1

Penacée: A Neural Net System for Recognizing On-line Handwriting.

I. Guyon [1]
J. Bromley [2]
N. Matić [3]
M. Schenkel [4]
H. Weissman [5]

ABSTRACT We report on progress in handwriting recognition and signature verification. Our system, which uses pen-trajectory information, is suitable for use in pen-based computers. It has a multi-modular architecture whose central trainable module is a Time Delay Neural Network. Results comparing our system and a commercial recognizer are presented. Our best recognizer make three times less errors on hand-printed word recognition than the commercial one.
Keywords: handwriting recognition, neural networks, on-line handwriting recognition, signature verification, Time Delay Neural Network, hybrid systems, pen-based computers, cursive handwriting.

1.1 Introduction

This paper reports on progress in the design of a multi-modular system which recognizes on-line handwriting and whose central module is a Neural Network. By *on-line* we mean that the input to the system is the pen trajectory, sampled at regular time intervals by a touch sensitive pad. In

[1] AT&T Bell Labs, 50 Fremont street, 6^{th} floor, San Francisco, CA 94105, USA, isabelle@neural.att.com.
[2] AT&T Bell Labs, Room 4G-338, Holmdel, NJ 07733, USA, jbromley@neural.att.com.
[3] AT&T Bell Labs, Room 4G-309-a, Holmdel, NJ 07733, USA, nada@neural.att.com.
[4] AT&T Bell Labs and ETH-Zürich, CH-8092 Zürich, Switzerland, schenkel@isi.ethz.ch.
[5] AT&T Bell Labs, presently at 12 Mordehai-Hetez st., Petah-Tikua, Israel, f67361@barilan.bitnet.

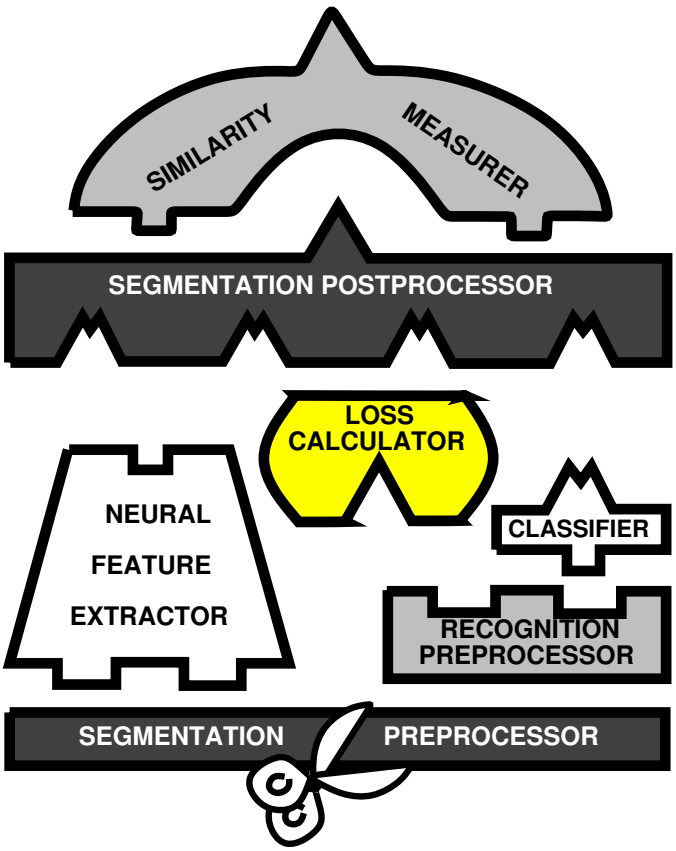

FIGURE 1.1. **Building blocks of the Penacée system.**

this article, we provide a synthesis of previously published work from our group [1, 2, 3, 4, 5, 6, 7] and a perspective on the on-going research.

There is urgent need for good recognizers to ensure the success of the first pen-based computers and pen-based personal communicators. In spite of the efforts of many companies and universities, the state-of-the-art on-line handwriting recognition accuracy has not yet reached a level which is acceptable to users. We tackle this problem from different angles. First, we address tasks of intermediate difficulty, but of real practical interest, such as the recognition of hand-printed words. Second, we introduce writer adaptation to fine tune the recognizer with examples of a particular user style.

The Penacée system (our Pen panacea...), is composed of several modules (preprocessor, classifier, segmentor, etc), as is classically done in pattern recognition [8]. The originality of using Neural Networks, and perhaps also the main advantage, is that the network itself can be decomposed into two modules [9]: a neural-network-based feature extractor and a classifier.

We make extensive use of our neural feature extractor which is a trainable module capable of producing a very good and compact feature representation. Our neural network is a Time Delay Neural Network (or $TDNN$) [10, 11]. It is a convolutional network which has several layers of local feature extractors. It is the one-dimensional version of the network used by the Optical Character Recognition group in our department [12, 13, 14] and is suitable for processing time varying signals, such as the pen-trajectory.

In Sect. 1.2, we introduce the various modules of our system and in Sect. 1.3 we present the results of applying it to isolated character recognitions, word recognition and signature verification.

1.2 Description of the Building Blocks

1.2.1 Recognition Preprocessor

The preprocessor converts the input to the system to a representation which facilitates the recognition process. Preprocessing techniques incorporate human knowledge about the task at hand, such as known invariances and relevant features. In this work, we use a rather crude preprocessing and rely mostly on the neural network to enforce invariances and extract features. Our preprocessing consists of fairly simple normalizations and extraction of low level local topological features such as line orientation.

Because the input data is the pen-trajectory, we face the choice of whether or not to use the dynamic information. It is possible get rid of the temporal parametrization of the data and represent patterns as pixel images. With such a representation, all the techniques used for OCR (Optical Character Recognition) are readily applicable. In this work, however, we encode patterns as a sequence of feature vectors, corresponding to the sequence of

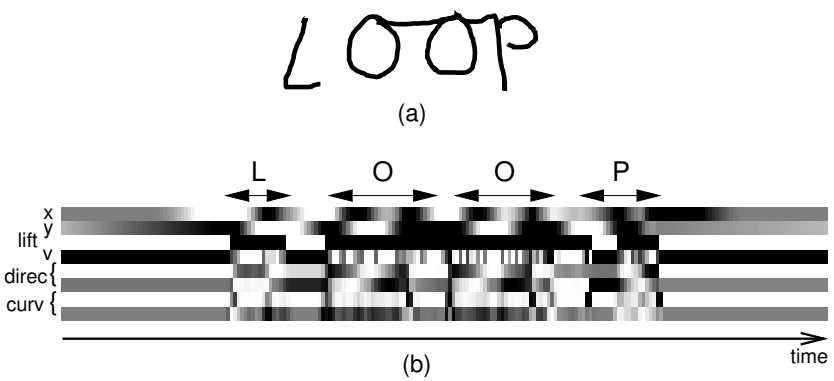

FIGURE 1.2. **Recognition Preprocessing.**
(a) The original word.
(b) The data as presented to the network. The feature vector in this case has 8 components encoding x and y coordinates normalized and rescaled, pen up or down information, speed of the pen, two components for direction of the pen trajectory and two components for its curvature.

drawing actions [1, 4].

In Fig. 1.2, we give an example of a preprocessed pattern with sequential encoding. The preprocessing is decomposed into normalization (centering, scaling, deskewing), resampling (to obtain a desired number of regularly spaced points along the trajectory), smoothing (to get rid of jittering) and feature extraction. In the resulting representation, each point on the trajectory is associated with a feature vector whose components are a subset of: x and y coordinates, direction of the trajectory, curvature of the trajectory, speed of the pen, acceleration of the pen, pen down or up position (touching or above the writing surface).

