INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

U·M·I

University Microfilms International
A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
313/761-4700 800/521-0600

Order Number 9405556

Exploration in on-line handwritten character recognition

Matić, Nada Petar, Ph.D.

City University of New York, 1993

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

Exploration in On-Line Handwritten Character Recognition

by

Nada P. Matić

A dissertation submitted to the Graduate Faculty in Engineering in partial fulfillment of the requirements for the degree of Doctor of Philosophy, The City University of New York

1993

This manuscript has been read and accepted for the Graduate Faculty in Engineering in satisfaction of the dissertation requirement for the degree of Doctor of Philosophy.

8/25/93
Date

Chair of Examining Committee

8/25/93
Date

Executive Officer

Prof. Sanghamitra Basu

Prof. Joseph Barba

Prof. Leonid Roytman

Dr. Isabelle Guyon

Supervisory Committee

THE CITY UNIVERSITY OF NEW YORK

Acknowledgments

This Thesis could not have been successfully completed without the help and assistance of many others. I would like to express my gratitude to all those who have contributed to this effort.

My adviser, Nenad Marinovich, deserves many thanks for his guidance, help, and direction over the years. His suggestion to look into the neural computing in signal processing inspired this dissertation.

I am indebted to Isabelle Guyon, with whom I started to work in summer 1991 at Bell Laboratories on neural networks based on-line character recognition. Most of the work in this Thesis, has been carried out with her direct involvement, both in experiments and as a reader and critic. Through her meticulous efforts, expertise and enthusiasm this thesis was greatly improved.

I would like to give special thanks to Larry Jackel, the Head of the Adaptive System Research Department at Bell Laboratories. He somehow always managed to find a way to support my work even when the situation looked desperate. The influence of other members of the Adaptive Systems Research Department is significant, in particular Vladimir Vapnik and John Denker. Thanks to Leon Bottou and Corinna Cortes for their Optimal Hyperplane programs. Thanks to all other members of the group.

Special thanks to Rita and Perigo for their enduring friendship, for helping me through the tough times and for "adopting" me when necessary.

Thanks to my parents whose support has meant more to me then they could ever imagine. Without them, none of this would have been possible. My final thanks to Petar who changed his way of life in order to be closer to me and help me finish the work.

The simulations were performed using Neural Network simulator "SN" written by L. Bottou and Y. Le Cun.

Table of Contents

Acknowledgements iii

Table of Contents v

List of Figures viii

1 Introduction 1

1.1 Handwriting Styles 4

1.2 Recognition Problems 7

1.3 Preprocessing and Data Representation 8

1.4 Recognition Techniques 12

1.5 Neural Network Approach 16

1.6 Contribution 20

1.7 Thesis Organization 22

2 Background 24

2.1 The Formal Neuron 26

2.2 Adaptive Single Layer Networks 27

2.3 Multi-layer Networks 34

2.4 Generalization in Neural Networks 39

2.5 Summary 50

3 TDNN for On-line Handwritten Character Recognition 51

3.1 The Method 51

3.2 Preprocessing 52

3.3 The Network Architecture 54

3.4 The Training 58

3.5 Training on Full Character Set 60

3.6 Summary 63

4 Cleaning The Large Database for Character Recognition 64

4.1 Motivation and Application 64

4.2 Problem Description 65

4.3 Emphasizing Ambiguous and Atypical Patterns 69

4.4 Proposed Solution - Super-supervised Learning 71

4.5 Experimental Results 75

4.6 Theoretical Foundations of Super-supervised Learning 78

4.7 Summary 85

5 Writer Adaptation 87

5.1 Existing Solution 91

5.2 Proposed Writer Adaptive System 92

5.3 The On-line Adaptation Version of the Optimal Hyperplane Classifier Training Algorithm 94

5.4 Training the Optimal Hyperplanes for Digits and Uppercase Letters 96

5.5 Different Adaptation Schemes 100

5.6 Final Adaptation System: Experiments and Results 101

5.7 Summary 107

6 Conclusions **113**

6.1 Advantages of the Method 114

6.2 Limitations of the Method 115

6.3 Future work 115

Bibliography **117**

List of Figures

1.1 Differences in European and North American writing style. . . . 5

1.2 Character "E" written with different number of components . . . 5

1.3 Examples of lowercase characters obtained from a touch terminal 6

2.1 (a) Real neuron (b) McCulloch-Pitts formal neuron(1943). 25

2.2 The Perceptron (F. Rosenbllat, 1957). 27

2.3 Linear separations. a) Perceptron b) Minmax separation. Supporting patterns are circled. 29

2.4 The ADALINE (B. Widrow, 1959). 33

2.5 The sigmoid unit. 35

2.6 The time-delay neuron 38

2.7 Analogy with curve fitting. 41

2.8 Eight possible ways to dichotomize three points in a plane. . . . 43

2.9 When the VC-dimension of the set of functions increases, the frequency of errors on the training set decreases. At the same time the width of the confidence interval ϵ increases. This two terms compete, and as a result the guaranteed risk (i.e. upper bound on the generalization error) has a minima for the optimal complexity of the classifier h^*. 45

2.10 Learning curves. Training and test error as a function of the number of examples m. The capacity h is fixed. 46

3.1 Estimation of the slope and the curvature. 53

3.2 Recognition preprocessing. The original characters "C" shown at the top has been resampled with 90 points. The intermediate representation, shown at the bottom consists of 7-component vectors containing information about pen up/down, direction, and curvature at each point. Each vector is represented as column of boxes whose size indicates the magnitude of a component and whose color indicates the sign. The intermediate representation is the input to the neural network. 55

3.3 Architecture of the Time Delay Neural Network (TDNN). Not all the neurons and all layers are represented. 56

3.4 Test results (output of the TESTBED). 62

4.1 Examples of meaningless and mislabeled patterns in the lowercase letters database. 67

4.2 Examples of atypical and ambiguous patterns in the lowercase letters database. 68

4.3 Emphasizing scheme. The least predictable examples are presented more often. 70

4.4 Block diagram of the cleaning scheme: super-supervision. 72

4.5 Flow diagram of the interactive version of the cleaning scheme. . 73

4.6 Recognition results for handwritten lowercase letters, for various levels of cleaning. The shaded area indicates the results obtained with the emphasizing scheme. 76

4.7 This small two-dimensional classification example illustrates the four kinds of suspicious patterns. Shaded areas represent regions of high probability density. The four subgraphs display various conditional and/or empirical probabilities. In the region of low $p(\mathbf{x})$, we find an *atypical* pattern $\mathbf{x}^1$ correctly labeled "a", and a *meaningless* pattern $\mathbf{x}^3$ (garbage) labeled "b". In the region of low $P(d = a|\mathbf{x})$, we find an *ambiguous* pattern $\mathbf{x}^4$ labeled "e" and a *mislabeled* pattern $\mathbf{x}^2$ labeled "c". Patterns $\mathbf{x}^2$ and $\mathbf{x}^3$ have been drown from the "distorted" distributions $P'(d = a|\mathbf{x})$ and $p'(\mathbf{x})$ respectively (see text). With the help of the decision function provided by the trained classifier (dashed line), suspicious patterns can be detected for further examination by a human "super-supervisor". 80

4.8 Percent error vs. percent of the patterns removed from the training set: (a) dumb cleaning; (b) smart cleaning. Constant $\beta = 0.5$. The guaranteed risk has been rescaled to fit the figure. 83

5.1 Examples of atypical writing styles. Various examples of the letters E are shown. Pen up segments are linearly interpolated and represented with small dots. An atypical E is shaded. 88

5.2 Minmax Training Algorithm: Usual decision boundary (like MSE) vs decision boundary as a result of Minmax training. Notice the role of the outlying patterns. 93

5.3 Optimal Hyperplane Classifier: a) Two class separation, "supporting patterns" are circled. The example "@" provided by a new user is misclassified. b) On-line adaptation, OH_{new} is generated. One of the old category-B patterns loses its status as a supporting pattern, while the new example becomes a category-A supporting pattern. 95

5.4 Writer adaptation system 103

5.5 Test set used for the adaptation. Writer No.1 104

5.6 Test set used for the adaptation. Writer No.2 105

5.7 Test set used for the adaptation. Writer No.3 106

5.8 Cumulative number of errors. Different adaptation schemes labeled 1 to 4. Baseline performance is labeled "no adaptation". Writer No.1. 108

5.9 Cumulative number of errors. Different adaptation schemes labeled 1 to 4. Baseline performance is labeled "no adaptation". Writer No.2. 109

5.10 Cumulative number of errors. Different adaptation schemes labeled 1 to 4. Baseline performance is labeled "no adaptation". Writer No.3. 110

5.11 Cumulative number of errors for all seven writers. Best adaptation scheme. All the available characters: uppercase, digits and 7 symbols. 111

1 Introduction

Automatic handwriting recognition is a difficult task. The variety in handwriting styles, with different sizes, slants, and stroke widths, even for small character sets such as numerals, present a research challenge. There is a great deal of study done in this field, which is of enormous practical importance. A number of methods for automatic handwriting can be found in the literature and in several surveys of the state of the art [1, 2, 3, 4].

Automatic handwriting recognition systems are classified into two categories according to the mode of data acquisition. In this work we are dealing with *on-line* systems, where recognition is performed while the user writes. On-line systems requires the use of a digitizing tablet. This is in contrast with *off-line* or Optical Character Recognition (OCR) systems [5] that use optical digitizing devices that generates pixel-maps. The mode of data acquisition has a significant effect on the other modules of the system like feature extractor, classifier etc. Digitizing tablet technologies are varied, the main family includes touch terminals that consist of a transparent touch-sensitive screen overlaid on a standard liquid crystal display (LCD). One can write on the touch sensitive screen using a writing device called simply a "pen". While the pen is touching the screen, points are returned , and displayed immediately under the pen (*electronic ink*). This device provides pen

trajectory information.

A number of so called notepad computers and pen-based personal communicators are available the market, which might address somewhat different applications, but their common property is the use of the pen-based interface. To be an attractive form of a human-computer interface, touch sensitive screens have to be coupled with a powerful and accurate *recognition module* that works in real-time.

An advantage of on-line devices is that they provide temporal or dynamic information of the handwriting. Each character can be considered as a sequence of basic components, *strokes*. A stroke is defined as writing from a pen-down to pen-up position. Dynamic information can be encoded in the form of *number* of strokes, *order* and *direction* of strokes, and the *speed* of the writing for each stroke. The presence of the dynamic information complicates the recognition process by introducing a new level of variations, but the additional information improves the recognition rate compared to the corresponding off-line approach. To prove that, experiments have been carried out, in the case of neural-network based recognition system, where the dynamic information from the on-line data has been removed, so that OCR recognition module can be used, and two modes compared [6]. It was found that the recognition system which incorporates the time information performs better, while using simpler preprocessing and smaller number of parameters.

Another advantage of on-line handwriting recognition is its interactivity. A large number of existing and potential applications are based on this property. One can, for example design a handwriting based editor where both input text and editing commands are handwritten, or one can use handwritten editing commands that are working on the text previously entered with a keyboard [7].

Also, this interactive property is the basis for a very important concept of the

user *adaptation* (i.e. fine tuning to a particular handwriting style). Systems that allows user adaptation are called *writer adaptive*, as oppose to the systems that are trained on many writers and are called *writer independent.* Recognition systems that allows adaptation, while still performing well on average writer, is a realistic solution for this hard task.

In this thesis we address specific problems connected to the *on-line handwritten isolated character recognition*, where the "core" of the automatic recognition system is based on **neural networks**. In particular we are exploring differences in the training patterns, in an effort to capture the tail of the data distribution. We have managed to use available patterns more efficiently, by recognizing the special role of the *most informative* (supporting) patterns that are typically outliers. They are examples of the rare writing styles that are under-represented in the training database. We have used two different approaches in order to pay special attention to the most informative patterns. One is to define and use a "super-supervised" learning procedure. This is applicable to the writer independent task. The second approach is to implement writer adaptation. By doing that we substantially improved the generalization performance, for both writer independent and writer adaptive tasks.

The problems explored in this thesis, are of real practical interest and intermediate difficulty. They are a part of the broader effort, namely building an experimental multi-modular system [8] based on the neural network, called Penacée. The ultimate goal of this system is to recognize unconstrained handwriting and achieve very high recognition rate on large variety of handwriting styles.

1.1 Handwriting Styles

There is a tendency that most people write particular alphabets in similar manner. This has a lot to do with a common education in certain regions. Despite our training, there is still a large variety in writing styles, and there are always writers with peculiar handwriting. In both problems that we are addressing, this peculiar, rare examples of handwritten characters are of the special interest. They represent outliers in pattern distribution, collected among large number of writers.

The origin of the writer [9, 10], can be one of the sources of static (size or shape) or dynamic (stroke number or stroke order) style variations. Japanese are known to be very consistent with stroke ordering, since they have learned to respect the rules in their own alphabet, as oppose to European and American writers. Some foreigners whose alphabet is not originally Latin, can have very atypical writing styles. European writers tend to use additional horizontal stroke on character "Z" and the number "7", as opposed to North American writers (see Fig 1.1). Left handed writers tend to use different style than the right handed. The number of components may vary a great deal for one character (see Fig.1.2). The order of components within one character may vary as well. Writers have a tendency to draw the vertical components before the horizontal ones, and the left before the right [9].

Examples of this variety of writing styles from the database that we are using in our experiments is given in figure 1.3. They are drawn from the database of *lowercase* characters entered on a a *Touch Terminal.* A data set of 11513 letters from approximately 250 writers was collected among the staff at AT&T. The writing style for our database was *hand-printed* characters, as oppose to the *cursive* writing, where several characters can be written with a single stroke.

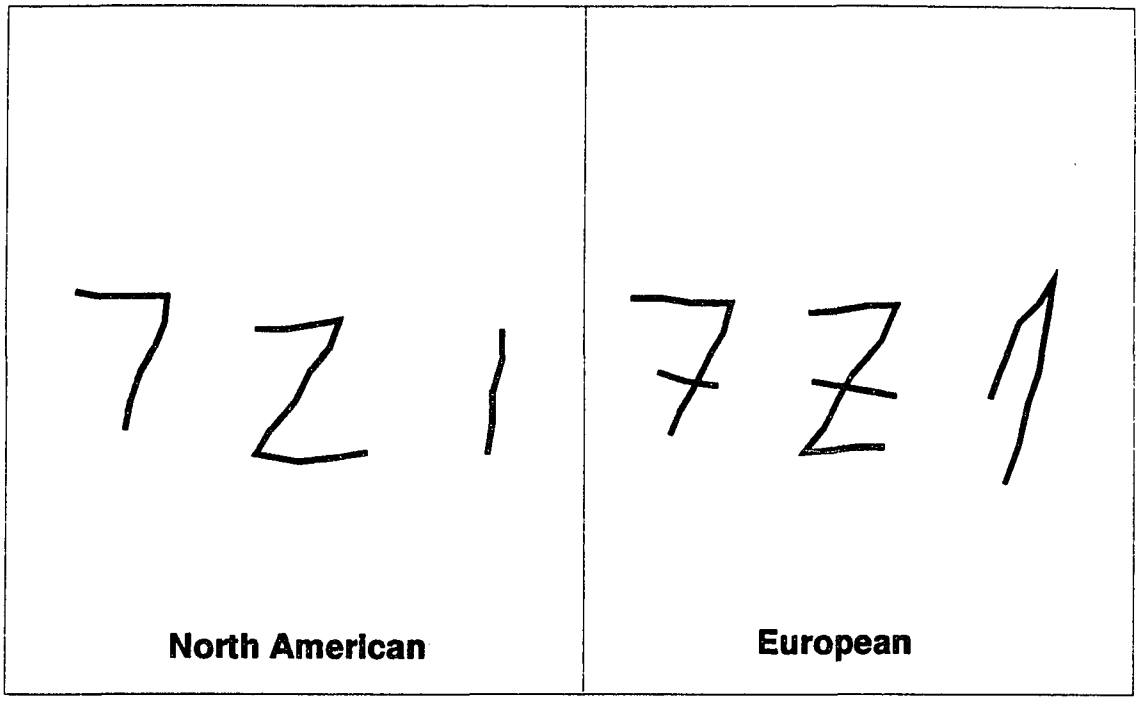

Figure 1.1: Differences in European and North American writing style.

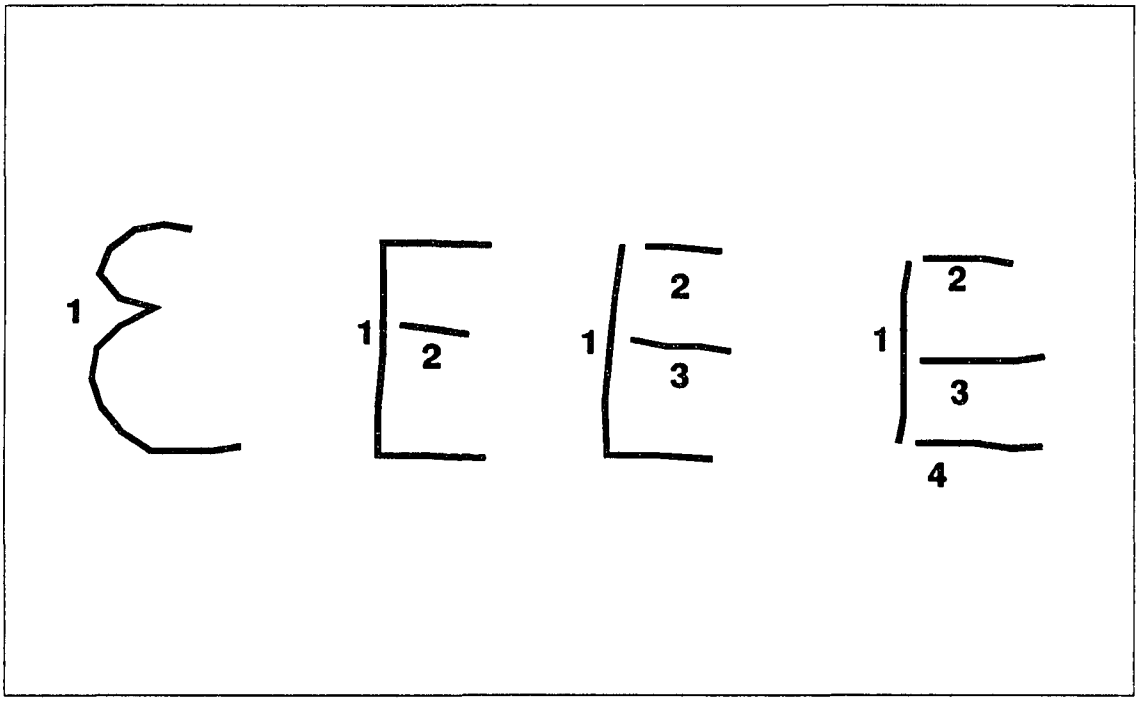

Figure 1.2: Character "E" written with different number of components

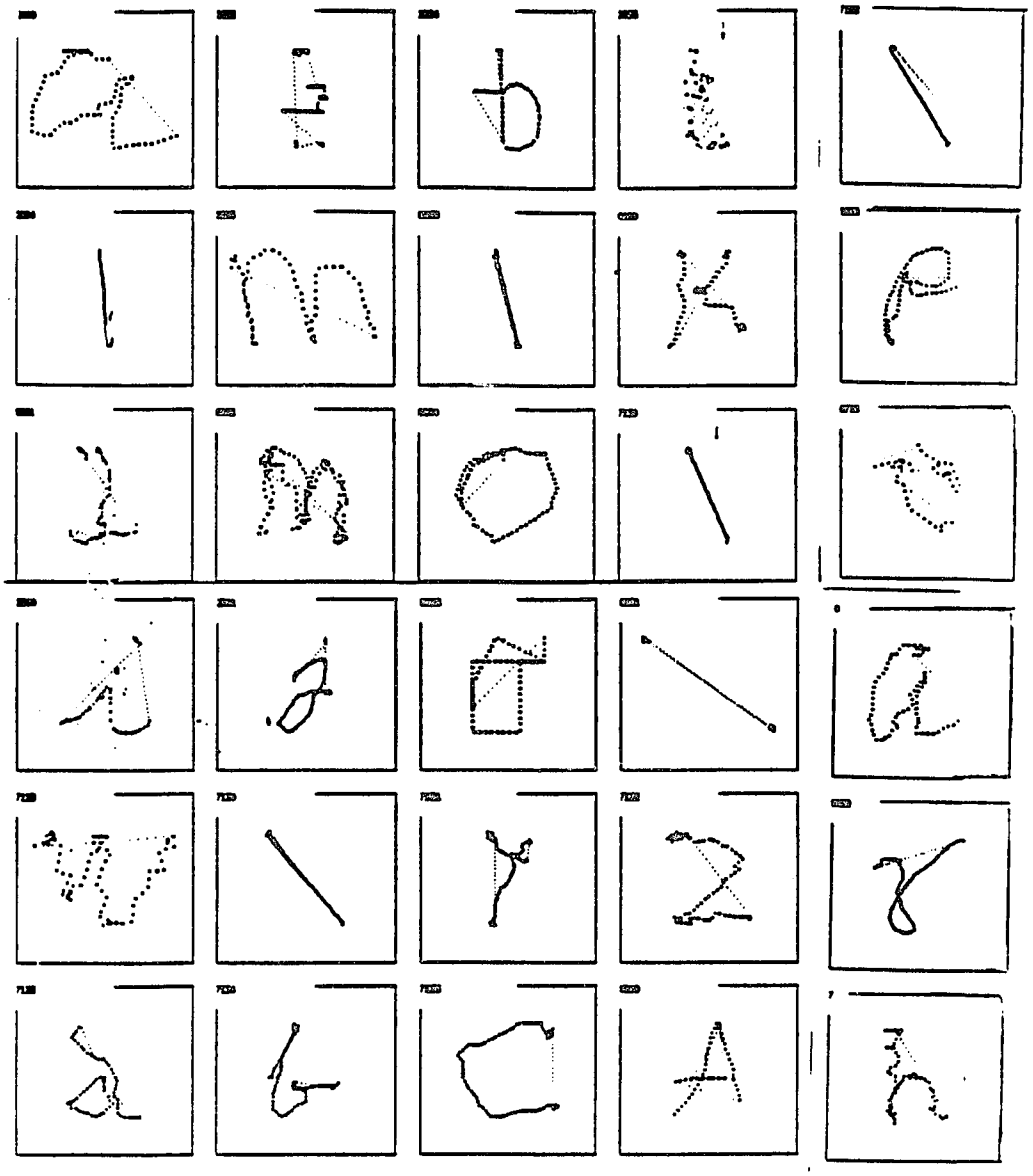

Figure 1.3: Examples of lowercase characters obtained from a touch terminal

1.2 Recognition Problems

Just by looking at some of the handwriting properties, one can anticipate different kinds of problems in handwriting recognition. Typically in on-line handwriting recognition, there may be less constrains on characters (e.g. their size and orientation may vary) which is an additional burden to the recognizer.

The problem that we explore in this work is introduced by the fact that a writer needs some time to adapt to the input device (e.g. tablet), which can result in a generation of meaningless characters. Writers can also introduce mislabeled patterns into the training data. Their presence in the training set can lead to the increasing error rate in the utilization phase. Such *garbage* patterns can be considered as outlying patterns or noise that are hard to distinguish from other *rare* patterns. We would like to pay special attention to rare patterns that are under-represented in the training database. They can be either poorly written characters or examples of a atypical writing styles. In both cases they represent a hard problem to the recognizer.

Another problem that we would like to address is the existence of *ambiguous* patterns. Different characters within the same category (i.e. uppercase, lowercase, numerals, punctuation symbols) often have similar shapes. Distinguishing such characters into correct classes is a difficult task. Examples are: U-V, C-L, I-L, n-h, q-g. Similar shapes exists among different categories. Sometimes without some context, it is hard to distinguish some characters and numbers. Examples are: O-0, I-1, l-1, Z-2, S-5, G-6. Also, there are upper and lower case characters with similar shapes: C-c, K-k, O-o, K-k, Y-y. In this case, the distinguishing factor can be the size of a character relative to the line spacing, or other characters size. But, when the recognizer has been trained on many different writers, where some of them are breaking common rules, even these hints, are not always helpful.

One of the problems that we are not dealing with is the *segmentation* (partitioning text into the individual characters). In unconstrained English handwriting, writers often tend to connect different characters together. Other Western languages are similar. For cursive writing and connected printed characters some form of character segmentation has to be applied before the recognition. Segmentation is a difficult task, particularly for cursive handwriting. Either temporal (e.g. time slice between two strokes) or spatial (e.g. distance between different strokes) information can be the basis for the segmentation. Segmentation can be performed before recognition, coupled with recognition, or after recognition [11]. We are just mentioning segmentation, since our task will be carried out on characters written into the boxes.

1.3 Preprocessing and Data Representation

A number of preliminary steps must be applied to the raw data in order to transform them into the form convenient for classification. The objective of the *preprocessing* is to reduce the useless factors of variability among patterns (e.g. noise reduction, normalization), and enhance factors of variability that will help discrimination among different classes (e.g. *feature extraction*).

Data representation using features usually reduces the complexity of the problem by extracting from mass of data just that information needed for the classification. Another form of data representation makes use of allographs (i.e. shape variants of a given letter) [12, 13, 14].

1.3.1 Noise Reduction

Depending on the model accepted, different noise reduction methods can be implemented. Some of them are:

Smoothing - usually averages a point with its neighbors, in order to reduce small imperfections [15] introduced by hardware or a writer.

Filtering - reduces the number of points. In order to obtain equally spaced points or to increase the number of points in the regions of greater curvature either resampling [6] or interpolation [16] can be applied.

Dot Reduction - reduces dots to a single points [17].

1.3.2 Normalization

The data representing the character can be normalized [16, 6] in terms of size, orientation, and position.

1.3.3 Feature Extraction

In the preprocessing stage typically suitable features are extracted from the patterns so that the later, classification phase can be viewed as partitioning in the feature space. Typically here the designer uses all the special knowledge about the problem at hand in order to choose a set of features that seems to be the most valuable in differentiating one pattern from another.

Different types of features can be extracted from the data, but the main categories are:

- Global features
- Local Geometrical and topological features

Global Features Extraction of the features can be done by applying some global transformation simultaneously on all data within the character input rep-

resentation frame. Fourier transformation can be used [18] for characters consisting mainly of curves. Walsh transformation and Karhunen-Loeve expansion have been used as well. Obtained features are in the form of series or spectra vectors and are often invariant to some global deformation (e.g. global translation or rotation).

Local Geometrical and Topological Features Often, the so called "local" features are extracted based on specific geometry or topology of the character [6]. Information about stroke directions, curvature, spacing between the points, speed of the pen are so called "non-descriptive" features, as oppose to the "descriptive" features like end points, intersections of line segments, number of branches and loops. These are all local features useful for further processing. The main advantage of local feature extraction are their high tolerance to style variations and distortions.

The recognition system for on-line isolated handwritten characters, that we used for our experiments is based on local feature extraction [6]. More details about the local feature extraction and preprocessing in this case will be presented in the Chapter 3.

Another new on-line handwriting recognition system [19], makes use of specific representation scheme called AMAP, in which the pen trajectory is used to construct a low-resolution multidimensional map that includes Geometrical and semi-dynamical local features. An AMAP can be viewed as a multidimensional array ($4X6X6$), where each dimension is associated with a local property of the trajectory: direction of motion θ, and X and Y position of pen. The value of each cell represents the "density" of features with a particular value of the parameters θ, X, Y in the character.

1.3.4 Allographs

Data representation can be based on *allographs*. Allographs are shape variants of given letter [12, 13, 14]. The same writer might select different shapes for the given letter in different conditions or in different context. Fundamental property of the allograph is its number of strokes. Thus, two allographs (e.g. two letters **a**) are different if their number of strokes differs. Each stroke interpretation has the general form: $\mathbf{name}(I_{stroke}/N_{stroke})$ where I_{stroke} is the position of the stroke in the sequence of strokes for a given allograph and N_{stroke} is total number of strokes. For example, a given stroke can be interpreted as representing one element of the set $\mathbf{a}(1/3), \mathbf{d}(1/3), \mathbf{o}(1/2), \mathbf{c}(1/1)$.

1.3.5 Coupled Oscillator Model

One way of describing allographs can be obtained by using the *coupled oscillator model* [20]. This representation assumes that the character formation can be viewed as the result of two orthogonal pendular movements. One is moving back and forth in the horizontal x direction while another is moving in in y direction. In addition, there is a constant motion to the right superimposed on the sinusoidal x motion. These can be expressed in terms of the velocities:

$$\begin{aligned} \dot{x} &= a\cos(\nu_x + \phi) + c \\ \dot{y} &= b\cos(\nu_y t) \end{aligned} \tag{1.1}$$

where a and b are the amplitudes of the oscillatory motion in the x and y directions, ν_x and ν_y are the motion frequencies in the x and y directions, ϕ is the phase difference between motion in the x and y directions. c is the constant motion (i.e. drift).

For each different character class, all the above parameters are different. According to Hollerbach [20], it is sufficient to model the different *strokes* with corre-

sponding set of amplitudes, frequencies and drifts. Each character (or character shape variant - allograph) then can be represented as a sequence of parameterized strokes.

1.4 Recognition Techniques

Characters can be recognized in many different ways. Most of the general pattern recognition techniques can be used. Classification techniques vary widely depending upon the features chosen, the way these features are extracted and upon the assumptions that can be made about the nature of the problem.

Classifiers for pattern recognition tasks can be roughly partitioned into two categories: *statistical* and *syntactic and structural.* While the syntactic and structural classifiers are based on the theory of formal languages statistical classifiers come from the statistical decision theory and can be partitioned into classifiers where the forms of the *discriminant functions* are known and *probabilistic* classifiers where the forms of underlying distributions are known.

In statistical setting vector $\mathbf{x}$ (e.g. feature vector) is assumed to be drawn randomly and independently from the input distribution characterized with the probability density function $p(\mathbf{x})$. In the k-class problem, the ith class is assumed to occur with *a priori* probability $P(y_i)$, $i = 1, 2, \ldots k$. For the patterns in the ith class, the feature vector is assumed to be governed by a conditional density $p(\mathbf{x}|y_i)$. The goal is to perform the classification. One way to do it is to select the category (i.e. class label) for which $P(y_i|\mathbf{x})$ is maximum. Unfortunately the forms of the probability distributions are often unknown and alternative approaches must be used to perform the classification.

