector machines (SVM) ≈ 1 %. Nearest neighbour algorithms have an error rate of approximately 5 %. All results are taken from an article of LeCun [11]. For more information about the tests, test results and the different classifiers the reader is referred to that article.

These results can be compared with approximately 3 % error rate for the HOSVD–algorithm when using 20 basis matrices, figure 2.9. The MNIST database was not used in the evaluation of the other algorithms. If we take into account that the other algorithms (our TD–classifier and TD with HOSVD) in this report actually do perform better than the HOSVD–algorithm and also that the USPS database is harder to classify than the MNIST database then we might have an error rate lower than 2 %. Of course these are pure speculations. But still 3 % in error rate is not bad.

To skip speculations here we present similar results for the USPS database from an other article [9]. The error rate for the different algorithms is as follows:

Human Performance	2.5 %
Neural Network	4.2 %
Support Vector Machine	3.0 %
Tangent Distance	2.5 %
Extended TD	2.4 %

After all the 3 % error rate in our complete TD–classification is fully acceptable. Especially if we take into account the simplifications we made by the approximation of the blurring operator and the numerical derivatives. The loss in performance is obviously acceptable. If we compare these results with the achievements in chapter 4, especially for the TD with HOSVD algorithm, which had a best error rate of approximately 5.5 %, we can conclude that it does not reach to the itemized results above. Even though, this achievement is still rather good.

Memory usage and amount of computations

In an application the actual performance is not the only important factor. Other important factors are memory usage and amount of computations. The TD–classifiers do have large memory requirements and rely on extensive computations. The methods with HOSVD have low memory requirements and moderate amount of computation. This is because the HOSVD is making a compression of the training set. Neural network algorithms are hard to build and they are time demanding in the construction process due to the iterative learning process. When the algorithm is ready it is very fast in performing the actual classification in the test phase. The memory usage for neural networks is moderate but can be high if many parameters are used and the problem is complex. Having this in mind the method of TD with HOSVD may very well be an option to be considered.

5.2 Summary and conclusions

In this report several classification algorithms were thoroughly analysed and tested. Three of them are especially interesting. Namely the classification algorithm by singular value decomposition of tensors, described in chapter 2, the tangent distance classifier, chapter 3 and the tangent distance classifier with incorporated higher order singular value decomposition, chapter 4. A brief summary with conclusions is following for each one of these methods.

HOSVD–classifier: A set of orthogonal basis matrices is calculated for every class. This is done by HOSVD of third order tensor containing the digits of the same class from the training set. By doing this an actual data compression is also done because the training set is reduced to the basis matrices for the different classes. The class of the set of basis matrices that describes an unknown digit as a linear combination in the best way is also giving the label of the unknown as an output of the algorithm. The best linear combination is determined by solving least squares problems, one for each class. The solution is particularly easy because of the orthogonality of the basis matrices. This algorithm has moderate performance, an error rate of 6 % (USPS) and 3 % (MNIST) in its best performance. It has low memory requirements and demands rather low amount of computations.

TD–classifier: The idea of this classifier is to incorporate different kind of invariant transformations, for digits within the same class, into a distance measure. Rotation, translation and scaling are examples of such transformations. These transformations were handled with the help of the tangent vectors, who are derived trough differential geometry. To get the tangent distance also here a least squares problem was solved. The TD was used in a simple nearest neighbour classifier, which computes the distances between an unknown digit and all the digits in the training set and takes the label of the digit that has the smallest distance to the unknown as an output for the algorithm. This algorithm has very good performance, error rate of 3 %, but is computationally heavy and requires large memory usage.

TD with HOSVD: In chapter 4 the theories for TD and HOSVD are combined. Several variations can be built but only one is of interest. It uses TD distance but here between the unknown digit and the ten virtual digits. The virtual digits are the best approximations to the unknown digit in terms of linear combinations of the basis matrices for the different classes. I.e. HOSVD is performed to get the basis matrices for every class. In this way only the basis matrices are needed to be stored, instead of the whole training set. The algorithm performs better than the HOSVD–classifier but not as good as the TD–classifier. On the other hand the algorithm has low memory requirements and not too extensive amount of computations.