Depending on the application, the preprocessing may slightly vary. For instance, it is debatable whether the representation should be invariant under changes in the speed of the pen. For writer independent character recognition, variations in the speed of the pen are a nuisance. Conversely, for signature verification, the exact dynamics of drawing actions are very precious for the detection of forgeries. Data collection devices sample the trajectory at regular time intervals (10-12 milliseconds). Some invariance with respect to the speed of the pen is obtained, for character recognition applications, by resampling to points regularly spaced in arc-length, as opposed to regularly spaced in time. For signature verification, the resampling preserves regular time intervals.

1.2.2 Neural Feature Extractor

Our networks consist of several layers of feature extraction followed by a classification layer. The neural feature extractor in Fig. 1.1, is the network up to its classification layer.

We use a convolutional neural network, the Time Delay Neural Network ($TDNN$). $TDNN$'s were first introduced for speech recognition and are well suited to sequential signal processing [10, 11].

We briefly sketch here the principles the $TDNN$ (Fig. 1.3). One layer of the network transforms a sequence of feature vectors into another sequence of higher order feature vectors in the following way.

A given neuron detects a particular local topological feature of the pen-trajectory. Its receptive field is restricted to a limited time window. The same neuron is reused along the time axis (Fig. 1.3, the same neuron is replicated in the time direction) to detect the presence or absence of the same feature at different places along the trajectory. By using several different neurons at each time step, the neural network performs the detection of different features (Fig. 1.3, the outputs of different neurons produce a new feature vector in the next layer, at a given time step).

The operations performed by one layer of the network are convolutional in essence (Fig. 1.4). Each neuron k in layer $\ell+1$ has an associated convolution kernel of height m (the number of features in layer $\ell+1$) and of width δ. The coefficients of the kernel are the neuron weights $w_{i,j}^{k,\ell}$, $i = 0, ...(\delta-1)$ and $j = 0, ...(m-1)$. The convolution of the states $f_j^\ell(t)$ of layer ℓ with kernel k is another sequence of states $f_k^{\ell+1}(t)$ of layer $\ell+1$ corresponding to confidence levels for the presence or absence of a given feature k along the pen trajectory. State sequences are parametrized by the discrete time variable t. The confidence levels are squashed by the neuron sigmoid function g[.]:

$$f_k^{\ell+1}(t) = g[\sum_{i=0}^{\delta-1} \sum_{j=0}^{m-1} w_{i,j}^{k,\ell} \ f_j^\ell(t-i)] \ . \tag{1.1}$$

with

$$g[x] = \alpha \tanh \beta x, \tag{1.2}$$

where tanh denotes the hyperbolic tangent, and where $\alpha = 1/\tanh(2/3)$ and $\beta = 2/3$.

The time component of our input representation is gradually eliminated by subsampling the convolution at each layer by a factor of two or three. To partially compensate for the loss of information, the number of features is gradually increased [15]. This is what we call a bi-pyramidal network architecture. Bi-pyramidal networks convert progressively time information into feature information.

The weights are adjusted during a supervised training session, using error back-propagation [16] which performs gradient descent in weight space with a mean-squared-error (MSE) loss function (see Sect. 1.2.6).

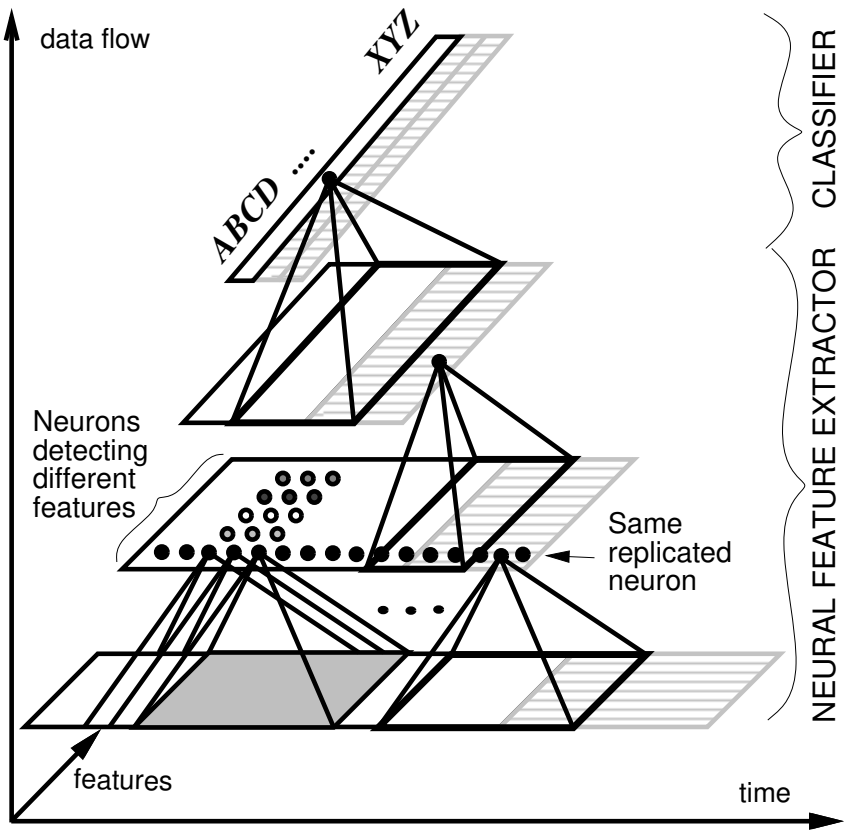

FIGURE 1.3. **Architecture of the Time Delay Neural Network ($TDNN$).** The connections between layers obey the following rules (not all neurons are represented):

- neurons are feature detectors with restricted input fields, limited in the time direction,
- in each layer, a set of neurons scans the output of the previous layer along the time axis, and produces higher-level feature vectors,
- the sequence of feature vectors is subsampled in time at each layer to obtain time contraction.

For isolated character recognition, the time contraction is such that characters, which fit into a fixed size input window, correspond to one output vector, for which the time dimension has been completely eliminated. The figure also shows that the convolution can be extented in time and so that, for a variable length input, a sequence of output vectors is produced. This is used in Sect. 1.3.2 for the recognition of entire words.

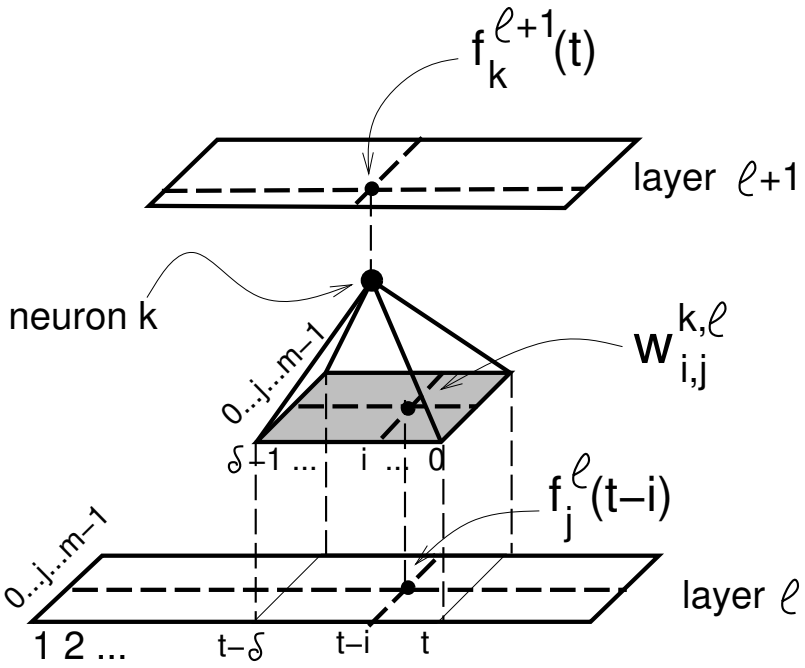

FIGURE 1.4. **Operations of one layer of the** $TDNN$**.**
Each layer ℓ produces an intermediate representation of the input pattern, as a sequence of feature vectors. We denote by $f_j^\ell(t)$ the j^{th} component of that vector at time t. The intermediate representation of layer $\ell+1$ is obtained by sweeping several "neuron feature detectors" over the intermediate representation of layer ℓ. Neuron k produces the sequence $f_k^{\ell+1}(t)$.