1.4.1 Classifiers Based on Discriminant Functions

Sometimes the forms for discriminant functions $D(\mathbf{x})$ are assumed or known [21], and patterns are used to estimate the values of parameters of the classifier. Though different procedures for determining discriminant functions can be used, none of them requires knowledge of the forms of the underlying probability distributions (i.e. $p(\mathbf{x})$ or $P(y_i|\mathbf{x})$).

They can have quite different forms:

Template Matching and Correlations

In Template Matching, templates (i.e. reference patterns) are often stored in terms of feature representation. With a chosen similarity measure, classification (matching) can be implemented on the basis of some criterion for the degree of similarity. Opposite of similarity between two patterns is their mutual distance in some metric, which can also be the basis for the classification. The measure of similarity [22] can be correlation (i.e. dot product) between the patterns or direction of cosines. The dissimilarity measure can be Euclidean distance or Mahalanobis distance. The degree of similarity can be obtained by using classifiers like Nearest-neighbor method or Parzen Windows. More complex, better suited distance is *elastic distance* used in elastic matching procedure (usually implemented with Dynamic programming [17]) or recently introduced *tangent distance* [23]. Tangent distance can be made locally invariant to any set of transformations of the input while at the same time it can be computed efficiently. These methods are often referred to as memory-based models, indicating the fact that they tend to memorize the response to the training data.

Linear Discriminant Functions

A discriminant function can be linear (i.e. linear classifier) in terms of the components of the input vector $\mathbf{x}$. In that case a decision surface represents a hyperplane. Classifiers based on linear discriminant functions are often simple and suitable for practical implementations.

A discriminant function can be "linear" in its parameters but not restricted to linear dependencies in $\mathbf{x}$. The decision surface in this case is nonlinear (e.g. quadratic when the discriminant function has quadratic dependence in $\mathbf{x}$). This type of discriminants are called *generalized linear discriminant functions* [21]. They include well known functions such as Perceptron (see section 2.2.1), polynomial classifiers and Radial Basis Functions. In reference [24] efficient algorithm has been proposed in order to train such classifiers.

Neural Networks

Neural networks represents a group of adaptive classifiers, that has been recently used for many real world problems. They make no assumptions concerning the shapes of underlying distribution and employ some form of the discriminant function. For example, Perceptron [25], a simple neural network classifier employs general linear discriminant function (see section 2.2.1). The more complicated (i.e. Multi-layer Perceptron) neural network can be viewed as an extension of discriminant function approach which is capable of forming arbitrary complex decision boundaries [26]. Sometimes higher-order (or sigma-pi) units can be substituted for the linear units in neural networks in order to get polynomial classifier [15]. Since we are going to use neural network classifier in our on-line recognizer, more details will be given in subsequent sections.

Tree-based Classifiers

Tree-based classifiers can also be viewed as a nonlinear discriminant function approach. Here, classifier is constructed by a series of simple greedy splits of the data into a subgroups. Each subgroup is then split recursively, so that a resulting classifier has a hierarchical binary tree structure [27, 28, 29].

1.4.2 Probabilistic Classifiers

These classifiers are specified in terms of parameters (mean and covariance) of the class probability distribution such as Gaussian or Gaussian mixture [21]. In other words, they require the knowledge of the forms of the underlying probability distribution [30]. Based on the form of the decision boundary, both linear and quadratic classifiers can be used for character classification.

Bayesian Classifiers

Bayesian classifiers characterize classes by their probability density functions on the input features and use Bayes's decision theory to form decision regions from these densities [21]. Bayesian rule can be found suitable when using Fourier coefficients to describe the characters [31]. Bayes decision rule is optimal in the sense that it minimizes the classification error, i.e. the average number of misclassifications. Though the knowledge of the form for the $P(y_i|\mathbf{x})$ is desirable, it is often replaced using Bayes rule with $P(y_i)$, $p(\mathbf{x}|y_i)$ and $p(\mathbf{x})$.

Other Probabilistic Classifiers

In some tasks it is essential to know *a posteriori* probabilities $P(y_i|\mathbf{x})$, not just the class label, in order to use them as inputs to the higher level decision making

module. For example, the Multi-layer Perceptron trained with backpropagation scheme (see the next section) can learn the best Mean Squared Error approximation to the probabilities $P(y_i|\mathbf{x})$. If some additional constraints are imposed, the network provides output values bounded between 0 and 1 while all output values sum to one. This approach is known as "softmax" scheme [32].

1.4.3 Syntactic and Structural Classification Method

This approach is used to classify the patterns in terms of the constituent basic elements and their relationships. Formal grammars are considered a useful formalism for this structural approach. The existence of a recognizable "structure" is essential for the success of these methods [33, 34, 35].

1.5 Neural Network Approach

Neural networks are well known as a powerful learning tools in a variety of problems, especially for the cognitive tasks that are characterized by a high degree of uncertainty and variability, such as: vision, speech, or language processing. The developments in learning algorithms for neural networks have provided potential alternatives to traditional pattern recognition techniques which make far less restrictive assumptions about the structure of the input patterns [36]. Recently, it has been shown [37], that neural networks are now the method of choice for off-line isolated handwritten character recognition.

The neural networks models are partitioned into two basic categories: *static networks* and *dynamic networks*. Static networks are characterized by node equations that are memoryless. That is, their output is a function only of the current input, not of the past or future inputs or outputs.

Dynamic networks, on the other hand, are systems with memory. Their node equation are typically described by differential or difference equations. They can be categorized into three different groups: networks with feed-forward dynamics, networks with output feedback, and networks with state feedback.

Static networks will be the type of neural networks used in our thesis. They implement nonlinear transformation of the form $\mathbf{y} = F(\mathbf{x})$. Typically $\mathbf{x} \in \Re^n$ and $\mathbf{y} \in [0,1]^m$ or $\mathbf{y} \in \Re^m$, where n and m are integers that represent the dimension of $\mathbf{x}$ and $\mathbf{y}$, respectively.

Most of the current neural network learning procedures are supervised: The *Learning Machine* is presented with the training pairs in the form of an input vector and a desired output vector. The learning process then consists in finding parameters (weights of the network), which minimize a measure of the dissimilarity between the actual output vector and the desired output vector. This is done without the help of some guiding learning rule, on the contrary, we can say that the rule is induced from a set of training pairs. The newly acquired knowledge of the network is encoded in its weights. The adaptation process (adjustment of weights) typically minimizes a certain objective (criterion) function in order to adjust weights of the network. If this objective function is smooth and differentiable, minima can be found by using the iterative gradient technique: at each iteration, the gradient of the objective function with respect to the parameters is computed, then the parameters are changed by a small quantity in the opposite direction of the gradient. The learning procedures we use in our experiments are supervised.

In contrast, in *unsupervised* learning, the networks are presented with the input samples only, while their desired outputs (targets, class label etc.) are unknown. Here, the objective function involves discrete variable (class labels) and cannot be optimized using iterative gradient technique. In this case "clustering" techniques are used, and input samples are grouped into the classes which are

self-similar. Networks trained in this fashion are called self-organizing networks (see references [22, 38]).

Backpropagation

The most widely used neural network training procedure is *backpropagation.* The specific way that the gradient is computed characterizes backpropagation as a training algorithm. More details about this training algorithm will be given in the next chapter. In practice, most of the backpropagation networks are composed of layers of "neurons" which compute a weighted sum of their inputs, followed by a nonlinear sigmoid function. Besides input and output layer there are one or several "hidden" layers of neurons. The common architecture is the *feed-forward* network , where each neuron receives input only from the units in the layer below it. Backpropagation provides a way of training such multi-layer structure, so that a contribution of "hidden" layer neurons to the objective function (typically mean squared error) is taken into account. To accomplish this, the error of the output is back-propagated recursively to each lower layer. The derivation of backpropagation algorithm was made possible by the use of a non-linear function that is differentiable. Such multi-layer structure is capable of learning arbitrary complex decision surface.

Methods for Improving Generalization

In tasks like character recognition, it is not sufficient that the network has learned training pairs, the network has to be able to *generalize* well, i.e. to correctly classify the inputs that it has not seen. If the network is too small, it is not capable of learning the training set efficiently. On the other hand, if the network is too big, it will learn the training set without difficulty, but it will not be able to generalize properly. The problem is similar to that of a curve fitting. With

a small number of experimental points, high order polynomial can fit even noise points, but be very irregular between the experimental data points. Reducing the number of parameters in the network is very important issue.

A large number of application use multi-layer feed forward networks successfully without any specialization. As the task at hand becomes more complex one has to *specialize* the architecture of the network. Backpropagation based procedures has been developed for highly specialized tasks. Complex neural networks now exists that are composed of modules replicated in space (for image recognition [39, 40]), or replicated in time for sequence recognition, in particular speech [41] and on-line character recognition [6]. How can some of these specializations be accomplished? One way is to introduce the concept of *weight sharing*, where several neurons share the same weights.

A time delay neuron is a neuron that is receiving inputs that have been delayed in time. Another interpretation of the same concept is to replicate the same neuron along the time axis, and constrain the weights of the group of replicated neurons to be the same. In the network of such units called Time-Delay Neural Network(TDNN) [42, 41], each neuron takes input from a *local* "receptive field" of the layer below it. Though it is capable of processing the dynamic signal TDNN is a static, feed-forward network that can be trained with backpropagation algorithm that is only slightly modified to account for the averaging of the weights updates of the group of neurons with "shared weights".

Another way of reducing the number of free parameters in the network, and therefore improving generalization is to minimize the size of the network during training. This can be accomplished through *weight decay*. This scheme amounts to minimizing a compound objective function, where besides usual error term (like mean-square error), additional term that is proportional to the sum of the squares of the weights exists. This is a regularization term. This scheme forces useless weights to decay exponentially to zero, during the training.

Regularization can be accomplished through a weight pruning procedure applied after the training, so called *Optimal Brain Damage* procedure [43].

In general, network minimization and regularization will reduce the *effective capacity* of the neural network, and improve generalization.

1.6 Contribution

The work presented in this thesis is dealing with the problem of on-line handwritten isolated character recognition. A Time Delay Neural Network which incorporates feature extractor and classifier in a single trainable system [6] has been used as a *Learning Machine*. Our experiments are carried out on a database of handwritten characters, entered on a touch terminal. The training procedures are supervised.

Our contribution is in developing systematic procedures and methods for improving the recognition performance for such a task. The recognition performance improvement is where the major effort in research and development in the field is directed. The reason for that is the fact that in on-line systems, speed is not as critical, as is in the off-line case, since the recognizer speed has to keep up with a speed of writing. The average writing rates are 1.5 - 2.5 characters/sec in English alphanumeric characters. For most recognition algorithms, this performance requirement can be met with current microprocessors. With speed constrain removed in on-line systems, one is left with the problem of accuracy.

Two research directions are explored in this thesis. One is in the case of the writer independent task, while the second is in writer adaptive environment. The first direction (applied to writer independent task) is to study the effect of the presence of mislabeled or even meaningless patterns in large training databases, in particular, how their presence in the training phase affects generalization performance

in utilization phase. This problem is interesting for many problems in pattern recognition that involve training by examples, and where patterns are collected and labeled according to the class to which they belong by a human supervisor. Such *garbage* patterns in the training data result in performance degradation of the classifier. Especially, training procedures which *emphasize atypical patterns* (in an effort to capture the tail of the distribution) require eliminating substantially all garbage patterns from the database, otherwise procedures will emphasize garbage patterns as well. Cleaning the database is essential to such algorithms. Having that problem in mind we have developed a systematic and automatic method of *cleaning the database*, in order to improve generalization performance of the classifier. The core of the cleaning methodology is the new concept of *"super-supervised"* learning. We applied cleaning by using super-supervised learning on the database of *lowercase* isolated handwritten characters entered on the touch terminal to illustrate the usefulness of the method.

While the goal of the first research direction is to improve the recognition system that has been trained on many different writers (i.e. writer independent), and is expected to perform well on *average writer*, the second direction that we explore, makes use of such writer independent recognizer, in order to fine tune the system to the specific writer. A writer can also be allowed to introduce new symbols, for his specific needs. As a result we introduce *writer adaptation* using the multi-module architecture. The first module is writer-independent TDNN, without its last layer, whose role is to perform feature extraction. The last layer of the network is replaced by a new type of classifier (i.e. second module): *Optimal Hyperplane* [44]. This combination provides fast adaptation, where the user needs to give only a few additional examples of his own handwriting. After the adaptation phase, there is no recognition speed degradation compared to the writer-independent case. All this is accomplished with a modest additional memory, so that the practical implementation is feasible with several versions of

the customized recognizer built in the same system.

What is common for both research tasks is the specific role of the informative (supporting) patterns. They bring a lot of new information into the learning task, so it is appropriate to treat them with special attention. In the first task, in order emphasize them and improve the recognition, we have to clean the non-informative outliers first. In the second task, adaptation is actually performed by introducing a new set of the most informative patterns (they characterize particular writing style). In the retraining process they will determine new decision boundaries.

1.7 Thesis Organization

The thesis is divided into 6 chapters. In Chapter 2 the previous research on multi-layer feed-forward neural networks is reviewed to provide general context for the recognition techniques used in our system. The formal neuron, adaptive single layer networks and multi-layer networks are described in sections 1, 2 and 3, respectively. The issues concerning the ability of the neural network to generalize, and conditions for which we could expect to achieve it are described in the section 4. In Chapter 3 we describe a highly specialized Time-delay Neural Network (TDNN) designed for the on-line isolated character recognition task. We train such a network to recognize full character set, digits, uppercase, lowercase and punctuation characters without any contextual information. Chapter 4 contains our method for the automatic cleaning of the large database for character recognition and in Chapter 5 we describe our writer adaptive system. In Chapter 6, we conclude by discussing the merits and limitations of our approach to solving the problem of the very high recognition accuracy, required for the realistic applications of the on-line handwritten recognizers. We also suggest other

possible extensions, where some of the technique described in this thesis may be applied.

2 Background

One of the dreams of the computer age is to build machines that can think as humans do. The question is how closely must the computer's internal representation and processes resemble those of a human brain in order to achieve the goal? Indeed the idea of building an intelligent machine out of artificial neurons has been around for quite some time. First version of artificial (formal) neuron (figure 2.1.b), was introduced by McCulloch and Pitts [45], and it represents a simplified version of real (biological) neurons (figure 2.1.a). A formal neuron sums its inputs, weighted by coefficients (called synaptic weights) and turns on, or "fires", if the net input is greater then a threshold. According to a hypothesis of Hebb [46] the connectivity of the brain is continually changing as an organism learns. That change is characterized by increasing conductance of particular synapse, if the sending neuron repeatedly causes receiving neuron to fire. Formal neurons learn by modifying their synaptic coefficients using algorithms inspired by Hebb's principle.

In 1958 Rosenblatt [25] showed how an network of McCulloch-Pitts (M-P) neurons with adjustable synapses (weights) can be "trained" to classify certain set of patterns. Rosenblatt called these networks Perceptrons. From Nilsson 1965 book [47] it is clear that the concept of layered machine (that is, of Perceptrons

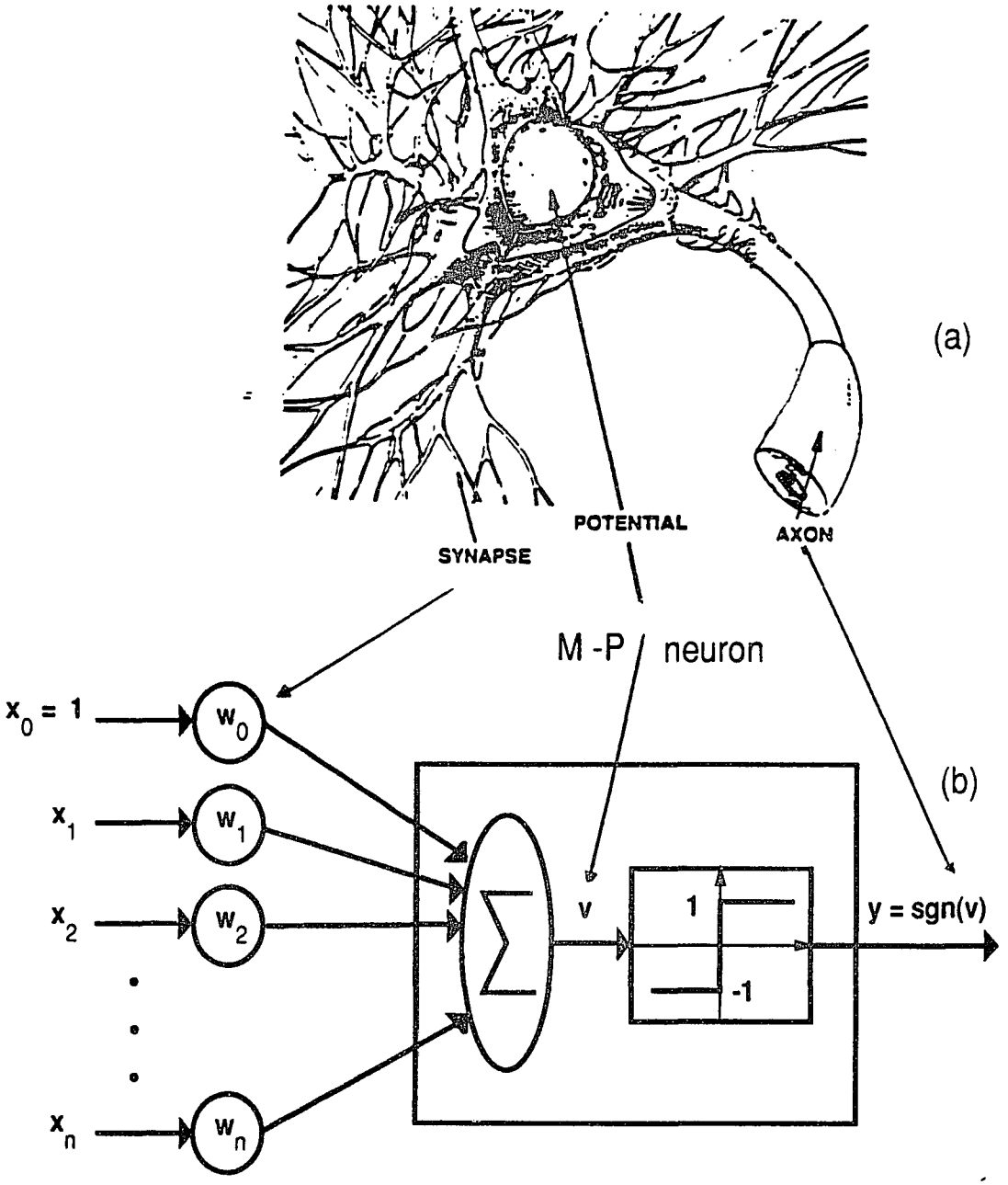

Figure 2.1: (a) Real neuron (b) McCulloch-Pitts formal neuron(1943).

with "hidden" layers) is well understood. The remaining problem was, whether the training of this structure could be carried out successfully, and what the training procedure would be. The analysis of the Perceptron limitations done by Minsky and Papert [48] was discouraging for the whole field, so in the 1970's the research in neural networks came to virtual halt, and the focus shifted to alternative approaches in AI that were more promising. In the early 1980's there has been a resurgence of interest in neural networks. There are several reasons for this, including the appearance of faster digital computers, interest in building massively parallel computers and networks for modeling the complexities of some difficult problems, and most importantly, the discovery of powerful neural network learning algorithms [49, 50, 51, 52].

In this review of the neural networks models, we are going to focus on *supervised* learning procedures that are applied to the feed-forward static networks.

A lot of material from this chapter can be found in in [53, 47, 54, 48].

2.1 The Formal Neuron

The formal neuron (figure 2.1.b) has binary inputs and outputs. It computes a sum of its inputs x_i weighted by the synaptic coefficients w_i. The weighted sum "potential" v is input to the *activation function* in the form of a threshold. If it exceeds the threshold, the neuron output y is $+1$, otherwise, it is -1. The bias weight w_0, connected to a constant input $x_0 = +1$ can be used to control the threshold level.

The formal neuron can be trained using learning principle proposed by Cooper *et al* [55] and is known as Hebb's rule. The idea is to increment (or decrement) each weight w_i by a given amount $\eta > 0$. (Note that the product of the states of

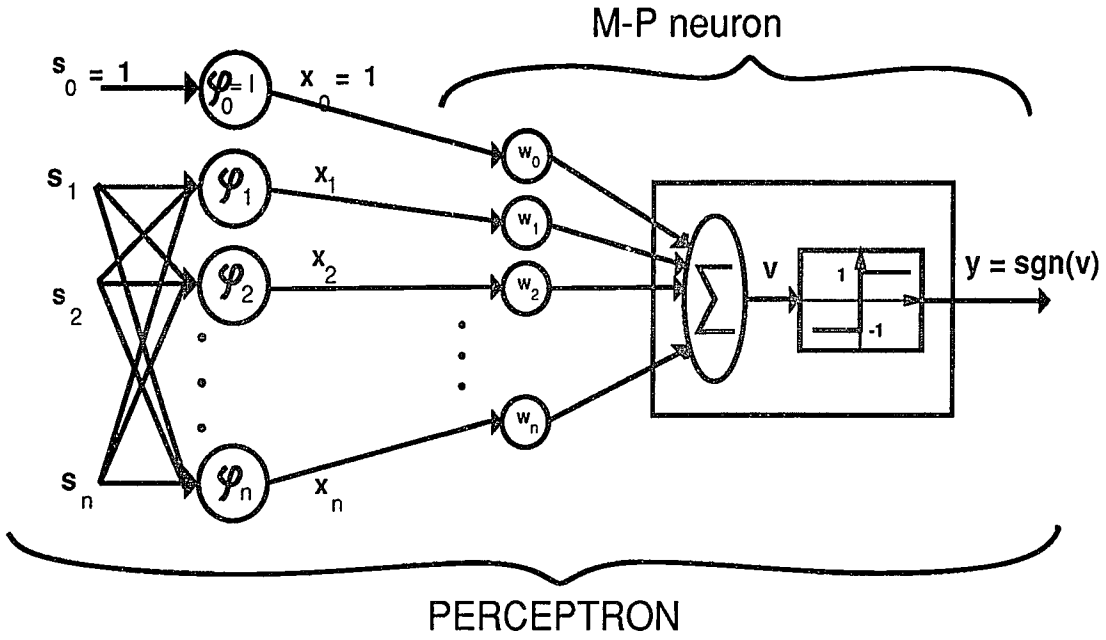

Figure 2.2: The Perceptron (F. Rosenbllat, 1957).

the input unit x_i^p, and the desired output d^p of the neuron is +1 or -1) :

$$\Delta_p w_i = \eta x_i^p d^p \tag{2.1}$$

2.2 Adaptive Single Layer Networks

2.2.1 The Perceptron

The Perceptron (see figure 2.2) is a supervised *Learning Machine* that was introduced by Rosenblatt [25]. It consists of a first layer of units performing an initial computation (this can be considered as a preprocessing), or a change of representation from $\{s_i\}$ to $\{x_i\}$. The second layer consists of a single formal neuron. Only the weights of the formal neuron are adaptive. The purpose of the

first layer is to change the representation, so that neuron in the second layer can learn the desired mapping. The Perceptron forms two decision regions separated by a hyperplane:

$$w_0 + w_1 x_1 + w_2 x_2 + ... + w_n x_n = 0$$

The transformed inputs $(x_1 \ldots x_n)$ and weights are typically real valued. Initially weights are arbitrary, and so is the response of the Perceptron on particular input. In order to get the desired response, the weight adjustment needs to take place, through the training procedure. Note that the Perceptron belongs to the classifiers linear in their parameters but it s not restricted to linear dependency in its input vector (see figure 2.2 and input vector **s**).

During training the desired value of the output d^p is compared to the output of the Perceptron y^p. A Perceptron *objective function* E can characterize how good a set of weights are. It is defined as a sum of distances of the misclassified input vectors x_i^p $(p = 1, 2, \ldots m; i = 1, 2, \ldots n)$ from the decision surface:

$$E = \sum_{p \ \text{"missed"}} |v^p| \tag{2.2}$$

To get better set of weights than the current, one would like to *minimize* E. The Perceptron learning algorithm aiming at minimizing the above criterion is a form of local search algorithm known as on-line gradient descent (*stochastic gradient*). The stochastic gradient, as oppose to "batch" version of the the minimizing procedure updates the weights after each input-output pair presentation. It requires less memory. Updating the weights is defined by:

$$\Delta_p w_i = -\eta x_i^p (y^p - d^p) \tag{2.3}$$

$\eta > 0$, is the scale factor that controls the rate of learning.

To make connection with traditional AI, we can say that Perceptron learning algorithm is a search algorithm. It begins with random initial state and finds

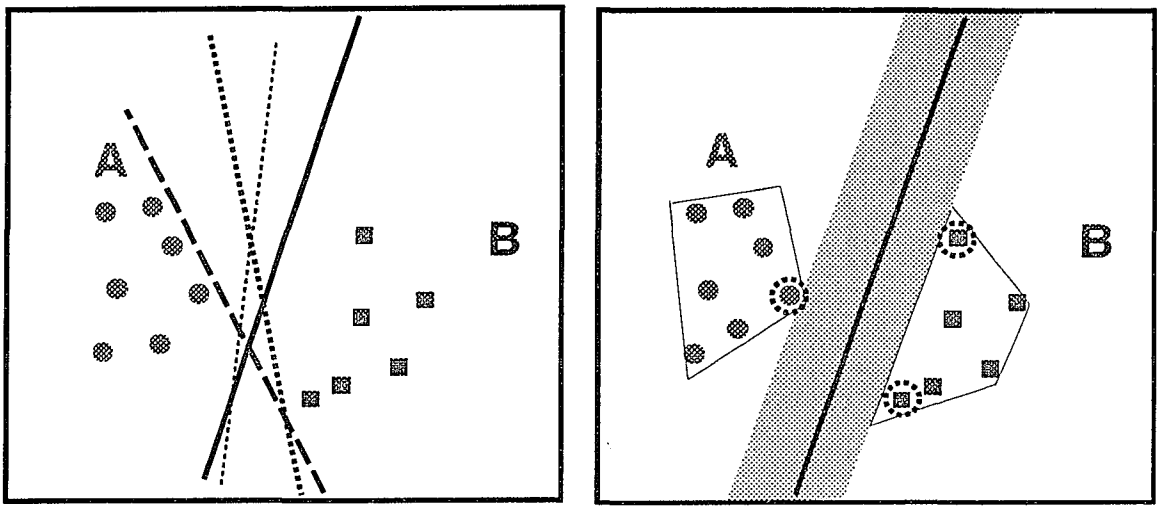

Figure 2.3: Linear separations. a) Perceptron b) Minmax separation. Supporting patterns are circled.

solution state. The search space is simply all of the possible assignments of real values to the weights of the Perceptron, and the search strategy is gradient descent.

The severe limitation of the Perceptron, pointed out by Minsky and Parpert [48], is that it is capable of learning only linearly separable problems in $\mathbf{x}$ space. Unfortunately, most problems do not supply such nice data. Indeed the Perceptron is incapable of learning to solve some very simple problems (in $\mathbf{x}$ space), a well known example is the XOR task [48].

2.2.2 Optimal Hyperplane

From the above discussion we can see that the Perceptron can be viewed as a classifier, capable of providing linear discriminant function (see section 1.4.1).

Let us assume that we have limited number of training examples from two classes A and B. The Perceptron learning algorithm will find a solution weight vector that corresponds to one out of many errorless separations (see figure 2.3.1). The question is is how to constrain the solution vector in order to obtain the boundary that provides best generalization. One posibility is to seek the solution weight vector that maximizes the minimum distance from the training patterns to the separating hyperplane (see figure 2.3.2). Such "Minmax" training procedure is known to lead to the maximum margin [44].

In this section we provide a general background for the linear classifier whose training procedure can be considered as a special Perceptron training that provides muximum margin separation between two classes. That classifier is called Optimal Hyperplane (OH) classifier [44]. The OH classifier is parameterized by a *subset* of training patterns that are closest to the decision boundary, so-called "supporting patterns" (see figure 2.3.2) . It has a small effective capacity while at the same time it has good generalization capability [24]. In Chapter 5 we will use this classifier for our adaptation experiments.

For a two class problem the OH classifier is defined by a hyperplane, whose direction $\mathbf{w}$ can be determined by a training procedure described in reference [44]. The hyperplane decision boundary is given by:

$$\mathbf{w} \cdot \mathbf{x} + b = 0 \tag{2.4}$$

and separates a set of pattern vectors $\mathbf{x}$, of dimension n, that belong to two classes A and B. The input to the training algorithm is a set of m examples $\mathbf{x}^i$ with labels y^i:

$$(\mathbf{x}^1, y^1), (\mathbf{x}^2, y^2), \ldots, (\mathbf{x}^m, y^m)$$

where $y^i = 1$ if $\mathbf{x}^i \in A$ and $y^i = -1$ if $\mathbf{x}^i \in B$.

Since there are an infinite number of equivalent classifiers which differ only by the norm of their weight vector $\mathbf{w}$, if the training set is linearly separable, there is a solution such that

$$\begin{aligned} \mathbf{w} \cdot \mathbf{x}^i + b \geq 1 \text{ if } \mathbf{x}^i \in A \\ \mathbf{w} \cdot \mathbf{x}^i + b \leq -1 \text{ if } \mathbf{x}^i \in B \end{aligned} \tag{2.5}$$

for all training patterns $\mathbf{x}^i$.

It can be shown [44] that the OH classifier, which maximizes the margins between training examples and the class boundary, is the solution of the optimization problem:

$$\min_{\mathbf{w}} \|\mathbf{w}\|^2$$

under the condition (2.5).

