Interactive Systems Labs

Carnegie Mellon University
Pittsburgh, PA, USA

University of Karlsruhe
Germany

Model-Based Recovery of Dynamic Information from Static Handwriting

DIPLOMA THESIS

Ralph Groß

Supervisors:
Prof. Dr. Alex Waibel
Dipl.-Inform. Stefan Manke

January 1998

Hiermit erkläre ich, daß ich diese Diplomarbeit selbständig verfaßt und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Pittsburgh, den 31.1.1998

[illegible]

Ralph Groß

Zusammenfassung

Im Bereich der Handschriftenerkennung kann zwischen On-line und Off-line Systemen unterschieden werden. On-line Systeme erhalten ihre Eingabe in Form einer zeitlich geordneten Sequenz von x-y Koordinaten der Stifttrajektorie, während Off-line Systeme auf einer statischen Repräsentation, üblicherweise einer Bitmap, arbeiten. Die Forschungsbemühungen der vergangenen Jahre haben gezeigt, daß On-line Systeme deutlich bessere Erkennungsraten erzielen als Off-line Systeme. Die Ursache hierfür ist vor allem das Vorhandensein der zeitlichen Schreibinformation bei On-line Erkennern. Die vorliegende Arbeit präsentiert ein modell basiertes System zur Rückgewinnung der zeitlichen Schreibinformation aus einer Off-line Repräsentation von handgeschriebenen Worten. Auf die daraus resultierenden Daten kann ein On-line Handschriftenerkenner angewendet werden.

Das vorgestellte System extrahiert während der Vorverarbeitung eine abstrakte Repräsentation des Eingabewortes. Mit Hilfe dieses sogenannten Segmentgraphen kann das Grundproblem, als Suche nach einem besten Pfad durch den Graphen formuliert werden. Die Hauptaufgabe dabei ist die Entwicklung einer Strategie zur Festlegung der Ausgangskante aus einem Knoten des Segmentgraphen bei gegebener Eingangskante. Die in der vorgelegten Arbeit realisierte Lösung benutzt ein Knotenmodell, das in Form eines vorwärtsgerichteten neuronalen Netzes implementiert ist. Basierend auf der im Rahmen dieser Arbeit entwickelten Merkmalsrepräsentation erreichen die mit Backpropagation trainierten Netze eine Erkennungsrate von 93.8%. Unter Verwendung der Ausgaben des Knotenmodells errechnet eine nachgeschaltete Suchkomponente die wahrscheinlichsten Pfade durch den Segmentgraphen. Aus der Liste der Gesamtwortpfade wird abschließend unter Benutzung eines On-line Handschriftenerkenners der beste Pfad ausgewählt.

Um die Leistung des vorgestellten Systemes zu messen, wird ein On-line Handschriftenerkenner auf die errechneten Daten angewandt. Dabei wurden Erkennungsraten von 84%, 77.3% und 67.8% unter Verwendung von Wörterbüchern mit 120, 1 000 und 5 000 Worten erzielt.

Abstract

Research efforts conducted in the field of handwriting recognition resulted in a number of systems for on-line and off-line data. It is a known fact that on-line systems perform significantly better than off-line systems. This can be attributed to the lack of dynamic writing information in the case of off-line data. This thesis presents a model-based system for the extraction of dynamic writing information from static representations of handwritten words. The resulting data can be used as input to an on-line handwriting recognizer.

The proposed system generates an abstract representation of the input word during preprocessing. Using this so-called segment graph, the task is transformed into the search for an optimal path through the graph data structure. The main problem within this task is the determination of the outgoing edge of a node in the segment graph given the input edge. The solution proposed here uses a node model, implemented by a feedforward neural network. Together with the feature representation developed in this thesis, the neural network achieves a classification accuracy of 93.8%. With the help of the node model a subsequent search algorithm calculates the best paths through the segment graph. The optimal path is chosen using an on-line handwriting recognizer.

In order to evaluate the presented approach, an on-line handwriting recognizer is applied to the data produced by the system. The recognizer achieves accuracies of 84%, 77.3% and 67.8% with dictionary sizes of 120, 1 000 and 5 000 words.

Acknowledgments

This work could not have been done without the help from a number of people. First I would like to thank Prof. Alex Waibel for the opportunity to conduct my research at our lab at Carnegie Mellon University, Pittsburgh. I would also like to thank: Stefan Manke for his advice and guidance during this project. Klaus Ries for the countless discussions we had on many related subjects. They helped me develop a deeper understanding of the problems involved in this project. Wolfgang Hürst for the productive cooperation we had during his stay at Carnegie Mellon. Matthias Denecke for technical advice and proofreading of this thesis. My fiancee Aleksandra for all the support she gave me especially during the last months of this project. Last but not least, my parents who supported me in every possible way during all the years of my studies.

Contents

Chapter 1

Introduction

1.1 Motivation

Despite the large number of computers in business offices and private homes and the omnipresence of telephones, a high percentage of our daily communication is still done by exchanging machine-printed and handwritten notes on paper, such as letters, letter envelopes, faxes or bank checks. Distribution and further processing of these messages can be accomplished more efficiently by computers. However, converting the information captured on paper into a computer-accessible format is difficult. Researchers in the field of OCR (**O**ptical **C**haracter **R**ecognition) have been working for decades on the task of recognizing machine-printed texts. The software resulting from these efforts achieves high recognition accuracies when used under good conditions. The recognition of handwritten text on the other hand is still far away from this performance.

The increasing computer power available for mobile systems such as laptops or palmtops raises the interest in on-line handwriting recognition which deals with handwritten input made on pressure sensitive displays or digitizer tablets. This allows the design of new interfaces offering mouse, keyboard and/or pen as input devices. The recognition rates of systems processing on-line data are noticeably higher than those for off-line data. This can be attributed to the lack of *dynamic writing information* in the case of off-line data.
Two questions arise out of this observation. First, is it possible to retrieve the lost temporal information? Second, how can additional dynamic information be used to perform off-line recognition? This thesis aims at answering both of these questions by proposing a system for recovering dynamic writing information from static images of handwritten words. The resulting data is used as input to an on-line recognizer.

1.2 Problem Definition

The problems associated with the above outlined task become obvious if one tries to imagine how a virtual pen traces along a bitmap such as the one depicted in figure 1.1.

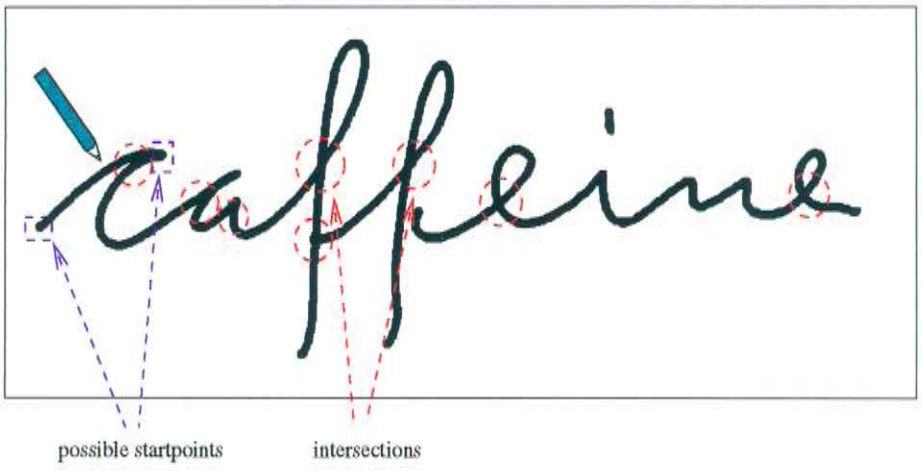

Figure 1.1: Example word with possible starting points and intersections highlighted

The first difficulty concerns identifying the starting point of the trace. The fact that English words are written from left to right and top to bottom suggests to search for the best starting point at the upper left end of the input image. As will be shown later on this heuristic does not cover all words, hence a more sophisticated algorithm has to be applied.

Following the edge out of the starting point, the next problem arises as the pen encounters an intersection. A decision strategy has to determine the edge on which the pen should leave the intersection again. This decision depends on the type of the intersection, the local context and the history of previous contacts with this node.

The overall goal is to find a consistent trace for the *whole* word. Any kind of decision strategy for intersections is likely to make mistakes. It is therefore necessary to employ a global strategy which combines the local decisions in intersections and optimizes for maximal coverage of the underlying word.

Finally, the input words as shown in figure 1.1 are given as binary images or bitmaps. This simple format encodes the image as a 2D raster grid. In order to be able to apply higher level algorithms, the input data has to be transformed into a more suitable data structure. The new representation should preserve as much information as possible while still being efficient to compute.

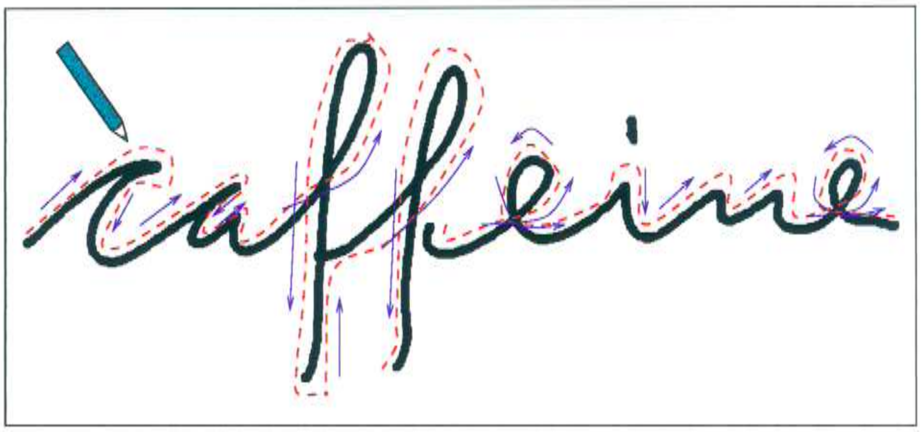

Figure 1.2: Example word with its trace

1.3 Contribution of this Work

This thesis provides solutions for all of the problems described in the last section. In particular new methods are proposed for the decision on local level, the identification of the correct starting point and the global search strategy. Furthermore, a new input representation is developed. The different contributions can be detailed as follows:

- **Segment Graph**
  The preprocessing stage extracts a new data structure, the so-called segment graph, from the image of the handwritten word. This representation preserves a maximum of information and is still efficient to compute.

- **Integrated Starting Point Search**
  The search for the best starting point is integrated into the overall search for the best path. Every visible endpoint is considered as potential starting point, therefore avoiding a separate heuristic.

- **Node Models**
  The decisions on local level are based on a statistical model for the nodes of the segment graph. These models are implemented by feedforward neural networks. Using a suitable feature representation developed as part of this thesis, the neural network achieves a classification accuracy of 93.8%.

- **Heuristic Word Search**
  The results of the neural networks are combined with a new heuristic search procedure. With the help of efficient pruning rules, this algorithm generates a list of path hypotheses. The optimal path is selected using a word model implemented by an on-line handwriting recognizer.

The result of all these efforts is a complete trace through the whole word as depicted in figure 1.2.

1.4 Organization of the Thesis

Following the introduction of some basic concepts in **chapter 2**, the related work from literature is reviewed in **chapter 3**. **Chapter 4** gives an overview of the whole approach. The different parts of the system are then described in greater detail: preprocessing (**chapter 5**), node models (**chapter 6**) and search (**chapter 7**). The results of the evaluation of the complete system are presented and discussed in **chapter 8**. The thesis concludes with a summary and an outline of possible future work in **chapter 9**.