Because of the convolutional structure of our network, the same neuron is replicated along the time axis. For the unfolded architecture (Fig. 1.3), one usually talks about "weight sharing" among the various replicas of the same neuron [17]. Weight sharing is enforced during training by averaging the weight updates of the various replicas.

1.2.3 Classifier

The classifier is often part of the neural network itself. On top of the feature extraction layers, the last fully connected layer performs the final classification (Fig. 1.3).

The last layer of the $TDNN$ consists of as many neurons as there are classes, $\{char_1, char_2, ...char_C\}$. During training, when a character of class $char_x$ is presented, we impose a desired values of -1 to all neurons, except to the neuron associated to the $char_x$ class which receives a $+1$ desired value. Thus, we bring back our C-class problem to C 2-class problems: each neuron is trained to separate one class versus all the other ones. After training, the classification is made according to the largest output value. The outputs of the neural network may be used to estimate the *posterior* probabilities $P(char_x|input)$ [6]. In that case, the classification is made according to the maximum *a posteriori* probability.

We make the distinction between neural feature extractor and classifier for two reasons:

- The neural feature extractor can be used independently to provide a compact representation of the data. In applications such as writer adaptation (Sect. 1.3.1) and signature verification (Sect. 1.3.3), patterns are stored for later use in this representation.

- Convolutional networks are good at extracting features, but there may be more suitable classifiers for the task at hand, that can be used instead of the last layer of the network [9]. For instance, a K-nearest neighbor classifier or an Optimal Margin classifier [18] replace the last layer for writer adaptation in Sect. 1.3.1.

1.2.4 Segmentation Preprocessor and Postprocessor

Our neural network isolated character recognizer is part of a larger system which can perform word recognition. The segmentation techniques that we use to separate a word into characters are closely related to Dynamic

[6] Given our choice for the sigmoid function (equation 1.2) and desired values, the outputs of the network are between -1.7 and $+1.7$. They are rescaled between 0 and 1 to obtain probability estimates.

Time Warping and Hidden Markov Models, used for speech recognition [19]. Their resemblance with neural networks has been pointed out by several authors [20, 21, 22, 23] but will not be emphasized here.

We distinguish several levels of difficulty in the segmentation problem (see Fig. 1.5). If characters are entered in boxes or if they are clearly spaced, the segmentation is trivial and can be handled independently of the recognition process. But, in the absence of boxes, people usually do not space their characters uniformly, which results in many segmentation ambiguities. To address this harder task, our strategy is to let the *segmentation preprocessor* make guesses and provide the recognizer with several likely segmentation hypotheses. The final segmentation and recognition decisions are then taken all at once by the *segmentation postprocessor*, using the recognition scores.

In our nomenclature, the segmentation preprocessor isolates segments of pen trajectory called *tentative characters*. A *stroke* is an elementary segment between a pen down and the next pen up. The segmentation preprocessor may take advantage of such pen-lifts and spaces between strokes and/or minima of the speed of the pen to determine *tentative cuts*. Alternatively, the segmentation preprocessor can simply sample regularly windows of the input signal.

Tentative characters usually overlap considerably. The segmentation postprocessor uses an *interpretation graph*. The nodes of this graph contain the tentative character recognition scores and the transitions between nodes favor particular character chainings. This avoids reusing the same part of a character multiple times and allows the implementation of frequencies of character successions particular to a given language.

We use the recognition scores provided by the neural network to provide estimates of the *posterior* probabilities of the various character interpretations $char_x$ given the input: $P(char_x|input)$. The transition coefficients are estimates of $P(char_y|char_x)$, determined independently of the recognition process. Using (abusively) the assumption of independence between the recognition of the various characters, the segmentation postprocessor computes probability estimates of character sequence interpretations as:

$$P(char_x, char_y, ...|input) = P(char_x|input).P(char_y|char_x).P(char_y|input)... \tag{1.3}$$

The output of the segmentation postprocessor is the best path in the graph, corresponding to the most probable word (or character sequence). Our implementations will be described in Sect. 1.3.2.

1.2.5 Similarity Measurer

So far, we have assumed that the neural network would be used to estimate the *posterior* probability $P(char_x|input)$. But one may be interested in having the network estimate $P(input_1 \sim input_2)$ that the estimate of the

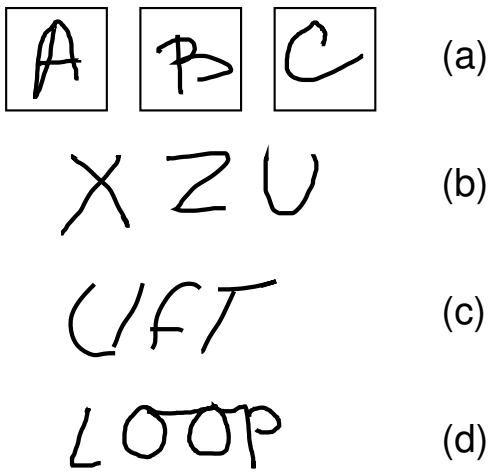

FIGURE 1.5. **Writing styles handled by the Penacée segmentors.** (a) Boxed characters. (b) Clearly spaced character. (c) Ambiguous spacing but consistent pen-lifts between characters. (d) Connected characters.

similarity between two patterns. Such a network has found applications to classification problems, using a K-nearest neighbor classifier or a kernel classifier [24] and to verification problems such as signature verification [4] (Sect. 1.3.3) and finger-print verification [25].

Similarity measures are very problem dependent and must usually be designed by experts. As an alternative, we tailor the similarity measure to the task at hand by training a neural network classifier to separate pairs of similar patterns and pairs of dissimilar ones.

For instance, take two identical neural feature extractors net_1 and net_2. The first one encodes $input_1$ into $output_1$ and the second one $input_2$ into $output_2$. Add on top a *similarity measure module* which computes the dot product between $output_1$ and $output_2$ to obtain the degree of similarity between the two patterns. We named a system with such an architecture a "siamese" neural network [4] (see Fig. 1.2.5).

During training, similar patterns (e.g. two genuine signatures from the same person) are given a large positive desired degree of similarity. Conversely, dissimilar patterns (e.g. a signature and its forgery) are associated with a negative or zero desired degree of similarity. The network is trained with the back-propagation algorithm by back propagating errors through the *similarity measure module* (Sect. 1.2.7). The constraint that $net_1 = net_2$ is enforced during training by averaging the weight updates of corresponding weights in net_1 and net_2.

Other *similarity measure modules* can be used instead of the simple dot product, such as a Gaussian similarity measure $\exp(-d^2/\sigma^2)$, where d is the Euclidean distance between $output1$ and $output2$. One could also imagine using an elastic matching module to perform better time alignment. We use in Sect. 1.3.3 a normalized dot product: $\cos(output_1, output_2)$.

1.2.6 Loss Calculator

The loss calculator computes a penalty function or *loss function* which measures the distance of the system from its objective. Combined with an appropriate optimization technique such as gradient descent, the loss function guides the system during training towards our goal. A very commonly used loss function is the square loss:

$$L_{sq} = (output - desired)^2,$$

where *output* is the actual output of the system and *desired* its corresponding desired value. For instance, for a classification problem with two classes A and B, all elements of class A are assigned a desired value $+1$ and all elements of class B a desired value -1. After training, if the output of the system is positive, the input pattern is classified as A, otherwise as B.

The goal of learning is to get best generalization performance. For a classification problem, this means that after training the classification error rate on examples not used for training (test set) must be low. It is not clear

whether minimizing a loss function using only training examples can yield good generalization. However, both theoretical studies and experimental results [26] show that good generalization can be achieved if the "capacity" of the learning system is matched to the number of training examples. If the capacity is too large, the system can easily learn the training examples, but usually exhibits poor generalization (overfitting). Conversely, if the capacity is too small, the system is not capable of learning the task at all (underfitting).