After training, in the *classification phase*, patterns are classified according to the rule:

$$\begin{cases} \mathbf{x} \in A & \text{if} \quad w \cdot \mathbf{x} + b \geq 0 \\ \mathbf{x} \in B & \text{if} \quad w \cdot \mathbf{x} + b < 0 \end{cases}$$

The minimization problem can be solved directly with numerical methods. Unfortunately that is impractical when the dimensionality of the input space is high. The minimization problem can be transformed in the dual space using the Lagrangian:

$$L = \frac{1}{2}\mathbf{w}^2 - \sum_{i \in A} \alpha^i (\mathbf{w} \cdot \mathbf{x}^i + b - 1) + \sum_{i \in B} \beta^i (\mathbf{w} \cdot \mathbf{x}^i + b + 1) \tag{2.6}$$

where $\alpha^i \geq 0$ and $\beta^i \geq 0$ are Lagrange multipliers of the constraints (2.5) and they satisfy the conditions:

$$\begin{aligned} \alpha^i \cdot (\mathbf{w} \cdot \mathbf{x}^i + b - 1) = 0, \text{ for } \mathbf{x}^i \in A \\ \beta^i \cdot (\mathbf{w} \cdot \mathbf{x}^i + b + 1) = 0, \text{ for } \mathbf{x}^i \in B \end{aligned} \tag{2.7}$$

The optimization problem is now equivalent to finding the maximum of the Lagrange function with respect to the parameters α^i and β^i. It can be shown that the shortest vector $\mathbf{w}$ is given in the form:

$$\mathbf{w}^\star = \sum_{i \in A} \alpha^i \mathbf{x}^i - \sum_{i \in B} \beta^i \mathbf{x}^i \tag{2.8}$$

where the summation runs only over the supporting patterns $\mathbf{x}^i$ that satisfy the equality:

$$\mathbf{w} \cdot \mathbf{x}^i + b = y^i \ . \tag{2.9}$$

Recently, the extension of the Optimal Hyperplane method has been proposed [24] that allows maximum margin training of various non-linear classifiers that are linear in their parameters but not restricted to linear dependancy on their input. Examples are Generalized Perceptron and polynomial classifiers (sigma-pi unit networks, see section 1.4.1).

2.2.3 The ADALINE

Another linear machine (see figure 2.4) is ADALINE (ADaptive LINear Element) introduced in 1959 by Widrow [56]. It is a Perceptron where threshold unit (hard limiter) has been replaced with purely linear unit. Its training algorithm proposed by Widrow and Hoff [54], known as Least Mean Squares (LMS), is:

$$\Delta_p w_i = -\eta x_i^p (v^p - d^p) \tag{2.10}$$

Similarly to the Perceptron learning algorithm, it is on-line (or *stochastic*) gradient descent algorithm. It minimizes the mean-squared-error (MSE) objective function:

$$E = \sum_{p=1}^{m} (v^p - d^p)^2 \tag{2.11}$$

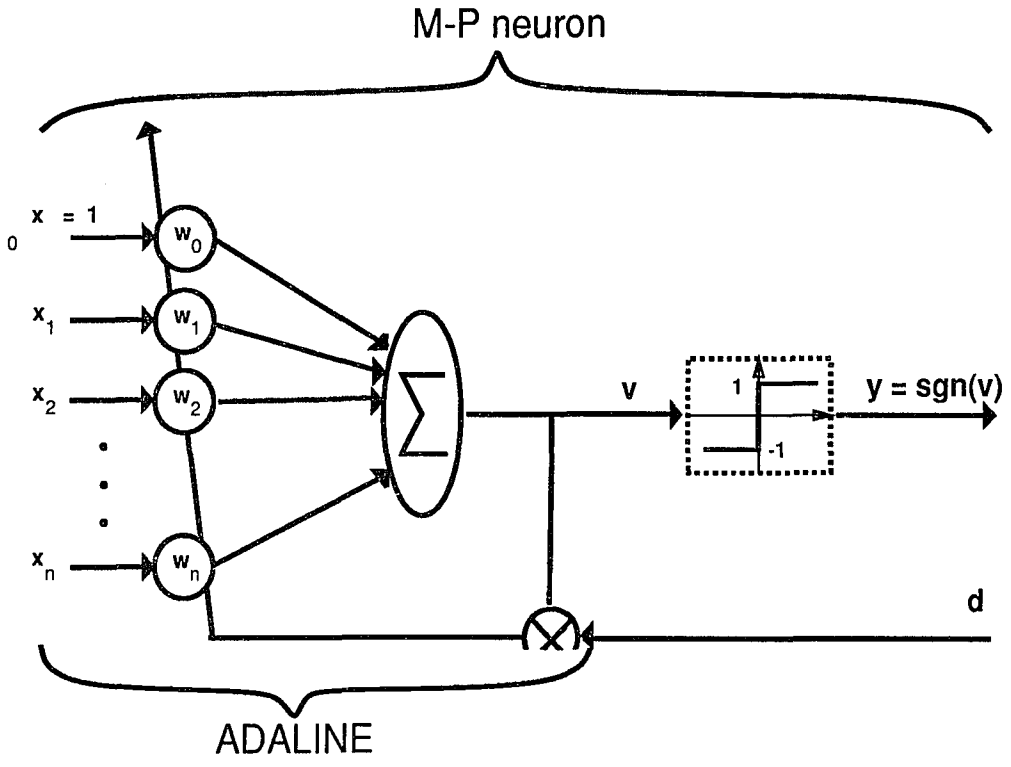

Figure 2.4: The ADALINE (B. Widrow, 1959).

2.2.4 Types of Activation Function

When they were introduced, McCulloch-Pitts neurons and Perceptrons where associated with a nonlinear threshold activation function. (see figure 2.1)

Later, other types of nonlinearities were introduced, and they typically represents smoothed version of a threshold function . They are *continuous* and *differentiable* and they produce continuous states of output units.

Examples are, logistic function:

$$f(v) = \frac{1}{1 + e^{-v}} \tag{2.12}$$

or the sigmoid function unit [57], where the activation function is a scaled hyperbolic tangent:

$$f(v) = a \tanh(bv) \tag{2.13}$$

where a and b are positive parameters.

The training rule for the sigmoid unit (see figure 2.5) is so-called *"delta-rule"* , or Widrow-Hoff algorithm applied to the sigmoid unit rather than to linear unit:

$$\Delta_p w_i = -\eta x_i^p (y^p - d^p) f'(v^p) \tag{2.14}$$

where $y^p = f(v^p)$. This rule minimizes the mean-squared-error objective function:

$$E = \sum_{p=1}^{m} (y^p - d^p)^2 \tag{2.15}$$

2.3 Multi-layer Networks

2.3.1 The Multi-layer Perceptron (MLP)

Multi-layer Perceptrons (also called multi-layer networks), or feed-forward nets with one or more layers of nodes (so called "hidden layers"). These additional

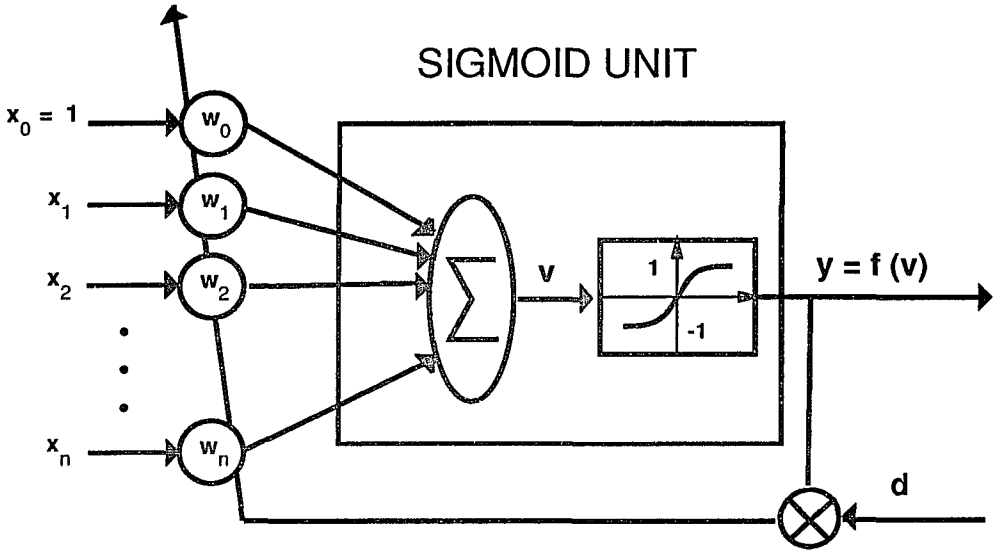

Figure 2.5: The sigmoid unit.

layers contain nodes that are not directly connected to both inputs and output nodes.

A multi-layer Perceptrons can perform (represent) any Boolean function by setting the weights to specified values. (the proof is based on the fact that a single formal neuron can implement 2 input NAND gate). From that follows that multi-layer Perceptrons, solve our knowledge representation problem of the XOR task. However, multi-layer networks cannot be trained with the learning rules specified in previous sections. Why? The training algorithms we mentioned so far use the desired values of the output units to calculate the error and adjust weights. But now we have hidden units. When there is an error at the output of the network, hidden units are also "responsible" for this error. How to adjust their weights in order to respect this "shared" responsibility? This is known as the *credit assignment problem*, whose solution will be described in the next section.

2.3.2 Training Multi-layer Networks - Backpropagation

The introduction of the *backpropagation* training procedure [49, 50, 51, 52] solves the problem of training multi-layer networks. It performs on-line (*stochastic*) gradient descent in weight space. The key element for introducing backpropagation is the use of continuous differentiable nonlinearities. The backpropagation algorithm minimizes the mean-squared-error, that is, the sum of the squared errors e_p, over the training set (p = 1 ... m), where the error is defined as a difference between the actual outputs y^p of the system and the desired outputs d^p. e_p is:

$$e_p = \sum_{j \in output} (y_j^p - d_j^p)^2 \tag{2.16}$$

and the MSE (same as equation 2.15 but for multiple outputs):

$$E = \frac{1}{p} \sum_p e_p \tag{2.17}$$

Weights are updated after the presentation of each pattern p by backpropagating the error measures layer by layer, from the output back to the input according to the following rule ("generalized delta rule"):

$$\Delta_p w_{ji} = \eta \delta_j^p y_i^p \tag{2.18}$$

where:

$$v_j^p = \sum_i w_{ji} y_i^p \tag{2.19}$$

and:

$$y_j^p = f(v_j^p) \tag{2.20}$$

Deltas (e.g. units errors) are calculated as follows:

$$\delta_j^p = -(y_j^p - d_j^p) f'(v_j^p) \tag{2.21}$$

if j is an output unit, and:

$$\delta_j^p = f'(v_j^p) \sum_k \delta_k^p w_{kj} \tag{2.22}$$

if j is a hidden unit.

This is a recursive definition in which the unit error is determined by the derivative of its nonlinear function $f(v_j^p)$ multiplied by the weighted sum of the errors to which the unit sends activation via outgoing connections. A given error term δ_k^p in the summation is, in fact, weighted by the strength of the connection pointing from the hidden unit j to the unit k that is the source of the delta. The errors are first computed for the output units, during forward propagation pass, and these are then propagated backward, (so called "backpropagation pass") to all units pointing to the output unit in the layer below. Network with 3 layers of neurons trained with backpropagation can learn arbitrary complex nonlinear decision surfaces. More details about comparison of standard pattern recognition techniques, and neural nets can be found in [36].

2.3.3 Time-varying signals - TDNN

In the case, of the time-varying input signals, one way to handle the problem would be to introduce dynamic networks, whose nodes are described by differential or difference equations. But instead of using the network that is truly dynamic, it is possible to use a static network to process time series data by simply converting the temporal sequence into a static pattern by unfolding the sequence over time. That is, time is treated as another dimension of the problem. This can be accomplished by feeding the inputs delayed in time. It is convenient to introduce time-delay neurons (see figure 2.6). A Time-delay neuron is a neuron whose inputs has been delayed in time. A network of multiple layers, where each layer is composed of time delayed neurons is called Time Delay Neural Network (TDNN) [42, 41].

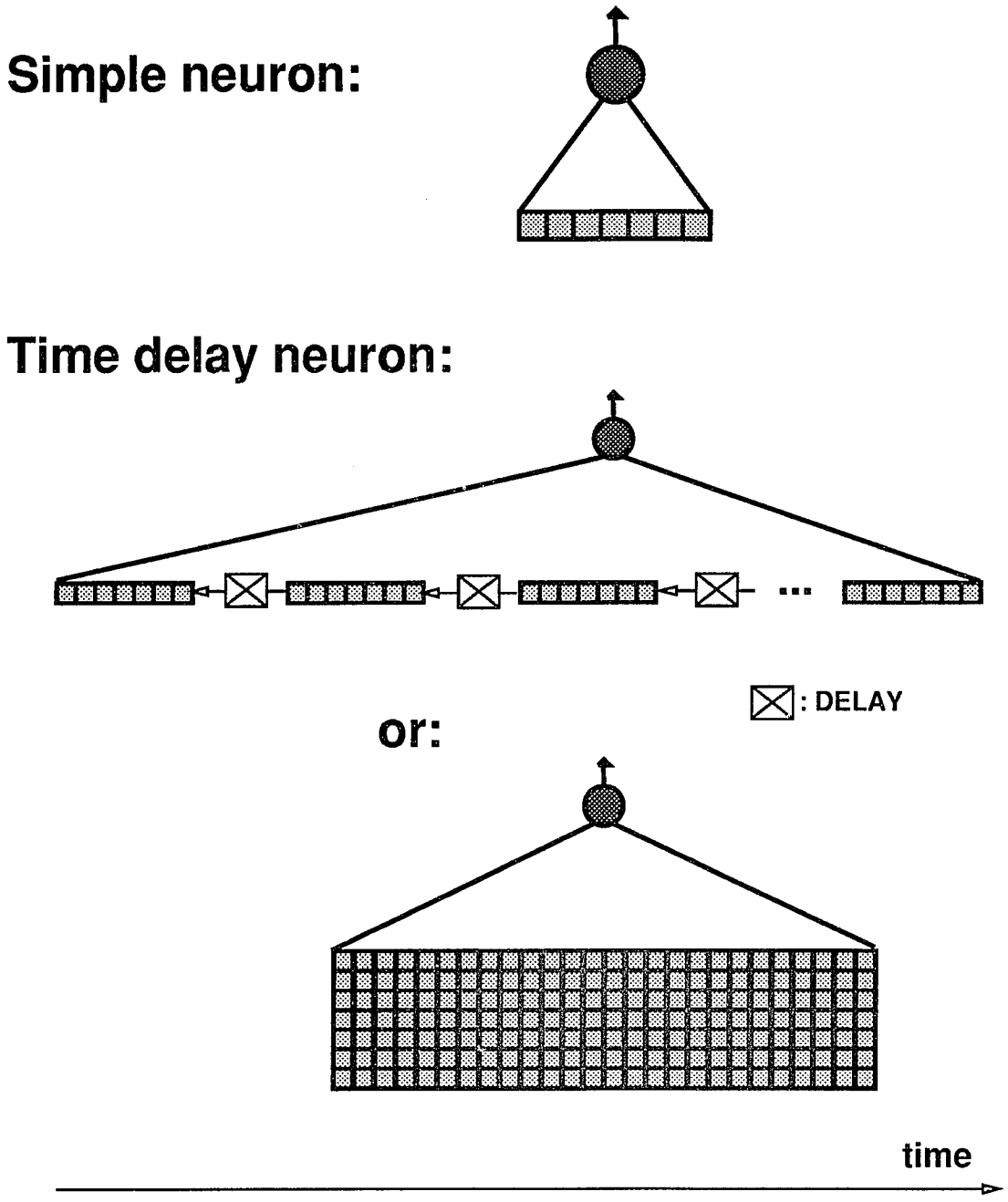

Figure 2.6: The time-delay neuron

TDNN is therefore capable of modeling the system where the output has *finite* temporal dependence on the input, that is:

$$u(k) = F[\mathbf{x}(k), \mathbf{x}(k-1), ..., \mathbf{x}(k-n)] \tag{2.23}$$

when the function $F(.)$ is a weighted linear sum, this architecture is equivalent to a linear *finite impulse response* (FIR) filter. because there is no feedback in this network, it can be trained using the standard backpropagation algorithm.

2.4 Generalization in Neural Networks

In this section we give a short overview of the elements of learning theory. We describe several issues and the limitations of the Multi-layer Perceptron (MLP) in the context of learning process. We discuss methods that can improve generalization in neural networks.

The question that can be raised is why and when, the MLP can be considered as powerful learning machine. That is, how trough design and learning one can produce MLP that generalize correctly to new cases after training on a sufficiently large set of examples from some domain. In much of the research, there is no formal definition what it means to generalize correctly.

Some of the issues that need to be addressed are:

1. What class of problems can neural network learn?
2. Under what conditions do neural networks generalize?

From the previous discussion we have concluded that MLP is capable of learning almost everything. Theoretical results indicate that MLP s capable of forming arbitrarily close approximation to any continuous nonlinear mapping (given a

set of examples, the backpropagation algorithm can be called upon to learn the mapping at the example points). Unfortunately this is true only if the size of the network grows arbitrarily large [58]. The second condition is that infinite amount of training data is available. Since this requirements are unrealistic in practice, one has to focus on the generalization capabilities of the neural networks in practical problems.

2.4.1 Choosing the Network Size

Choosing the proper size of the network for the given amount of training data is important. If the network is too small it will not be capable of forming the good model of the problem. On the other hand, if the network is too big then it may be too capable [59]. That is, it may be able to implement numerous solutions that are consistent with the training data, but exhibit poor performance on the data that it has not been trained on. In other words the network will not be able to *generalize* well and the solution is likely to be a poor approximation to the actual problem.

To grasp the problem of designing and training of the neural network, one can make analogy with curve fitting (see figure 2.7).

The problem is to fit experimental points with a polynomial. If one has limited number of points, fitting with a high-order polynomial can lead to ridiculous interpolation. The curve includes all the experimental points (including possible noise points). Unfortunately, the curve makes no sense between the experimental data points. The polynomial has too many free parameters, and the available data points are insufficient to constrain the solution in the desired direction (*underdetermination*).

If we try to fit the same number of experimental data with a line, we can find a regression solution, that does not include all the experimental data points, but

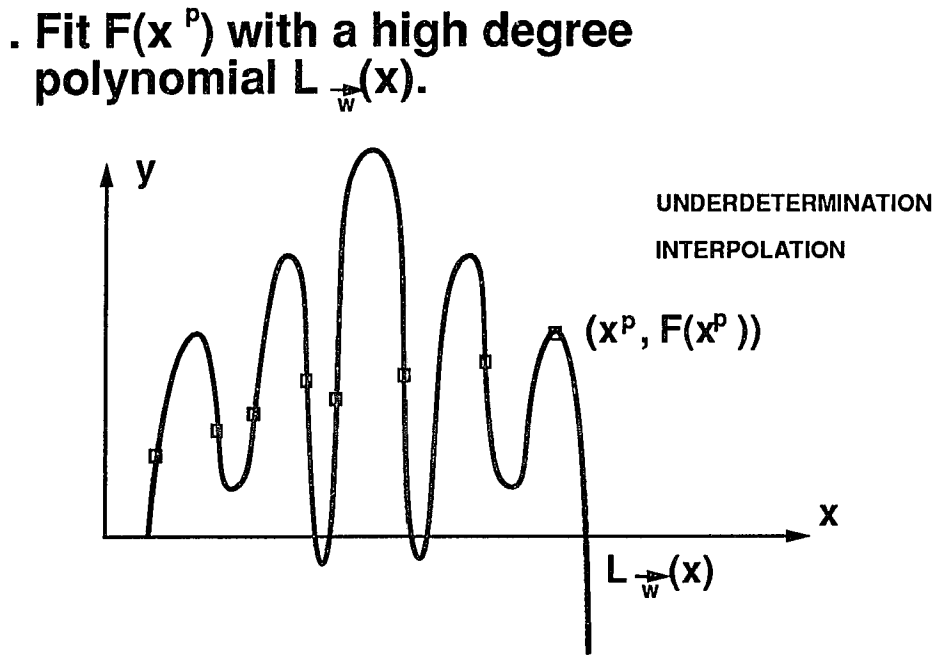

Figure 2.7: Analogy with curve fitting.

makes sense for the points between the experimental ones. In this case we have more experimental points then free parameters (*overdetermination*), but that allows for reasonable generalization.

The conclusion of this analogy is that the number of *free parameters* is one (arguably oversimplified) measure of the complexity of the model.

Both theoretical studies and experimental results [15] show that good generalization can be achieved if the so called "capacity" of the neural network is matched to the number of training examples. The capacity of a neural network is on the other hand related to the number of free parameters [59]. Before discussing different methods for controlling the capacity of the neural networks we will give some basic ideas about what is the capacity of the adaptive classifier.

2.4.2 Capacity and VC-dimension

The *capacity* of an adaptive classifier (e.g. neural network) is a measure of the richness of its functional capabilities. It indicates how large is the set of the different functions $F(\mathbf{x}, \mathbf{w})$ which can be implemented by varying the parameters w of the classifier within all allowed values? One quantitative measure of the capacity of the classifier is the VC-dimension [44].

The VC-dimension of a set of indicator functions is the maximum number h of vectors which can be shattered in all possible 2^h ways using functions in the set.

For *linear* decision rules the VC-dimension is $h = n + 1$, (where n is the dimensionality of the input space). For example, the VC-dimension for the linear Perceptron with two inputs ($n = 2$) is 3. In this case it is equal to the number of free parameters. While it is possible to find a set of 3 points that can be linearly dichotomized in all 2^3 ways (see figure 2.8) it is not possible to find a set of 4 points that can be linearly dichotomized in all 2^4 ways.

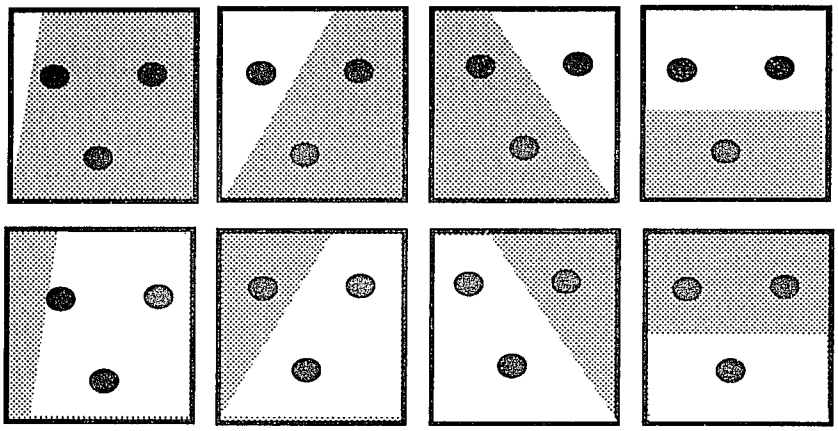

Figure 2.8: Eight possible ways to dichotomize three points in a plane.

Unfortunately, to obtain the exact value of the VC-dimension for other, more complex classifiers is a very hard task.

One of the most commonly used principles in learning problems is the principle of *empirical risk minimization* [44]. According to this principle in order to obtain good estimate of the *generalization error* one should guide the learning process so as to minimize the number of errors on the training set E_{train}.

The theory of empirical risk minimization [44] was developed in the 1970s. In the pattern recognition case, this theory provides a bound on the probability of the generalization error E_{gen}, when the classification function is chosen among a set of functions with finite VC dimension. For a set of classification functions $\{F(\mathbf{x}, \mathbf{w})\}$, with probability $1 - \eta$, for a number of training examples $p > h$, simultaneously for all classification functions, the generalization error E_{gen} is smaller than the following *guaranteed risk*:

$$G = \nu(p) + \epsilon(p, h, \eta) \tag{2.24}$$

where $\nu(p)$ is the frequency of errors on the training set, and ϵ is the confidence interval, which depends on the number of training examples p, on the

VC-dimension h of the set of functions, and on η. Confidence interval ϵ is proportional to:

$$\epsilon_0 = [h(ln2p/h + 1) - \eta/12]/p \tag{2.25}$$

for small training error and to $\sqrt{\epsilon_0}$ for training error close to 1 [44].

The capacity (VC-dimension h) has important role in this dependence (see figure 2.9). When the VC-dimension of the set of functions increases, the frequency of errors on the training set decreases. At the same time the width of the confidence interval ϵ increases. This two terms compete, and as a result there exist optimal VC-dimension value that leads best (minimal) generalization error.

Another important parameter that determines the value of the guaranteed risk is the number of training patterns. For the fixed capacity h, the difference between generalization error and training error depends on the size of the training set p (see figure 2.10). The generalization error always exceeds the training error, but the difference between them decreases towards 0 as the size of the training error tends to ∞. Unfortunately in learning tasks, we usually have limited amount of training data and we are not in the regime where the difference between the two terms is negligible.

Being limited with amount of empirical data one has to try to control the capacity. One way to accomplished that is through architectural modifications. In the next section several methods for improving generalization through architectural modification or through modification of the learning algorithm will be described. In this thesis we show that the efficient utilization of the available training data (paying special attention to rare examples of the handwriting) can also improve the generalization. More details about that approach will be given in chapters 4 and 5.

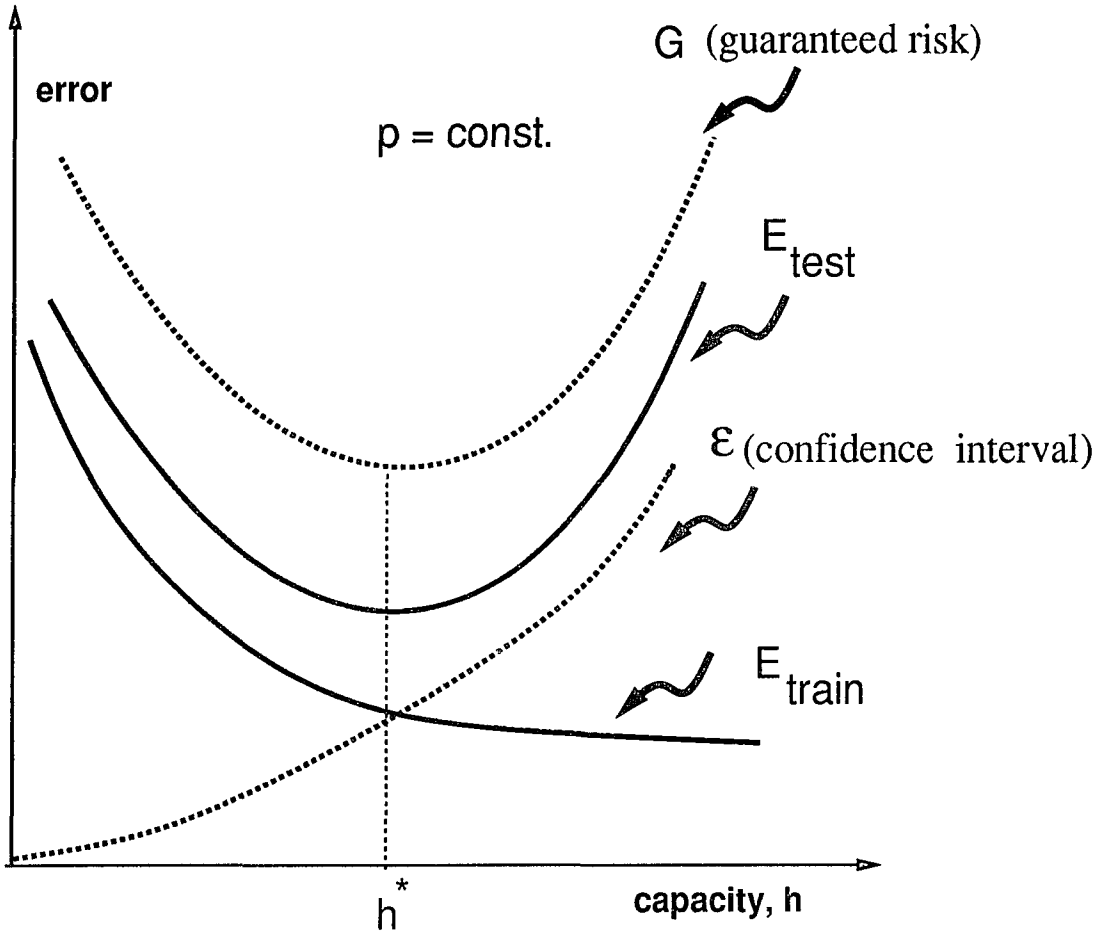

Figure 2.9: When the VC-dimension of the set of functions increases, the frequency of errors on the training set decreases. At the same time the width of the confidence interval ϵ increases. This two terms compete, and as a result the guaranteed risk (i.e. upper bound on the generalization error) has a minima for the optimal complexity of the classifier h^*.

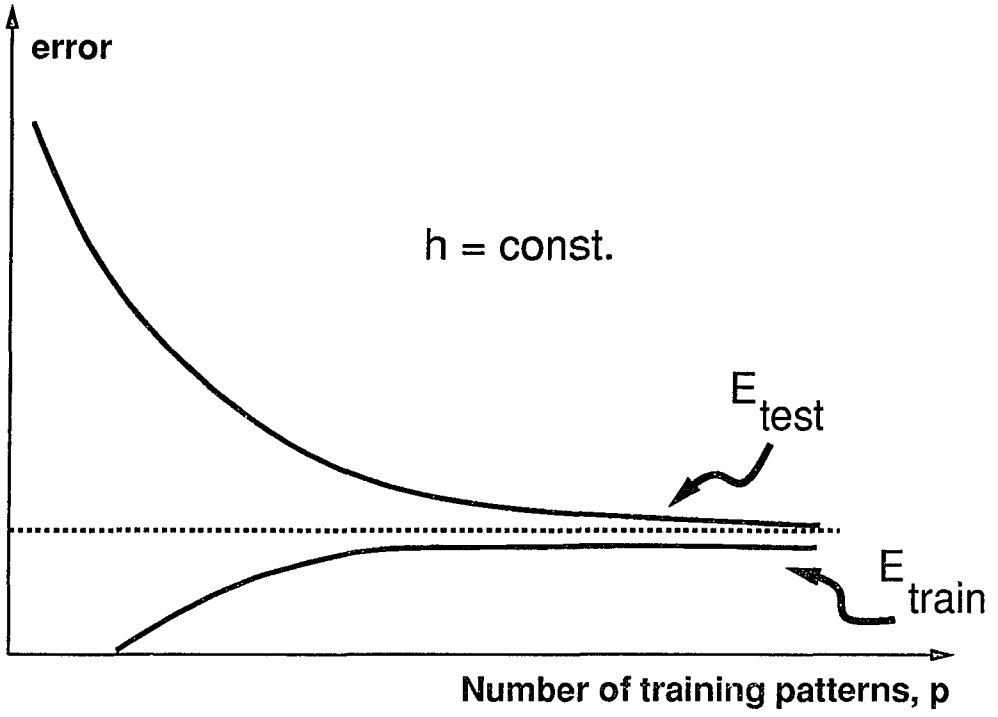

Figure 2.10: Learning curves. Training and test error as a function of the number of examples m. The capacity h is fixed.