Chapter 2

Background

This chapter reviews some basic techniques and notations used within this thesis. After a more detailed introduction to handwriting recognition in general, the on-line handwriting recognizer **NPen^{++}** is described. The chapter continues with a list of properties of cursive script, which are used in many parts of this work. As stated in the introduction, the proposed system uses neural networks to model the nodes in the segment graph. This chapter reviews the basic concepts of artificial neural networks. After introducing perceptrons as building blocks for the type of neural networks relevant to this work, multilayer networks and the backpropagation training algorithm are presented. The overview concludes with some extensions to standard backpropagation and remarks on the representational power of feedforward networks[1].

2.1 Handwriting Recognition

2.1.1 Handwriting Recognition Taxonomy

Research in handwriting recognition can be classified into different groups. The following sections highlight the major points which define different problem areas within the field of handwriting recognition (see [Sen94], [Kas95]). This taxonomy only includes systems based on the Latin character set.

On-line versus Off-line

The first distinction to be made is concerned with the type of input data that the recognition system deals with.
On-line recognition systems receive their input data from touch-sensitive devices such as graphic tablets or touch-screens. The data consists of the time ordered sequence of x-y coordinates, sometimes augmented by related parameters such as pressure, velocity or acceleration. As an one-dimensional stream of information, handwriting data

[1]The part on neural networks was written using [Mit97], [HKP91], [Bis96] and [Zel94].

is similar to speech data. Therefore, techniques developed in the context of speech recognition systems have been successfully applied to on-line handwriting recognition. This includes Hidden Markov Models [BNNB94], time-delay neural networks [MFW95], [SGH94] and even speech recognizers [SMSC94]. Tappert et al. [TSW90] published an extensive survey article on on-line handwriting recognition.
On the other hand *off-line* recognition systems get an image of the handwriting obtained from a scanner or a video camera as input. The recognition process and the production of the data do not have to take place at the same time. The type of input makes off-line handwriting recognition a subset of *Optical Character Recognition* (OCR), although OCR systems usually only deal with machine printed documents. Refer to [SLN$^+$93] for an overview of recent advances in off-line handwriting recognition.

Writer Independence

Different people have significantly different handwritings and it is not likely that two words, written by two writers, look the same. Therefore, recognizers have to handle a large amount of variability. There are different ways of dealing with this variability. One possible solution is to require sufficient amounts of data from each writer in order to train the system for that specific user. The result is a *writer dependent* system, which usually shows the best recognition performance. If the amount of data available from a user is limited or if the application does not allow a training period, a *writer independent* system has to be used. These recognizers are usually trained with data from many different writers in order to capture as much variation as possible. The recognition performance of a writer independent system is usually significantly lower than the performance of a writer dependent system (when used on data provided by the user on which the writer dependent system was trained on).

Writing Style

Besides the variability found between the handwriting of different writers, there are also significantly different ways how a single writer can write a word. Figure 2.1 shows the word "caffeine" written in 4 writing styles: boxed, printed, cursive and a mixture of printed and cursive.
The task of recognizing cursive words turns out to be the most challenging one for on-line and off-line recognizers. The major difficulty lies in the segmentation of a cursive word into its letters. This is similar to the problem of segmenting continuous speech into its component words. In both speech recognition and handwriting recognition, it has been found that systems performing segmentation *during* the recognition process usually perform better than systems that try to do the segmentation within a preprocessing step.

Dictionary Size

Virtually all existing handwriting recognition systems use a language model, usually in the form of a dictionary to restrict the search space. The recognizers are not able

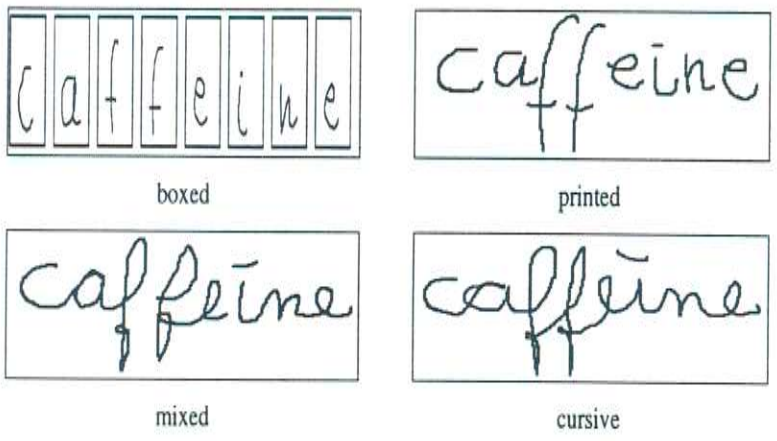

Figure 2.1: Examples for the different writing styles boxed, printed, mixed and cursive

to recognize arbitrary character sequences, but only words out of this fixed set. As the recognition task becomes harder with larger dictionaries, the performance of a given system always has to be seen in relation to the size of the dictionary in use. The literature describes systems for both on-line and off-line recognition, which can handle dictionary sizes of up to 100 000 words (see [MFW96], [Sen94] as examples for on-line and off-line systems, respectively).

2.1.2 Dynamic Writing Information

Due to the lack of standardized test sets, it is hard if not impossible to compare the performance of different recognizers. This applies in particular to the comparison of on-line with off-line recognition systems. Experience shows that on-line recognizers usually achieve higher recognition accuracies than off-line recognizers. On-line data provides the recognition system with an ordering of the input coordinates which makes it easy to distinguish overlapping stroke segments or strokes[2]. This advantage seems to outweigh additional variability only visible in on-line data (e.g. in writing speed, direction, acceleration or stroke order).
The results of a number of experiments suggest that the higher performance of on-line systems can mostly be credited to the availability of temporal information. Mandler et al. [MOD85] have shown that the conversion of on-line data into equivalent off-line data leads to a degradation in recognition performance. In older experiments by Fujimoto et al. [FKH$^+$76] the opposite effect is noticed. They manually estimated directional information from static images and used that data as input to an on-line recognizer. The recognition rates obtained through that procedure are higher than those resulting from the use of an off-line recognizer on the original data. Finally, psychology experiments conducted by Freyd [Fre83] revealed that humans can recognize static characters faster if they are distorted in a manner consistent with the drawing

[2]The usage of the term *stroke* is not consistent in the handwriting recognition literature. Throughout this thesis it denotes the sequence of coordinate points between a pen-down and a pen-up.

method than if they are distorted in an inconsistent manner. This suggests that even for humans the use of dynamic information enhances the perception of static forms (citations following [DR95]).

2.2 Properties of Cursive Script

Some basic characteristics of cursive words are valuable for the task of this thesis (see [DR95]). These "rules" also apply to printed characters or words, although they are not as significant as they are for cursive script. The scope of these considerations is again limited to words formed out of the Latin character set.

- *Writing Direction*
  The *overall* writing direction is from left to right. Strokes beginning in one zone (upper, middle or lower) usually end in or below that zone.
- *Energy Minimization*
  The stroke order is *influenced* by the attempt to minimize the energy required to produce them. This can be seen in minimal changes of curvature and smooth continuations at junctions.
- *Local Continuity*
  All parts of an uninterrupted isolated stroke segment follow the same direction.

As emphasized throughout this listing, the properties are not valid for every word. This for example can be seen in the so-called *delayed stroke problem* (see [Hue97]). It denotes strokes such as i-dots or t-crosses, whose placement in the time ordered sequence is not consistent among writers. They may occur right after the correspon-ding character or at the end of the input after the whole word was written. The principle of energy minimization does not seem to apply in this case. Nevertheless, these properties give important leads on how to design a system for the retrieval of temporal writing information.

The literature also contains work on the mathematical modeling of handwriting and the handwriting generation system (see [Hol81], [ST94], [Pla89], [Pla95]). As will be shown in chapter 4, the system presented here uses a different model.

2.3 On-line Handwriting Recognizer NPen^{++}

NPen^{++} is a writer independent, large vocabulary on-line handwriting recognizer ([MFW95], [MFW96], [Gro97] for a detailed description). The system combines a robust preprocessing with a Multi-State Time Delay Neural Network (MS-TDNN) for the recognition of single words. Figure 2.2 gives an overview of the system.

During preprocessing, the time-ordered sequence of x-y coordinates is normalized to reduce meaningless variability. The procedures applied during this stage include resampling, smoothing and a baseline normalization. For each data point of the normalized coordinate sequence, a 17-dimensional feature vector is calculated. The idea is to capture low-level topological information and leave the extraction of high-level features to the connectionist recognizer. The most important features are direction and curvature of the curve described by the pen motion and the so-called *context bitmaps* (first proposed in [MFW94]). The context bitmaps provide a coarse-grained view of the vicinity of each data point. They are able to model temporal long range and spatial short range dependencies as occurring in pen trajectories.

The MS-TDNN architecture was originally proposed for continuous speech recognition tasks [HW92]. It combines the high accuracy pattern recognition capabilities of a Time Delay Neural Network [WHH$^+$89] with a non-linear time alignment algorithm (dynamic time warping) in order to find an optimal alignment between stroke and characters in handwritten words. The characters are modeled by a three state Hidden Markov Model (HMM) (for an introduction to HMMs [Rab89]). In order to guarantee a fast search even for large dictionaries, the MS-TDNN architecture is extended to a tree-based TDNN. This allows real-time performance with dictionary sizes up to 100 000 words [MFW96]. **NPen^{++}** achieves a word accuracy of 92% using a 20 000 word dictionary.

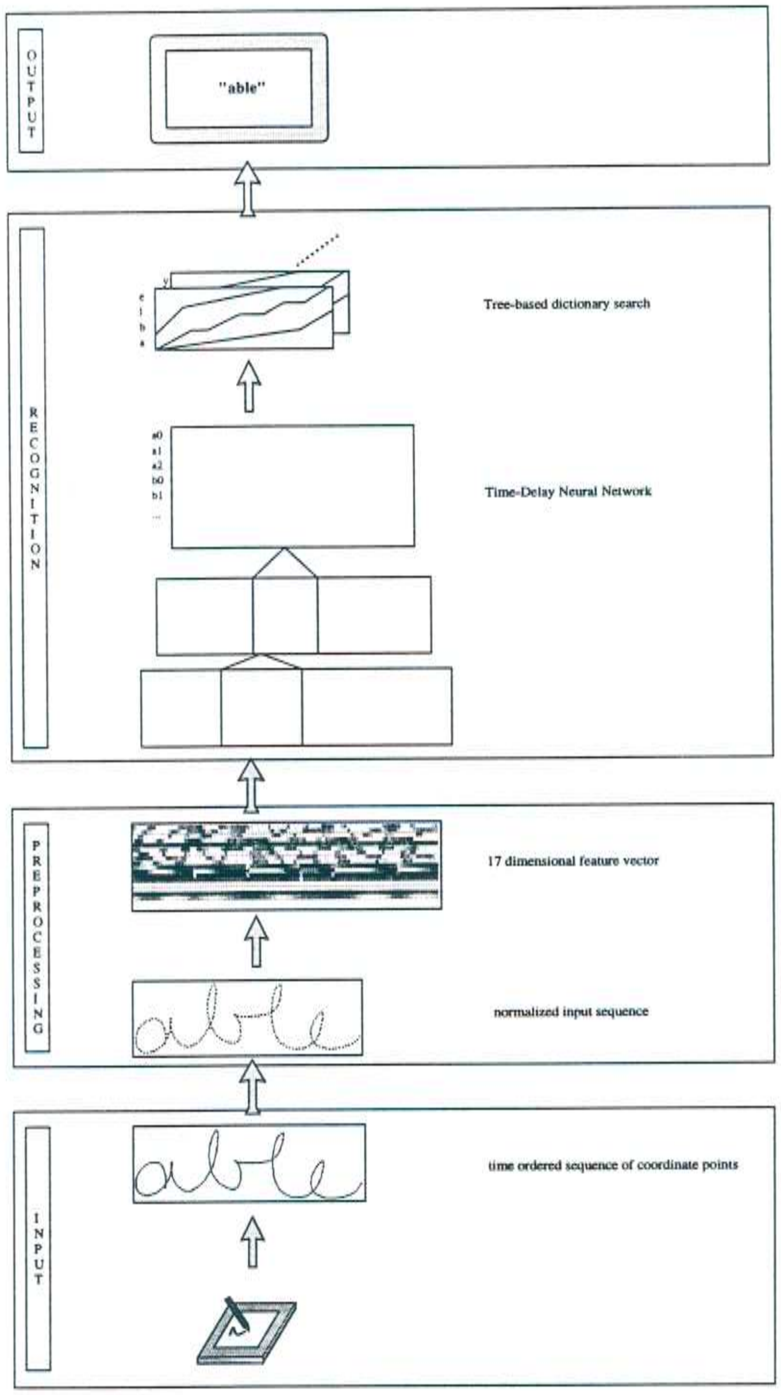
OUTPUT
"able"
RECOGNITION
Tree-based dictionary search
Time-Delay Neural Network
a0
a1
a2
b0
b1
PREPROCESSING
17 dimensional feature vector
normalized input sequence
INPUT
time ordered sequence of coordinate points