The capacity of a neural network is related to the number of free parameters [27]. This is a relatively small number for convolutional networks such as the $TDNN$, compared to fully connected networks and to networks with local connections but with no weight sharing [17]. More recent work talks rather of an "effective capacity" which incorporates properties of the input space, network architecture and training algorithm [26]. A simple way of affecting the capacity during training is to modify the loss function by adding, for instance, a weight decay term which pulls the weights to zero.

In our present work, the capacity control is handled by "early stopping". Before training, 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 1.2). The initial effective capacity is smaller than what the number of tunable parameters suggests: it is equal to the number of free parameters of the equivalent linear system. During training, as the weights increase, use is made of the non-linearity of the squashing function and the capacity progressively increases until it reaches an optimum which is detected by cross-validation. We do cross-validation by computing the performance of the system on a small set of patterns, distinct from the training set and the test set.

Training algorithms usually minimize the average loss over all training examples, with respect to the weights of the network. Such is the case for the original back-propagation algorithm [16] which minimizes the Mean-Square-Error (MSE) or average square loss. This technique is widely applied and well suited to a large variety of problems. An alternative strategy is to minimize the maximum loss over all training patterns ("minimax" training) [28]. This technique is guaranteed to capture the tail of the distribution of input patterns but is very sensitive to outliers [18]. Better generalization than with MSE minimization is obtained with minimax training after careful screening of eventual mislabeled or meaningless patterns which have been inserted in the training data [3] (see Sect. 1.3.1).

1.2.7 Global Optimization Techniques

Once the various modules described in the previous sections are assembled into a system, the problem arises of optimizing all the parameters (i.e. the weights of the network) with respect to the final objective determined by the loss function. In reference [29], a general framework has been proposed

to achieve the global optimization of a multi-modular architecture. We present here a particular example which illustrates the main ideas.

We train our neural network with gradient descent, for which the weight updates are computed as:

$$\Delta w = -\epsilon \frac{\partial L}{\partial w} \ , \tag{1.4}$$

where L is a loss function, w a weight of the network and ϵ is a small positive quantity (the gradient step or learning rate). The back-propagation algorithm permits chaining the computation of the partial derivatives from the output to the input of the network [16].

Consider the case of the "siamese" neural network, described in Sect. 1.2.5. To compute the weight updates and use the chaining rule of back-propagation, one needs to compute first the partial derivative:

$$\frac{\partial L}{\partial output_i} \ , \tag{1.5}$$

where i is either 1 or 2. Assume that we use a cosine similarity measure and a squared loss:

$$L(input_1, input_2, \mathbf{w}) = [\cos(output_1, output_2) - desired]^2 \tag{1.6}$$

$$= [(\frac{output_1.output_2}{\|output_1\|\|output_2\|}) - desired]^2 \ , \tag{1.7}$$

where $desired = 1$ if patterns are similar and 0 otherwise. Then, using the condensed notation S for $\cos(output_1, output_2)$, X for $output_1$, Y for $output_2$ and D for desired, we obtain:

$$\frac{\partial L}{\partial X} = \frac{\partial L}{\partial S}\frac{\partial S}{\partial X} \tag{1.8}$$

$$= 2(S-D)\frac{1}{\|X\|\|Y\|}(Y - S\frac{\|Y\|}{\|X\|}X), \tag{1.9}$$

from which all other partial derivatives with respect to the weights of the network can be computed using back-propagation.

1.3 Applications

In this section we present solutions to problems in on-line handwriting recognition and signature verification which involve various combinations of the modules previously described.

We have designed the Penacée system so that ultimately it would handle unconstrained handwriting of any style. We report first the results we have obtained for the writer independent recognition of digits, uppercase and

lowercase letters, written in boxed mode. Then, we explain the techniques that we have used to handle the recognition of unsegmented hand-printed uppercase words (run-on mode). We report the results of two methods of segmentation: $INSEG$ (for INput SEGmentation) and $OUTSEG$ (for OUTput SEGmentation). We show that the two methods have complementary advantages. We propose a combination of $INSEG$ and $OUTSEG$ and preliminary results indicating that our final system may be able to handle unconstrained handwriting. Finally we present an application of "siamese" neural networks to signature verification.

1.3.1 Isolated Character Recognition

The simplest task that we have addressed is that of isolated character recognition which involves the combination of modules shown in Fig. 1.6. The specifications of the neural feature extractor (a $TDNN$ up to its second last layer) are summarized in table 1.1. We report several experiments with the same neural feature extractor but different classification layers. We always use one output neuron per character interpretation or class. The desired output values for a given interpretation $char_i$ are -1 for all the neurons, except for the i^{th} neuron which has desired output 1. We train the network with back-propagation.

The experiments are performed with data collected in the cafeteria of an AT&T facility. Approximately 250 writers contributed to at least one complete set of the 10 digits, the 26 uppercase letters and the 26 lowercase letters. We address the writer independent task, meaning that different sets of writers are used for training and testing.

In reference [1], we report results obtained on digits and uppercase letters. A set of approximately 12,000 examples is used for training and 2,500 examples for testing. A $TDNN$ with 36 outputs in its classification layer is trained to recognize either digits or uppercase letters. This means that during training, no error is back-propagated from the uppercase letter neurons if a digit is presented, and *vice versa*. At utilization time, the *a priori* information whether the character to be recognized is a digit or an uppercase letter is needed. For instance, the neuron with maximum activation among the digit output neurons is selected if the character to be recognized is known to be a digit. We obtain an error rate on the test set of 3.4% error (2.3% if tested on digits only and 3.8% if tested on uppercase letter only).

In reference [2], we propose a training algorithm (the "emphasizing scheme") which enables atypical patterns, such as characters written by left handed people, to be learnt. The method is a simple way of improving the information theoretic learning efficiency. It consists, during back-propagation training, in presenting more often the least predictable patterns (i.e. the ones with largest squared loss). The method is related to minimax learning procedures (Sect. 1.2.6) such as Optimum Margin classification [18] and to Boosting techniques [30]). Using this method, the error rate is reduced

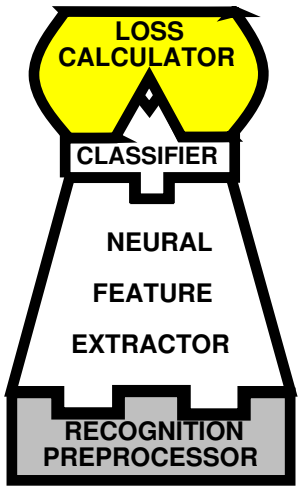

FIGURE 1.6. **The architecture of the isolated character recognizer.**

input length	subsampling steps	δ/m 1^{st} layer	δ/m 2^{nd} layer	δ/m 3^{nd} layer	δ/m 4^{nd} layer
90	3.3.2	8 / 7	6 / 10	4 / 16	5 / 24

TABLE 1.1. $TDNN$ **feature extractor for character recognition.**
The subsamplings steps are time subsampling steps from one layer to the next. The time scale is thus contracted by a factor 18. The kernels have dimensions δ by m (Fig. 1.3). The neural feature extractor has 34,660 connections in its unfolded version (Fig. 1.3) but only 3,106 independent weights, due to "weight sharing" (Sect. 1.2.2).

from 3.4% to 2.8%. Even more importantly, the variation in error rate for different writers is substantially decreased.

In reference [3], we present results on lowercase letters. In this study, we propose a "super-supervised" learning technique the purpose of which is to detect undesirable outliers (mislabeled or meaningless patterns introduced by accident into the database). During the supervised learning session which utilizes the class labels stored in our database, a human "super-supervisor" double checks the labels of the patterns which are hardest to learn and are therefore suspicious. We show that the "emphasizing scheme" works best when the data is cleaned by a "super-supervisor". A $TDNN$ with the same feature extraction layers as specified in table 1.1, but with only 26 neurons in the classification layer (for the lowercase letters), is trained with approximately 9,500 lowercase letters and tested with 2000 letters from different writers. Our final error rate of 6.9% error is considerably better than the initial error rate of 11% error which is obtained with standard back-propagation and without cleaning the data.