It can be concluded that rather good and simple algorithms can be constructed with some, not too extensive, work. Any of the algorithms above, espcially the last one, can be considered for use in a real application.

Bibliography

[1] Rikard Berthilsson. Character recognition. Technical report, Center for Mathematical Sciences, Lund University, Box 118, 221 00 Lund, Sweden.

[2] Bart de Moor, Lieven de Lathauwer, and Joos Vendewalle. A multilinear singular value decomposition. *Siam Matrix Anal. Appl.*, 21(4):1253–1278, 2000.

[3] Lars Eldén. Data compression of multi-way arrays by least squares. Manuscript, March 2002.

[4] Lars Eldén. Decomposition of a 3-way array of data from the electronic nose baseline–corrected data. Manuscript, March 2002.

[5] Lars Eldén and Linde Wittmeyer-Koch. *Numerisk analys - en introduktion.* Studentlitteratur, 3rd edition, 1996. Swedish book.

[6] G. Golub and C. Van Loan. *Matrix Computations.* Johns Hopkins University Press, 3rd edition, 1996.

[7] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning.* Springer-Verlag New York, 2nd edition, 2001.

[8] Geoffrey E. Hinton, Peter Dayan, and Michael Revow. Modelling the manifolds of images of handwritten digits. Technical report, Department of Computer Science, University of Toronto, Canada, 1997.

[9] Daniel Keysers, Jörg Dahmen, Thomas Theiner, and Hermann Ney. Experiments with an extended tangent distance. In *Proceedings 15th International Conference on Pattern Recognition*, volume 2, pages 38–42, Barcelona, Spain, September 2000.

[10] Per-Ola Kristensson. Design and evaluation of a shorthand aided soft keyboard. Master's thesis, Linköpin University, August 2002.

[11] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document reconition. *IEEE*, November 1998.

[12] Shunji Mori, Hirobumi Nishida, and Hiromitsu Yamada. *Optical Character Recognition.* John Wiley Sons, 1999.

[13] Richard O.Duda, Peter E. Hart, and David G. Stork. *Pattern Classification*. John Wiley Sons, 2nd edition, 2001.

[14] Patrice Y. Simard, Yann A. Le Cun, John S. Denker, and Bernard Victorri. Transformation invariance in pattern recognition – tangent distance and tangent propagation. 2000.

[15] Svante Wold. Pattern recognition by means of disjoint principal components models. 8:127–139, March 1975.

[16] Åke Björck and Germund Dahlquist. Numerical methods and scientific computations. Volume 2, Working copy, August 2001.

Appendix A

Proofs and Definitions

This section is intended to give proofs of greater importance, definitions and other specific details that are omitted for simplicity reasons in the report.

A.1 Tensor representation

Exact relation between a tensor and the matrix representing its unfoldings is given in the definition below.

Definition A.1 *An arbitrary Nth order tensor $\mathbf{A} \in \mathbb{C}^{I_1 \times \cdots \times I_N}$ and its unfolding $\mathbf{A}_{(n)} \in \mathbb{C}^{I_n \times (I_{n+1} I_{n+2} \dots I_N I_1 \dots I_{n-1})}$ are related in this specific way: Each element from $\mathcal{A}$ with index $(i_1, i_2, ..., i_N)$ is placed in $\mathbf{A}_{(n)}$ with row index i_n and column index*
$(i_{n+1} - 1) I_{n+2} I_{n+3} \dots I_N I_1 I_2 \dots I_{n-1} + (i_{n+2} - 1) I_{n+3} I_{n+4} \dots I_N I_1 I_2 \dots I_{n-1} + \dots$
$+ (i_N - 1) I_1 I_2 \dots I_{n-1} + (i_1 - 1) I_2 I_3 \dots I_{n-1} + (i_2 - 1) I_3 I_4 \dots I_{n-1} + \cdots + i_{n-1}$.

A.2 The n–mode product of a tensor

The general definition of the n–mode product is as follows.