Figure 2.2: Overview of the **NPen**$^{++}$- system (from [Hue97])

2.4 Neural Networks

Neural Networks are highly parallel information processing systems. They consist of a large set of densely interconnected units (*cells* or *neurons*), where each unit produces a single real-valued output out of a number of real-valued inputs. The result of this calculation serves either as input to one or more other cells or is part of the system output.
The study of Artificial Neural Networks (ANN) started in the context of Biology, where artificial neurons are used to model the basic units in biological information processing systems. Their capability of *learning* how to solve certain tasks from training examples makes them interesting to computer science.

2.4.1 Perceptrons

Structure

One basic unit used in certain ANNs is the *perceptron* as shown in figure 2.3.

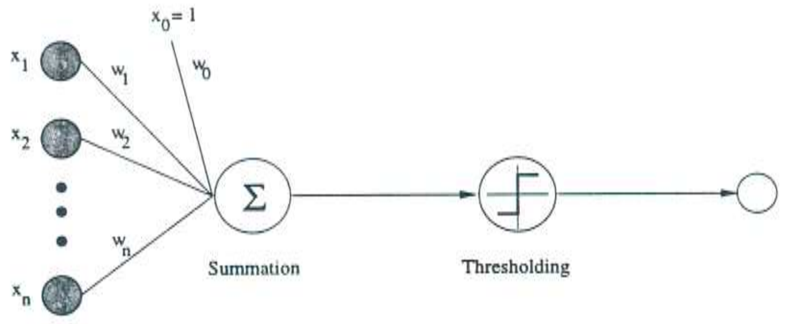

Figure 2.3: Perceptron (from [Mit97])

The perceptron calculates a linear combination of its real-valued inputs and produces the output 1 if the result of the combination exceeds a certain threshold and -1 otherwise. For given input values $x_1, \ldots, x_n$, the output $o(x_1, \ldots, x_n)$ is computed as:

$$o(x_1, \ldots, x_n) = \begin{cases} 1 & \text{if } \; w_0 + w_1x_1 + w_2x_2 + \ldots + w_nx_n > 0 \\ -1 & \text{otherwise,} \end{cases}$$

where the $w_0, \ldots, w_n$ are so-called *weights* which determine the contribution of each x_i to the output. The above stated condition is equivalent to

$$w_1x_1 + w_2x_2 + \ldots + w_nx_n > -w_0,$$

$-w_0$ being the threshold which the weighted sum of input values has to surpass. In order to simplify the notation, an additional input x_0 with value 1 is added. Therefore, the condition can be written as

$$\sum_{i=0}^{n} w_i x_i > 0 \tag{2.1}$$

Given a set of input values and corresponding output values the learning task consists of finding a weight vector which enables the perceptron to output the correct values for the given input.

Sigmoid Unit

It is important for the backpropagation training algorithm, which will be introduced later in this chapter, that the transfer function of a neuron is differentiable. This is not the case for the perceptron as the weighted sum of the inputs is thresholded using a discontinuous step function. One possible alternative is a *sigmoid* unit as show in figure 2.4.

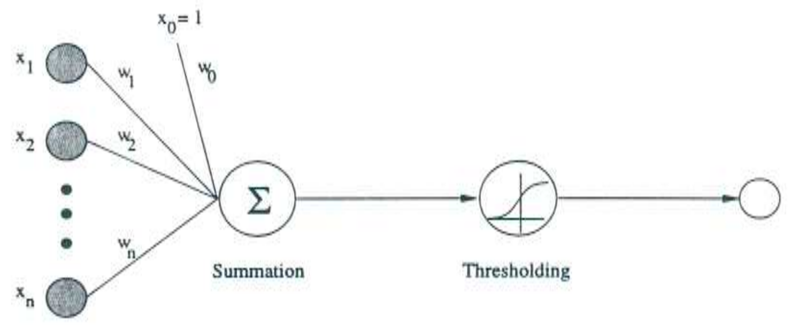

Figure 2.4: Sigmoid unit (from [Mit97])

This unit only differs in the transfer function which is applied to the weighted sum of inputs. For inputs y, the output of the *sigmoid* function is computed as

$$\sigma(y) = \frac{1}{1 + \mathrm{e}^{-y}}$$

Figure 2.5 compares the sigmoid and the *tanh* function, the latter being sometimes used in place of the sigmoid function.

2.4.2 Multilayer Networks

The simple neurons, as described in the last section, are combined into networks in order to support the solving of more complicated problems. Figure 2.6 shows an example of a typical structure. This kind of networks using perceptrons as building blocks are sometimes called *multi layer perceptrons*.
The network is organized into three layers. Each circle in the diagram stands for the output of a single network unit and the lines entering the node are the inputs (lines

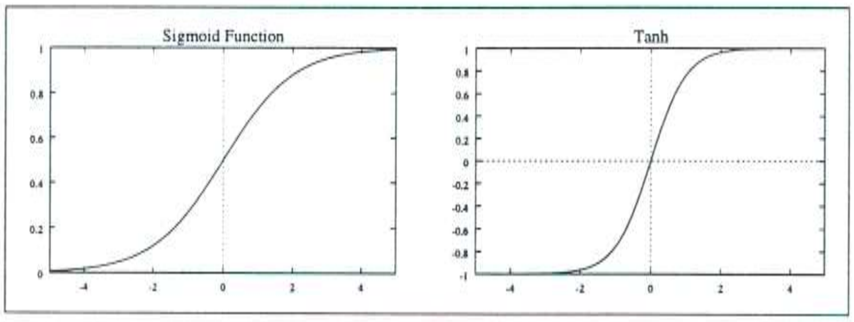

Figure 2.5: Activation functions sigmoid and tanh

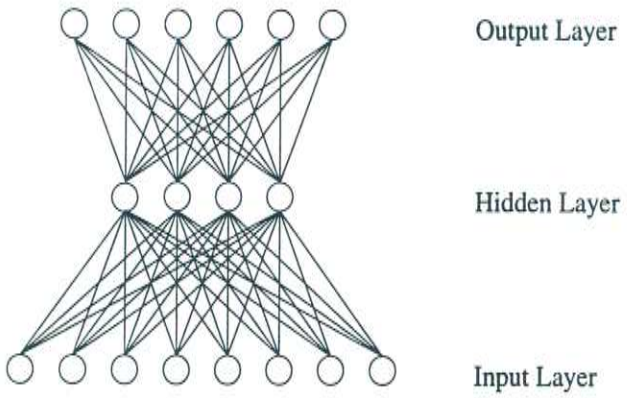

Figure 2.6: Example of a neural network

into the nodes of the input layer are omitted). As the nodes in the second layer are not visible from the outside, they are called *hidden units*, their layer *hidden layer*. The information flow within the network is directed from the input layer to the output layer. This class of networks is called *feedforward* networks. Notice furthermore, that the graph is acyclic.
The question left to discuss is how to train such a network.

2.4.3 Learning in Neural Networks

The optimal choice for network structure and corresponding training algorithm depends heavily on the type of the learning problem the neural network is supposed to solve. Generally, it is possible to distinguish between two types:

- *supervised learning*
  For each sample in the training set an output pattern is given. The task for the neural network is to learn to produce the correct output for a given input pattern of the training set. Supervised learning includes the special case of *reinforcement learning*. The only feedback given to the network is whether the

output is correct or not, not what the output pattern should be.

- *unsupervised learning*
  In this case only input patterns are available to the network. The network is supposed to find similarities within the given data and to categorize the input accordingly.

The reminder of the chapter will focus on the task of supervised learning.

Gradient Descent

There are a lot of ways how a neural network can learn how to solve a given problem. In most cases, the method of choice is to initially define a net structure with a fixed number of neurons and fixed connections between them and to alter the weights associated with the connections.
In the given context, the objective of the training procedure is to minimize the difference between the net output and the target output by changing the weight matrix $W = \{w_{ik}\}$ of the network. As the network structure stays constant throughout the training, the difference or error E the network makes is a function of the weight matrix $E(W)$.
A good heuristics for the training is to alter the weights so that the error function is changed in the direction of the steepest descend, i.e. in the direction of the negative gradient. The algorithm implementing this idea is called *gradient descend.* Due to the fact that gradient descent only uses local information, it is only guaranteed to find a local minimum, which might be suboptimal. Practice has shown that in the case of neural network training gradient descent produces excellent results in many cases.

Backpropagation Training Algorithm

The backpropagation algorithm learns the weights in a neural network by applying gradient descent. The algorithm proceeds as follows [Mit97]:

```
• Create a feedforward network of desired structure
• Initialize weights with small random values
• Until the termination condition is met, do
        For each sample ⟨$\vec{x}, \vec{t}$⟩ of the training set, do
        Propagate input forward through the network
          1. Give input $\vec{x}$ to the network and calculate the output of each unit
        Propagate the errors backward through the network
          2. For each output unit calculate the error between the actual output and
             the target output
          3. Calculate the error for each hidden unit
          4. Update each weight according to its contribution to the overall error
```

This version of the backpropagation algorithm were each weight is updated after *every* training sample is called *on-line* or *stochastic* training. The counterpart is *batch* or *off-line* training. There the weight update is done only once after every sample of the training set was presented to the network.

2.4.4 Remarks on Backpropagation Learning

The mechanisms behind gradient descent learning in ANNs are still only poorly understood. Therefore, the training of an ANN involves a number of trials with different learning parameters. The following section compiles some heuristics and variations often used in the context of the backpropagation algorithm.

Learning Rate

The learning rate η in the backpropagation algorithm moderates the degree to which the weights change in each step. The correct selection of η is crucial for the success of the training. If the value of η is too high, the network might miss good minima or jump out of them. If the value is too low, the training time might be unacceptably long. One suggestion to overcome this problem is to start with a relatively high value (e.g. 0.9) and to reduce the value slowly (e.g. to 0.1). The best solution for a given problem has to be found experimentally.