In references [2, 5], we address the problem of writer adaptation. No matter how good a writer independent system becomes, there will always be too many variations in writing styles to ensure very high recognition accuracy for all writers. It is therefore important to allow a given writer to fine tune the system to his own handwriting. We have developed techniques which allow the same neural feature extractor to be kept and retrain only the classification module. The simplest method consists in using a Nearest Neighbor classifier instead of the last layer of the network. A selection of prototypes are stored in the representation of the neural feature extractor. Patterns are classified according to the class of their nearest prototype. When the system fails to recognize a pattern, an additional prototype is generated. New classes can be introduced, allowing for customizations of the recognizer by adding new sets of symbols and gestures. Performing Nearest Neighbor classification in the representation of the neural feature extractor, as opposed to the network input representation, is advantageous for two reasons: few patterns are required for adaptation (usually only one or two), thanks to the robustness of the representation; little storage is required, thanks to the compactness of the representation. Using an Optimal Margin classifier instead of the Nearest Neighbor classifier, the same accuracy is retained while recognition speed is improved and fewer prototypes are needed. This enables us to reach very high recognition accuracies for most writers (less than 1% error), with fast adaptation and at the expense of no recognition speed degradation.

1.3.2 Hand-printed Word Recognition

In references [6, 7], we address the harder task of recognizing hand-printed words that are not *a priori* segmented into letters. This task involves the combination of modules shown in Fig. 1.7. We report results obtained with

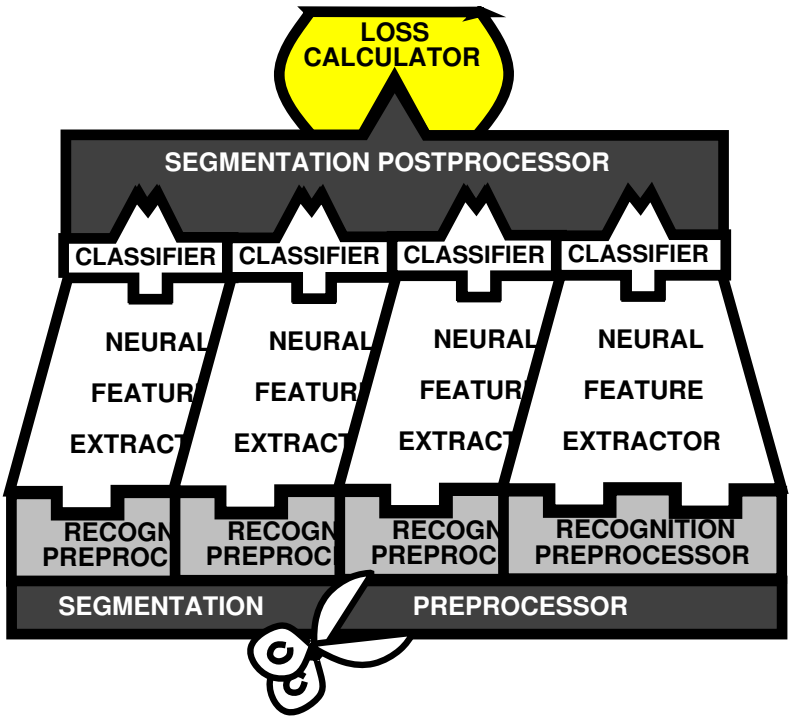

FIGURE 1.7. **The architecture of the hand-printed word recognizers.**

two methods of recognition-based segmentation. The methods are designed to work in "run-on mode" where there is no constraint on the spacing between characters. While both methods use a neural network recognition engine and a graph-algorithmic post-processor, their approaches to segmentation are quite different.

The experiments are carried on data from a large variety of writers which was collected in the cafeteria of an AT&T facility. We collected approximately 9000 one to five letter words, that we separated into 8000 for training and 1000 for testing. Short words (one to three letters) are random combinations of all letters and longer words legal English words. Another set of 600 English words of any length (from an 80,000 word dictionary) is also used for testing. Our best result is 11% word error rate on that last set, using lexical checking.

INSEG

We call our first method INput SEGmentation or $INSEG$, because of the key roll played by the segmentation preprocessor. The $INSEG$ technique

is inspired by earlier work in Optical Character Recognition [14]. A set of heuristics is used to determine likely segmentation points, or "tentative cuts". For the recognition of on-line hand-printed characters, pen-lifts are very natural tentative cuts. Our segmentation preprocessor uses both pen-lifts and spaces to cut the data stream and define a set of "tentative characters", usually a small superset of the valid characters (Fig. 1.8).

An interpretation graph is built with the recognition scores of all tentative characters (Fig. 1.8). The transition probabilities are simply set to the inverse of the preceding node fan-out and do not include information about the relative position of the tentative characters [31] nor the frequencies of letter successions in English [32]. The segmentation postprocessor searches the K-best paths in the graph. In the case of the figure, the best path is "UFT". But if the word is written by an English speaking person, we would rather select the second best path "LIFT" which is a valid English word. In practice, we use up to 20 best paths and try to match them with the closest English words using the "ispell" program [33]. The words thus obtained are reordered, using the interpretation graph. The valid English word with highest score is selected. Using as recognition engine the $TDNN$ described in Sect. 1.3.1, we obtain a word error-rate on the short-word data of 29% without lexicon and of 23% with lexicon.

We have found that better recognition rate is obtained by optimizing the overall system: recognizer and segmentor. This is done by retraining the $TDNN$ with both examples of valid characters and counter examples of characters. These counter examples correspond to mistakes of the segmentation preprocessor, such as incorrect splitting of characters or incorrect groupings of strokes. If the correct interpretation is not the best path in the interpretation graph, the segmentation postprocessor pursues its search until the n^{th} path gives the correct interpretation. All the incorrect paths up to the $(n-1)^{th}$ are used as counter examples to retrain the network. With such retraining, much improvement is obtained. The word error-rate without lexicon drops to 18% and with lexicon to 15%, on the short word test set.

Our best performance are obtained using retraining and by combining two different neural networks with a voting scheme: a $TDNN$ and a 2-dimensional convolutional network [34]. On the short word test set, we reach, without lexicon, a word error-rate of 13% and, with lexicon, 10%. On the long word test set, we reach an error-rate of 23% without lexicon and 11% with lexicon. It should be emphasized that the test sets are not cleaned from mislabeled and meaningless patterns which are introduced during data collection. The error rates we obtain are more than three times smaller than the error-rate obtained by a commercial recognizer tested on the same data.