2.4.3 Practical Methods for Improving MLP Generalization

Improving Generalization Through Pruning

We now review, how different modifications in architecture of the network, influence generalization capabilities. One can start with a small size of the network (restrictive architecture - small capacity) and "grow a network". Approaches that use such a technique include Cascade Correlation [60], the Group Method of Data Handling [61, 62], Projection Pursuit [63, 64] . In the growth process, additional nodes are created during the training process.

Another possibility is to start with large network (large capacity) and then apply a pruning technique that destroys weights and/or nodes that does not contribute to the solution. This can be justified, since it is generally true that there is a large amount of the redundant information contained in the weights of a fully connected MLP. Thus it seems plausible that we could eliminate weights from the network, and at the same time retain the functional capability needed to solve the problem. We discuss this "reduction" techniques in more details.

Though the simplest approach to pruning is to delete the smallest weights in the network, this however has some drawbacks, since the solution can be quite sensitive to these weights. A better approach is to use the **Optimal Brain Damage** technique for reducing the size of the learning network by selectively deleting weights [65]. In OBD, the network is first trained using the Backpropagation algorithm. Then the weights with smallest *saliency* are deleted. After this, the reduced size network is re-trained to obtain the final solution. The saliency for weight w_{ji} is defined as:

$$s_{ji} = \delta E w_{ji}^2 \tag{2.26}$$

where δE measures the sensitivity of the objective function to small perturbations

in w_{ji}. This procedure avoids the deletion of small weights which have a large influence on the solution. In OBD , the sensitivity measure is approximately:

$$\delta E \approx \frac{\partial E}{\partial w_{ji}^2} = \sum_p \frac{\partial^2 E_p}{w_{ji}^2} = \sum_p r_j^p (y_i^p)^2 \tag{2.27}$$

The approximation is obtained by expanding δE in a Taylor series and dropping a second-order cross terms as well as all high-order terms. The first order terms drop out automatically because the network has converged to local minimum where first-order terms are zero.

Deleting a law-saliency parameter is defined as setting it to 0 and freezing it there. In some applications it has been found that the number of weights can be reduced by a factor of four [65]. The recent extension of the OBD method which uses the full Jacobian (instead of just the diagonal elements) is described in [66]

.

Improving Generalization Through Weight Sharing

Another way to reduce the number of weights in the network is to *specialize* the network to the task at hand. For example, let us consider the task of recognizing handwritten characters. One way to do it is to perform shape recognition by detecting and combining local features. We can require the network to do so by constraining the connections in the first few layers to be *local*: individual neuron process input only from a local "receptive field" on the layer below. In addition, if the feature detector (implemented by a neuron) is useful on one part of the image, it is likely to be useful on other parts of the image as well. Therefore it can be useful to constrain the set of weights of different neurons to be equal. In other words, we create a collection of neurons that perform the same operation on different parts of the image. These neurons may be seen as collectively performing a *convolution* of the input image with a small-size

kernel, followed by squashing function. The number of *free* parameters for this entire collection of neurons is equal to the size of the kernel (the size of one of the receptive fields). Experiments showed the advantage of weight sharing for two-dimensional shapes in the context of backpropagation [52, 40], as well as in one-dimensional version known as Time-Delay Neural Networks, successfully applied to speech [42, 41], and on-line character recognition [6]. Shared-weight network are even more efficient when combined with the idea of subsampling: the network is composed of succession of shared-weight layers with decreasing spatial resolution [67].

Improving Generalization Through Reguralization

Several automatic techniques have been proposed to minimize the *size of the network* during training in data-dependent way. The goal is to improve generalization. The simplest one is known as **weight decay** [68, 69] Weight decay can be viewed as a way of reducing the *effective number of weights* in the network by encouraging the learning algorithm to seek solutions that use as many zero (or near zero) weights as possible. Weight decay relies on the implicit assumption that the small weights provide more noise-resistant solutions (as this can be proven in the linear single-layer case). Intuitively, keeping the weights small will cause the sigmoid functions to operate in the linear region, thereby reducing the *effective* number of parameters in the system (i.e. multi-layer linear system can be reduced to a single layer system). This is accomplished by adding an extra regularization term to the objective function that penalizes the network for using nonzero weights. The compound objective function is of the form:

$$E(w) = \sum_p E_p(w) + \frac{\lambda}{2} \sum_i w_i^2 \tag{2.28}$$

where the first term is the usual squared error criterion, and the second term is new term involving a sum over all the weights in the network. The λ parameter is

a small positive constant that is used to control the influence of this term relative to the squared error term. The learning algorithm derived from this criterion is a simple extension of the backpropagation algorithm. In fact, the only modification is to subtract an extra term of the form $\eta\lambda w_i(k)$ (see equation 2.18), each time the i-th weight is updated.

Many other regularization terms have been proposed. Some force the weights to form clusters of similar values [70], some tend to eliminate small useless weights by pulling harder towards 0 on small weights than on the larger weights [71]. In general, regularization terms will reduce the effective *capacity* of the neural network and improve the generalization performance.

In neural networks with sigmoid units the capacity control can be obtained by **early stopping** of the training session before reaching the minimum of the training error. Here, before the training starts, the weights are initialized with small random values such that the total input of the neurons lie in the linear part of the sigmoid squashing function (equation 2.5). The capacity increases progressively during training. The optimum is detected by cross-validation. Cross-validation can be done by computing the performance of the system on a small set of patterns, distinct from the training set and the test set. This form of regularization is somewhat equivalent to a weight decay [15]. In our experiments we use this form of regularization.

2.5 Summary

In this chapter we reviewed the supervised neural networks and the corresponding learning procedures. We discussed the issues concerning the ability of the neural network to generalize well. We described some practical methods for improving generalization in neural networks.

3 TDNN for On-line Handwritten Character Recognition

In the previous chapter, we reviewed the multi-layer feed-forward neural networks. We also mentioned that in the case of time-varying input signals it is convenient to introduce time-delay neurons, and a network of multiple layers, where each layer is composed of such neurons called Time Delay Neural Networks (TDNN). In this chapter we first describe the on-line handwritten isolated character recognition system that we have been working on [6]. The system was designed to recognize either digits or uppercase letters.

We then present the results, of training such a network (with small architectural modifications) on digits, uppercase, lowercase letters and punctuation symbols simultaneously.

3.1 The Method

In on-line recognition system [6] characters that are entered on a touch terminal are represented as a sequences of $[x(t), y(t)]$ coordinates. Characters were entered

in disjoint boxes, therefore no segmentation was done in the preprocessing stage. The system is trained to be *writer independent.* Either digits or uppercase letters are considered. Recognition is not based on any syntax, semantic or contextual information. After preliminary preprocessing stage, feature extraction and classification is performed using Time Delay Neural Network. This is multi-layer feed-forward network, trained with backpropagation algorithm. The network is highly constrained in order to capture local topological features.

3.2 Preprocessing

As described in section 1.3 the goal of the preprocessing is to build some invariance, and to provide salient features. It is the stage where *a priori* knowledge (to the extent possible) of the task at hand is incorporated.

Handwriting is captured by a touch-sensitive pad as a sequence of coordinates $[x(t), y(t)]$ regularly spaced in time. Sampling rates range between 50 and a 100 points per second, depending on the pad. Each character is composed of one or or several strokes.

The preprocessing is designed to preserve the temporal structure of the input data. The resulting representation is a sequence of vectors ordered with increasing time. The first steps of the preprocessing, *resampling, centering and rescaling*, significantly reduce the variability within classes by removing time and scale distortions . The resampled characters are represented as a sequence of points $[x(n),\ y(n)]$, regularly spaced on the trajectory, as opposed to the input sequence, which was regularly spaced in time. In the last step of the preprocessing encoding of certain local parameters (slope and curvature) of the handwriting is done (see figure 3.1), in order to describe characters in a way that will help discrimination between different classes in the recognition phase.

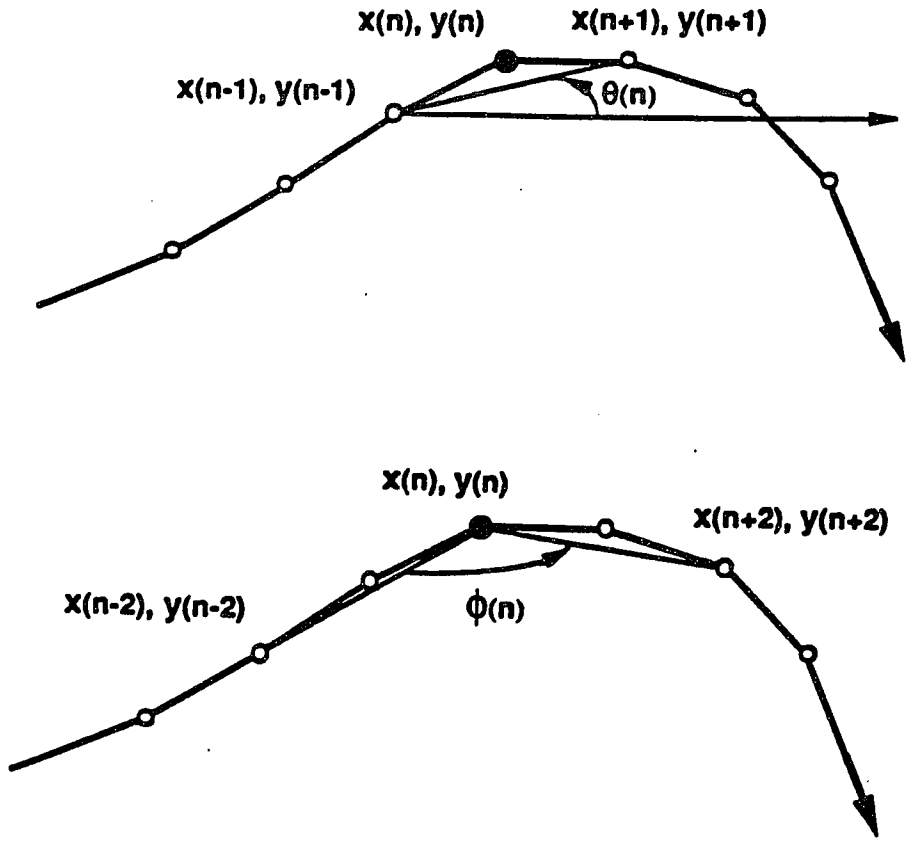

Figure 3.1: Estimation of the slope and the curvature.

The direction of a stroke is encoded by the direction cosines of the tangent to the curve at point n (see figure 3.1). These parameters can also be thought of as the first derivatives dx/ds and dy/ds, where $ds = \sqrt{dx^2 + dy^2}$. The local curvature is measured by the angle between two elementary segments: $\phi(n) = \theta(n+1) - \theta(n-1)$ as shown in figure. This angle is encoded by its cosine and sine, which can be computed from the direction cosines of $\theta(n-1)$ and $\theta(n+1)$ through trigonometric formulas.

The outcome of the preprocessing is an *intermediate representation* for each character. This representation consists of a sequence of 90 feature vectors ($n = 90$). Each of these vectors have the following components:

$$\begin{cases} f_0(n) &= penup(n) \\ f_1(n) &= \dfrac{x(n) - x_0}{\delta_y} \\ f_2(n) &= \dfrac{y(n) - y_0}{\delta_y} \\ f_3(n) &= \cos\theta(n) \\ f_4(n) &= \sin\theta(n) \\ f_5(n) &= \cos\phi(n) \\ f_6(n) &= \sin\phi(n) \end{cases} \tag{3.1}$$

where $penup(n)$ is one if the pen is down and zero otherwise, δ_y is the height of the character and x_0 and y_0 the coordinates of the center of the character.

3.3 The Network Architecture

This intermediate representation encodes local features (figure 3.2). It is input to the neural network whose task is to extract more complex, global features, and perform classification. The network is highly specialized Time Delay Neural Network or TDNN [42]. Figure 3.3 shows its architecture.

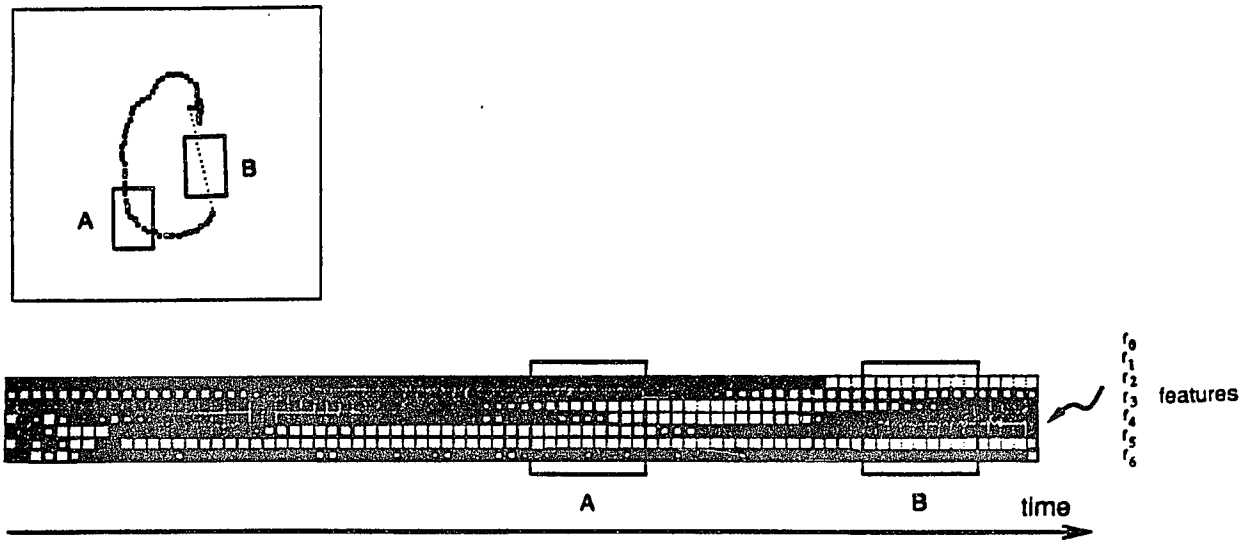

Figure 3.2: Recognition preprocessing. The original characters "C" shown at the top has been resampled with 90 points. The intermediate representation, shown at the bottom consists of 7-component vectors containing information about pen up/down, direction, and curvature at each point. Each vector is represented as column of boxes whose size indicates the magnitude of a component and whose color indicates the sign. The intermediate representation is the input to the neural network.

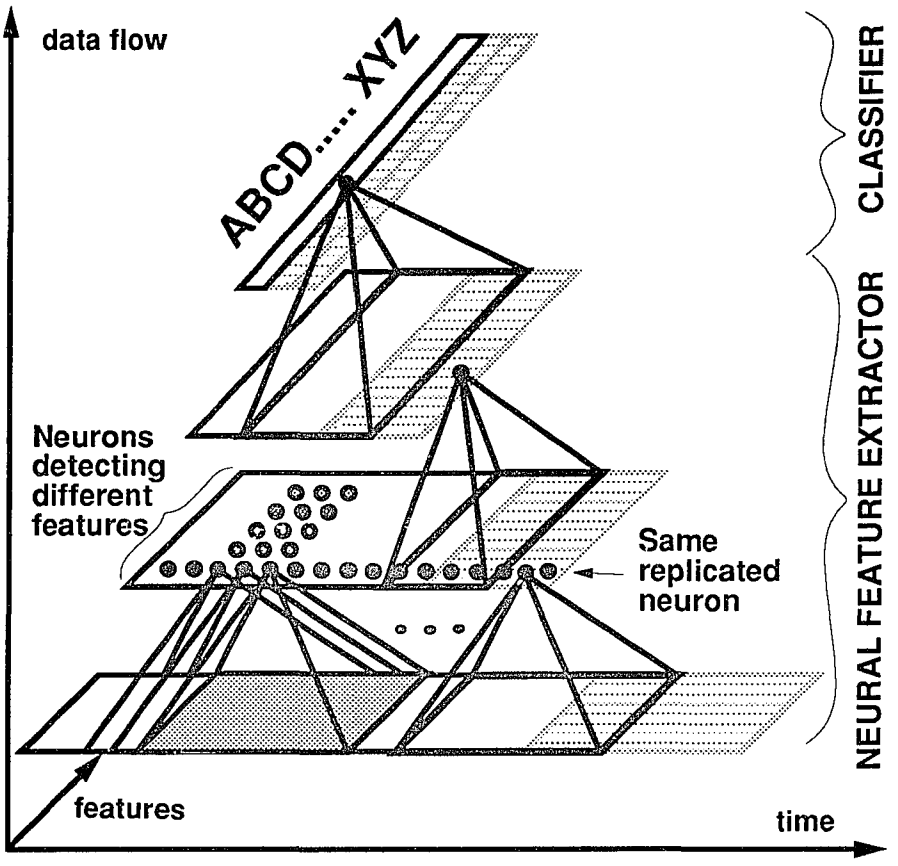

Figure 3.3: Architecture of the Time Delay Neural Network (TDNN). Not all the neurons and all layers are represented.

It is a feed-forward *layered* network, having four layers of weights connecting five layers of units (the input as a degenerate "layer #0"). The connections are restricted to be *local.* A particular unit has a receptive field in the layer below it limited in time direction. Since this network should look for the sequences, the receptive field of unit i will have considerable overlap with the receptive field of unit $i-1$.

The network is constrained to be *convolutional.* (i.e. the concept of weight sharing has been applied). The weights connecting one feature vector in one layer to its receptive field in the layer below it are the same as the weights connecting the next feature vector to the next receptive field. By doing that, the designer is giving to the network prior knowledge that problem at hand has certain time-invariant properties. The same feature can appear at different positions in time. This specialization greatly reduces the complexity of the network (described by a number of free parameters), at the same time it improves generalization. The network which is described in [6] has 34,066 connections, but only 4450 independent weights.

Weight sharing is combined with idea of *subsampling* [5, 67]. Each layer has a coarser time representation than the preceding layer. Only one every s values in the convolution is kept (and actually computed).

There is some kind of tradeoff between the loss of time resolution introduced by subsampling and the increase in the number of features in the subsequent layers. This is called "bi-pyramidal" scaling. The output layer consists of only one feature vector of N coefficients (N is the number of classes) that can be considered as the ultimate, most abstract features. More details about the specifications of the network can be found in reference [6].

3.4 The Training

The neuron used is a sigmoid unit whose activation function is a scaled hyperbolic tangent:

$$f(v) = a \tanh bv$$

where $a = 1.716$ and $b = 2/3$ [72]. By convention, the input x_0 is always set to 1, so that the weight w_{i0} serves as a bias value.

It should be noticed that we use a symmetric activation function (odd sigmoid function). Non-symmetric activation functions (ranging between 0 and 1 for instance) yield much slower convergence [73].

The choice of the parameters a and b of the sigmoid has to be made in conjunction with the choice of the target values of the output layer and the initialization of the weights.

- It is important to choose target values within the range of the sigmoid. With this choice of a and b, target values can be chosen conveniently to be -1 and $+1$. Different authors use target values for desired outputs equal to the asymptotic values of the activation function. The problem is the target values on the asymptotes tend to drive the weights to infinity, and slow down learning by orders of magnitude. The generalization performance is also affected.
- Initial values of the weights should not be too big (that might saturate the units and result in very small gradients). Initial values of the weights should not be too small (this can cause the gradients to be very small, and the learning to be initially very slow; the collapse with all weights going to 0 might also occur). The value of the weighted sum v (or potential) should stay in the linear part of the sigmoid. This imposes the requirement that random initial values of the weights should be smaller for units with many

inputs and larger for units with few outputs. In practice, with the values of a and b chosen above, weights are initialized with random values, uniformly distributed between $-2.4/F_i$ and $2.4/F_i$, where F_i is the total number of inputs to unit i.

The weights are adjusted during a supervised training session which performs gradient descent in weight space with a mean-squared-error (MSE) objective function (see section 2.3.2). It is a modified version of the standard backpropagation algorithm where the learning rate is adjusted during training with the help of the diagonal approximation of the Hessian matrix [74]. The target values are binary. The algorithm cycles through the training patterns and a modification of the weights occur at each presentation of a patterns (*stochastic gradient* procedure). In addition, the training algorithm has to average the weight updates of the group of neurons with "shared weights".

The capacity control is obtained by *early stopping* of the training (see section 2.4.3). For this purpose a validation set, distinct from both the training set and the test set, is defined. The training is stopped when the mean-squared-error on the examples of the validation set stop decreasing significantly or starts increasing. The network which gives the smallest mean-squared-error on the validation set is kept and re-tested with an independent test set of examples from different writers. The result of this last test is what is called the performance of the network.

3.5 Training on Full Character Set

The network we described, was trained [6] originally to recognize either digits or uppercase letters. Since there are several situations in which digit cannot be distinguished from the corresponding capital letter, for instance zero versus "O", one versus "I", five versus "S" and six versus "G", the contextual information was required, i.e. it was necessary to know *a priori* whether the input should be interpreted as a digit or letter, so the network is not required to make this distinction.

For this reason the output was split into two fields, one devoted to digits, and one devoted to letters. The rest of the network was not divided so that (in all layers except the last) training on digits would modify the recognition of letters and vice versa. The number of outputs is equal to the number of digit and uppercase classes (36).

During the training, if the desired output is a digit, no error is backpropagated from the output units of the letter fields, and vice versa. For example if the desired output is digit zero, there was no penalty if the letter "O" is activated. With this specific training very high recognition rates have been obtained on both digits, and uppercase letters.

There are some situations where the information about the context (i.e. digits or uppercase letter) is not available. That is the reason why in the following experiment we tried to train the network, so that no prior information about the context is assumed. In addition to that we expanded the character classes that the network handles. We included both lowercase letters and punctuation symbols. As oppose to the previous network the output field is now treated as one entity, and the network is trained so that *there is penalty* if the desired output is for example zero, and the letter "O" is activated, (i.e. corresponding error is backpropagated).

The training set increased from 12,000 digits and uppercase letters from 250 writers, to 30,000, digits, upper and lowercase letters and special symbols (punctuation) from 250 writers. The database was collected among the staff in AT&T. The number of outputs of the network increased as well (since corresponding number of classes increased from 36 to 92). On order to match the capacity of the classifier to this somewhat changed task, we had to enlarge the network size. Total number of connections in the network is 94,272. The number of independent weights is 27,561.

To handle larger this increase the overall size of the network increased as well.

After the training, on this training set, we retrained the network on another chunk of the training data. The size of this additional training set is 9,500 characters from 10 writers. Data has been provided by GO corporation.

In the training we excluded (didn't train on) the following punctuation characters: "-", "=", "/", "." . We didn't test on this set of characters as well. We tested the results on the training set that of 3318 digit, uppercase and punctuation characters from 7 writers. The data was provided by the character recognition group at IBM Watson Research Center. The performance on the test set is tested on the TESTBED designed by I. Guyon. The results (output from the TESTBED) are given in the figure 3.4. Two tests were performed. In the first test, some confusions among characters were allowed (i.e. zero can be recognized as "O"). The list of legal confusions is given in the figure 3.4. The overall performance was 87.12% on the test set. The second test was performed with no confusions allowed. The overall performance on the test set was 79.03%. For the comparison, results on the same test set (no confusions allowed) with a commercial recognizer was 65.79%.

We measured raw performance (the simplest classification scheme consisting of picking the maximum output as the resulting class).

```
================================================================================
            *** NADANET ***   P E N   R E C O G N I Z E R
================================================================================

Designer name: Nada Matic
Creation date: 1992
Preprocessing: Fixed-length-convert, 7 features
Number of frames in input: 90
Network name: ALL-GO.fp
Net trained with actual targets, style:  all
Test performed by: isabelle
Database: IBM corporation database: BL+FORMAT/ibm-iso.db
Map : The whole database (3318 patterns)
Symbols excluded: ("-" "=" "/" ".")
Total time spent: 366 seconds

=============> TEST RESULTS

LIST OF "LEGAL" CONFUSIONS: ('("0" "O") '("0" "o") '("1" "i") '("1" "l") '("1" "
I") '("1" "|") '("1" "!") '("1" "/") '("1" "\") '("1" "'") '("1" "`") '("1" ",")
 '("2" "z") '("2" "Z") '("3" "z") '("3" "Z") '("5" "s") '("5" "S") '("6" "G") '(
"9" "g") '("9" "q") '("c" "C") '("f" "F") '("i" "l") '("i" "I") '("i" "|") '("i"
 "!") '("i" "/") '("i" "\") '("i" "'") '("i" "`") '("i" ",") '("i" ";") '("j" ";
") '("j" "J") '("k" "K") '("l" "I") '("l" "|") '("l" "!") '("l" "/") '("l" "\")
'("l" "'") '("l" "`") '("l" ",") '("l" "L") '("m" "M") '("n" "N") '("o" "O") '("
p" "P") '("s" "S") '("t" "T") '("u" "U") '("v" "V") '("w" "W") '("x" "X") '("y"
"Y") '("z" "Z") '("I" "|") '("I" "!") '("I" "/") '("I" "\") '("I" "'") '("I" "`"
) '("I" ",") '("+" "t") '("|" "!") '("|" "/") '("|" "\") '("|" "'") '("|" "`") '
("|" ",") '("!" "/") '("!" "\") '("!" "'") '("!" "`") '("!" ",") '("/" "\") '("/
" "'") '("/" "`") '("/" ",") '("\" "'") '("\" "`") '("\" ",") '("'" "`") '("'" "
,") '("`" ",") '("." "-") '(":" ";") '(":" "!") '("(" "[") '(")" "]") '("C" "(")
)

CHARACTER CATEGORY          ANY    digit  upper  lower  symbol
Total Chars (TC)        [   3128     378   2610      0     140]
Well recognized chars   [   2725     301   2333      0      91]
 in perc TC             [  87.12   79.63  89.39    NaN   65.00]
Missed Chars (MC)       [    403      77    277      0      49]
 in perc TC         *** [  12.88   20.37  10.61    NaN   35.00]

================================================================================

Designer name: Nada Matic
Creation date: 1992
Preprocessing: Fixed-length-convert, 7 features
Number of frames in input: 90
Network name: ALL-GO.fp
Net trained with actual targets, style:  all
Test performed by: isabelle
Database: IBM corporation database: BL+FORMAT/ibm-iso.db
Map : The whole database (3318 patterns)
Symbols excluded: ("-" "=" "/" ".")
Total time spent: 348 seconds

=============> TEST RESULTS

LIST OF "LEGAL" CONFUSIONS: (empty)

CHARACTER CATEGORY          ANY    digit  upper  lower  symbol
Total Chars (TC)        [   3128     378   2610      0     140]
Well recognized chars   [   2472     277   2126      0      69]
 in perc TC             [  79.03   73.28  81.46    NaN   49.29]
Missed Chars (MC)       [    656     101    484      0      71]
 in perc TC         *** [  20.97   26.72  18.54    NaN   50.71]
```

Figure 3.4: Test results (output of the TESTBED).

3.6 Summary

In this chapter we described the Time Delay Neural Network (TDNN) designed for the on-line handwritten character recognition task [6]. The network was originally trained to recognize either digits or uppercase letters. The contextual information was required. We removed that constraint, modified the network slightly, and trained the TDNN on full character set. We tested the performance of the network on the independent test set, using the TESTBED. We tested the performance in two cases. First no confusions among characters were allowed, then the list of legal confusions was used to determine final classification performance. We compared the performance of the TDNN with a commercial recognizer performance on the same test set and showed the superiority of our technique.

4 Cleaning The Large Database for Character Recognition

In this chapter we explore the role and the quality of the training data and its influence on the performance of the writer independent TDNN recognizer described in the previous chapter. Since the amount of the training data is always limited, we developed a methodology that allows to use the available data more efficiently. That is, our methodology allows to extract the maximum information from the data while at the same time use efficiently computer time during the training. The central role in the methodology is the introduction of the concept of *super-supervised* learning, where another level of efficient supervision is imposed on the standard supervised learning scheme. While in our experiments we make use of the backpropagation, re-supervision can be applied on top of various learning schemes that involves learning from examples.

4.1 Motivation and Application

Many methods in pattern recognition involve training by examples. Typically data (patterns) are collected and labeled by a human supervisor. In most practical cases, training databases contain mislabeled patterns and may even include some meaningless patterns. Training such *garbage* patterns degrades performance

of the classifier. In particular, training procedures which emphasize atypical patterns will emphasize garbage patterns as well. Cleaning the database is essential to such algorithms.

We propose in this thesis the cleaning methodology that uses the classifier itself in the process of training, to identify suspicious patterns so that a new human *super-supervisor* can be queried. Without undue effort, the *super-supervisor* will be able to browse through the suspicious patterns, eliminate garbage patterns and keep the valid ones. Experiments are carried out on the problem of writer independent isolated handwritten character recognition, entered on a touch terminal with Time delay Neural Network (see previous chapter) which incorporates the feature extractor and classifier in a single trainable system as a *Learning Machine.*

4.2 Problem Description

During data collection, several kinds of errors can be introduced into the database:

- Hardware failures can cause the insertion of *meaningless* patterns.
- Human failures can cause the insertion of *mislabeled* patterns.

Keeping these patterns in the training database might severely degrade the performance of the classifier.

To get better reliability, several persons could check the database. This procedure provides confidence levels for the labels, but it is impractically laborious for reasonably large databases.