Momentum Term

The size of the weight update during training is proportional to the size of the gradient of the error function. Therefore, the training stagnates on flat plateaus of the error function. One common variation of the backpropagation algorithm which deals with this problem alters the weight update rule by making the update in the nth iteration depend partially on the update occurring during the $(n-1)$th iteration:

$$\Delta w_{ji}(n) = \eta \, \delta_j \, x_{ji} + \alpha \Delta w_{ji}(n-1) \tag{2.2}$$

The constant α with $0 \leq \alpha < 1$ is called *momentum*. In order to visualize the effect of adding a momentum, imagine the gradient descent search trajectory being a (momentumless) ball rolling down the error surface. With the new update rule from equation (2.2) the ball has a momentum that tends to keep it rolling in the same direction. This sometimes helps the training procedure to "roll over" small minima or to increase the step size in regions where the absolute value of the gradient is small.

Generalization

In any real world application the data used during the training stage of a neural network only samples a small fraction of the input space. The objective behind the training is to learn the characteristics of the distribution of the input data and not the characteristics of the given set of training samples. The network is supposed to

generalize from the training data to the distribution of the input data. To ensure generalization the error that the network makes during training is usually also measured on an independent *validation* or *test* set. Figure 2.7 shows a typical example. The error over the training set is constantly decreasing until it reaches almost zero, whereas the error over the test set only decreases up to a certain point and then increases again. Beyond this minimum, the neural network is said to *overfit* the training data. It picks up idiosyncrasies of the samples in the training set which are not representative of the general distribution of the input data.
The most obvious solution to the problem of overfitting is to simply stop training when the error over the validation set reaches its minimum. Of course this is only possible if enough data is available to set up an independent validation set. A different approach to the problem is to alter the error function in a way that prevents the network from learning very complex decision surfaces. This will be discussed in the next section.

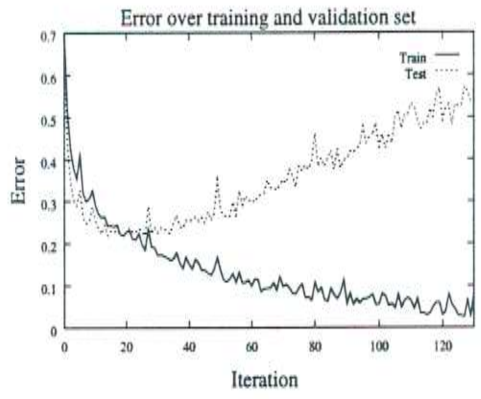

Figure 2.7: Example error curves over training and test set

Alternative Error Functions

The weight update rule for the backpropagation algorithm is derived using the sum of squared errors of the network as error function. Of course, this is not the only possible choice. In fact, any differentiable function that is minimized when its arguments are equal to each other could be used. In the following two variations of the error function are given. Note that for both functions a new weight update rule has to be derived.

- *Adding a penalty term for weight magnitude*
  Once common variation adds a penalty term to the error function (from [Mit97]):

$$E(\vec{w}) = \frac{1}{2} \sum_{d \in D} \sum_{k \in outputs} (t_{kd} - o_{kd})^2 + \gamma \sum_{i,j} w_{ji}^2$$

  Here D denotes the training set. This definition of the error function forces the gradient descent search to seek weight vectors with small magnitudes. This reduces the risk of overfitting.

- *Minimizing the cross entropy of the network with respect to the target values*

  It can be shown that in a specific context the sum-of-squares error function is the optimal choice if the target data follows a Gaussian distribution (see for example [Bis96]). For classification problems where the neural network is supposed to learn to output the correct class label given a certain input vector, the output is usually encoded using a 1-of-c coding scheme: the network is trained to output a 1 in the unit associated with the class of the training sample and 0 in all the other output units. The underlying target function might be best modeled by outputting the probability that a given sample belongs to a certain class. It turns out that in a specific context the best probability estimates are produced by networks trained with the so-called *cross entropy* error function, defined as :

$$E = -\sum_{d \in D} \sum_{i} t_{d,i} \log o_{d,i} + (1 - t_{d,i}) \log(1 - o_{d,i}) \tag{2.3}$$

  Here, i iterates over the output classes. As the output values are required to lie in the range $[0, 1]$ and to sum to unity, a different activation function is chosen for the units in the output layer. The output of unit i is calculated as:

$$y_i = \frac{e^{x_i}}{\sum_i e^{x_i}} \tag{2.4}$$

  where x_i denotes the input into unit i. This function is called *softmax function*.

2.4.5 Representational Power of Feedforward Networks

Feedforward Networks have been proven to be very useful for a variety of tasks. The range of existing applications goes from handwriting recognition, steering of autonomous vehicles and face recognition to credit rating and evaluation of marketing decisions. Some results of theoretical work show that limited depth feedforward networks are capable of representing a large set of functions [Mit97]:

- *Continuous functions*

  Every bounded continuous function can be approximated with arbitrarily small error (under a finite norm) by a network with two layers of units.

- *Arbitrary functions*

  Any function can be approximated to arbitrary accuracy by a network with three layers of units.

Chapter 3

Related Work

This chapter gives an overview of related work relevant to the thesis. In comparison to the large amount of publications in the field of handwriting recognition in general, the number of publications dealing specifically with the dynamic writing information is relatively small. The review covers systems operating on the *stroke level* and on the *word level*. The approaches classified into the latter group aim at recovering the writing information of a *whole* word, which is not the case for the systems of the first group. The stroke-level systems are restricted to the processing of single characters or single strokes within a word. The most important systems are compared to the approach proposed in this thesis.

3.1 Stroke-level systems

3.1.1 Lee and Pan

Lee and Pan [LP92] describe a system for the tracing of signatures. They stress that the intention is not the retrieval of the *original* sequence as written by a particular person or group, but the transformation of a given 2-D spatial pattern into a 1-D temporal pattern *in a consistent manner*. Their approach consists of four stages:

- *Preprocessing*
  The main step is the application of a thinning procedure on the original image. This results in a one pixel[1] wide skeleton of the signature. Several thinning-artifact-removal and noise-reduction procedures are applied.
- *Tracing*
  They use a set of low-level (for determining the next pixel) and high-level (for global stroke order) rules to find a possible trace.
- *Critical-point segmentation and normalization*
  In order to characterize the signature, they apply a multi-resolution critical-point segmentation and normalization.

[1] *Pixel* denotes a point on a 2D raster grid

In contrast to most of the other approaches dealing with the recovery of temporal information, Lee and Pan do not convert the skeleton of the signature into a more abstract graph representation. Instead, they directly operate on the pixel level.
The proposed rules use the properties described in section 2.2. For the decision on how to continue the trace in a junction point, they implement a criterion which minimizes the directional variation between incoming and outgoing branches. In order to optimize the global stroke sequence, two performance measures are introduced. These indices enforce the writing direction within a stroke and over the whole signature to follow a top-bottom-left-right trend.
The authors apply their method to 20 signatures. They evaluate the resulting traces manually and conclude that they are very similar to sequences a human would have produced. However, they admit having problems with letters, such as "a", "d", "g" or "q", which involve counterclockwise circular movements.

3.1.2 Boccignone et al.

Boccignone et al. [BCCM93] follow a similar approach for handwritten characters. The objective of their work is not the recovery of the *true* order in which the strokes were written, but the proper connection or separation of contiguous line pieces at junctions.
Their method is based on the assumption that consecutively drawn strokes share more similar features than strokes which, although they may be joint, are not temporally subsequent. Starting from the bitmap of the character, they construct a skeleton using Medial Axis Transformation. This thinning algorithm preserves information about the thickness of the original curve. In order to be able to compute the directions of lines merging at a line joint, they approximate the skeleton with a piecewise continuous curve. As curves produced by thinning algorithms show a lot of distortions, especially around branch points (those points being the most important for the task discussed here) the authors apply a number of correction algorithms to the polygonally approximated skeleton. The tracing step uses three features to determine which line segments should be grouped together at branch points:

- angle between the segments
- ratio between the length of the shortest and the longest segment of each possible line pair
- absolute value of the difference between the average widths of each line pair

The underlying assumption is that the optimal trace is the one which most closely follows the energy minimization criterion.
For each feature, a function determining the probability that the underlying segment pair has been drawn sequentially is *empirically* constructed. A linear combination integrates the resulting functions into a single score. The ordering of the stroke segments is then derived on a purely local basis. For every possible pair of segments in a given branch point, the continuity score is calculated and the pair with the highest score is selected. In case four edges are joining in a point, this rule is slightly altered.

The authors evaluate their method on a test set of approximately 6600 handwritten characters, containing upper and lower case letters in hand printed and cursive style. They report that their method finds the correct result in 97% of all test cases. As mentioned above, the proposed method only decides which stroke segments have been drawn sequentially. Therefore, a sequence containing two consecutive segments in the wrong order would still be considered correct.

3.1.3 Huang and Yasuhara

Huang and Yasuhara [HY95] propose a method for the recovery of the drawing order of *single-stroke* cursive script. The scope of the input data is further restricted by excluding strokes containing junctions of three or more stroke segments and strokes containing segments which are traced more than once.
After applying Medial Axis Transformation to the original image, the authors examine the Eulerian paths in the resulting skeleton. They propose a good continuity criterion, which takes the whole stroke into account and not only local regions around branch points. This definition aims at overcoming the problem that thinning algorithms tend to be error prone around the crucial junction points. The authors report that they tested their method extensively. However, they do not offer any supporting evidence, such as number of test cases or success rates, for this claim.

3.1.4 Lüdemann-Ravit

The author presents an approach [LR95] which is based on decision trees produced by the symbolic learning algorithm C4.5 [Qui93]. Using a standard thinning procedure, the system first derives a word skeleton. With the help of this data structure the task is formulated as a classification problem: given the feature representation of a node determine the outgoing edge. The bitmap data used as input for the system is produced from on-line data, hence the original sequences are available and can be used for the generation of the decision trees. With the feature set determined by the author the trained classifier achieves accuracies between 90% (writer independent) and 95% (writer dependent). However, the system does not provide a search component for the whole word. After the starting point is manually selected, the skeleton is traced until the classifier outputs a pen-up. In order to continue, a new starting point has to be manually chosen.
The system of Lüdemann-Ravit is similar to the approach presented in this thesis in that it tries to solve the problem using a classifier for the nodes of a graph structure. The feature representation developed by Lüdemann-Ravit is based solely on the word skeleton produced by a thinning algorithm. These algorithms tend to produce errors especially around intersections. The approach proposed in this thesis uses the complete crossing region as feature, therefore avoiding similar problems. Furthermore the system of Lüdemann-Ravit does not contain a search procedure for the recovery of the writing order of a whole word.