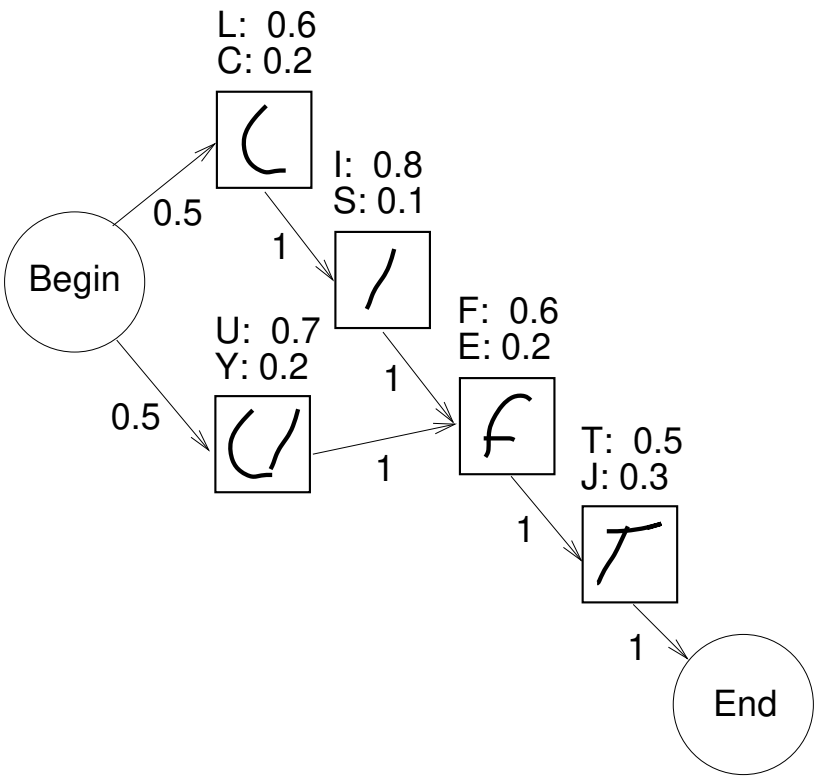

FIGURE 1.8. **Example of an interpretation graph produced by the** *INSEG* **system.**
The first two best scores are shown on top of each box. The transition weights indicated below the arrows are just the inverse of the preceding node fan-out. The best path "UFT" has a score of 0.10. The second best path "LIFT" has a score of only 0.07, but it will be preferred when lexical checking is used.

OUTSEG

The $INSEG$ system is well suited for the recognition of characters separated by pen-lifts. For connected handwriting, we prefer an alternative segmentation technique which does not presume of any segmentation point prior to recognition. We call this other technique OUTput SEGmentation, or $OUTSEG$.

The segmentation preprocessor presents to the recognizer a window of the input signal. The window is shifted in time to show the next tentative character. Therefore, the task of the segmentation preprocessor is trivial.

Scores for the different tentative characters are obtained as an ordered sequence of score vectors (Fig. 1.3). Both the location and the interpretation of the characters are determined by the segmentation postprocessor which handles therefore the segmentation *per se*. Hence the name $OUTSEG$.

How does the segmentation postprocessor proceed? If no character is present in the input window, all the network outputs are below a certain activation threshold. Conversely, if a character is present in the input window, there is a high activation value for the neuron corresponding to the correct interpretation. We avoid however the very delicate problem of tuning thresholds by filtering the sequence of output scores. We use a digital filter which implements a model of character duration and of duration of the spacing between characters. In Fig. 1.9, we show the sequence of network outputs and the interpretation path selected by our postprocessor. Several high scores are filtered in this path by the duration modeling.

We have used first a $TDNN$ trained on isolated characters only, similar to the one described in the previous section. We obtain with that network a very large error rate: more than 80% word error, with or without lexicon, on the short word database.

As in the case of the $INSEG$ method, we have found it important to retrain the $TDNN$ with entire words. With retraining, the network learns both to recognize characters and to detect transitions between characters by giving a low score to all outputs, when the window is not centered on a character. We use for that purpose the position invariant training technique proposed in reference [35, 36]. We have also tried to optimize the overall system, recognizer plus segmentor, by back-propagating errors through the duration model, with similar success. After retraining the word error rate on the short word data reduces to 21% without lexicon and 17% with lexicon. On the long word data we obtain 49% without lexicon and 21% with lexicon.

The error rates of $OUTSEG$ are about twice smaller than that of the commercial recognizer tested on the same data. They are however not quite as good as that of $INSEG$ on this task. This is understandable since uppercase printed letters are relatively easy to segment with a good heuristic segmentation preprocessor. In more than 95% of the words of our database, letters are separated by spaces and/or pen-lift. Although we do not have

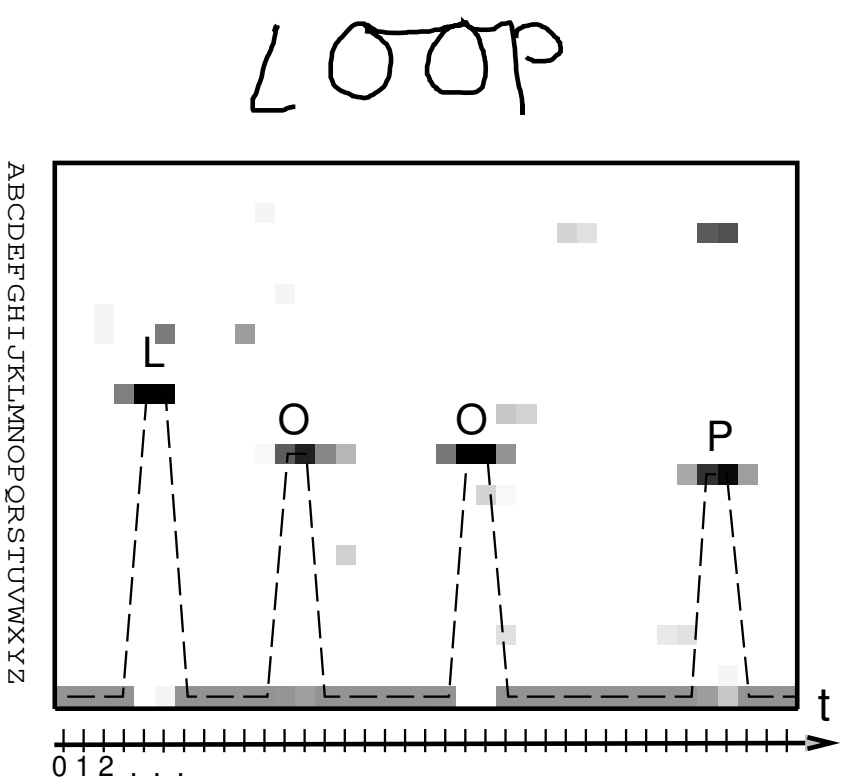

FIGURE 1.9. **Example of an output from the** *OUTSEG* **system.**
Each row in the matrix represents the score of one output unit for a different input window position in time. Each column represents the scores of all units for one particular position of the time window. The darker the boxes the more confident the unit is in its interpretation. The last row in the matrix is one minus the sum of the all other scores in a given column. It represents the probability that no character is detected. The dashed line indicates the best path found by the postprocessor, using a duration model of the characters and the character transitions.

direct evidence, the experiments performed in reference [37] are an indication that on cursive handwriting, for which good tentative cuts are difficult to determine, $OUTSEG$ would work better than $INSEG$.

Combination of $INSEG$ and $OUTSEG$

We would like our end system to be able to recognize printed characters, cursive script and also mixed styles. Currently existing systems are either good at recognizing printed letters or cursive. We propose a way of combining $INSEG$ and $OUTSEG$ into a single system which has the good features of both systems and permits mixed styles to be handled. We call this new method $INOUTSEG$.

To extend $INSEG$ to cursive recognition, one would need to introduce more tentative cuts than pen-lifts, perhaps minima of the speed of the pen. But, multiplying the number of tentative cuts is unnecessary for the recognition of hand-printed characters, and can only result in performance degradation. It is also unclear whether tentative cuts other than pen-lifts could be found. On the other hand, $OUTSEG$ can spot character locations without making use of heuristic tentative cuts, but makes more mistakes than $INSEG$ when characters are nicely separated by pen-lifts.

Our idea is to rely on the $INSEG$ preprocessor to determine tentative cuts in input space that are reported in the $OUTSEG$ output space (see Fig. 1.10). We then detect inconsistencies between the $INSEG$ and $OUTSEG$ segmentations. When $OUTSEG$ finds multiple characters between two consecutive tentative cuts, additional "virtual cuts" are introduced to separate them. The list of tentative cuts and virtual cuts thus obtained is subsequently used to build an interpretation graph, such as the one described in Fig. 1.8. This $INOUTSEG$ graph is filled with the recognition scores of $INSEG$ and $OUTSEG$ combined with a voting scheme.

In the example of Fig. 1.10, the $INSEG$ system uses a neural network trained to recognized uppercase and lowercase letters and the $OUTSEG$ system uses a network trained to recognize cursive words. The figure shows that $OUTSEG$ cannot recognize the uppercase letter "M" and makes an insertion error at the end of the word. On the other hand, $INSEG$ cannot separate the connected letters "ed". The combined system however recognizes correctly the word. Our system is still under development and experimental results will be reported elsewhere.