While some data characteristic are known to affect the learning in specific ways (i.e. size of the training set, class distribution etc.) the effects of the aforementioned problems is open research problem. There exist classical methods for

handling the outliers. *Robust statistics* [75, 76], in a broad interpretation, includes different methods that are relatively insensitive to departure from distributional assumptions of outliers (e.g. sample censoring). This robustness is often obtained by modifying a least square scheme with one that limits the influence of outliers. In our case, suspicious patterns may not be garbage patterns; if they are valid patterns that happen to be difficult to learn, they are extremely valuable (the most informative) and we would like to emphasize their role. That is the reason why the tools of robust statistic are not applicable in this case. The problem of presence of erroneous or noisy data has been studied in the field of *symbolic learning.* In a detailed study [77], Quinlan artificially corrupted data input to ID3 symbolic algorithm for learning from examples. He introduced varying degrees of corruption, sometimes to attribute values (pattern can be described as a tuple of attributes), and sometimes to labels (class membership values), to show how severely they can degrade accuracy. Despite the fact that many symbolic and neural learning algorithms address the same problem of learning from examples, similar study has not been published in neural networks learning. In our experiments with neural network as a Learning Machine, we show that similar behavior is obtained.

As we mentioned before, it is dangerous to limit the influence of all outliers because some of them may not be garbage patterns; if they are valid patterns that happen to be difficult to learn, they are extremely valuable. Such a situation arise in two different cases:

- The pattern is *atypical* but meaningful and well labeled.
- The pattern is *ambiguous*; reasonable questions can exist about the label.

In order to distinguish them from the garbage we developed a selective procedure, that keeps valuable outliers in the database, without limiting but rather

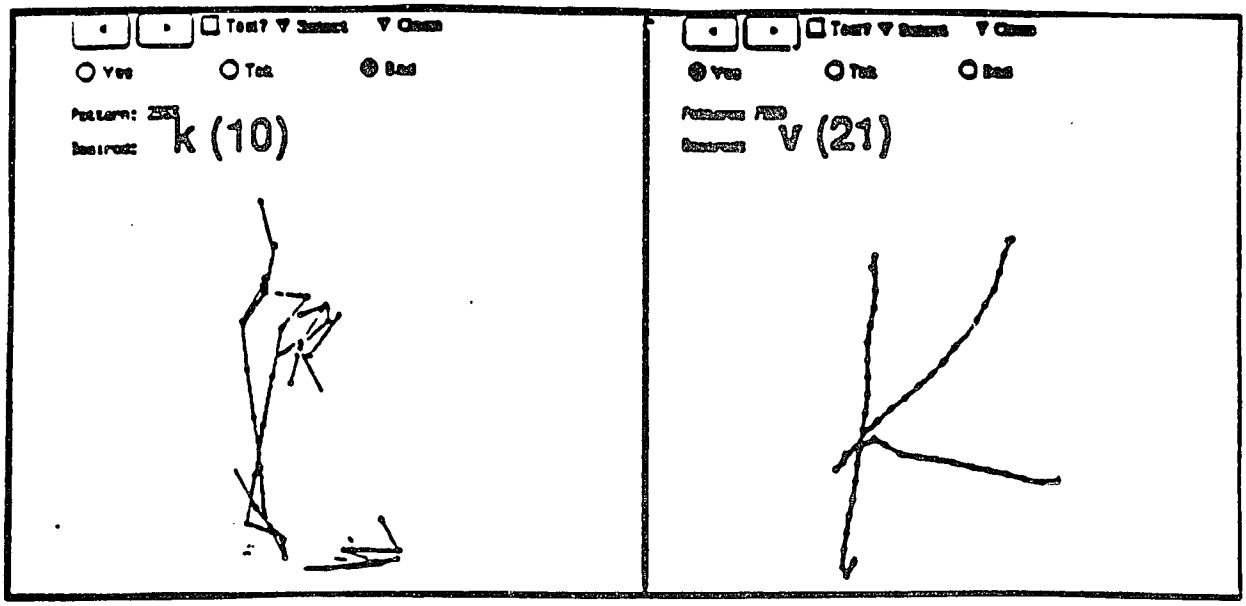

Figure 4.1: Examples of meaningless and mislabeled patterns in the lowercase letters database.

emphasizing their influence, while rejects garbage. To do a good job at cleaning the database, it is necessary to insert a human *super-supervisor* in the cleaning process. To alleviate this task, we use the assistance of the trainable classifier itself, which points out a *restricted number* of suspicious patterns.

Figure 4.1 shows examples of meaningless and mislabeled patterns that are present in our database of handwritten lowercase letters. These patterns are considered garbage and removed from the database. Figure 4.2 shows examples of valuable outliers, from the same database, i.e. atypical and ambiguous. We will keep them in the database, in order to capture the tail of the distribution and to improve the recognition rate.

It is interesting that in some tasks in handwriting special training on garbage patterns helps the recognition. For example in reference [75] the task is to recognize automatically segmented zip-codes. In this case "garbage" is produced by imperfect segmentation process, where some digits are cut in several segments, while others are joined together. "Garbage" here does not represent any "poorly"

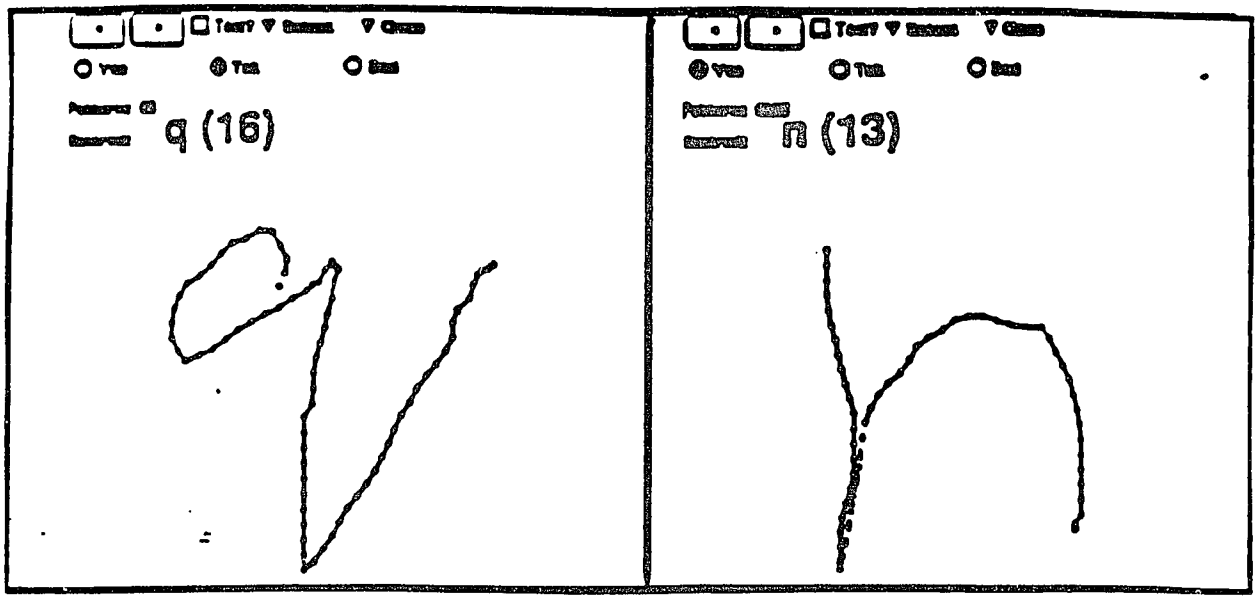

Figure 4.2: Examples of atypical and ambiguous patterns in the lowercase letters database.

written digit, consequently it has no label (as oppose to the case we discuss where "garbage" patterns are random labeled). Here, the segmentation "garbage" is extracted from the training database by human supervisor. It is then used for the negative training of the neural network (target values for all available classes are set to -1). Such training procedure helps the neural network to reject the garbage typically produced by an automatic segmenter for string of digits, while maintaining its performance level at classifying digits. Unfortunately, this approach is good only for the "specific" source of unlabeled garbage, and can not be used in general case.

Another idea, would be to introduce extra class for the garbage patterns, and then to train the network with the garbage. Again, that would be possible only in the case where we have consistent source of "garbage" patterns which is not realistic and general enough case.

4.3 Emphasizing Ambiguous and Atypical Patterns

Before describing the cleaning (query) system, we outline the learning-with-emphasis procedure, which is intended to capture the tail of the distribution of the training patterns.

An ideal classifier will: (i) perform very well on all *typical* patterns, and (ii) perform reasonably well on *atypical* and *ambiguous* patterns. Most algorithms lead to classifiers satisfying (i). In reference [10], it was shown that by emphasizing difficult patterns, it is possible to get (ii) with no degradation of (i).

Emphasis can be applied to all adaptive training schemes which cycle several times through all the patterns before converging, such as gradient descent techniques. Typically, such techniques minimize an error function such as the mean-squared-error (MSE) cost function (see equation 2.17). We focus on "on-line" learning procedures; that is, a modification of the classifier parameters happens after each presentation of a pattern. On-line gradient descent is affected by the order and frequency of presentation of the patterns. The least predictable example (i.e. with largest squared error) is the one carrying most *new* information [79, 80]. In character recognition, for instance, one should inter-mix examples of different classes and from different writers, as opposed to presenting batches of similar patterns. Proper shuffling of the examples greatly improves the speed of convergence.

Performance can be further improved by increasing artificially the frequency of presentation of the patterns that are least predictable (most informative). At the extreme, it is conceivable to train only on the least predictable pattern (in the squared error sense) every time one cycles through the training set. This time-consuming scheme converges asymptotically to a solution that minimizes

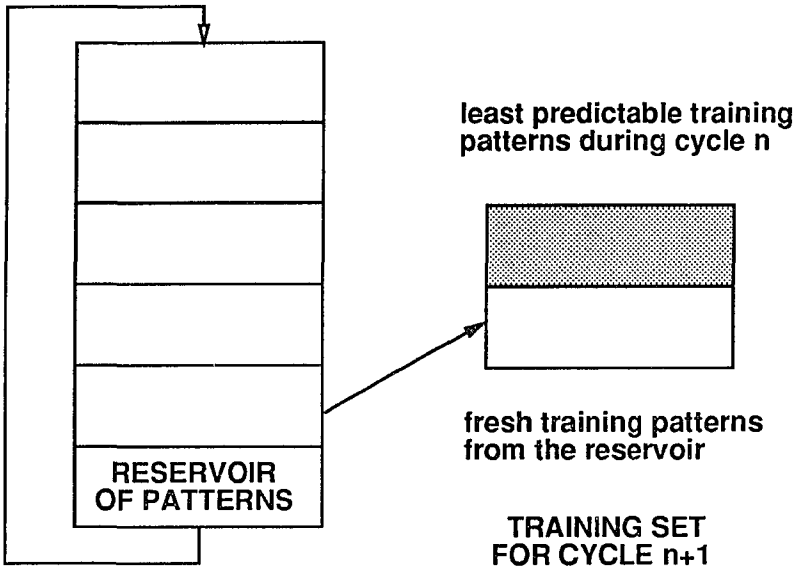

Figure 4.3: Emphasizing scheme. The least predictable examples are presented more often.

(over all training patterns) the maximum squared error (as opposed to the mean square error; see [81]).

A more practical implementation of this idea was proposed in [10] (figure 4.3). Training patterns are put into a reservoir. Only a small subset of training patterns is used at each training cycle. At the end of a cycle, patterns in the subset are sorted in order of increasing predictability. The least-predictable half is kept (e.g. the patterns with the largest squared error) and the second half is replaced by fresh patterns taken from the reservoir.

Garbage patterns will fit the description of the least predictable patterns. On the other hand, they carry no useful information. So, the danger of this scheme is that if the database contains *mislabeled* or *meaningless* patterns, emphasis will be put on them as well as on the highly important *atypical* and *ambiguous* patterns. The

significant improvements obtained with the emphasizing scheme [10] compared with "classical" procedures, motivated the design of an efficient cleaning method.

4.4 Proposed Solution - Super-supervised Learning

Super-supervised learning refers to filtering the suspicious patterns (meaningless, mislabeled, ambiguous, or atypical) using a trainable classifier. The super-supervised learning starts by training the classifier using the data labeled by supervisor I in the usual way (e.g. standard supervised learning). In addition to the basic classification task, the classifier divides the patterns, according to some predefined criteria, into two categories: the predictable versus unpredictable. The latter are declared suspicious and sent to supervisor II who decides what to do with the pattern.

A block diagram of the procedure is presented in figure 4.4. The classifier has already received some preliminary training. Supervisor I provided the labels that appear in the training database before the cleaning operation is started. When a subsequent pattern is presented, an error measure (such as the one computed from equation 2.16) is obtained by comparing the output of the classifier with the label provided by supervisor I. If this error exceeds a certain threshold, supervisor II will be queried before the pattern is incorporated into the training set. The point is to get maximum information per unit effort from supervisor II; only a few patterns are presented to supervisor II.

We propose two complementary implementations of this scheme for the cleaning the database application: interactive cleaning and batch cleaning.

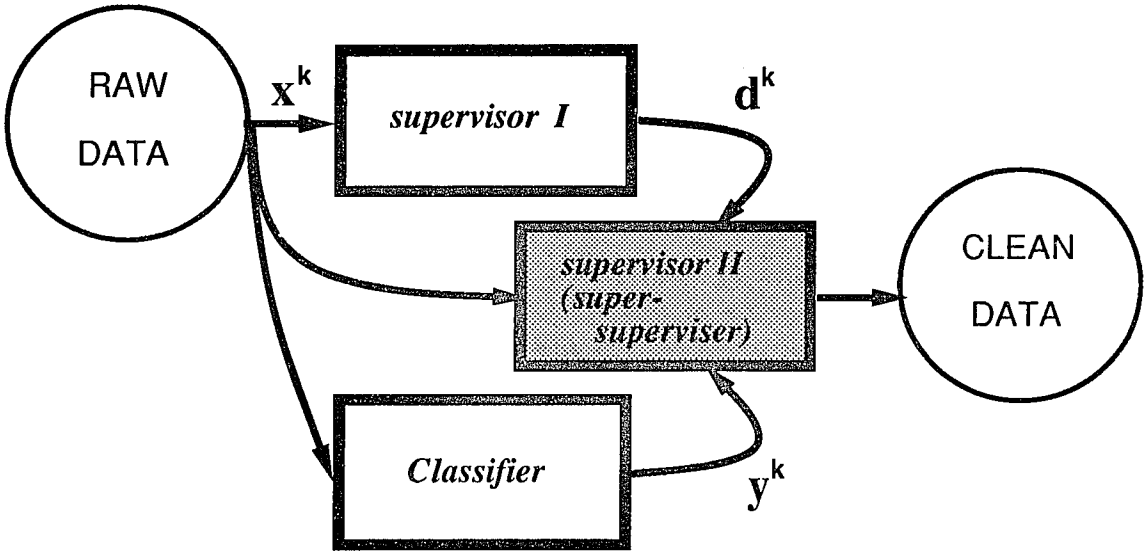

Figure 4.4: Block diagram of the cleaning scheme: super-supervision.

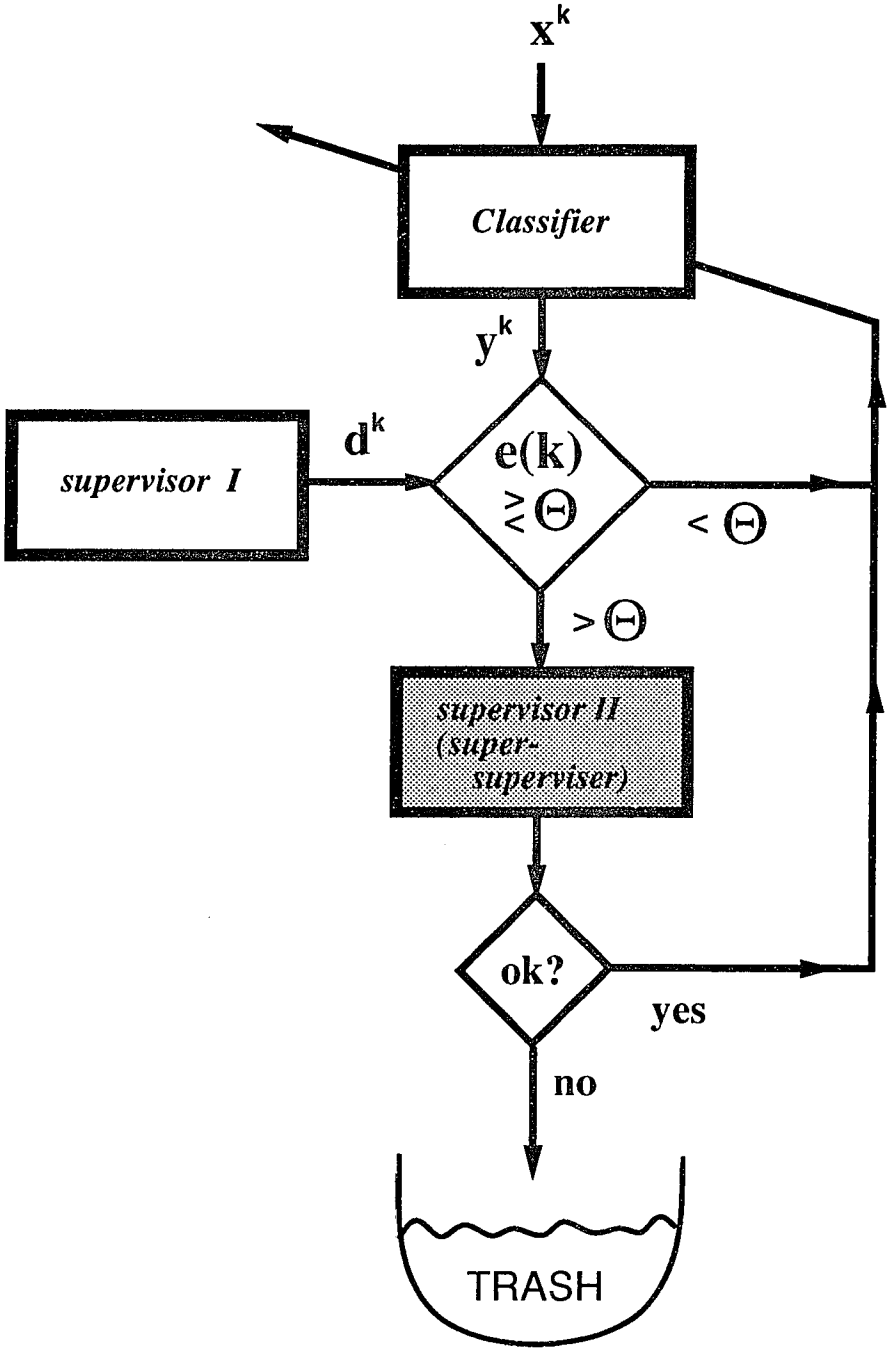

Figure 4.5: Flow diagram of the interactive version of the cleaning scheme.

Interactive Cleaning "Interactive super-supervision" has the advantage of combining cleaning and training in a single session. A flow diagram of this method is presented in figure 4.5. Initialization is performed by training the classifier with a few clean examples.

After initialization, at step k of the cleaning process, a new pattern $\mathbf{x}^k$ is presented to the classifier. The prediction of the classifier is compared to the label provided by supervisor I and the error $e(k)$ is computed. If the error is below a given threshold θ, the pattern is immediately incorporated into the training set. Otherwise, the pattern is sent to supervisor II for checking. Depending on the decision of supervisor II, the pattern is either incorporated into the training set or discarded.

When the session ends, both training and cleaning are completed. However, knowing an appropriate value for the threshold θ is required.

Batch Cleaning The batch cleaning scheme, in contrast, does not require knowing θ. Training is performed first on all the patterns. An index of predictability of the patterns is then computed, perhaps based on the remaining error $e(k)$ after training. Batch cleaning can be easily combined with the emphasizing scheme. In that case, a good index of predictability is the time each pattern spent among the patterns that are hardest to learn (shaded area in figure 4.3). Patterns are sorted in order of increasing predictability. Supervisor II checks the patterns starting with the least predictable ones. Cleaning can be stopped when undesirable patterns become very rare.

After batch cleaning, a new classifier has to be trained with clean data. In principle, this new classifier should be checked to make sure no new outliers have turned up.

Bootstrap Cleaning The third version of the cleaning scheme combines the best features of the interactive and batch versions; we call this the bootstrap method. Cleaning and training are performed using a small part of the available data. The resulting network constitutes a partial solution that is highly selective for identifying outliers in a larger subset of the training data. The process can be iterated as necessary, with ever larger amounts of training data, until the whole reservoir of patterns is cleaned. To make the task of supervisor II comfortable, we have developed a user-interface for bootstrap cleaning, where different cleaning parameters, such as size of the pattern chunks, the threshold, or the number of training iterations can be easily defined.

4.5 Experimental Results

In our application, we consider a problem of writer independent recognition of lowercase isolated letters, entered on a *Touch Terminal* that provides pen trajectory information. A data set of 11513 letters from approximately 250 writers was collected among the staff at AT&T. The contributors were instructed to write a certain set of letters, so the task of supervisor I (preliminary labeling of the data) was trivial — which explains our concern with minimizing the burden on supervisor II. The data set was partitioned into a training set of 9513 characters and a test set of 2000 characters from disjoint set of writers. For batch cleaning, the training session was stopped after approximately 250,000 example presentations.

Our classifier is a Time Delay Neural Network (TDNN) described in the previous chapter [6]. The number of outputs correspond to the number of lowercase characters (26).

In figure 4.6 we present the results given by the bootstrap cleaning procedure. The cleaning was done while training the TDNN with plain back-propagation (no

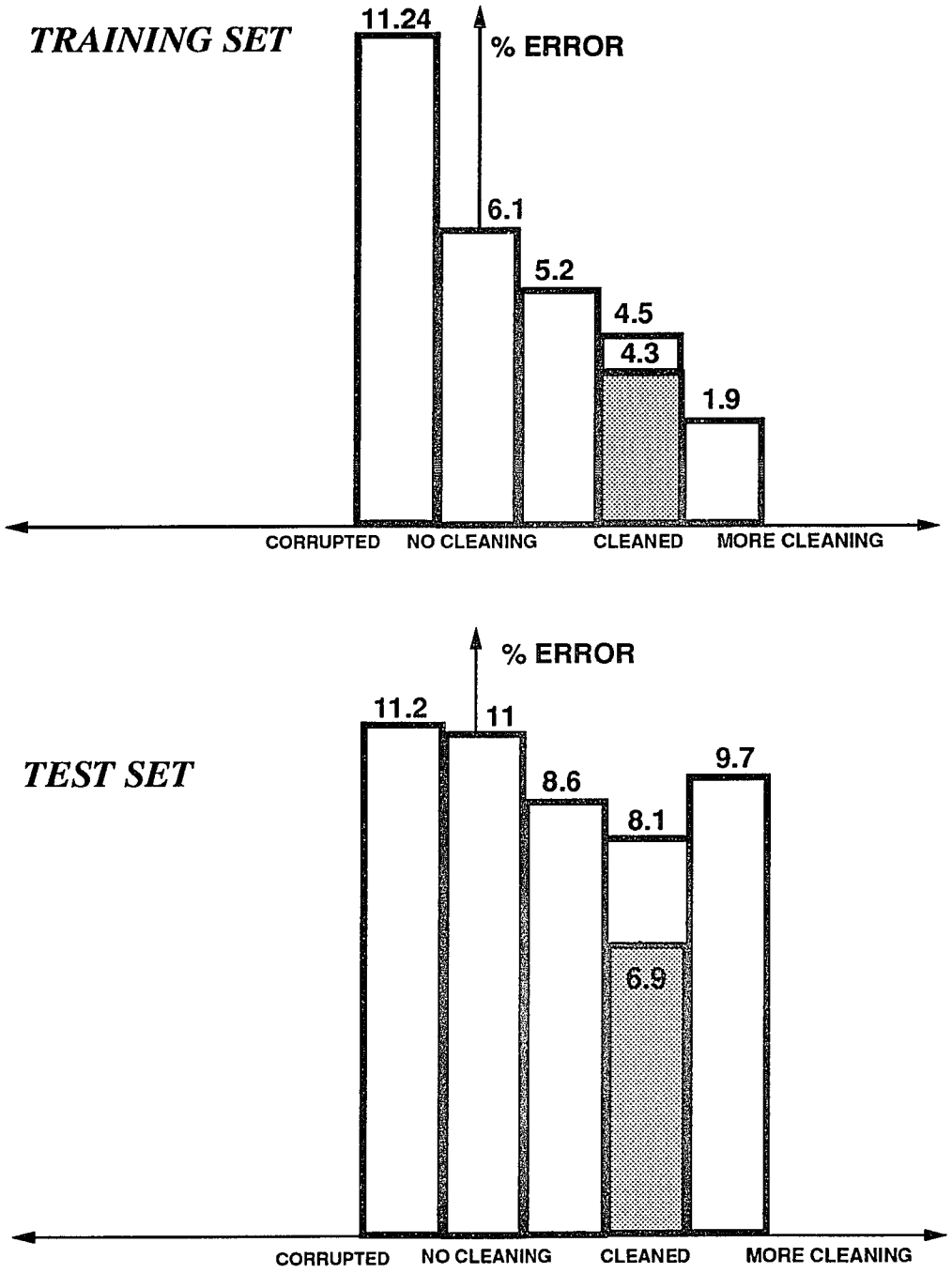

Figure 4.6: Recognition results for handwritten lowercase letters, for various levels of cleaning. The shaded area indicates the results obtained with the emphasizing scheme.

emphasizing scheme). The output unit with highest output value is considered to be the decision of the network.

Both training set and test set were cleaned, but, a distinction was made between these two sets. While all the *garbage* patterns were removed from the training set, we removed from the test set only the patterns that supervisor II deemed totally meaningless — typically resulting from a hardware failure during data collection. Poorly written, highly ambiguous, or mislabeled patterns were kept in the test set, to make our test results as conservative as reasonably possible. The supervisor II was free to decide whether or not questionable characters should be removed from the training set.

Starting with the "uncleaned" database, 3 stages of bootstrap cleaning were applied by the super-supervisor. Each stage was more strict, i.e. he lowered the tolerance for marginal-quality characters. To test the power of the cleaning technique, we also tried to corrupt the initial, uncleaned database, by artificially mislabeling 5% of the characters in the training set.

The results of figure 4.6 show the evolution of the error rate both on the training set and the test set. While the error rate keeps decreasing on the training set, the error rate on the test set goes through a minimum for a certain level of cleaning. After that point, any additional cleaning will result in an increase of the test error rate. This indicates that the super-supervisor has removed all meaningless and mislabeled patterns and starts removing useful atypical or ambiguous patterns. It is important to be able to detect this phenomenon, and avoid overcleaning. This minimum can be detected with a validation-set (independent from the test-set so as not to bias the final performance result). Another way is to use the predictions of learning theory [44]. In that case, no validation set is needed. This technique is further developed in section 4.6.

The emphasizing scheme was run on a freshly initialized TDNN. Experiments

were performed with no cleaning and with the optimal cleaning. The resulting improvement is shown by the shaded bar in figure 4.6. We see that the cleaning plus emphasis reduces the raw error rate almost by half. With unclean data, emphasis is even a little bit worse than the plain vanilla training algorithm. The full advantage of cleaning is therefore reached with the emphasizing scheme. Similarly, emphasizing scheme is not bringing any improvement if we do not clean the training database first.

We also tried batch cleaning. The index of predictability was the time each pattern spent in the "hard to learn" category during training (shaded area in figure 4.3). The results are comparable to those obtained with the interactive cleaning. The super-supervisor however, does not need to sit in front of the machine during training, nor set any *a priori* threshold.

4.6 Theoretical Foundations of Super-supervised Learning

We now make connections with Bayesian classification and the Vapnik-Chervonenkis (VC) theory [44]. We justify our method for cleaning the data based on an index of predictability. We describe a method to avoid overcleaning and check its validity experimentally.

4.6.1 Probabilistic Formalism

Our training problem can be formalized as follows. In an environment characterized by a probability density function $p(\mathbf{x})$, examples $\mathbf{x}^k$ are drawn randomly and independently. Supervisor I provides class labels[1] d_l with probability $P(d_l|\mathbf{x})$.

[1] l is the class index. Class labels are $d_l = 1$ if $l = \text{class}(\mathbf{x})$ and 0 otherwise.

In this framework, *atypical* patterns are drawn from regions of input space with small $p(\mathbf{x})$; *ambiguous* patterns correspond to regions of input space with small $P(d_l|\mathbf{x})$ (for instance $P(d_1|\mathbf{x}) \simeq P(d_2|\mathbf{x}) \simeq 0.5$).

In the following, $p(\mathbf{x})$ will be referred to as the "natural" probability distribution and $P(d_l|\mathbf{x})$ as the "natural" posterior probability.

Let us introduce now sources of distortion of these "natural" probabilities so as to account for hardware and human failures in the data collecting process. Hardware failures (*meaningless* patterns) cause distortions of $p(\mathbf{x})$, leading to the "distorted" distribution $p'(\mathbf{x})$. Human failures (*mislabeled* patterns) cause distortions of $P(d_l|\mathbf{x})$, leading to the "distorted" posterior probability $P'(d_l|\mathbf{x})$.

Meaningless patterns are drawn from regions of input space with possibly large $p'(\mathbf{x})$, but small $p(\mathbf{x})$. *Mislabeled* patterns correspond to regions of input space with possibly large $P'(d_l|\mathbf{x})$, but small $P(d_l|\mathbf{x})$.

As summarized in figure 4.7, four kinds of suspicious patterns have been inventoried, corresponding to either small $p(\mathbf{x})$ (atypical or meaningless, i.e. outliers) or to small $P(d_l|\mathbf{x})$ (ambiguous or mislabeled). But all the probabilities $p(\mathbf{x})$, $p'(\mathbf{x})$, $P(d_l|\mathbf{x})$ and $P'(d_l|\mathbf{x})$ are unknown and we have to rely only on the empirical data contained in the database itself to perform the cleaning.

4.6.2 Justification of the Cleaning Method

In our approach, we train a classification function $y_l(\mathbf{x})$ to approximate $P(d_l|\mathbf{x})$. We detect a suspicious pattern $\mathbf{x}^k$ based on an index of predictability involving for instance the error $e(k)$ (see equation 2.16) which reflects the discrepancy of the opinion of the classifier, $y_l(\mathbf{x}^k)$, and that of supervisor I, d_l^k.