3.2 Word-level systems

3.2.1 Doermann and Rosenfeld

Doermann and Rosenfeld [DR93, DR95] present extensive work on the recovery of temporal information from static handwriting. They observe that a large amount of the information necessary to determine the correct stroke order is destroyed by standard skeletonization. Hence, their system is based directly on the gray-scale image. In contrast to most of the other approaches presented in this review, their goal is to recover the *true* order in which the strokes were constructed. They develop a *stroke recovery platform*, which provides a hierarchical representation of stroke-like features in a document, reaching from pixel-level information up to an abstract stroke graph.
The first step of the proposed approach comprises the identification of maximum-gradient pixels. For each of those pixels, a scan line in direction of the gradient is generated to locate an opposite edge. In cases where an edge is found, properties of this cross-sections such as width or slope, are computed. Groups of cross-sections with consistent widths and orientations are then combined to candidate stroke segments. With the help of these groupings, the remaining non-stroke-like regions are classified as junctions, endpoints or noise. Using all previously collected information a partial stroke graph is constructed. In order to reconstruct the trajectory underlying the stroke segments, the non-stroke regions which are represented by nodes in the stroke graph are further examined. The interpretation of the intersection regions is based on:

- *smoothness* of segment pairings
  The score quantizing the smoothness of a segment pairing is calculated using properties of the boundaries of the involved stroke segments (angles, sign and orientation of the curvatures).
- *compatibility* of the stroke reconstruction with the region in question
- *constraints* imposed by higher level understanding of the writing
  This may include stroke continuity, temporal ordering or global positioning.

The interpretation is supported by temporal clues extracted from various stages of the stroke recovery platform. The authors distinguish between three different classes of clues:

- Local clues
- Regional clues
- Global clues

Local clues are directly obtained from the static image of the writing sample. They try to capture marks such as striations, endpoint intensity variations, feathers or hooks. *Regional clues* are derived from relationships between neighboring strokes captured

in the stroke graph. They include stroke segment curvatures in various zones (lower, middle and upper), distances between stroke endpoints and junction interpretations. *Global clues* contain the basic properties described in section 2.2. With the help of the temporal clues and the constraints mentioned above, a globally consistent interpretation of the motion which created the given sample of writing is derived.
The authors examined a set of approximately 1000 hand printed and cursive word images extracted from U.S. mail pieces. They claim that more than 90% of the images have some clues. No further results are reported.
The stroke recovery platform proposed by Doermann and Rosenfeld makes strong usage of information directly extracted from scanned bitmaps. Therefore, their approach is hard to compare with this thesis which is based on bitmaps created from on-line data. As the authors do not provide an evaluation of their method, it is not possible to judge the soundness of this approach.

3.2.2 Bunke et al.

Bunke et al. [BASS97] propose a system based on graph search. After applying a set of standard preprocessing algorithms (normalization, thinning, filtering) the word skeleton is split vertically in places where the connection between letters is assumed. The resulting pieces are represented by directed attributed graphs. In order to derive the drawing order, a set of likelihood criteria is defined. They aim at capturing the basic properties of cursive script concerning writing direction and energy minimization as described in section 2.2. The authors use a best-first search algorithm with pruning to generate possible traces through the different graphs. The best path is found by minimizing cost functions based on the likelihood criteria. The resulting coordinate sequences are then recognized with an on-line handwriting recognizer. After re-training the recognizer with data produced by their system, they achieve a recognition accuracy of 75.6% on a data set of 150 words (using a dictionary with 25000 words). Without training the system recognizes 26.1% of the test words. The words used for testing where written by cooperative writers.
The system of Bunke et al. is similar to most of the other work presented so far in that it uses a set of manually derived criterion functions to determine the quality of a certain path. The system proposed in this thesis differs from these approaches by using a model trained from data. In order to keep the search procedure computational feasible, Bunke et al. perform a segmentation step during preprocessing. Based on the experience gained in on-line and off-line handwriting recognition, it is questionable that this kind of segmentation works reliably on words which are not written by cooperative writers. The search algorithm proposed in this thesis is able to deal with whole words without performing a segmentation.

Chapter 4

System Overview

The purpose of the system presented in this report is to recover the dynamic writing information from static handwriting. This chapter gives an overview of **NTime**, which has been developed within this project. All data structures and algorithms mentioned here are covered in greater detail in subsequent parts of the report. This chapter also provides a description of the input data used during training and testing of the system.

4.1 Overview

Figures 4.1 to 4.3 depict the different components of **NTime** and the data structures used in the processing stages. During the first step of the preprocessing, the on-line data is converted into *bitmaps* using an algorithm called *scan conversion*. In order to obtain a more compact description the *contour* of the bitmap is calculated. Using this contour it is possible to calculate the angular variations along the outside boundary of the bitmap, which is useful for the identification of certain regions of interest (e.g. endpoints or intersections). All these different sources of information (together referenced as enhanced bitmap) are used to build an abstract representation of the underlying word. The so-called *segment graph* describes the given word as a set of graphs, where each connected component of the segment graph corresponds to a connected part in the given bitmap. Every subsequent module of the system makes use of this important data structure.
With the help of the segment graph, the task of recovering the writing information from a bitmap is transformed into the task of finding the best path through the graph. Assuming that the correct starting point is given and that no pen-up events occur, the main problem is the determination of the output edge of a node. This can be interpreted as a *classification* task with the node and the incoming edge as input and the outgoing edges as possible output classes. In **NTime** a neural network is used to *model* the nodes of a segment graph. The network is trained using labels derived from the on-line data.

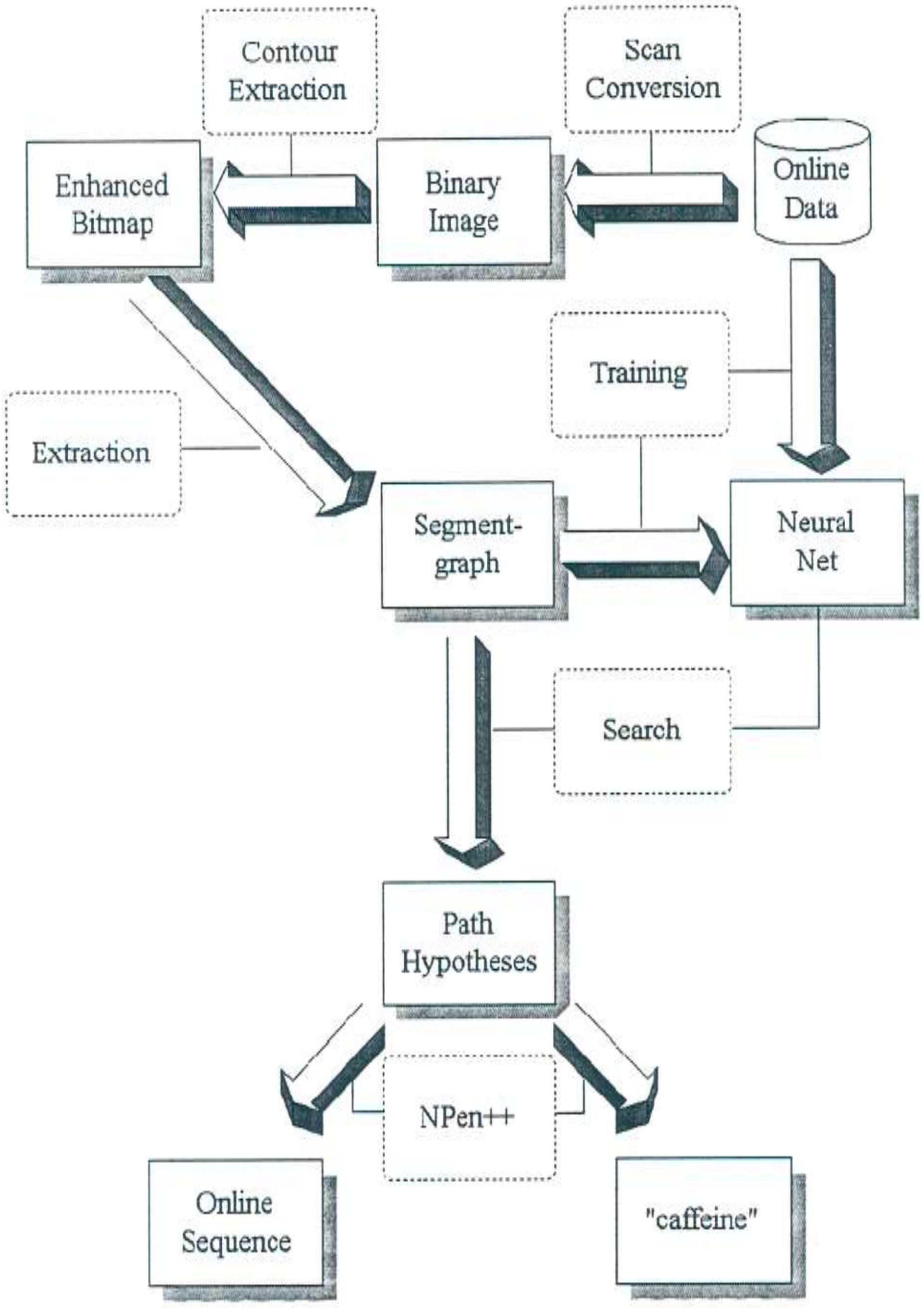

Figure 4.1: Overview of the NTime system

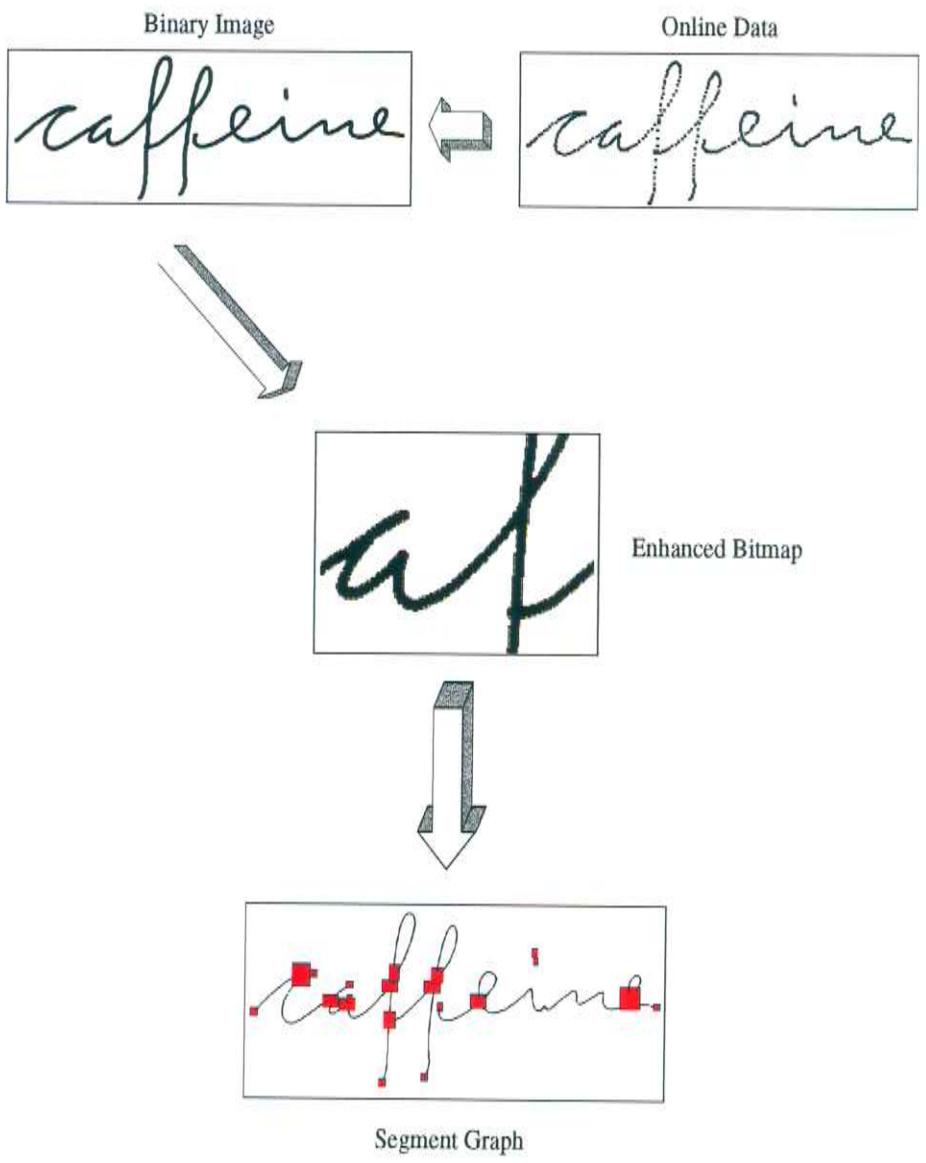

Figure 4.2: Visualization of the data structures used during preprocessing. The depicted data structures correspond to the top part of figure 4.1.