1.3.3 Signature Verification

The problem of signature verification is quite different from that of character recognition. For handwriting recognition, the correct interpretation must be discovered, not the identity of the writer. For signature verification, the correct interpretation (writer name) is given, but what needs to be discovered is the identity of the writer (good guy or forger?).

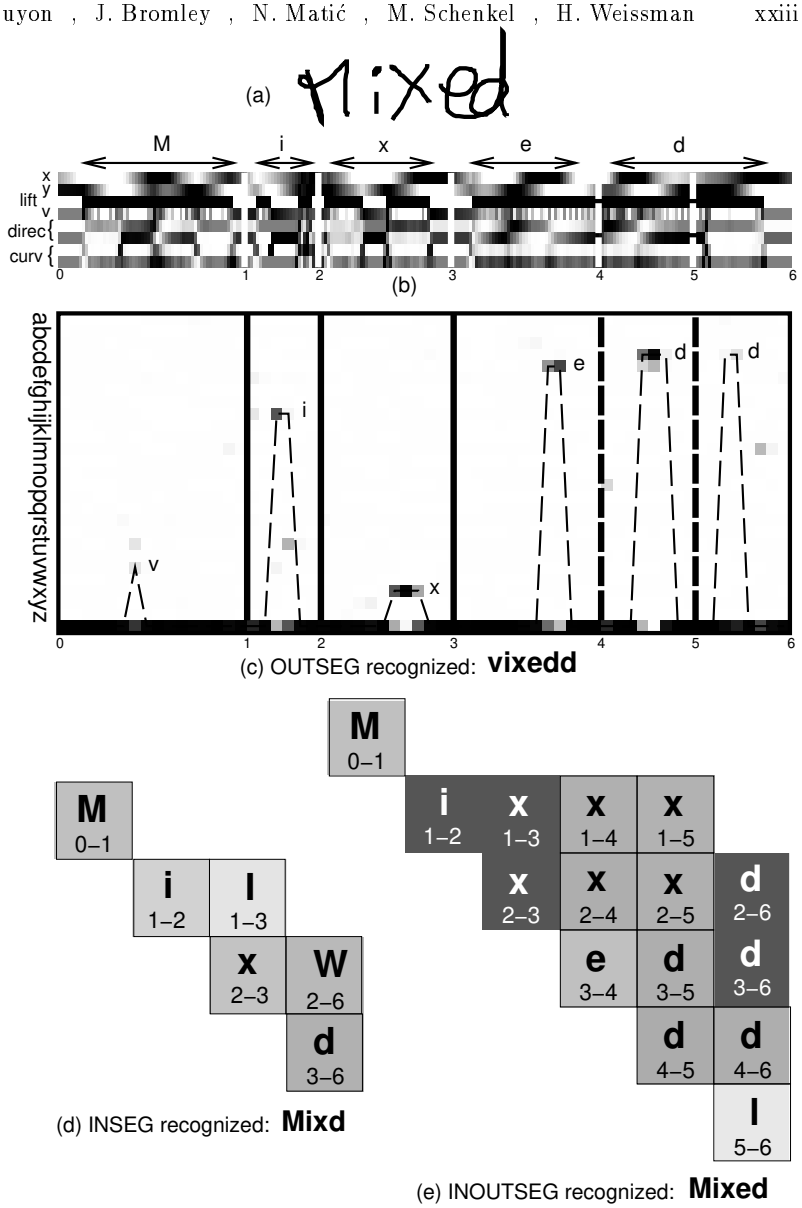

FIGURE 1.10. **Example of combination of input and output segmentation.**
(a) The original word. (b) The data as presented to the $OUTSEG$ network. (c) The sequence of output scores obtained by the $OUTSEG$ network. (d) The $INSEG$ interpretation graph. (e) The $INOUTSEG$ interpretation graph, including $INSEG$ tentative cuts, $OUTSEG$ virtual cuts and the scores of both $INSEG$ and $OUTSEG$ combined with a voting scheme.
In (b) and (c), the tentative cuts selected by $INSEG$ (coinciding with some pen lifts) are shown as bold vertical line; the virtual cuts provided by $OUTSEG$ are shown as vertical dashed lines. In (d) and (e) arrows are not represented; each box is associated with the tentative character delineated by two cuts whose numbers are at the bottom of the box; in each box, we show only the interpretation associated to the best score which intensity is indicated with the grey shading (the darker, the larger).

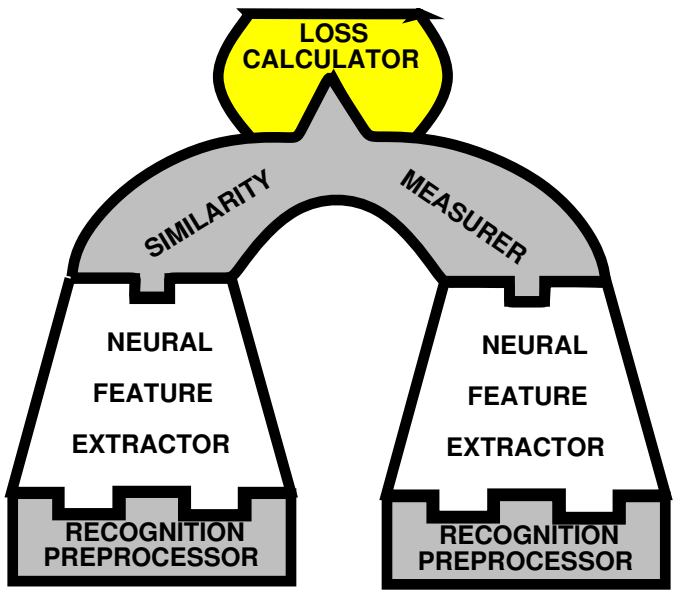

FIGURE 1.11. **Architecture of the siamese neural network.**

Most approaches to this problem rely on human expertise to device discriminatory similarity measures. In our approach, we train a "siamese" neural network (see sections 1.2.5 and 1.2.7 and Fig. 1.2.5) to learn a similarity measure from examples.

In reference [4], we address the signature verification task with the problem of reducing credit card fraud in mind. The credit card holder would give several samples of his own signatures at the bank and then an encrypted pattern of his signature would be stored on his credit card. At the retail site, his signature would be matched against the pattern stored on his card.

The experiments are carried out on data collected among the staff at AT&T and NCR and on data collected at Wright State University. A total of approximately 3300 signatures was obtained, including genuine signatures and forgeries. The amount of effort put into producing forgeries varied but reflect rather well the spectrum of real life data since a lot of forgers attempting to use somebody else's credit card produce effortless forgeries and may even use their own signature.

Training of the "siamese" network is performed by presenting pairs of signatures. We train the neural feature extractor to produce a representation of the signatures such that there is a small angle between two genuine

signatures from the same person and a large angle between two distinct signatures and between a signature and its forgery. The angle is measured by the "similarity measurer" (Fig. 1.11) by calculating the cosine between the output vectors of the two neural feature extractors (see Sect. 1.2.5). If both signatures are produced by the same person (genuine signatures), the desired cosine is one, otherwise it is negative or zero. The exact value of the desired value does not influence the performance significantly.

During utilization, the cosine similarity measure is replaced by a Gaussian similarity measure (Sect. 1.2.5). Credit card holders usually do not contribute to the data which is used to train the neural feature extractor. They provide only a few examples of their signatures which is encoded as the representation of the output of the neural feature extractor. We build a Gaussian model of these patterns, whose mean becomes the reference prototype for the user's signature and whose covariance matrix is used to determine the width and orientation of the Gaussian window.