Definition A.2 *The n–mode product of a third order tensor $\mathcal{A} \in \mathbb{C}^{I_1 \times I_2 \times \cdots \times I_N}$ by a matrix $\mathbf{F} \in \mathbb{C}^{J_n \times I_n}$, denoted by $\mathcal{A} \times_n \mathbf{F}$, is a $(I_1 \times \cdots \times I_{n-1} \times J_n \times I_{n+1} \times \cdots \times I_N)$–tensor. Letting $\mathcal{B} = \mathcal{A} \times_n \mathbf{F}$, the entries of $\mathcal{B}$ are given by folding $\mathbf{B}_{(n)}$, where $\mathbf{B}_{(n)} = \mathbf{F} \cdot \mathbf{A}_{(n)}$.*

A.3 Proof of Theorem 2.2 – HOSVD

Proof of the HOSVD–theorem presented here is for third order tensors only. The more general proof is in complete analogy with this one.

First of all we make the following observation: Theorem 2.2 can be represented with matrices by using the unfolding of $\mathcal{A}$ and $\mathcal{S}$ along any mode. Thus equation (2.9) is equivalent to the matrix equation

$$\mathbf{A}_{(n)} = \mathbf{U}^{(n)} \cdot \mathbf{S}_{(n)} \cdot (\mathbf{U}^{(n+1)} \otimes \mathbf{U}^{(n+2)} \otimes \cdots \otimes \mathbf{U}^{(N)} \otimes \mathbf{U}^{(1)} \otimes \cdots \otimes \mathbf{U}^{(n-1)})^T. \quad \text{(A.1)}$$

The $\otimes$ symbol denotes the Kronecker product of two matrices wich is defined in this way.

Definition A.3 *For arbitrary two matrices*

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix} \quad \textit{and} \quad \mathbf{B} = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1q} \\ \vdots & \vdots & & \vdots \\ b_{p1} & b_{p2} & \dots & b_{pq} \end{pmatrix},$$

the Kronecker product is given by

$$\mathbf{A} \otimes \mathbf{B} = \begin{pmatrix} a_{11}B & a_{12}B & \dots & a_{1n}B \\ \vdots & \vdots & & \vdots \\ a_{m1}B & a_{m2}B & \dots & a_{mn}B \end{pmatrix}.$$

Proof *HOSVD–theorem* The aim of this proof is to show that for every third order tensor $\mathcal{A} \in \mathbb{C}^{I_1 \times I_2 \times I_3}$ there are unitary matrices $\mathbf{U}^{(i)}$ and a tensor $\mathcal{S}$ with the relation and properties as stated in the theorem. Matrix SVD on the unfoldings $\mathbf{A}_{(i)}$ gives

$$\mathbf{A}_{(i)} = \mathbf{U}^{(i)} \cdot \Sigma^{(i)} \cdot \mathbf{V}^{(i)^T} \qquad \text{for} \qquad i = 1, 2, 3. \quad \text{(A.2)}$$

Define now the n–mode product

$$\mathcal{S} = \mathcal{A} \times_1 \mathbf{U}^{(1)^H} \times_2 \mathbf{U}^{(2)^H} \times_3 \mathbf{U}^{(3)^H}, \quad \text{(A.3)}$$

and examine $\mathcal{S}$. Unfolding $\mathcal{S}$ along the first mode we get

$$\mathbf{S}_{(1)} = \mathbf{U}^{(1)^H} \cdot \mathbf{A}_{(1)} \cdot (\mathbf{U}^{(2)} \otimes \mathbf{U}^{(3)}) =$$

$$= \mathbf{U}^{(1)^H} \cdot \mathbf{U}^{(1)} \cdot \Sigma^{(1)} \cdot \mathbf{V}^{(1)^H} \cdot (\mathbf{U}^{(2)} \otimes \mathbf{U}^{(3)}) =$$

$$= \Sigma^{(1)} \cdot \mathbf{V}^{(1)^H} \cdot (\mathbf{U}^{(2)} \otimes \mathbf{U}^{(3)}),$$