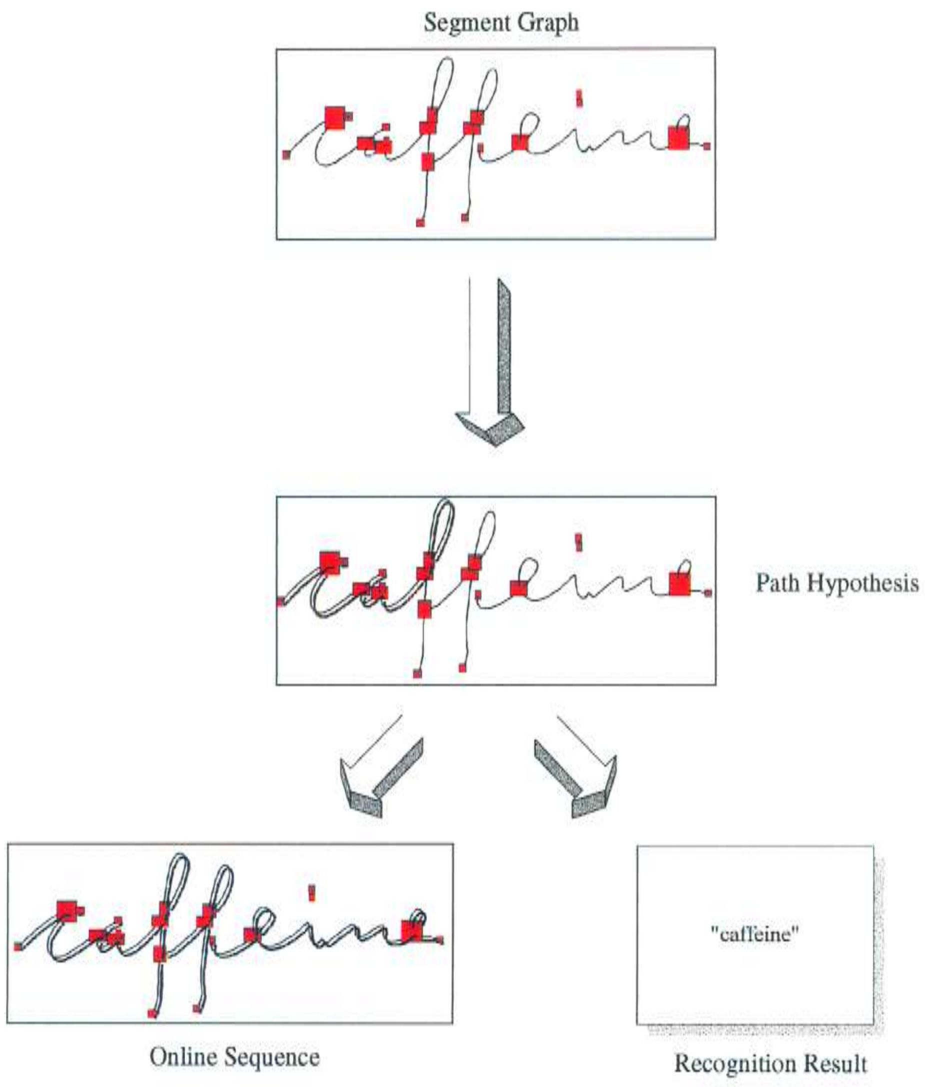

Figure 4.3: Visualization of the data structures used during search. The depicted data structures correspond to the bottom part of figure 4.1.

In order to determine a path through a connected component of the segment graph, the results of the model evaluation for the various nodes have to be combined. This is done using a heuristic approach. The path hypotheses calculated during the search process are finally evaluated using the **NPen**$^{++}$ handwriting recognizer. The sequence with the highest score is picked as the output. The result of the recognizer run, namely the word of the dictionary which most likely was written, is returned as a by-product.

4.2 Input Data

In order to keep the task computationally feasible, the assumption has to be made that the direction of the pen trajectory is only changed in intersections and not within an edge. This assumption seems reasonable, although not every word complies to it (see figure 4.4 for an example).

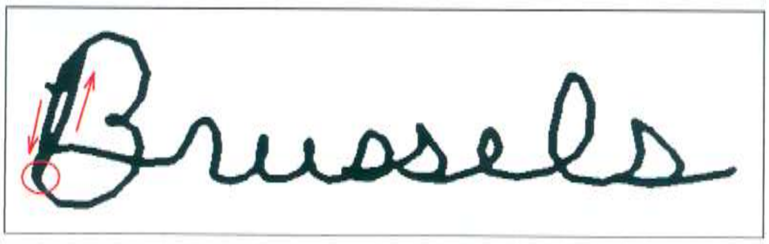

Figure 4.4: The pen direction is changed in the area marked by the red circle where no intersection is visible in the bitmap.

Furthermore it is assumed that pen-ups occur only in distinct places such as endpoints and nodes, and not within an edge. Following a pen-up the pen can only be put down in an endpoint.
As shown in chapter 1, the cursive and printed writing styles are fundamentally different. The system proposed in this work is currently limited to cursive words. The changes necessary to extend the range of input to printed and mixed style words are described in chapter 9. On-line data of good quality is used as source for the algorithms generating the bitmap representations. Figure 4.5 gives some examples of words in the data set.
Table 4.1 summarizes the distribution of words and samples used for training and testing of the system. Here *sample* refers to a feature vector calculated from a node in the segment graph. The two sets are build with words contributed from 160 different writers.

	Training	**Test**	**Total**
nbr words	1029	120	1149
nbr samples	24060	2740	26800

Table 4.1: Distribution of words and samples in training set and test set

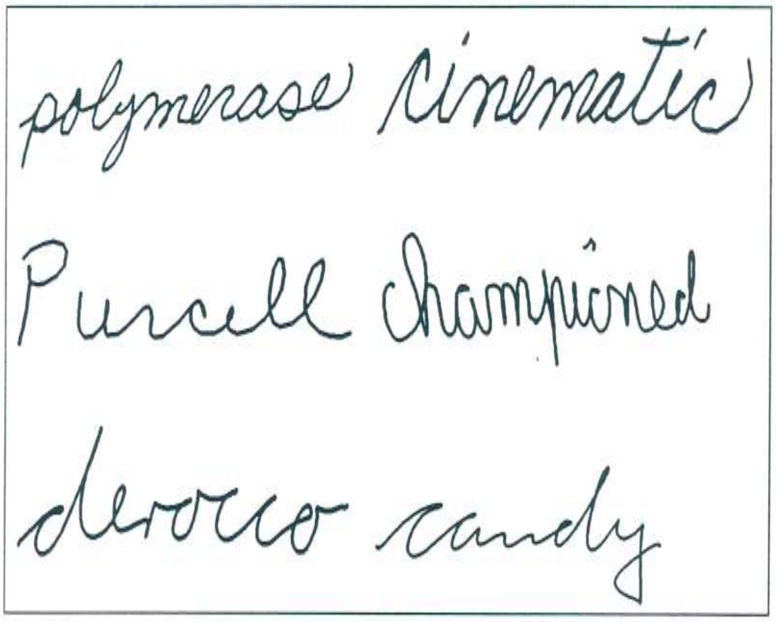

Figure 4.5: Examples of words in the data set

Chapter 5

Preprocessing

Most of the work which deals with the problem of recovery of dynamic writing information follows similar preprocessing strategies (see chapter 3). The major step is the application of a standard skeletonization or thinning algorithm in order to derive a one-pixel wide graph-like representation of the input bitmap. Unfortunately almost all skeletonization procedures have problems with intersections in the trajectory of the handwritten word. Therefore, a different approach was taken in the **NTime** system. The details of this algorithm are explained in section 5.3. This procedure relies heavily on the use of the contour of the input bitmap. The proposed system contains an efficient algorithm for contour calculation which is sketched in section 5.2. The chapter starts with a short explanation of the scan conversion algorithm used to convert the on-line data into binary images. To prove the efficiency of the preprocessing algorithms extensive test runs have been conducted. The results are summarized in section 5.4. The chapter concludes with a description of known problems.

5.1 Scan Conversion

The original data as collected for the on-line handwriting recognizer is a time ordered sequence of points from the pen trajectory. For the purpose discussed in this section, only the spatial information (i.e. the x-y coordinates of the data point) is used. In order to obtain a binary image of the on-line data the problem of calculating a line connecting the points of the input sequence has to be solved. This task constitutes a standard problem in Computer Vision, for which a number of algorithms have been proposed. Within this work the so-called *midpoint line algorithm* [FvFH90] is used. Given two endpoints (x_0, y_0) and (x_n, y_n), the algorithm computes the coordinates of the pixels that lie on or near an ideal line connecting the given endpoints. Figure 5.1 depicts this situation. It is assumed that the slope of the line to be approximated is between 0 and 1. All other cases can easily be reduced to this one by a suitable reflection.

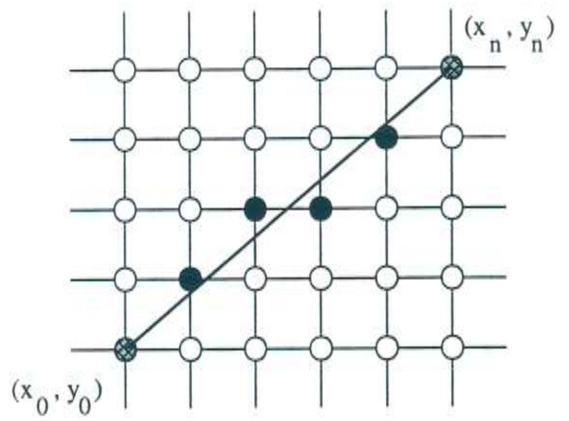

Figure 5.1: Example of a scan converted line between the two given endpoints (x_0, y_0) and (x_n, y_n). The solid black pixels are found by the algorithm.

The algorithm proceeds incrementally. With (x_p, y_p) being the last calculated point, the candidate pixels for the next point are (x_{p+1}, y_p) (called east pixel, E) and (x_{p+1}, y_{p+1}) (called northeast pixel, NE) (see figure 5.2).

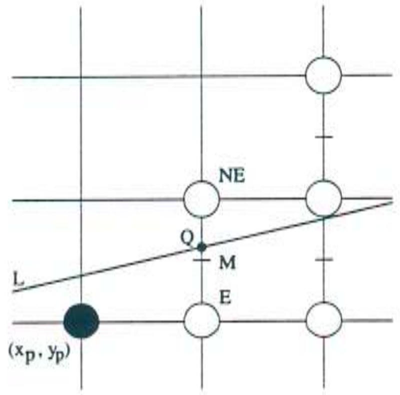

Figure 5.2: Details of the scan conversion algorithm. The procedure chooses pixel NE or E as follow-up pixel to (x_p, y_p), depending on which side of the middle point M the intersection point Q is located.