The siamese network fits quite nicely the problem. After training, a model signature can be encoded in the representation of the neural feature extractor. It is therefore both compressed significantly and encripted, which is suitable for storage on a credit card magnetic strip. During verification, the signature to be checked is processed at the retail place by an identical neural feature extractor and matched to its model. The model can be constantly updated with examples of successfully verified signatures.

In reference [4], several network architectures are tried. The networks are trained with 7700 signature pairs, 50% being pairs of genuine signatures from the same person, 40% genuine-forgery pairs and 10% pairs of different signatures. The networks are then tested on a separate set of signatures from different people comprising about 500 genuine signatures and 400 forgeries. For each person, a model signature in the neural feature extractor representation is built from six signatures. The model is tested with six other genuine signatures and forgeries.

By varying a rejection threshold on the output of the system, one can monitor the tradeoff between accepting too many forgeries and rejecting too many genuine signatures. Our best network permits detecting 80% of the forgeries while rejecting only 5% of genuine signatures.

1.4 Conclusion

We presented a neural network approach to solve several problems in online handwriting recognition. Our design choices make a consistent use of the sequential nature of the data, both in the preprocessing and the neural network architecture (a $TDNN$). The initial success of our writer independent isolated character recognizer motivated us to address tasks of greater difficulty. We reported here applications to signature verification and the recognition of hand-printed words with no spacing constraint between char-

acters. On this last task, our system outperforms a widely distributed commercial recognizer tested on the same data. While we are still improving our first system, we are developing the next generation which will try to recognize mixed styles including hand-printed letters and cursive.

Acknowledgments: The work described in this overview paper was conducted in Larry Jackel's department at Bell Labs and is the result of the joint effort of many researchers including Paul Albrecht, Yoshua Bengio, Jim Bentz, Bernhard Boser, Léon Bottou, John Denker, Donnie Henderson, Wayne Hubbard, Yann Le Cun, Annick Leroy, Cliff Moore, Craig Nohl, Howard Page, Ed Pednault, Doug Riecken, Eduard Säckinger, Roopack Shah, Vladimir Vapnik and Anne Weissbuch. Collaborations with our colleagues at ETH Zürich, and our colleagues at Bell Labs doing research on Optical Character Recognition and Multimedia Communications are greatfully acknowledged. The simulations were performed on the neural network simulator *SN* of Neuristique Inc. written by Léon Bottou and Yann Le Cun and on the network simulator $\nu - lisp$ written by Bernhard Boser.

1.5 References

[1] 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.

[2] 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.

[3] N. Matić, I. Guyon, L. Bottou, J. Denker, and V. Vapnik. Computer-aided cleaning of large databases for character recognition. In *Proceeding of ICPR*, Amsterdam, 1992. IEEE.

[4] J. Bromley, J. Bentz, L. Bottou, I. Guyon, L. Jackel, Y. Le Cun, C. Moore, E. Sackinger, and R. Shah. Signature verification with a siamese time delay neural network. In I. Guyon and P.S.P. Wang, editors, *Applications of Neural Networks to Pattern Recognition*, Singapore, 1993. World Scientific (to appear).

[5] N. Matić, I. Guyon, J. Denker, and V. Vapnik. Writer adaptation for on-line handwritten character recognition. In *Submitted to ICDAR'93*, Tokyo, 1993.

[6] M. Schenkel, H. Weissman, I. Guyon, C. Nohl, and D. Henderson. Recognition-based segmentation of on-line hand-printed words. In *Advances in Neural Information Processing Systems 5 (NIPS 92)*, San Mateo CA, December 1992. Morgan Kaufmann (to appear).

[7] 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.

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

[9] P. Gallinari. A neural net classifier combining unsupervised and supervised learning. In *Proceedings of the International Neural Network Conference*, volume 1, pages 375–378, Paris, July 1990. IEEE.

[10] 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.

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

[12] Y. Le Cun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Back-propagation applied to handwritten zipcode recognition. *Neural Computation*, 1(4), January 1990.

[13] O. Matan, C. J. C. Burges, Y. Le Cun, and J. Denker. Multi-digit recognition using a Space Dispacement Neural Network. In J. E. Moody et al., editor, *Advances in Neural Information Processing Systems 4*, Denver, 1992. Morgan Kaufmann.

[14] C. J. C. Burges, O. Matan, Y. Le Cun, D. Denker, L. D. Jackel, C. E. Stenard, C. R. Nohl, and J. I. Ben. Shortest path segmentation: A method for training neural networks to recognize character strings. In *IJCNN'92*, volume 3, Baltimore, 1992. IEEE.

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

[16] 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.

[17] 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.

[18] B. Boser, I. Guyon, and V. Vapnik. A training algorithm for optimal margin classifiers. In *Fifth Annual Workshop on Computational Learning Theory*, pages 144 – 152, Pittsburgh, July 1992. ACM.

[19] L. R. Rabiner. A tutorial on Hidden Markov Models and selected applications in speech recognition. In *Proceedings of the IEEE*, volume 77-2, February 1989.

[20] R. P. Lippmann. Review of neural networks for speech recognition. *Neural Computation*, 1(1):1–38, 1989.

[21] J. S. Bridle. Alpha-nets: A recurrent 'neural' network architecture with a hidden markov model interpretation. *Speech Communication*, 9(1):83–92, February 1990.

[22] H. Bourlard and C. Wellekens. Links between markov models and multilayer perceptron. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12-12:1167–1178, December 1990.

[23] E. Levin, R. Pieraccini, and E. Bocchieri. Time-warping network: A neural approach to hidden markov model based speech recognition. In I. Guyon and P.S.P. Wang, editors, *Applications of Neural Networks to Pattern Recognition*, Singapore, 1993. World Scientific (to appear).

[24] E. Sackinger and J. Bromley. Neural-network and k-nearest neighbour classifiers. Technical Report TM 51323-919819-01, AT&T Bell Laboratories, 1991.

[25] Baldi. P. and Y. Chauvin. Neural networks for fingerprint recognition. *To appear in Neural computation*, 1992.

[26] I. Guyon, V. Vapnik, B. Boser, L. Bottou, and S.A. Solla. Structural risk minimization for character recognition. In J. E. Moody et al., editor, *Advances in Neural Information Processing Systems 4*, Denver, 1992. Morgan Kaufmann.

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

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

[29] L. Bottou. *Une approche théorique de l'apprentissage connexioniste; Application à la reconnaissance de la parole.* PhD thesis, University of Paris XI, 1991.

[30] H. Drucker, R. Schapire, and P. Simard. Boosting performance in neural networks. In I. Guyon and P.S.P. Wang, editors, *Applications of Neural Networks to Pattern Recognition*, Singapore, 1993. World Scientific (to appear).

[31] E. Pednault. A hidden markov model for resolving segmentation and interpretation ambiguities in unconstrained handwriting recognition. Technical Report TM 11352-920929-01, AT&T Bell Laboratories, 1992.

[32] T. Fujisaki, H. S. M. Beigi, C. C. Tappert, M. Ulkeson, and C. G. Wolf. On-line recognition of unconstrined handwriting: a stroke-based system and its evaluation. *Preprint*, 1992.

[33] R. E. Gorin et al. $UNIX^{TM}$ man-page for ispell, version 3.0.06 (beta). 09/17/91.

[34] A. Weissbuch and Y. Le Cun. Private communication. 1992.

[35] J. Keeler, D. E. Rumelhart, and W-K. Leow. Integrated segmentation and recognition of hand-printed numerals. In R. Lippmann et al., editor, *Advances in Neural Information Processing Systems 3*, pages 557–563, Denver, 1991. Morgan Kaufmann.

[36] J. Keeler and D. E. Rumelhart. A self-organizing integrated segmentation and recognition neural net. In J. E. Moody et al., editor, *Advances in Neural Information Processing Systems 4*, pages 496–503, Denver, 1992. Morgan Kaufmann.

[37] D. Rumelhart and et al. Integrated segmentation and recognition of cursive handwriting. In *Third NEC symposium Computational Learning and Cognition*, Princeton, New jersey, 1992 (to appear).