From the considerations of the previous section, it is clear that such a method will eventually detect ambiguous and mislabeled patterns $\mathbf{x}^k$ which correspond

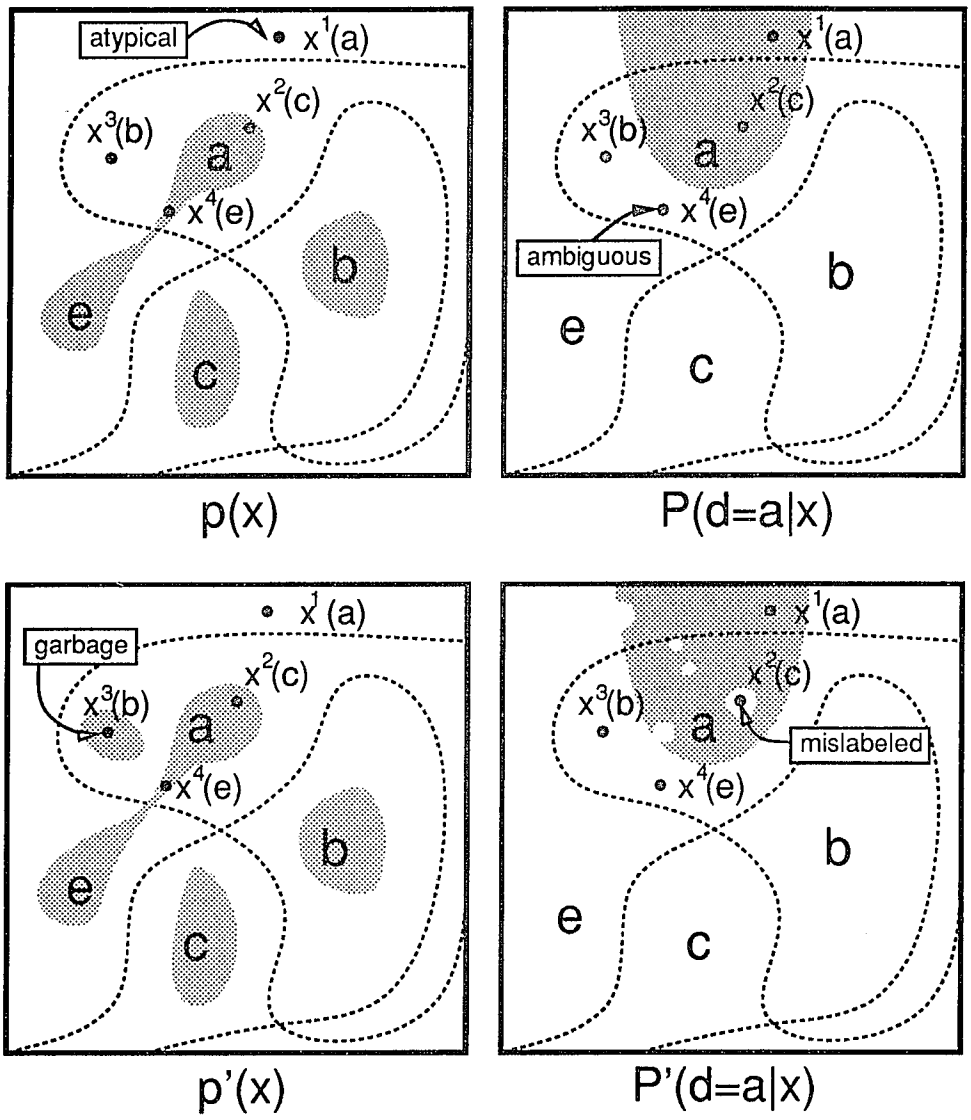

Figure 4.7: This small two-dimensional classification example illustrates the four kinds of suspicious patterns. Shaded areas represent regions of high probability density. The four subgraphs display various conditional and/or empirical probabilities. In the region of low $p(\mathbf{x})$, we find an *atypical* pattern $\mathbf{x}^1$ correctly labeled "a", and a *meaningless* pattern $\mathbf{x}^3$ (garbage) labeled "b". In the region of low $P(d = a|\mathbf{x})$, we find an *ambiguous* pattern $\mathbf{x}^4$ labeled "e" and a *mislabeled* pattern $\mathbf{x}^2$ labeled "c". Patterns $\mathbf{x}^2$ and $\mathbf{x}^3$ have been drown from the "distorted" distributions $P'(d = a|\mathbf{x})$ and $p'(\mathbf{x})$ respectively (see text). With the help of the decision function provided by the trained classifier (dashed line), suspicious patterns can be detected for further examination by a human "super-supervisor".

to small $P(d_l^k|\mathbf{x}^k)$. Outlying patterns such as atypical and meaningless patterns, which correspond to small $p(\mathbf{x}^k)$, can also be detected through the same mechanism. t is very difficult to obtain reliable predictions of $P(d_l|\mathbf{x})$ in regions of small $p(\mathbf{x})$, because of the lack of training data. Therefore, for very small values of $p(\mathbf{x})$, the prediction will be erratic. A totally random prediction would predict the label y^k provided by supervisor I with probability $1/c$ (c being the number of classes). Hence, the probability of not detecting an outlying pattern with our method will only be of the order of $1/c$. The method can further be refined by training N sets of c classifiers providing independent predictions, which reduces the probability of not filtering outliers to approximately $1/c^N$. Similar ideas appear in [80].

4.6.3 VC-predictions of Optimal Cleaning

Our experimental results indicate that there is a point of optimal cleaning yielding best generalization (see figure 4.6). In this section, we check experimentally the predictions of optimal cleaning obtained using the VC-bounds [44] (see section 2.4.2).

According to the VC-theory, in the case when m patterns has been removed from the original training set p, the formula 2.24 for the *guaranteed risk* can be written in the following form:

$$G = \nu(p-m) + \beta\frac{h(ln\frac{2(p-m)}{h}+1) + m(ln\frac{p}{m}+1)}{p-m} \tag{4.1}$$

where β is a constant which was experimentally determined by an independent study [82] ($\beta = 0.5$), $\nu(p-m)$ is the frequency of training errors when m suspicious patterns have been removed from the training set, p is the number of training patterns before cleaning and h is the VC-dimension.

As discussed in section 2.4.2 this bound is valid for small values of the frequency of training errors $\nu(p-m)$ and assumes that patterns are removed only from the training set. G predicts an upper bound on the frequency of errors on the original uncleaned test set.

G is the sum of two terms: the frequency of training errors $\nu(p-m)$, which decreases as the number of removed examples m increases, and a second term:

$$\epsilon = \beta \frac{h(ln\frac{2(p-m)}{h}+1) + m(ln\frac{p}{m}+1)}{p-m} \tag{4.2}$$

which characterizes the confidence with which we know the probability of classification error from $\nu(p-m)$. This last term increases when we remove training examples. The minimum of G indicates the point of optimal cleaning guaranteed by the VC-theory. If the frequency of the training error $\nu(p-m)$ is not small, then the second term in the equation 4.1 is replaced with the term $\epsilon/2(1+\sqrt{1+4\nu(p-m)/\epsilon})$.

The VC-dimension is difficult to estimate for large neural networks trained with learning schemes involving uncontrolled regularization. In our case, regularization happens in several different ways, including early stopping of the training session (see section 2.4.3). If training consisted of merely minimizing the number of errors on the training set, a good approximation of the VC-dimension would be the number of free parameters N_{free}. In the case of a TDNN, this number is the number of independent weights, not the number of connections. Because of our implicit regularization, the *effective* VC-dimension is only a fraction of N_{free}. Based on experiments performed in [15], we use the following heuristic formula:

$$h = \frac{N_{free}}{3} \ . \tag{4.3}$$

We performed the following experiments to check the validity of this result. Training of the TDNN was performed with all the training data (including

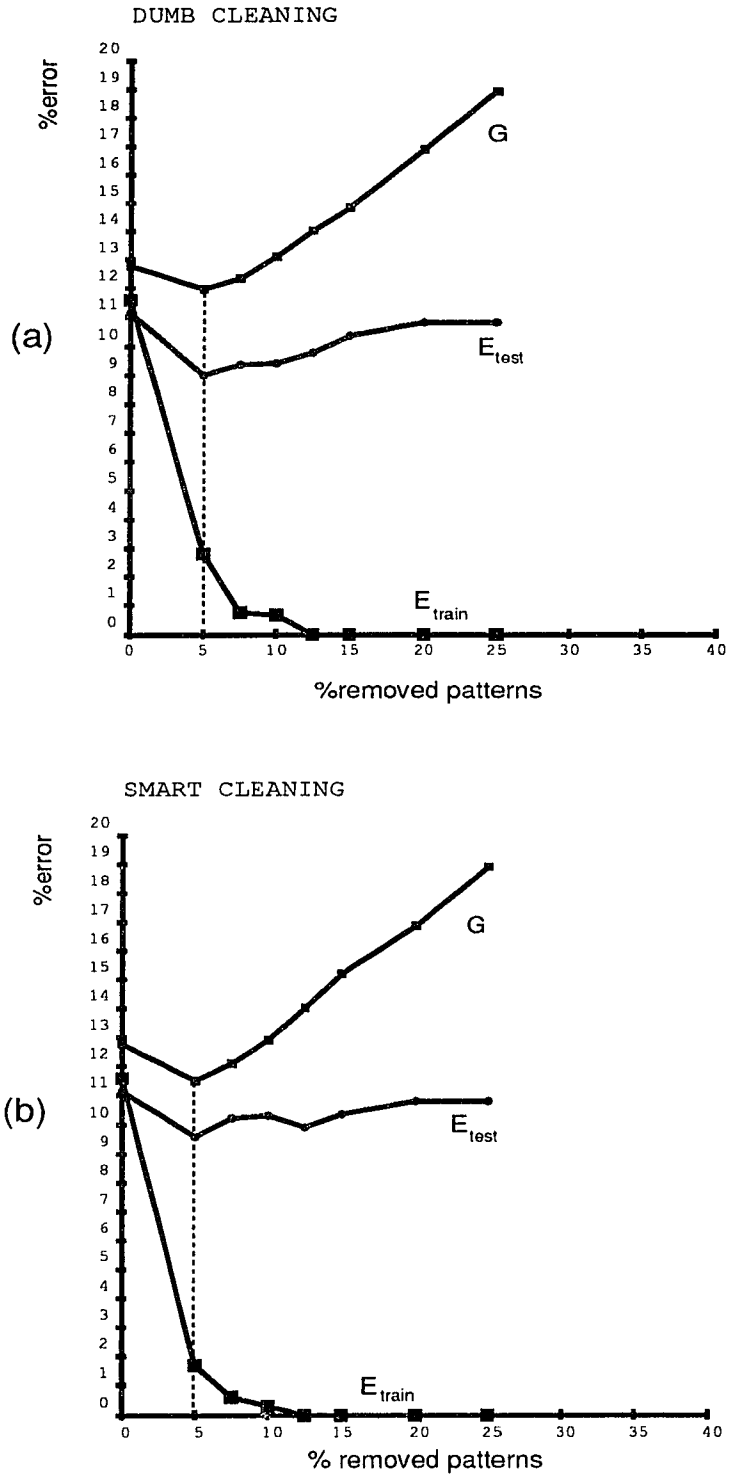

Figure 4.8: Percent error vs. percent of the patterns removed from the training set: (a) dumb cleaning; (b) smart cleaning. Constant $\beta = 0.5$. The guaranteed risk has been rescaled to fit the figure.

garbage patterns) using the emphasizing scheme. Patterns were then ranked according to decreasing index of predictability. A set of suspicious patterns selected from the least predictable top ranked patterns was removed from the training set. A freshly initialized TDNN was trained with the remaining patterns. Then, suspicious patterns were progressively incorporated in the training set, starting from the most predictable ones (least suspicious). A each step, partial retraining of the previous TDNN was performed with the clean training set augmented with some suspicious patterns.

Two different definitions of the set of suspicious patterns were considered: (i) the top 25% of the least predictable patterns (dumb cleaning); (ii) the top 25% of least predictable patterns that had been identified as suspicious by supervisor II in our previous study (smart cleaning). In figure 4.8 we show the results obtained in both cases.

The minimum of G coincides with the best generalization performance measured with the test error E_{test}. We checked the robustness of the prediction by varying the value of the VC-dimension h over a wide range:

$$300 < h < 5000 \; . \tag{4.4}$$

We did not measure any significant change in the position of the optimum. This result turns the VC-prediction into a very useful tool to prevent overcleaning, even in the absence of an accurate estimate of the *effective* VC-dimension.

We need to make two additional comments on the results of figure 4.8. The best test error is larger than the one presented in section 4.5. This is due to the fact that the test set contains here all the garbage patterns, including hardware failures, since VC-predictions hold only when cleaning is performed exclusively on the training set. Another observation is that in this experiment similar performance results were obtained with both smart cleaning and dumb cleaning.

By inspecting the top ranking patterns of the dumb cleaning, one can see that in this case large majority of the patterns were meaningless and mislabeled patterns. We believe that the difference in performance between smart and dumb cleaning is data dependent, and that on other databases the top ranking patterns in dumb cleaning can contain more ambiguous and atypical patterns.

Super-supervised learning (used in smart cleaning) provides possibility of taking different actions on "suspicious patterns", not just removal (as in the cleaning application). That has very important consequence. The example of such situation would be the case where certain class of characters has very special meaning in particular application, and the cost of inadvertent recognition of such character is very high (e.g. character which means erase file). It is imperative to train the recognizer so that no conceivable garbage pattern would lead to the inadvertent recognition of such character. Smart cleaning can be modified to handle such applications, as oppose to dumb cleaning that does not allow that.

4.7 Summary

By filtering suspicious patterns and querying a super-supervisor, we have demonstrated clear improvement in our handwritten lowercase letter training database. For the writer independent task, the final recognition rate on the test set was 93.1% (raw performance; no rejection allowed), and 95% when 3.2% of the patterns were rejected as unclassifiable, and 97% with 6.5% rejection. These results are better than the ones reported in a recent review paper [4] on a comparable task. Such high recognition rates could not have been achieved without the learning scheme which emphasizes ambiguous and atypical patterns, or without accurate cleaning of the training database. In other words, special treatment was applied to the most informative patterns. Our cleaning techniques are supported by theoretical derivations. Overcleaning can be prevented by computing

the point of optimal cleaning predicted by the Vapnik-Chervonenkis theory. The super-supervised learning does not have to be used exclusively for the cleaning purposes. It is a general method that allows different actions for different kinds of "suspicious" patterns.

5 Writer Adaptation

In the previous chapter, we discussed the role of informative patterns in the training process and how it affects generalization . We have developed a "super-supervised" technique that is able to identify the most informative patterns and distinguish them from undesirable outliers. We showed that the emphasizing training scheme works best when the data is cleaned by a super-supervisor. We applied that methodology to the *writer independent* recognition task. In this chapter, we address the second half of the problem. How to deal with informative patterns that the system has not been trained on so far and that user can provide on-line (i.e. examples of the rare writing styles). In other words, how to modify the writer independent recognition system in order to allow writer adaptation.

Even with improved writer independent recognition system, we can never have enough data, especially rare ones, to train the recognizer with, and expect the recognizer to perform excellent on all potential writers. On the other hand, with emergence of "pen-based' computers and their first real-world applications, the recognition accuracy requirements for on-line handwriting recognition have increased. Some users are ready to limit themselves to form filling applications and write only either digits or uppercase letters in boxes, in order to achieve recognition accuracy close to 99%. At the same time a lot of potential applica-

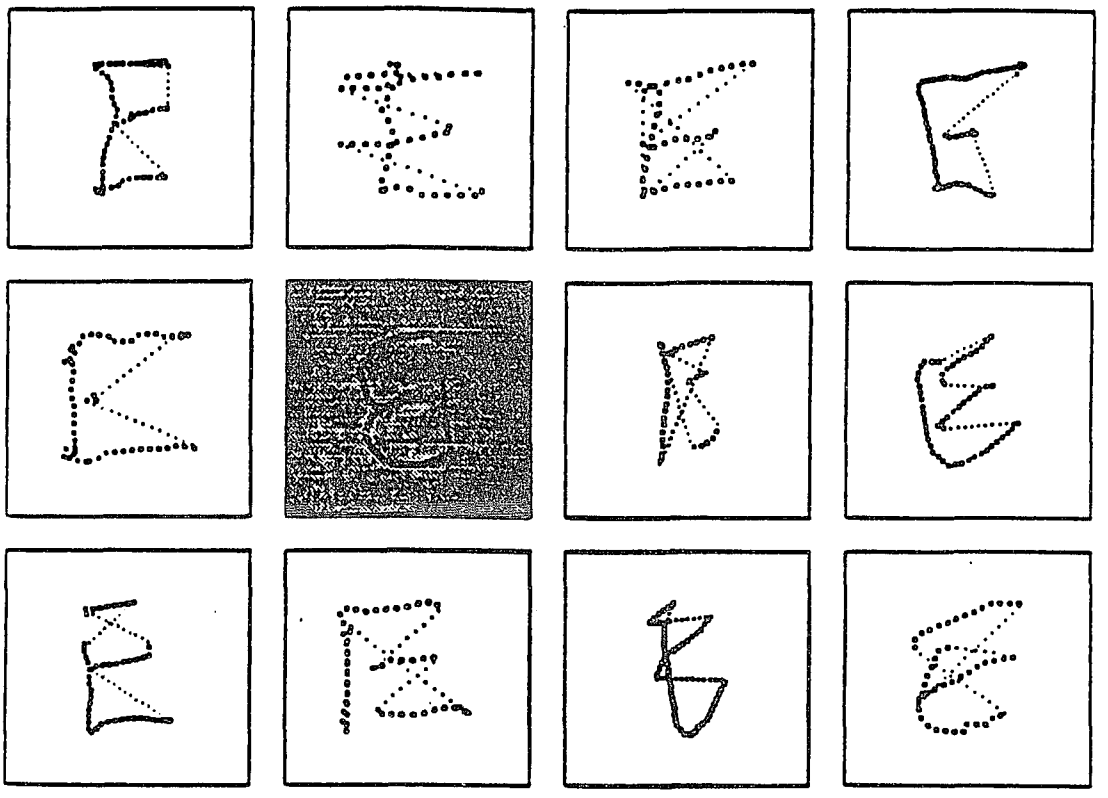

Figure 5.1: Examples of atypical writing styles. Various examples of the letters E are shown. Pen up segments are linearly interpolated and represented with small dots. An atypical E is shaded.

tions of on-line recognition systems are oriented to a specific user. Having that in mind, one can avoid the requirement that the system has to perform excellent for *any* writer, (i.e. in the writer-independent environment), and try to improve the performance by fine tuning of the system for a *particular* writer.

One of the main reasons why it is so hard to obtain the required accuracy for writer-independent systems is the large variety of writing styles (see figure 5.1). This is especially true for on-line handwritten characters. Here, the dynamics of the handwriting bring another level of variability to the already existing explosion of different handwriting styles due to geographical, social, and professional origin and to differences between left-handed and right-handed writers [9] (see chapter 1 for more details).

Two approaches can be taken in order to obtain very high recognition accuracy. First, the recognition system can be trained to be completely *writer-dependent* [83, 84], with a loss of generality as a price to be paid. Second, the system can be designed to be *writer adaptive*. In writer-adaptive systems the writer starts using the writer independent system and provides more examples of his own handwriting while using it, in order to fine tune the recognizer to his own style.

We have identified a few desired features for a good writer adaptive system:

- Fast adaptation phase, where the user needs to give only a few additional examples of his own handwriting.
- Possibility to easily add new classes.
- No recognition speed degradation.
- Modest additional memory requirements, especially since several versions of customized recognizers might be needed on the same system.

We designed our *supervised* writer-adaptive system with these requirements in mind. Our goal was to preserve as much as possible of the writer independent recognizer. This was possible because the writer-independent TDNN recognizer (see Figure 3.3) can be viewed as a combination of two separable modules: *Feature extractor* and *classifier* The neural feature extractor consists of successive layers up to the last layer of the network. It can be used independently to provide a compact representation of the input data. The classifier is the last fully connected layer of the TDNN (see figure 3.3).

In reference [85] it was suggested that for some type of problems it is more appropriate to use more powerful classifiers instead of the last layer of weights a neural network. The first module (i.e. feature extractor) is performing a clustering task of input data into its last layer. The data are now input to the

second module, classifier, whose choice depends on the problem at hand. The input pattern is propagated through the truncated network (without its last layer) and stored in a high-level representation.

In the writer adaptive system [10], the TDNN feature extractor is combined with k-nearest neighbor classifier. This adaptation scheme fulfills the first two of the above requirements, while the others are only partially fulfilled.

Our implementation of the supervised writer adaptive recognition system, combines TDNN feature extractor and the Optimal Hyperplane (OH) classifier [44] (see section 2.2.2), which can be easily retrained for adaptation purposes. The training set for the adaptation consists of examples of rare writing styles and only a *small subset* of the original training set. The Optimal Hyperplane is a linear classifier, therefore, after the recognition system is customized, there is no recognition speed degradation compared to the writer-independent case. This approach satisfies all the above-mentioned requirements, and provides fast and accurate adaptation.

In this chapter, we first describe the existing writer adaptive system that employs k-nearest neighbor classifier. We then present the on-line version of the Optimal Hyperplane classifier training algorithm. We employ Optimal Hyperplane training procedure to generate a number of OH classifiers for the multi-class problem. In section 5.4 we compare the performance of the writer-independent TDNN recognizer with the recognizer that consists of TDNN feature extractor and OH Classifier. The comparison is performed before the adaptation takes place. We propose four different adaptation schemes. We describe the adaptation experiments and compare the performance results for all four proposed schemes. Finally, using the best adaptation scheme, we perform experiments and present the adaptation results for 7 different writers. Our test set consists of both predefined classes characters (i.e. digits and uppercase letters) and new symbols. We conclude this chapter with an analysis of our adaptation method

and give further directions.

5.1 Existing Solution

The writer adaptation system [10], is based on the already existing TDNN described in section 3.3, trained to recognize characters from a large variety of writers. In the following, this network will be referred as the WIN (Writer Independent Network). As already described, *truncated* WIN TDNN (without its last layer), can be considered as a feature extractor. It will be referred as a WIN-core. For the adaptation purposes TDNN feature extractor is combined with a k-nearest neighbor classifier.

This classification scheme compares the unknown pattern with all known reference patterns on the basis of some criterion for the degree of similarity (see section 1.4), between two patterns $\mathbf{x}$ and $\mathbf{x}^k$. In this particular case, the direction cosines were used as a measure of the non-metric similarity:

$$cos\theta = \frac{(\mathbf{x}, \mathbf{x}^k)}{\|\mathbf{x}\|\|\mathbf{x}^k\|} \tag{5.1}$$

The last layer of the writer-independent TDNN is reused to avoid unnecessary training efforts: the weight vectors of these neurons are kept as *prototype* vectors for the corresponding classes. If a character is misclassified by the system, the user may decide to introduce it as a new example. This new example can be either member of the pre-existing classes or a new symbol. The new example is stored in WIN-core representation. It is labeled by the user and joins the set of examples of the corresponding class already collected, including prototypes for the pre-existing classes. The best adaptation results are obtained with 3-nearest neighbors.

Although the adaptation works well (often single example of the new character is enough to retrain the particular class), this implementation has some drawbacks:

Memory requirements are increasing, since too many characters get stored. As a consequence, recognition time increases. The computational time for the k-nearest neighbor amounts to computation of the similarity of the example vector and all the sample vectors. This is in contrast to the last two features identified as desirable for the good adaptation system.

5.2 Proposed Writer Adaptive System

5.2.1 What is the "Optimal" Separation Among Classes?

In section 1.4, we have briefly described different approaches used in pattern recognition, in order to "optimally" solve the problem of separation among different classes. But it is important to notice, that the separation of data points can be optimal (or sub-optimal) only with respect to some criterion [22]. This criterion can be the desired value of the *loss function*, which measures the distance of the system from its objective. Training algorithms usually minimize the *average loss* over all training examples with respect to the weights of the network. In the previous chapter, that was the case with a neural network (TDNN) trained with backpropagation algorithm, trained to minimize Mean-Square-Error (MSE) or average square loss. This is quite acceptable for a lot of cases.

An alternative strategy is to minimize the *maximum loss* over all training patterns(see section 2.2.2). Such "Minmax" training [44] is guaranteed to capture the tail of the distribution (see figure 5.2). Therefore it is more suitable for the adaptation purposes, where the goal is to obtain a decision boundary, which is determined by the extreme patterns (i.e. examples of the rare writing styles).

To overcome the adaptation problems mentioned in the previous section, we replaced the last layer of the trained TDNN with a different type of classifier,

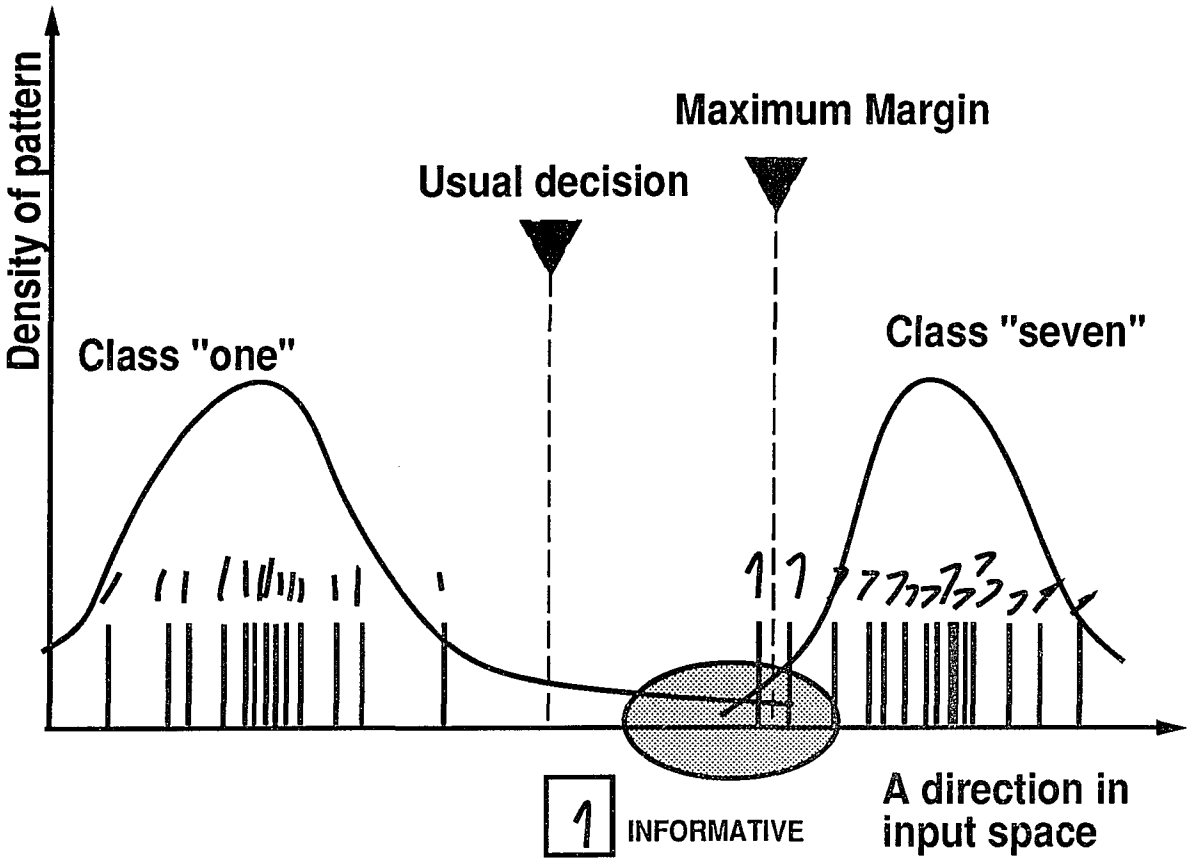

Figure 5.2: Minmax Training Algorithm: Usual decision boundary (like MSE) vs decision boundary as a result of Minmax training. Notice the role of the outlying patterns.

whose properties will lead to better memory utilization, without recognition speed degradation. In this adaptation scheme the recognition time for the input characters will be the same as with the original network. The classifier is the *Optimal Hyperplane Classifier* [44] described in section 2.2.2.

5.3 The On-line Adaptation Version of the Optimal Hyperplane Classifier Training Algorithm

In the section 2.2.2 it was shown that the maximum separation between the classes is obtained with the hyperplane whose norm vector $\mathbf{w}$ is parameterized with a subset of the training patterns (see equation 2.8). This patterns are called supporting patterns.

The property that the OH is expressed as a linear combination of the supporting patterns only facilitates adaptation.

In order to describe the on-line adaptation version of the Optimal Hyperplane classifier training algorithm we will look at the simple two-class example. Let us assume that an OH is generated (as described in section 2.2.2) to separate two classes A and B. (See figure 5.3a). We call such a hyperplane and the set of its supporting patterns the "default"(OH_d). Its direction is specified by a vector $\mathbf{w}_d^\star$:

$$\mathbf{w}_d^\star = \sum_{i \in A-default} \alpha^i \mathbf{x}^i - \sum_{i \in B-default} \beta^i \mathbf{x}^i \tag{5.2}$$

Let us assume that a new example of class "A" is presented for classification. If it is misclassified (see figure 5.3a), then the retraining scheme generates OH_{new}.