Let Q be the intersection point of the interpolation line L and the vertical grid line $x = x_{p+1}$. The algorithm calculates on which side of L the midpoint M of the line connecting E and NE lies. If M is located above the line, pixel E is closer to L and should consequently be selected, otherwise NE is the optimal choice. If L passes through M the lower point E is picked. A criterion which determines the current location of M can easily be computed out of the function definition for the line L (for the details refer to [FvFH90]). In order to obtain a line of width greater than 1 the algorithm is extended to write multiple pixels in each step. Furthermore, the ends of line segments are rounded to ensure smooth connections between adjoining segments. In a final step, the resulting bitmap is filtered and thresholded using a Gaussian filter mask. The actual values used for the different steps in the scan conversion are summarized in table 5.1. They have been found to produce the best results. Figure 5.3 shows an example generated by the algorithm. The time consumption of the different steps of the scan conversion are summarized in section 5.4.

Parameter	**Value**
Pen Width	7
Filter Mask Size	7
Threshold Value	0.5

Table 5.1: Parameter values used for scan conversion

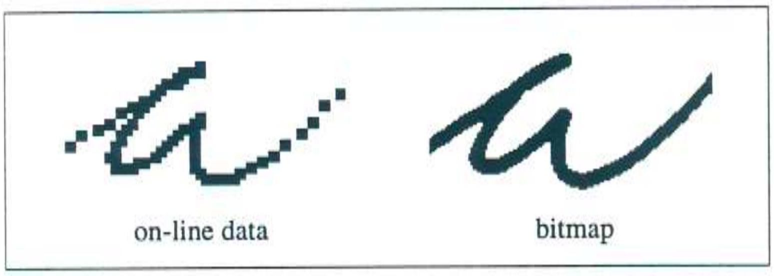

Figure 5.3: Scan conversion algorithm applied to on-line data

5.3 Segment Graph Extraction

The most important step during preprocessing is the derivation of an abstract description of the input image. A more compact representation reduces the amount of information to be processed and facilitates the analysis of the given word. The standard strategy for this problem is the application of a thinning algorithm on the input image. The following section discusses standard thinning methods and their shortcomings and then presents the algorithm used in **NTime**.

5.3.1 Standard Thinning Algorithms

The term *thinning algorithm* refers to a class of methods which reduce a given elongated pattern into a line drawing representation. The result is a one pixel wide *skeleton* of the input image. The literature on document and image processing contains a vast amount of thinning algorithms (refer to [LLS92] for a survey). Two different approaches to thinning can be identified: *topological thinning* and *medial axis extraction from distance maps* [OI92].

Algorithms belonging to the first class iteratively test the topological relevance of each pixel. This is either done by performing a raster scan over the image or by following the object contour. If the pixel is not needed to preserve the current topology, it is deleted.

The methods of the second group aim at extracting the *medial axis* of the object. For a given set S in the plane with boundary B, it is always possible to find the closest neighbor on B of any point X belonging to S. If X has more than one of those neighbors it belongs to the medial axis. Possible ways of calculating this axis include the computation of the centers of largest inscribed disks or the computation of the Voronoi Diagram of the boundary points.

Algorithms for both approaches are designed to handle arbitrarily shaped objects. When applied to handwritten words, they exhibit a number of typical problems (see figures 5.7 and 5.8). The examples were generated using the algorithm proposed in [OI92][1]. In the skeleton depicted in figure 5.7 two so-called *elongation artifacts* can be seen. This artifact appears when two line segments converge into a point with a small angle. The length of the elongation depends on the thickness and the converging angle of the involved line segments (refer to [AHD96] for further explanations).

In figure 5.8, the second common error made by standard thinning algorithms is shown. This *bifurcation artifact* may occur in points where two or more lines are crossing. The original junction is thinned into two (or more) connected crossing points. In the case of two intersecting lines, the bifurcation artifact can be seen as two overlapping elongation artifacts in opposite direction. Since junctions in the word skeleton are especially important for the problem addressed in this work, bifurcation artifacts severely affect the quality of the resulting skeleton for subsequent processing steps.

[1]This was done using the publicly available software package of the author.

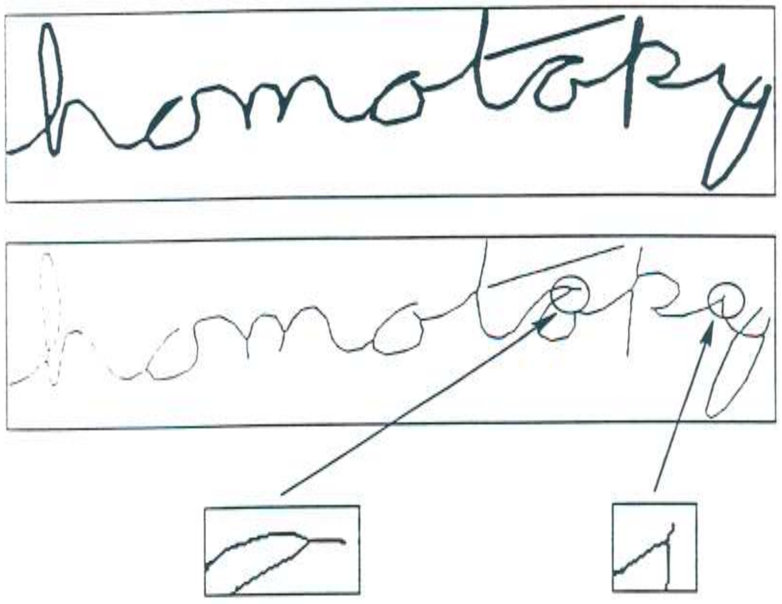

Figure 5.7: Elongation artifact in skeletons of handwritten words

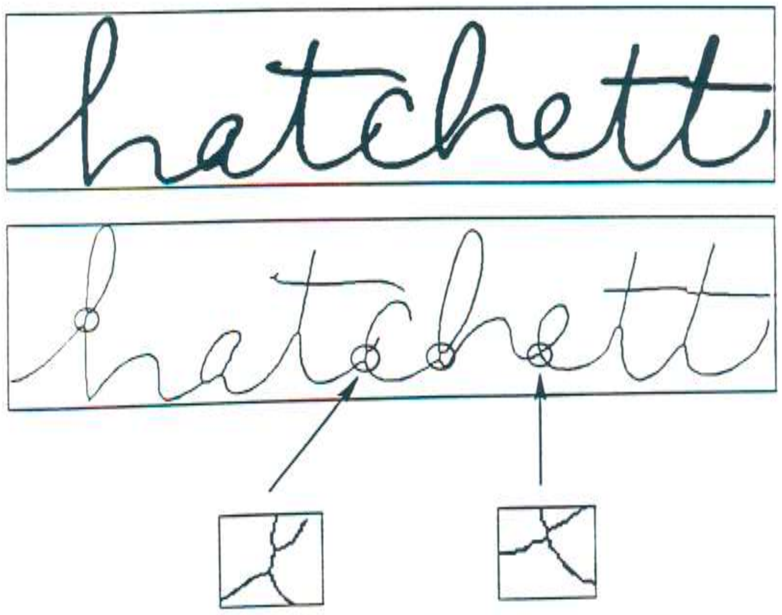

Figure 5.8: Bifurcation artifact in word skeletons

5.3.2 Thinning by Line Following

In order to avoid the problems of standard thinning algorithms, the proposed system uses a different approach (introduced in [CP92]) which was specifically designed to deal with handwritten input. The main idea is to use two pointers P_L and P_R located on either side of the current line to define a rectangular window. The pointers are moved along the contour and the middle points of the consecutive windows are joined to form the skeleton. Figure 5.9 depicts this situation. The following sections explain the different parts of the line-following algorithm: window definition, starting point localization, window moving, handling of intersections and the resulting segment graph.

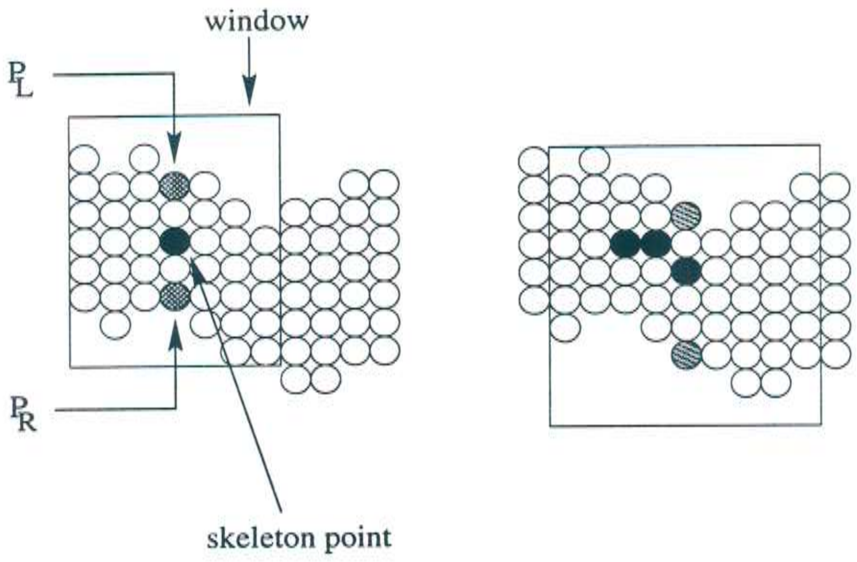

Figure 5.9: Moving window with skeleton points

Window Definition

The window definition takes the distance of the pointers P_L and P_R to each other into account. The offsets D_x and D_y used to determine the window corners are defined as:

$$D_x = \max(|P_{Lx} - P_{Rx}|, 2)$$
$$D_y = \max(|P_{Ly} - P_{Ry}|, 2)$$

(with $P_L = (P_{Lx}, P_{Ly})$ and $P_R = (P_{Rx}, P_{Ry})$)
Figure 5.10 visualizes this definition. By using D_x for the offset in y-direction and D_y for the x-direction it is ensured, that the currently important dimension is stressed.

Window Moving

The process of moving the window is accomplished by moving the pointers P_L and P_R. As stated earlier, the contour finding algorithm returns a linked list of contour points. The points are ordered counterclockwise around the given object and clockwise around holes. Therefore, subsequent contour points can simply be determined by traversing the linked list. Based on the results of Chouinard and Plamondon [CP92] a step size of 2 was chosen. In order to be able to move around curves where the pointers have different distances to cover, a special moving scheme has to be used. In each step the new *possible* locations P'_L and P'_R for P_L and P_R are obtained by determining the contour points with distance 2 to the original pointers. Then the Euclidean distances $P'_L \longleftrightarrow P'_R$, $P_L \longleftrightarrow P'_R$ and $P'_L \longleftrightarrow P_R$ are calculated. The point pair with the *smallest* distance is picked as new P_L and P_R. Figure 5.12 depicts an example of this process.