and

$$\mathbf{S}_{(1)} \cdot \mathbf{S}_{(1)}^H =$$

$$= \Sigma^{(1)} \cdot \mathbf{V}^{(1)^H} \cdot (\mathbf{U}^{(2)} \otimes \mathbf{U}^{(3)}) \cdot (\mathbf{U}^{(2)} \otimes \mathbf{U}^{(3)})^H \cdot \mathbf{V}^{(1)} \cdot \Sigma^{(1)^H} =$$

$$= \Sigma^{(1)} \cdot \mathbf{V}^{(1)^H} \cdot \mathbf{V}^{(1)} \cdot \Sigma^{(1)^H} = \Sigma^{(1)} \cdot \Sigma^{(1)^H} =$$

$$= \operatorname{diag}((\sigma_1^{(1)})^2, (\sigma_2^{(1)})^2, \dots, (\sigma_{I_1}^{(1)})^2).$$

The same computations along the other modes give us similar diagonal matrices

$$\mathbf{S}_{(2)} \cdot \mathbf{S}_{(2)}^H = \mathrm{diag}((\sigma_1^{(2)})^2, (\sigma_2^{(2)})^2, \ldots, (\sigma_{I_2}^{(2)})^2),$$

$$\mathbf{S}_{(3)} \cdot \mathbf{S}_{(3)}^H = \mathrm{diag}((\sigma_1^{(3)})^2, (\sigma_2^{(3)})^2, \ldots, (\sigma_{I_3}^{(3)})^2).$$

It is clear that the raw vectors and the columns of $\mathbf{S}_{(i)}$ are orthogonal. This means that the subtensors of $\mathcal{S}$ along every mode ($i = 1$, 2 and 3) are orthogonal giving all–orthogonality. The squared norms of the subtensors are given by the diagonal entries i.e. $\sigma_j^{(i)}$ ($j = 1 \ldots I_i$, $i = 1$, 2 and 3) are the norms of the subtensors and we know from matrix SVD that they decrease with index. This gives the ordering property of $\mathcal{S}$ and completes the proof. □

A.4 Least squares problem

Proof *General solution of least squares problem (2.21)* Reshaping $\mathbf{X}$ and $\mathbf{A}_i$ to vectors allows us to rewrite equation (2.21) in a standard form for least square problems. Let $\mathbf{x}$ and $\mathbf{a}_i$ be the column forms of $\mathbf{X}$ and $\mathbf{A}_i$ respectively. Then

$$\min_{\alpha_i} \|\mathbf{X} - \sum_{i=1}^{k} \alpha_i \mathbf{A}_i\|_F = \min_{\boldsymbol{\alpha}} \|\mathbf{x} - \mathbf{A}\boldsymbol{\alpha}\|_2, \tag{A.4}$$

in which $\mathbf{A} = \begin{pmatrix} \mathbf{a}_1 & \mathbf{a}_2 & \ldots & \mathbf{a}_k \end{pmatrix}$ end $\boldsymbol{\alpha} = \begin{pmatrix} \alpha_1 & \alpha_2 & \ldots & \alpha_k \end{pmatrix}^T$. The problem is overdetermined because $\mathbf{A}$ has more rows than columns. It is well–known that the solution of (A.4) is given by the normal equations,

$$\mathbf{A}^T\mathbf{A}\boldsymbol{\alpha} = \mathbf{A}^T\mathbf{x}. \tag{A.5}$$

Because of the orthogonality of the columns of $\mathbf{A}$, $\mathbf{A}^T\mathbf{A}$ is a diagonal matrix with the diagonal values $\mathbf{a}_i^T\mathbf{a}$. Finally we get the elements of $\boldsymbol{\alpha}$ to be

$$\alpha_i = \frac{\mathbf{a}_i^T\mathbf{x}}{\mathbf{a}_i^T\mathbf{a}_i}.$$

Reshaping $\mathbf{x}$ and $\mathbf{a}_i$ back into matrices and using the scalar product notation we can write

$$\alpha_i = \frac{\langle \mathbf{X}, \mathbf{A}_i \rangle}{\langle \mathbf{A}_i, \mathbf{A}_i \rangle}.$$

□