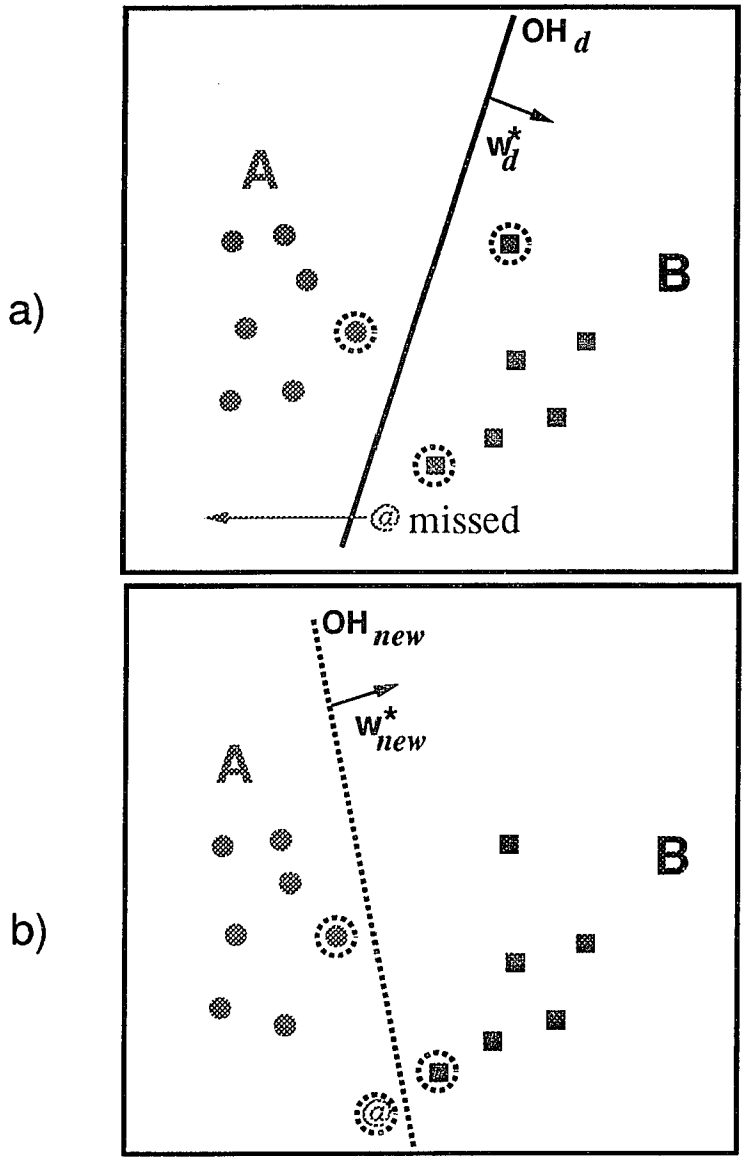

Figure 5.3: Optimal Hyperplane Classifier: a) Two class separation, "supporting patterns" are circled. The example "@" provided by a new user is misclassified. b) On-line adaptation, OH_{new} is generated. One of the old category-B patterns loses its status as a supporting pattern, while the new example becomes a category-A supporting pattern.

The training set for OH_{new} consists of the "default" supporting patterns augmented by the new example. The weight vector before retraining is initialized to $\mathbf{w}_d^\star$. The resulting OH_{new} is a linear combination of a "new" set of supporting patterns, including the new example. Its direction is specified by the vector $\mathbf{w}_{new}^\star$:

$$\mathbf{w}_{new}^\star = \sum_{i \in A-new} \alpha^i \mathbf{x}^i - \sum_{i \in B-new} \beta^i \mathbf{x}^i \tag{5.3}$$

The OH_{new} now represents the default classifier for potential new misclassified patterns.

In the case of the K-class problem, we use K OH classifiers where a single OH is trained for each possible category to distinguish members of that category from all other categories.

5.4 Training the Optimal Hyperplanes for Digits and Uppercase Letters

In order to build the complete OH *classification module* (OHC) for our writer adaptive system, we train a number of hyperplanes, one for each uppercase letter class and one for each digit class (i.e. predefined classes) using the training algorithm given in the section 2.2.2, now applied to multiple classes. For example, in order to train OH for class "A", two training sub-sets are used. All examples of letter "A" belong to one sub-set (i.e. positive examples). All other uppercase letters examples: "B" "C" ... "Y" "Z", belong the second sub-set (i.e. negative examples). In other words each "uppercase" OH separates one uppercase letter from all other uppercase letters. The same procedure is applied for for digits.

Training database

The training of the OHC is performed on the database collected among the staff at AT&T. The training set consists of 16,000 examples of digits and uppercase letters from 250 writers. The size of the training set for uppercase letters is 11,000. The size of the training set for 10 OH for digits is 5000. We have trained 26 OH, each one corresponding to one uppercase letter and 10 OH, each one corresponding to one digit.

Recognition Results Before Adaptation: TDNN *vs* OHC

Before the adaptation, we have measured the performance of the two different recognizer:

- TDNN trained on digits and uppercase letters.
- TDNN-core coupled with OHC module trained on the same training set.

The performance was measured on the independent test set of 2,000 uppercase letters.

The results obtained with "classical" TDNN recognizer shown in the Table 1 are compared to these obtained with OHC module working on the TDNN-core representation. The performance is given for each uppercase class separately. Associated with a class performance of the OHC is corresponding number of "support patterns" $< m >$ for the particular classifier. In the last column of the table the number of training patterns $< p >$ is split into the number of positive and negative examples used in the training of each corresponding OH classifier.

CLASS	TDNN performance	WIN-core+OHC performance	<m>	<p>
A	98.07 %	96.15 %	89	442/10471
B	94.33 %	96.15 %	90	390/10523
C	93.10 %	96.60 %	85	378/10535
D	98.11 %	98.11 %	94	372/10541
E	98.18 %	94.54 %	102	447/10466
F	100.00 %	98.18 %	104	406/10507
G	93.87 %	97.95 %	97	375/10538
H	94.33 %	96.20 %	90	401/10512
I	94.44 %	96.29 %	101	446/10467
J	92.72 %	94.54 %	100	395/10518
K	100.00 %	100.00 %	86	428/10485
L	98.18 %	100.00 %	80	418/10495
M	92.98 %	96.49 %	74	389/10524
N	94.54 %	94.54 %	83	421/10492
O	100.00 %	100.00 %	76	524/10389
P	100.00 %	100.00 %	71	417/10496
Q	98.14 %	94.40 %	79	396/10517
R	100.00 %	98.14 %	75	408/10505
S	100.00 %	100.00 %	70	416/10497
T	100.00 %	100.00 %	95	476/10437
U	96.42 %	98.14 %	83	419/10494
V	91.22 %	91.22 %	88	431/10482
W	98.21 %	94.54 %	69	417/10496
X	98.21 %	94.64 %	94	443/10470
Y	96.42 %	96.42 %	93	392/10521
Z	100.00 %	100.00 %	99	399/10514

From the results obtained, we can conclude that the performance of this two recognizers are quite similar before the adaptation takes place. That is what one might expect since they differ only in the type of the linear classifier used on top of the feature extractor (TDNN-core) and since the TDNN was trained with the emphasizing training scheme.

The VC-dimension (capacity) of the OH classifier is related to the number of support patterns [44, 24]. The range of the number of support patterns for each class is [69 - 104]. The VC-dimension of the second classifier, that is, fully connected last layer of the TDNN is equal to the dimension of the transformed input pattern or 120 in our case (see section 2.4.2. Though OHC has somewhat smaller capacity than the second classifier, the real advantage of using it instead of fully connected last layer of the TDNN comes from adaptation.

Recognition Results Before Adaptation: TDNN *vs* Optimum Margin Classifier

We trained second order polynomial classifier module instead of the linear OHC module in order to combine it with the TDNN-core. In order to do that we have used the extension of the OH classifier training procedure described in reference [24]. It is called *Optimum Margin* training algorithm. This algorithm operates on a large class of decision functions that are linear in their parameters but not restricted to linear dependencies in the input components. Polynomial classifiers fall into this class. Like in linear case, decision boundary is a combination of support patterns only.

The training set was the same like in the linear case. The performance was measured on the independent test set of 2,000 uppercase letters.

The performance for the second order polynomial Optimum Margin classifier was a little better than the performance of the OHC. The 97.34% *vs* 97% on the same

test set. But the number of supporting points increased as well. The range of the number of support patterns for each class in this case is [94-239]. If we wont to use polynomial classifier as a part of our adaptive recognizer, than the total memory requirements for the system will increase as well. Since one of identified desirable properties for the "good" writer adaptive system is modest memory requirement, we have decided that linear OHC will be a better solution.

5.5 Different Adaptation Schemes

In order to perform adaptation using the on-line adaptation version of the OHC training algorithm described in 5.3, we tried four different schemes.

1. For each misclassified pattern "i" with the desired label "X" retrain the OH_X with the training set that consists of all supporting patterns from $OH_{X-default}$ plus pattern "i". Pattern "i" is a positive example.
2. For each misclassified pattern "i" with the desired label "X" , if the answer of the classification module (i.e. normalized maximum of all OH classifiers) was "Y": (a) Retrain the OH_X with the training set that consists of all supporting patterns from $OH_{X-default}$ plus pattern "i". Pattern "i" is a positive example. (b) Retrain the OH_Y with the training set that consists of all supporting patterns from $OH_{Y-default}$ plus pattern "i". Pattern "i" is a negative example.
3. Same as scheme 2) plus: Retrain all the OH classifiers (besides "Y") whose normalized distance from the corresponding hyperplane was larger than the normalized distance for the classifier "X". Pattern "i" is the negative example.
4. For each misclassified pattern "i" with the desired label "X": (a) Retrain the OH_X with the training set that consists of all supporting patterns from

$OH_{X-default}$ plus pattern "i" plus all previous misclassified patterns whose desired label was "X". Pattern "i" and previous misclassified patterns are positive examples. (b) Retrain the OH_Y with the training set that consists of all supporting patterns from $OH_{Y-default}$ plus pattern "i". Pattern "i" is the negative example.

The same procedure is to be applied for a new symbol that a user introduces. In that case the new Optimal Hyperplane Classifier that discriminates among new symbol class and all current existing classes has to be trained first.

One can see, that the adaptation process consists of gradually changing the initial set of supporting points (one per class), with the new set that reflects the change in writing style from average to a particular.

It is important to say that this retraining of the Optimal Hyperplanes is performed only in adaptation phase. Following that, in utilization phase no additional computation is required for the recognition process then before adaptation, but much better performance is expected for a particular writer.

5.6 Final Adaptation System: Experiments and Results

Our final adaptation system consists of (see Figure 5.4):

- The writer independent TDNN described in the section 3.3 without its last layer (WIN-core). The WIN-core is used as a preprocessor that provides high level feature representation.
- A a set of 36 OH classifiers replacing the last layer of the TDNN. Initially default 36 hyperplanes are generated for the writer independent task as

described in section 5.4. They are split into two groups: 26 hyperplanes are trained to separate one upper-case letter from other upper-case letters and 10 hyperplanes are generated to separate one digit class from all other digit classes. The maximum output in the field designated by contextual information indicates the classification result.

When the pattern is presented, it is first transformed to the abstract internal representation provided by the WIN-core. That is a vector with the dimension 120. The transformed pattern is then presented to the last layer (i.e. Optimal Hyperplanes) for final classification. In case of a classification error, the user can provide the desired label and adaptation takes place, as described in the previous section. If an example of a non-predefined class (e.g. an arbitrary symbol, for instance symbol "+", previously unknown to the WIN) is presented to the system, a new OH for that class is initialized with that pattern as a positive example. Later, it is adapted as any other hyperplane.

The resulting classifiers had a total of 3130 supporting patterns before the adaptation started (see section 5.4). They were stored in the WIN-core representation (of 120 coefficients).

Adaptation tests were performed on data provided by seven writers distinct from the training-data writers. The data were provided by the character recognition group at IBM Thomas Watson Research Center. There were 450 characters per writer, including uppercase letters and digits on which the WIN was trained, plus 7 symbols (namely "+", "-", "*", "/", ")", "(", and "=") unknown to the WIN-core. The original data written by three out of seven different writers are given in the Figure 5.5 to Figure 5.7 respectively. Whenever a character was missed, it was used together with its label to fine tune the corresponding hyperplanes.

Figure 5.8 to 5.10 presents the results obtained with 4 different adaptation schemes (see section 5.5) together with baseline performance (no adaptation) for

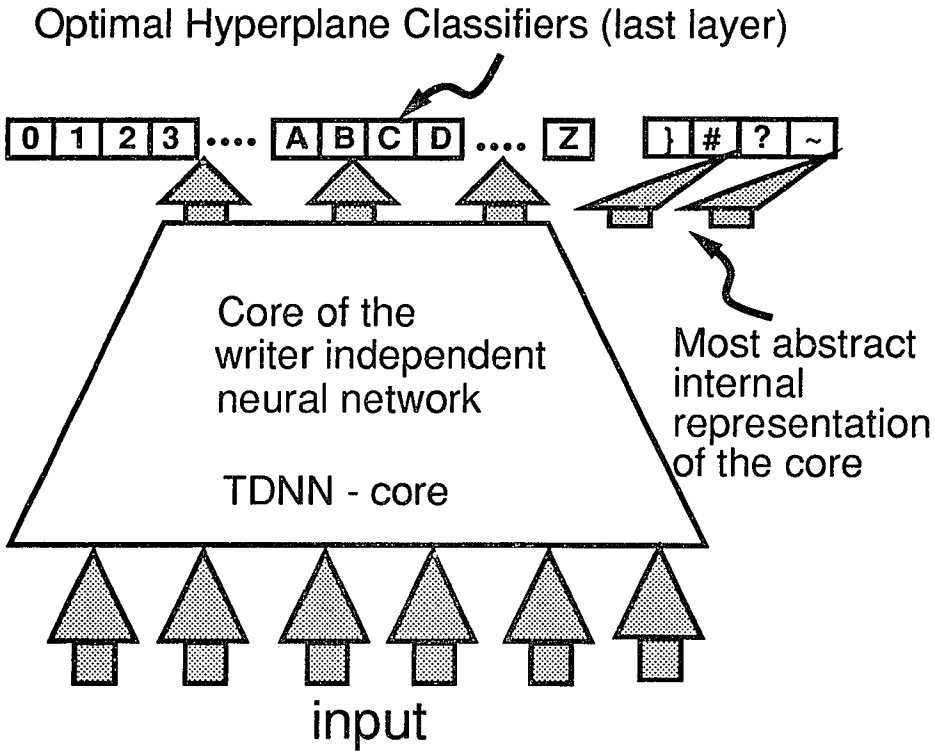

Figure 5.4: Writer adaptation system

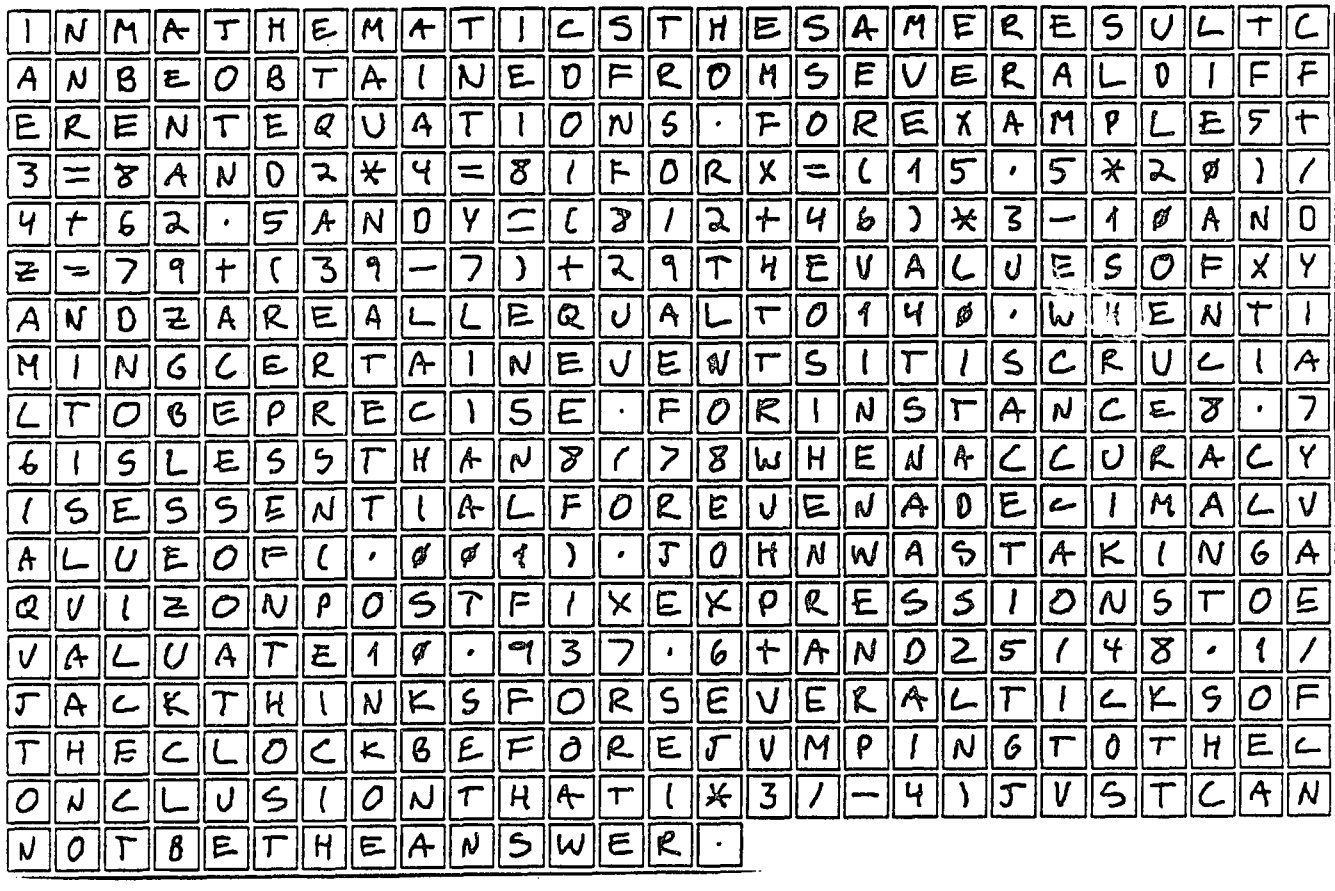

Figure 5.5: Test set used for the adaptation. Writer No.1

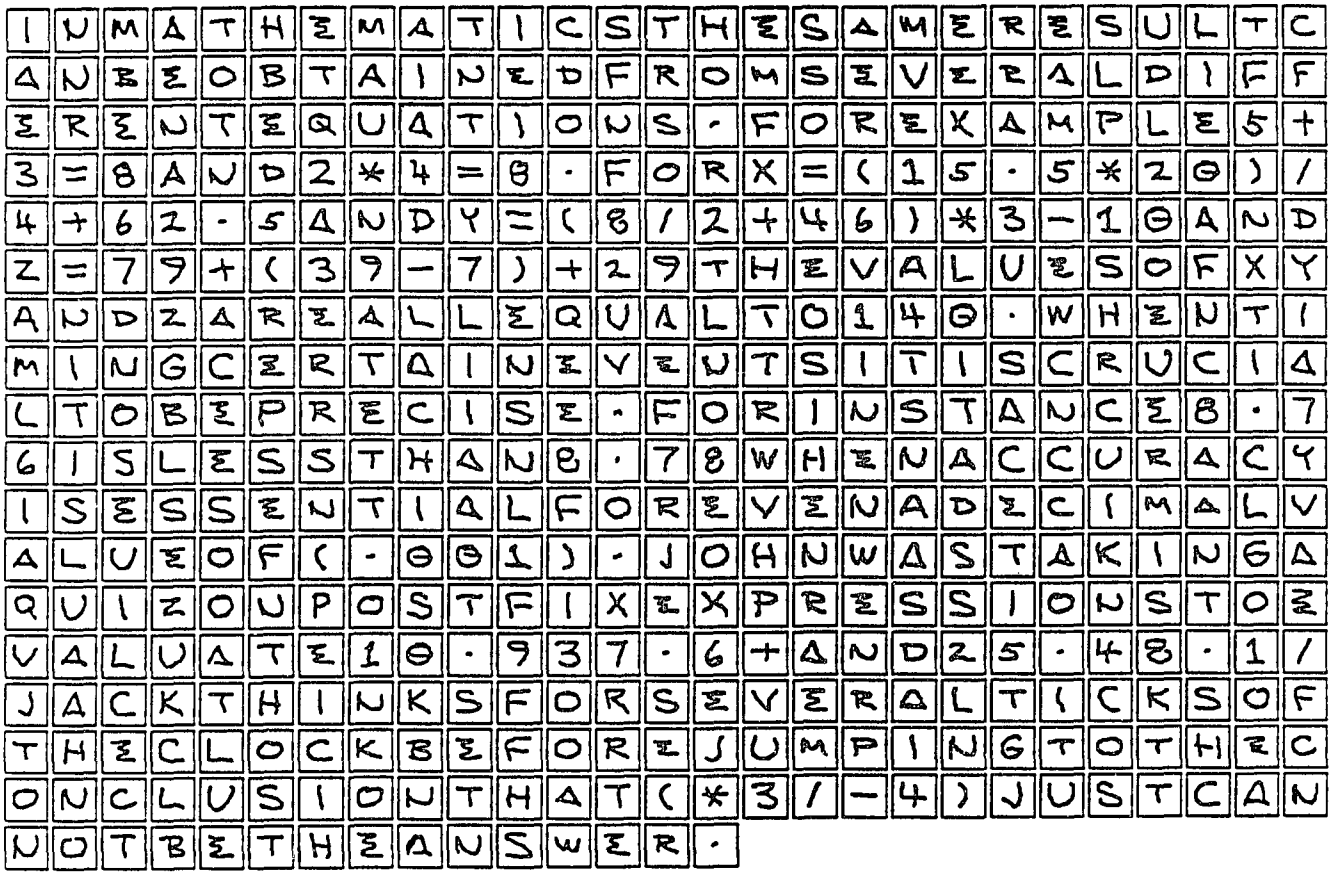

Figure 5.6: Test set used for the adaptation. Writer No.2

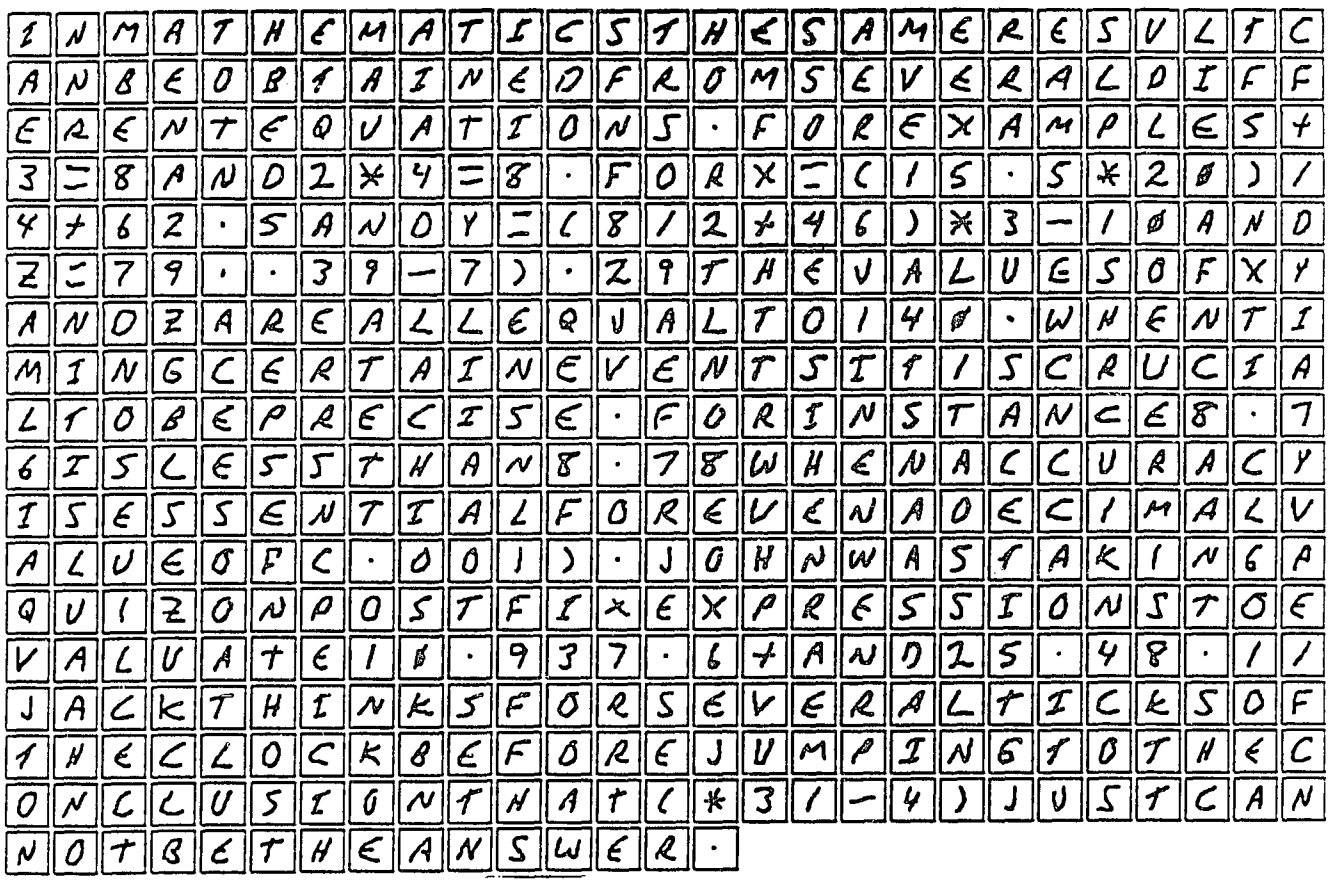

Figure 5.7: Test set used for the adaptation. Writer No.3

the first three writers. This tests were performed only on predefined classes (digits and uppercase). In each graph the total number of errors since the adaptation started is plotted against the number of examples presented for the adaptation. The slope of the curve gives the estimate of the error rate. The slope decreases as the system progressively adapts to the writer.

The second adaptation scheme has overall best performance in terms of accuracy and computing complexity, therefore it is chosen for our final adaptation system.

The final test was performed on all 7 writers with 450 characters per writer with all the available characters in the test set: uppercase, digits and new symbols ("+", "-", "*", "/", ")" "(", and "="). Results are shown in figure 5.11. Again, the total number of errors since the adaptation started is plotted against the number of examples presented for the adaptation. At the end of adaptation, the error rate averaged over the seven writers is 2.5%. One can notice that there are important differences between writers, some reaching easily less than 1% error and some remaining around 4% error. The total number of examples needed to learn the 7 new symbols and the writer-dependent variants of uppercase letters and digits never exceeded 40 in this experiment. What is interesting is that the number of "supporting patterns" never increases as the system adapts; indeed it actually decreases slightly.

5.7 Summary

In this chapter we presented the implementation of a writer adaptive recognition system, where user provides on-line examples of his own handwriting. Rare, informative patterns that are misclassified by the default system are used for adaptation purposes. The system allows fast learning of the new symbols as well. The adaptation phase is fast for both predefined classes and new classes.

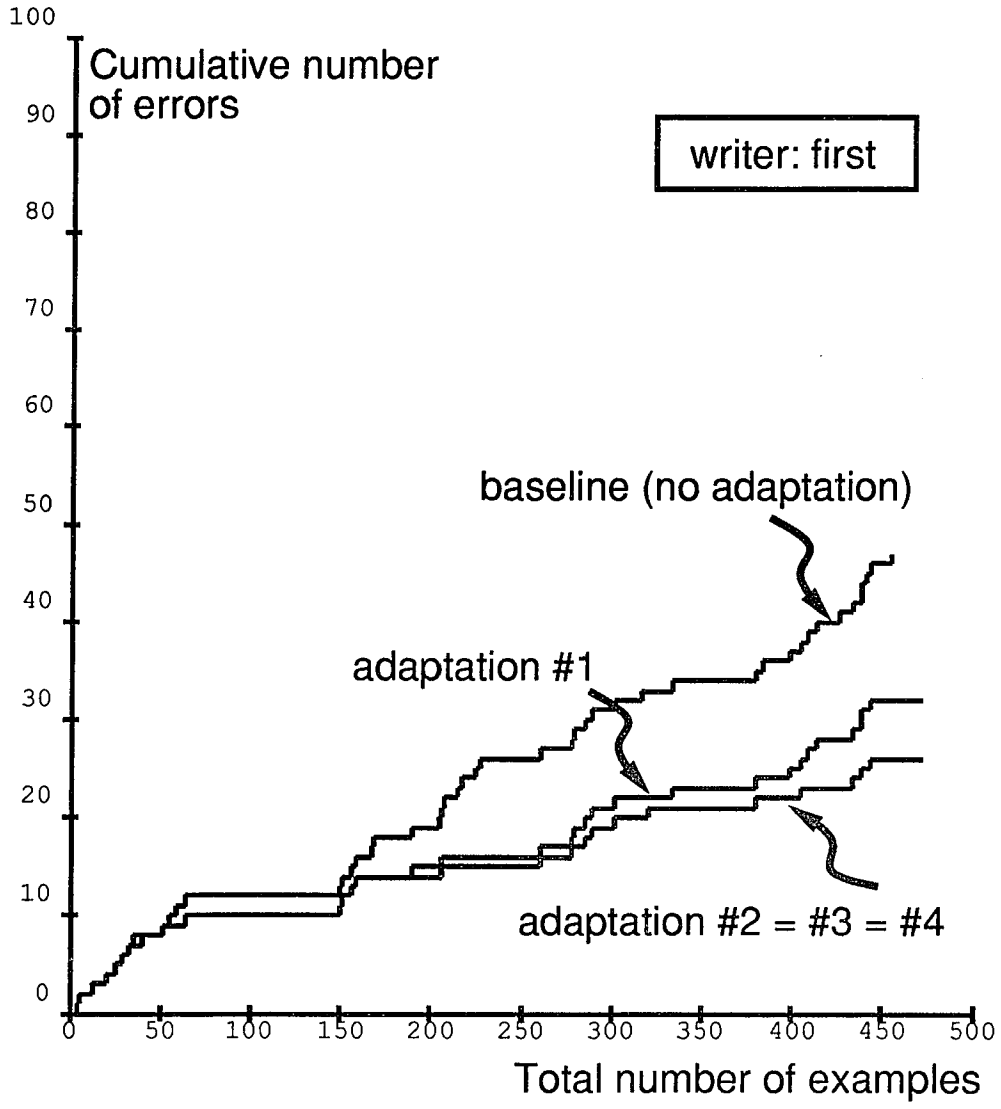

Figure 5.8: Cumulative number of errors. Different adaptation schemes labeled 1 to 4. Baseline performance is labeled "no adaptation". Writer No.1.

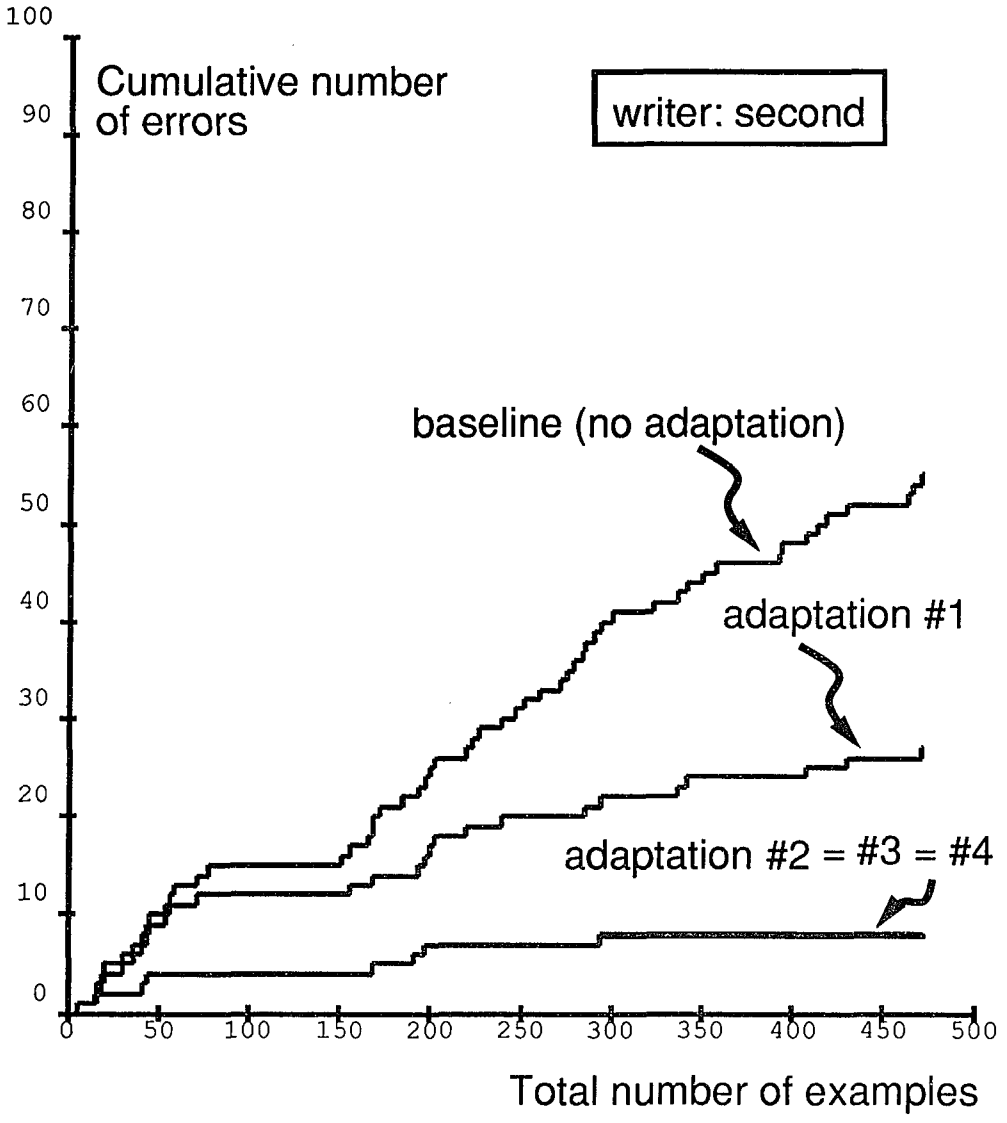

Figure 5.9: Cumulative number of errors. Different adaptation schemes labeled 1 to 4. Baseline performance is labeled "no adaptation". Writer No.2.

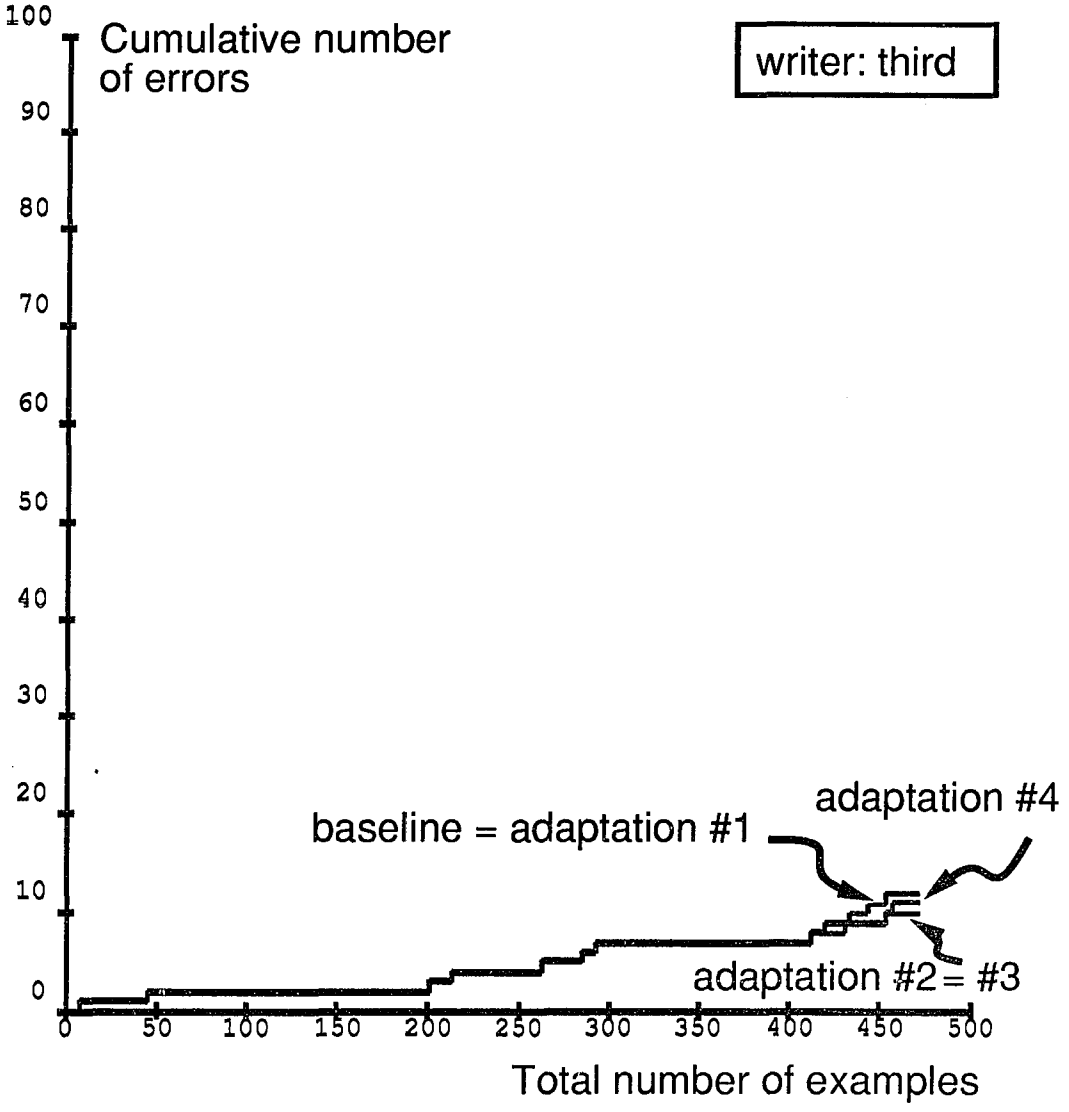

Figure 5.10: Cumulative number of errors. Different adaptation schemes labeled 1 to 4. Baseline performance is labeled "no adaptation". Writer No.3.

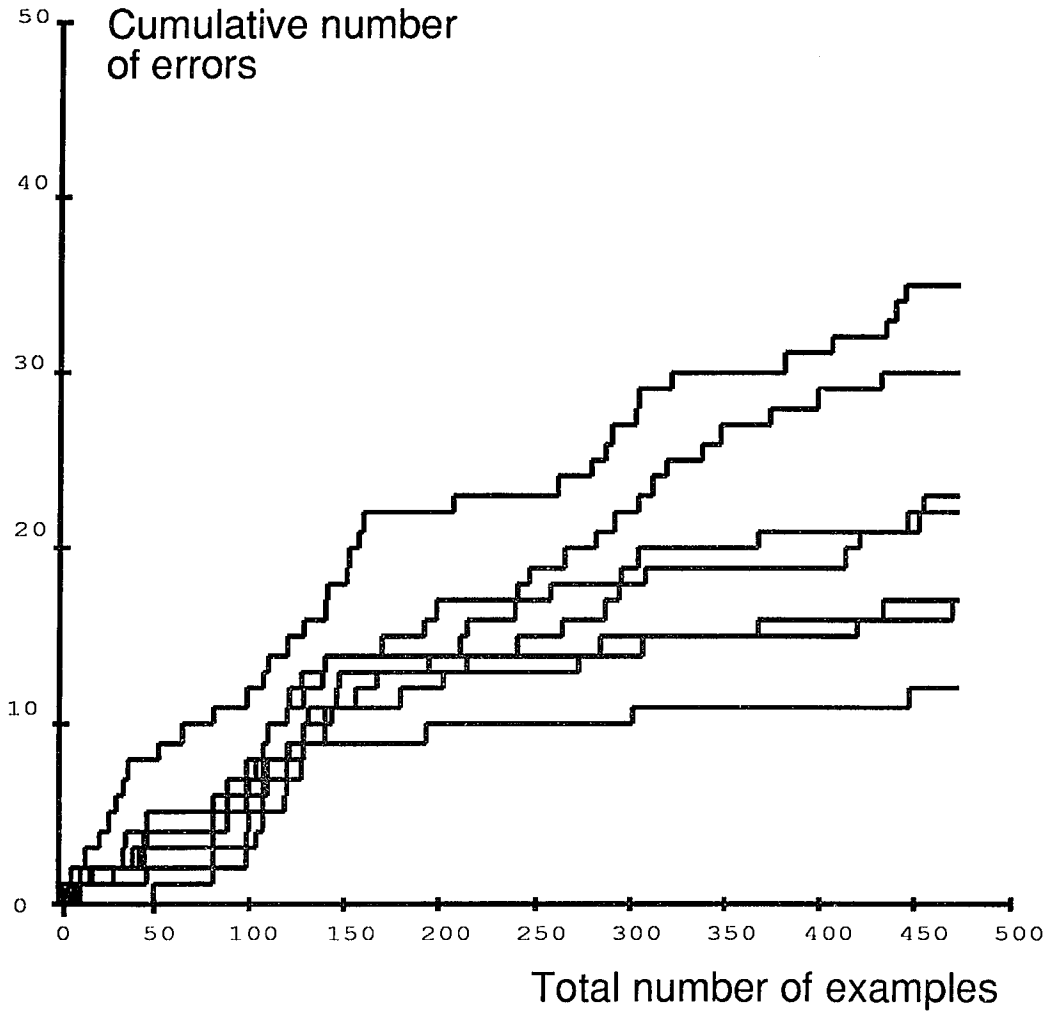

Figure 5.11: Cumulative number of errors for all seven writers. Best adaptation scheme. All the available characters: uppercase, digits and 7 symbols.

The recognition of the character (after adaptation) requires the same amount of time as in the writer independent case. Only modest additional memory is required for each customized version of the recognition system. In order to implement adaptation we used multi-module recognizer architecture. The first module consists of the writer independent TDNN without its last layer. Its role is to extract features from the task at hand. Second module is trainable classifier. We used Optimal Hyperplane classifier, and extended its training procedure to allow user adaptation. We tested the performance of our writer-adaptive system on the independent test set of characters provided by seven writers. Analysis of the adaptation results shows that the adaptation to a particular writer is successful, only few examples of the rare writing styles are required, and for most writers at the end of adaptation around 1% which is acceptable recognition accuracy for the on-line adaptation task.

6 Conclusions

This thesis investigates different ways to improve the performance of the on-line handwritten character recognition using the Time-delay Neural Network as a central module of the recognizer. Recognition of handwritten characters is hard task, even if we look at the limited case of recognizing isolated characters. If "pen-based" computing devices are to be widely accepted, very high recognition accuracy of the recognition module is required. To achieve that accuracy level, often different methods have to be applied in parallel. Alternatively one can limit the complexity of the task (i.e. adaptive vs. general recognizer) and try to obtain the high accuracy level in that particular case.

In this thesis we took a similar approach. We limit our task to isolated character recognition. Then we show that using the available data in an efficient way leads to the substantial recognition improvement. In particular, we have focused on the most informative patterns in the training data distribution. They are examples of the rare writing styles that are typically under-represented in the training database. We have used two different approaches in order to pay special attention to these patterns.

In our the writer-independent system we have developed a systematic 2 step computer-aided process. First, non-informative (e.g. meaningless and misla-

beled) patterns are removed from the training database. The rare and unusual patterns that remain after the meaningless or mislabeled patterns have been removed are then emphasized with a special training procedure, since they are extremely informative representatives of the tail of the data distribution. Using this approach we have reduced the generalization error by almost a factor of two (6.9% error for lowercase letters).

We introduce writer adaptation to allow a specific user to provide, on-line, examples of character styles that do not exist in the database. To do that, we use the writer independent network without its last layer as a preprocessor to the Optimal Hyperplane Classifier [44]. Through the adaptation, we retrain the Optimal Hyperplanes with the set of most informative patterns of the default database, augmented by the examples provided on-line by a specific user. Without degradation in speed, the error rate after adaptation is 1% to 2% for most of the writers.

6.1 Advantages of the Method

The super-supervised learning and its application to database cleaning, is a general methodology that can be applied on top of almost any sort of learning scheme that includes learning from examples in order to improve the performance of the *Learning Machine*. It facilitates efficient use of available data, efficient use of the computer time during the training and efficient use of the supervisor.

Since sometimes the objective of the recognition system is not to minimize necessarily the number of errors but the *cost* of the errors, super-supervised learning can be applied to tasks other than cleaning where special attention (and actions) have to be paid to the patterns for which inadvertent recognition has very high cost.

Our writer-adaptive module achieves very high accuracy for most of the writers. It allows a fast adaptation phase, where the user needs to give only a few additional examples of his own handwriting. New symbols can be added easily. There is no speed degradation compared to the default recognizer (before the adaptation) The implementation has modest memory requirements, so that several versions of customized recognizers might be needed on the system.

6.2 Limitations of the Method

Our methodology has been developed for the isolated characters only. Good isolated character recognizer can be the basis for the unconstrained handwriting recognition, but that is not the only way to solve the problem.

Our writer adaptation is supervised. This can be tedious for the user, since he has to spend some time to train the system (i.e. he has to provide the labels for the misclassified characters). Some form of semi-supervised learning, or unsupervised learning, is more comfortable solution for any user of the system. One way to do that is to incorporate context and/or syntax information. This often requires the existence of the "higher-level" knowledge module in the system that provides tentative labeling. It is often desirable to train such a module with the already existing recognizer. In our particular case it is not completely clear, how to achieve such combined training.

6.3 Future work

Our experiments on the role of the most informative patterns suggests a number of issues for further research. It would be interesting to use the super-supervised learning in problems other than the cleaning of the training database. Another

research direction would be to apply the methodology to learning from the examples algorithms other than backpropagation. Possible extensions would be to try the methodology on completely different databases, where sources of corruption are different (e.g. speech).

In writer adaptation, further direction is to try to use similar architecture for unsupervised adaptation. The first step is to try adaptation without the help of the high level knowledge, using the clustering property of the experimental data provided on-line. Secondly, the module that provides tentative labeling using contextual information should be included.

Bibliography

[1] S. Mori, K. Yammamoto, and M. Yasuda. Research on machine recognition of handprinted characters. *IEEE Trans. Patt. Anal. Machine Intell.*, 6:386–405, 1984.

[2] C.Y. Suen, M. Berthod, and S. Mori. Automatic recognition of handwritten characters - the state of the art. *Proc. of the IEEE*, 68:469–487, April 1980.

[3] C.C. Tappert, C.Y. Suen, and T. Wakahara. On-line handwriting recognition - a survey. In *9th International Conference on Pattern Recognition*, pages 1123–1130. IAPR, IEEE Computer Society Press, 1988.

[4] C. C. Tappert, C. Y. Suen, and T. Wakahara. The state of the art in on-line handwriting recognition. *IEEE Trans. Patt. Anal. Machine Intell.*, 12(8):787–808, 1990.

[5] Y. Le Cun. Generalization and network design strategies. In R. Pfeifer, Z. Schreter, F. Fogelman, and L. Steels, editors, *Connectionism in Perspective*, Zurich, Switzerland, 1989. Elsevier.

[6] I. Guyon, P. Albrecht, Y. Le Cun, J. Denker, and W. Hubbard. Design of a neural network character recognizer for a touch terminal. *Pattern Recognition*, 24(2), 1991.

[7] A. Leroy and I. Guyon. A pen editor prototype. Technical report, AT&T Bell Laboratories Technical Report No:11389-921123-33TM, 1993.

[8] I. Guyon, J. Bromley, N. Matić, M. Schenkel, and H. Weissman. Penacée: A neural network system for recognizing on-line handwriting. In Van Hemmen and et al., editors, *Models of Neural Networks*. Springer-Verlag, to be published.

[9] J.R. Ward and T. Kuklinski. A model for variability effects in handprinting with implications for the design of handwriting character recognition systems. *IEEE Trans. Syst., Man, Cybern.*, 18:438–451, 1988.

[10] I. Guyon, D. Henderson, P. Albrecht, Y. Le Cun, and J. Denker. Writer independent and writer adaptive neural network for on-line character recognition. In S. Impedovo, editor, *From pixels to features III*, Amsterdam, 1992. Elsevier.

[11] H. Weissman, M. Schenkel, I. Guyon, C. Nohl, and D. Henderson. Recognition-based segmentation of on-line run-on hand-printed words: Input vs. output segmentation. *Submitted to Pattern Recognition*, 1993.

[12] L. R. B. Schomaker and H. L. Teulings. A handwriting recognition system based on the properties and architectures of the human motor system. In C. Y. Suen, editor, *Frontiers in Handwriting Recognition*, Montréal, 1990. CENPARMI, Concordia University.

[13] H. L. Teulings and L. R. B. Schomaker. Unsupervised learning of prototype allographs in cursive-script recognition using invariant handwriting features. In S. Impedovo, editor, *From pixels to features III*, Amsterdam, 1992. Elsevier.

[14] K. A. Sizov. Recognition of symbols and words written by hand. In *Proceedings of the International Conference on Document Analysis and Recogntion*, volume 2, Saint-Malo, France, 1991. IAPR, IEEE, IEE.

[15] I. Guyon, V. Vapnik, B. Boser, L. Bottou, and S.A. Solla. Structural risk minimization for character recognition. In J. Moody and et al., editors,

NIPS-4, San Mateo CA, 1992. IEEE, Morgan Kaufmann.

[16] Brown M.K. and Ganapathy S. Preprocessing techniques for cursive script word recognition. *Pattern Recognition*, 16:447–458, 1983.

[17] C.C. Tappert. Adaptive on-line handwriting recognition. In *7th International Conference on Pattern Recognition*, pages 1004–1007, 1984.

[18] Impedovo S. Marangelli B. and Fanelli A.M. A fourier descriptor set for recognizing nonstylized numerals. *IEEE Trans. Syst., Man, Cybern.*, SMC-8:640–645, 1987.

[19] Y. Le Cun, Y. Bengio, A. Weisbuch, H. Weissman, and L. Jackel. On-line handwriting recognition using coarse-coded spatial representations. In *Snowbird Conference on Neural Networks and Computing*, 1993.

[20] J. Hollerbach. An oscillation theory of handwriting. *Biological Cybernetics*, 39:139–156, 1981.

[21] R.O. Duda and P.E. Hart. *Pattern Classification And Scene Analysis.* Wiley and Son, 1973.

[22] T. Kohonen. *Self-Organization and Associative Memory (2nd edition).* Springer-Verlag, New York, 1987.

[23] P. Simard, Y. Le Cun, and J. Denker. Efficient pattern recognition using a new transformation distance. In J. Moody and et al., editors, *NIPS-5*, San Mateo CA, 1993. IEEE, Morgan Kaufmann.

[24] B. Boser, I. Guyon, and V. Vapnik. An algorithm for optimum margin classifiers. In *Proceeding of the ACM workshop COLT'92*, Pittsburg,PA, 1992. IEEE.

[25] F. Rosenblatt. *Principles of Neurodynamics.* Spartan Books, New York, 1962.

[26] D.R. Hush and B.G. Horne. Progress in supervised neural networks. *IEEE Signal Processing Magazine*, 1:8–39, 1993.

[27] J.H. Breiman, R.A. Friedman, and C.J. Stone. *Classification and Regression Trees.* Wadsworth&Brooks, Pacific Grove, CA, 1984.

[28] J.R. Quinlan. Simplifying decision trees. *International J. Man-Machine Studies*, 27:221–234, 1987.

[29] J.R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[30] C. W. Therrien. *Decision, Estimation and Classification: An Introduction to Pattern Recognition and Related Topics.* Wiley, 1989.

[31] H. Arakawa. On-line recognition of handwritten characters-alphanumerics, hiragana, katakana, kanji. *Pattern Recognition*, (16):9–21, 1983.

[32] J. S. Bridle. Training stohastic model recognition algorithms as networks can lead to maximum mutual information estimation parameters. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2 (NIPS 89)*, pages 211–217, San Mateo CA, 1990. IEEE, Morgan Kaufmann.

[33] K. S. Fu. *Syntactic Methods in Pattern Recognition.* Academic Press, New York, 1974.

[34] P. J. Ye, H. Hugli, and F. Pellandini. Techniques for on-line chinese character recognition with reduced writing constraints. In *7th ICPR, International Conference on Pattern Recognition*, volume 2, pages 1043–1045, 1984.

[35] G.Y. Tang, P.S Tseng, and C.C. Hsu. A microcomputer system to recognize handwritten numerals using a syntactic-statistic approach. In *7th ICPR, International Conference on Pattern Recognition*, volume 2, pages 1061–1064, 1984.

[36] Lippmann R.P. Pattern classification using neural networks. *IEEE Communications Magazine*, pages 47–64, November 1989.

[37] G. E. Hinton and Y. Le Cun. Developments in artificial neural networks and their computing requiements. Technical report, AT&T Bell Laboratories Technical Report No:11359-920724-18, 1992.

[38] G. A. Carpenter and S. Grossberg. The art of adaptive pattern recognition by a self-organizing neural network. *Computer*, 21, 1988.

[39] Y. Le Cun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Handwritten digit recognition with a back-propagation network. In David Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2, (Denver, 1989), 1990. Morgan Kaufman.

[40] Y. Le Cun. Generalization and network design strategies. Technical Report CRG-TR-89-4, University of Toronto Connectionist Research Group, June 1989.

[41] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. Lang. Phoneme recognition using time-delay neural networks. *IEEE Trans. Acoust., Speech, Signal Processing*, 37:328–339, March 1989.

[42] K. J. Lang and G. E. Hinton. A time delay neural network architecture for speech recognition. Technical Report CMU-cs-88-152, Carnegie-Mellon University, Pittsburgh PA, 1988.

[43] Yann Le Cun, J. S. Denker, and S. Solla. Optimal brain damage. In David Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2, (Denver, 1989), 1990. Morgan Kaufman.

[44] V. Vapnik. *Estimation of Dependences Based on Empirical Data.* Springer-Verlag, New York, 1982.

[45] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[46] D. O. Hebb. *The Organization of Behavior.* Wiley, New York, 1949.

[47] N. J. Nilsson. *Learning Machines.* McGraw-Hill, New York, 1965.

[48] M. Minsky and S. Papert. *Perceptrons.* MIT Press, Cambridge, Mass, 1969.

[49] P. Werbos. *Beyond Regression.* Phd thesis, Harvard University, 1974.

[50] D. B. Parker. Learning-logic. Technical report, TR-47, Sloan School of Management, MIT, Cambridge, Mass., April 1985.

[51] Y. Le Cun. A learning scheme for asymmetric threshold networks. In *Proceedings of Cognitiva 85*, pages 599–604, Paris, France, 1985.

[52] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In *Parallel distributed processing: Explorations in the microstructure of cognition*, volume I, pages 318–362. Bradford Books, Cambridge, MA, 1986.

[53] I. Guyon. Neural networks and applications tutorial. In K. M. Decker, editor, *Parallel Architectures and Applications*, volume 207:(3-5). Physics Reports, North Holland, 1991.

[54] B. Widrow and S. D. Stearns. *Adaptive Signal Processing.* Prentice-Hall, 1985.

[55] L. N. Cooper, F. Liberman, and E. Oja. A theory for the acquisition and loss of neuron specificity in visual cortex. *Biological Cybernetics*, 33:9–28, 1979.

[56] B. Widrow and M. E. Hoff. Adaptive switching circuits. In *IRE WESCON Conv. Record, Part 4.*, pages 96–104, 1960.

[57] D. E. Rumelhart, J. L. McClelland, and the PDP research group. *Parallel distributed processing: Explorations in the microstructure of cognition. Volume I.* Bradford Books, Cambridge, MA, 1986.

[58] G. Cybenko. Approximation by superpositions of sigmoidal functions. *Mathematics of Control, Signals, and Systems*, 2:303–314, 1989.

[59] E. B. Baum and D. Haussler. What size net gives valid generalization? *Neural Computation*, 1(1):151–160, 1989.

[60] S. E. Fahlman and C. Lebiere. The cascade-correlation learning architecture. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2 (NIPS 89)*, pages 524–532, San Mateo CA, 1990. IEEE, Morgan Kaufmann.

[61] A. R. Barron and R. Barron. Statistical learning networks: A unifying view. In E.J. Wegman, D.I. Ganz, and J.J. Miller, editors, *Computing Science and Statistics: Proc. of the 20th Symposium on the Interface*, pages 192–202, 1989.

[62] A.C. Ivakhnenko. Polynomial theory of complex systems. *IEEE Trans. Syst., Man, Cybern.*, 1:364–378, 1971.

[63] J.H. Friedman and W. Stuetzle. Projection pursuit regression. *J.Amer.Stat.Assoc.*, 76:817–823, 1981.

[64] J.H. Friedman and J. W. Tukey. A projection pursuit algorithm for exploratory data analysis. *IEEE Transaction on Computers*, 23:881–889, 1974.

[65] Y. Le Cun, J. S. Denker, and S. Solla. Optimal brain damage. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2 (NIPS 89)*, pages 598–605, San Mateo CA, 1990. IEEE, Morgan Kaufmann.

[66] B. Hassibi and D.G. Stork. Second order derivatives for network pruning: Optimal brain surgeon. In J. Moody and et al., editors, *NIPS-92*, San Mateo CA, 1993. IEEE, Morgan Kaufmann.

[67] L.-Y. Bottou. Master's thesis, EHEI, Universite de Paris 5, 1988.

[68] S. J. Hanson and L. Y. Pratt. Some comparisons of constraints for minimal network construction with back-propagation. In D. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 1, Denver, 1988, 1989. Morgan Kaufmann.

[69] G. E. Hinton. Connectionist learning procedures. Technical report, Carnegie-Mellon University, Pittsburgh PA, 1987.

[70] S. J. Nowlan and G.E. Hinton. Simplifying neural networks by soft weight sharing. *Neural Computation*, 4(4):473–493, 1992.

[71] A.S. Weigend, D.E.. Rumelhart, and B.A. Huberman. Back-propagation, weight elimination and time series prediction. In *Proc. of the 1990 Connectionist Models Summer School*, pages 65–80, San Mateo, 1989. Morgan Kaufman.

[72] Y. Le Cun, L.D. Jackel, B. Boser, J.S. Denker, H.P. Graf, I. Guyon, D. Henderson, R.E. Howard, and W. Hubbard. Handwritten digit recognition: Application of neural network chips and automatic learning. *IEEE Communications Magazine*, pages 41–46, November 1989.

[73] I. Kanter, Y. Le Cun, and S. Solla. Second-order properties of error surfaces: learning time and ge neralization. In R. Lippmann, J. Moody, and D. Touretzky, editors, *Advances in Neural Information Processing Systems 3 (NIPS*90)*, Denver, CO, April 1991. Morgan Kaufman.

[74] S. Becker and Y. Le Cun. Improving the convergence of back-propagation learning with second-order methods. Technical Report CRG-TR-88-5, University of Toronto Connectionist Research Group, September 1988.

[75] P. J. Huber. *Robust statistics.* Wiley, New York, 1981.

[76] D. C. Hoaglin, F. Mosteller, and J. W. Tukey, editors. *Understanding Robust and Exploratory Data Analysis.* Wiley, New York, 1983.

[77] J.R. Quinlan. The effect of noise on concept learning. *Machine Learning:An artificial intelligence approach*, 2, 1986.

[78] J. Bromley and J. Denker. Improving rejection performance on handwritten digits by training with rubbish. *Neural Computation*, 5(3):367–370, 1993.

[79] S. A. Solla, E. Levin, and N. Tishby. A statistical approach to learning and generalization in layered neural networks. *Proceeding of the IEEE*, 78:1568 – 1574, 1990.

[80] H.S. Seung, M. Opper, and H. Sompolinsky. Query by committe. In *Proceeding of the 1992 Workshop on Computational Learning Theory*, Pittsburg,PA, 1992. IEEE.

[81] W. Krauth and M. Mezard. Learning algorithms with optimal stability in neural networks. *J. Phys. A: Math. gen.*, 20:L745, 1987.

[82] Y. Le Cun, E. Levin, and V. Vapnik. A method for experimental evaluation of the effective vc-dimension. *private communication*, 1991.

[83] C. C. Tappert. Speed, accuracy, flexibility trade-offs in on-line character recognition, IBM Research Report RC13228, 1987.

[84] E. Mandler. Advanced preprocessing technique for on-line script recognition of nonconnected symbols. In *Proceedings of the 3rd Int. Symp. Handwriting Comput. Appl.*, pages 64–66, 1987.

[85] M. de Bollivier, P. Gallinari, and S. Thiria. Multi-module neural networks for classification. In *Proceedings of the International Neural Network Conference*, volume 2, pages 777–780, Paris, July 1990. IEEE.