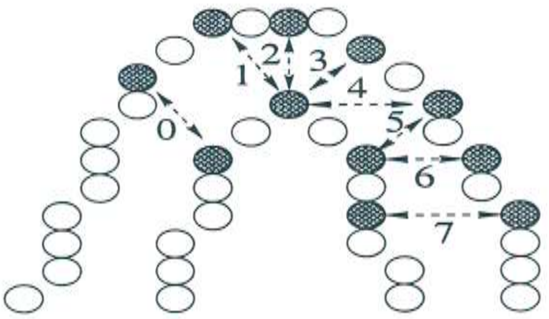

Figure 5.12: Sequence of point locations along a curve (after [CP92])

Handling of Intersections

After each movement, the situation in the window is analyzed. This is done by following the contour and the window boundary from P_R to P_L as shown in figure 5.13. The number of times a window edge is traversed can be used to determine the type of the underlying intersection:

- one time
  the line continues
- two times
  T shaped intersection (as in the example)
- three times
  X shaped intersection
- four and more times
  freely shaped intersection

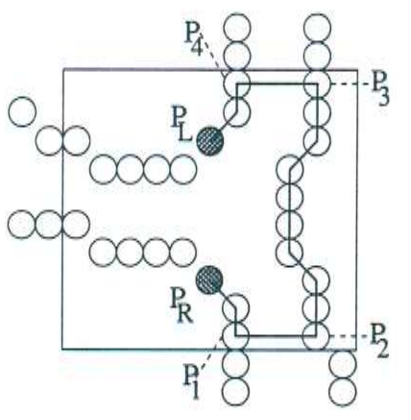

Figure 5.13: Analysis of moving window (after [CP92]). The procedure follows the contour from point P_R to point P_L and counts the number of times the window boarder is encountered in order to determine the situation in the window.

This categorization together with the distribution of discontinuity points around the intersection could be used to derive the correct thinning. In order to preserve as much information as possible the intersections are not thinned. Chapter 6 will show how the whole intersection is used as feature for the node model. Therefore only the location of the starting points of the outgoing branches (points P_1, P_2 and P_3, P_4 in figure 5.13) is important. The window which "discovered" the intersection is stored as node window and new windows are installed for every branch detected during the analysis. Figure 5.14 shows this situation for the example introduced in figure 5.13.

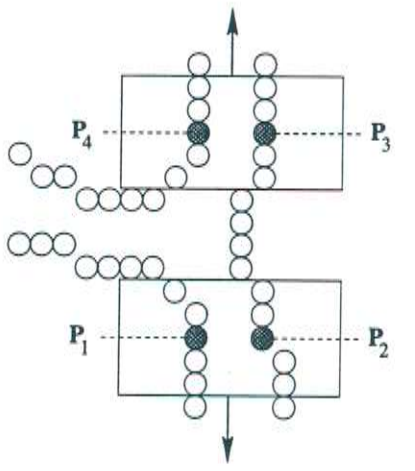

Figure 5.14: Two windows emerging out of a "window split"

already visited earlier. If this is the case, the corresponding node in the graph is retrieved and the connection for the actual branch is added. This procedure also identifies loops as branches where start- and endpoint are located in the same node. Figure 5.15 shows an example. The edges of the component graphs are denoted by red dotted lines. The black lines correspond to the points obtained by thinning the lines of the original bitmap. They are not part of the segment graph and are only shown to facilitate the understanding of the segment graph. The red rectangles representing the nodes in the graph have position and dimension of the windows which first encountered the underlying intersection.

5.4 Performance

The following tables 5.3 and 5.4 summarize the time consumption of the various preprocessing steps. The numbers are based on a test run over the data set described in section 4.2[2]. The average sizes of the main data structures used during preprocessing were also calculated. The results are compiled in table 5.5.

[2]The experiments were conducted on an AMD K6 233MHz PC running under Linux.

Performance of Preprocessing	
Scan Conversion	**Avg. Time Consumption [ms]**
Point Calculation	139.3
Gaussian Filtering	4445.4
Thresholding	227.9
Total	4812.6
Preprocessing	
Contour Calculation	345.2
Curvature Calculation	161.7
Move Windows	105.0
Analyze Windows	391.9
Total	1003.8

Table 5.3: Performance evaluation of preprocessing

Performance of Preprocessing	
Scan Conversion	**Rel. Time Consumption**
Point Calculation	2.9%
Gaussian Filtering	92.4%
Thresholding	4.7%
Total	100%
Preprocessing	
Contour Calculation	34.4%
Curvature Calculation	16.1%
Move Windows	10.5%
Analyze Windows	39.0%
Total	100%

Table 5.4: Performance evaluation of preprocessing

	avg. number of points
contour length	3855.2
	avg. number of nodes
nbr. endpoints	8.73
nbr. intersection nodes	10.95

Table 5.5: Average sizes of preprocessing data structures

5.6 Contribution of this Thesis

This chapter described the preprocessing algorithms used within the **NTime** system. The combination of a number of known procedures results in a new representation specific for bitmaps of handwritten words. This so-called segment graph preserves a maximum of information for subsequent parts of the system. The efficient implementation of the preprocessing algorithms keeps the computational costs low.

Chapter 6

Node Models

The purpose of this thesis is to develop a system which recovers the dynamic writing information from static handwriting. Using the segment graph described in the last chapter, this task is transformed into the problem of finding an optimal path through this data structure.[1] The main question to be answered is how to proceed with a given path in nodes of the graph.
This chapter addresses the decision strategy implemented in **NTime**. After restating the task as classification problem, a feature representation for the node model is developed. The chapter continues with the description of the training processes for the neural networks which implement the model and concludes with an evaluation.

6.1 Formulation as Classification Problem

Assuming that a path into a node of the segment graph is given, the task is to determine the outgoing edge of the node or to decide for a pen-up. This can be seen as classification problem with the node and the incoming edge as input and the possible outgoing edges (including pen-up) as target classes. As pen-up labels comprise only 0.5% of all labels in the data set, they are not further considered as output class. Figure 6.1 depicts the resulting classification problem. In the above situation the node (together with the input edge) should be mapped to output class "1". Figure 6.2 shows the example in a more abstract way.
As with most real world problems, it is not possible to directly determine a concise form for this mapping. Therefore, a *statistical classifier* trained on examples of nodes and corresponding classification is applied. Due to their robustness and their known ability to perform well in pattern classification tasks, a neural network model was chosen as classifier.

[1] More specifically through the component graphs of the segment graph.

6.2 Features of the Model

In order to be able to successfully apply a neural network model, a suitable representation of input and output values has to be found. The system proposed in this work combines a set of features taking the properties of cursive script (see section 2.2) and the results found in the related work (see chapter 3) into account. The following features are used:

- Bitmap representation of the node
- Local context of the node
- Node configuration
- History of edge usage
- Energy necessary to connect different edges

The following sections describe these features in greater detail.

6.2.1 Bitmap Representation

The work of Doermann and Rosenfeld [DR95] proves that a lot of clues for the extraction of temporal information are observable *directly* in the bitmap. As these clues are beyond mere connectivity information, most of them get lost during the conversion of the original bitmap into a skeleton. An example for this situation can be found in figures 6.3 and 6.4. The magnified node region of the character "d" clearly shows that the pen passed twice along the upper arch of the loop. Figure 6.4 depicts the corresponding part in the skeleton of the word[2]. The clue visible in the original bitmap is not present in the skeleton. In order to overcome this situation the neural network is provided with a down-sampled bitmap of the node region. Some examples for the resulting representation are shown in figure 6.5. Notice that the thicker arch in the last "d" is visible in the feature representation. Experiments show that a bitmap size of 6×6 gives the best results.

[2]The skeleton was produced using the software package introduced in section 5.3.

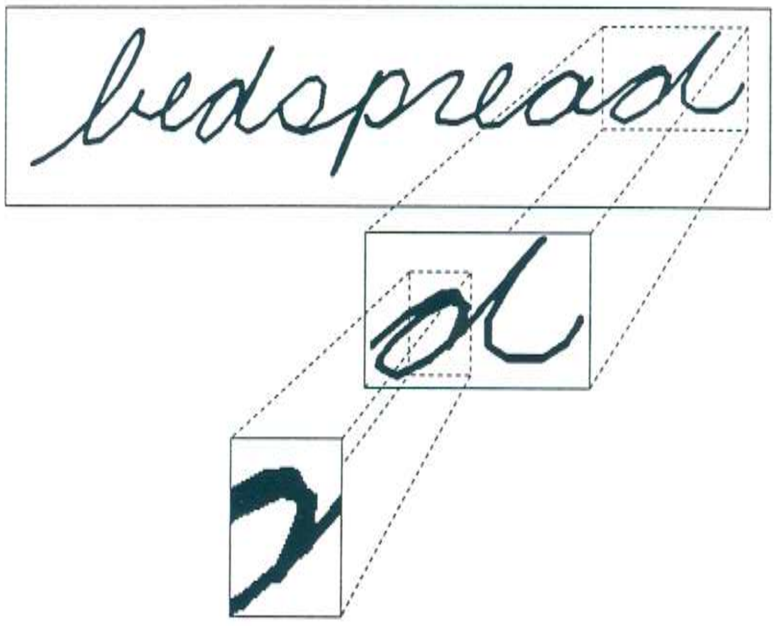

Figure 6.3: Magnification of an intersection with a retraced line.

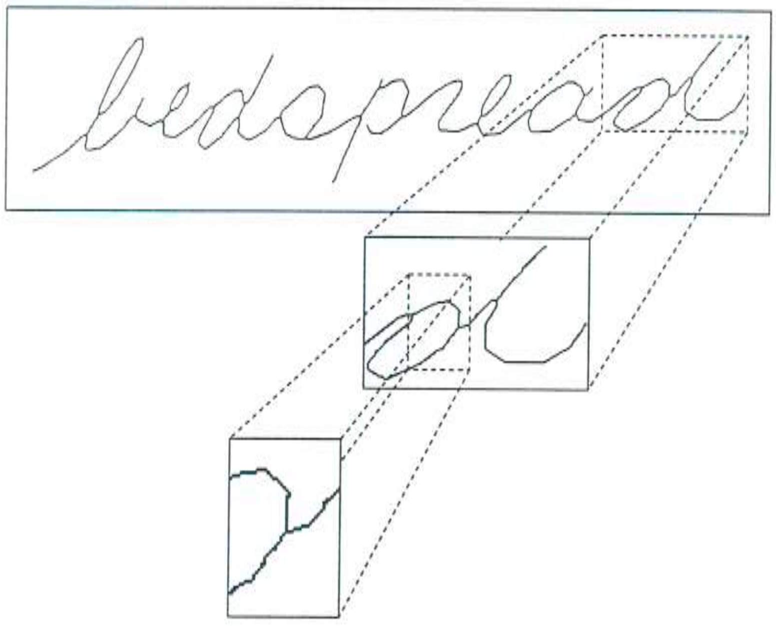

Figure 6.4: Magnification of the corresponding intersection in the skeleton

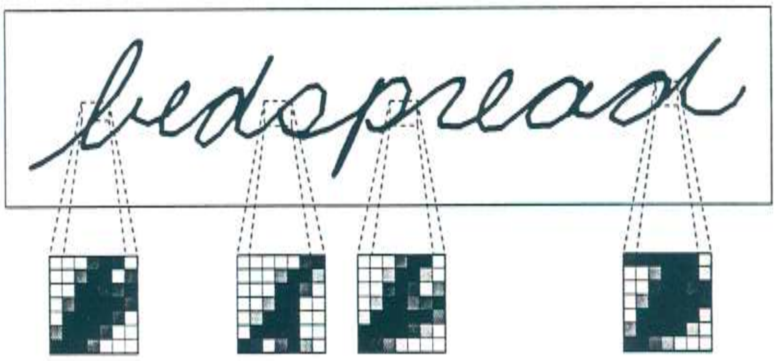

Figure 6.5: Bitmap representation of nodes

6.2.2 Local Node Context

The decision on how to leave a given node can not be made based only on a bitmap image of the node. Figure 6.6 shows an example of two almost identical bitmap representations from nodes in completely different contexts.

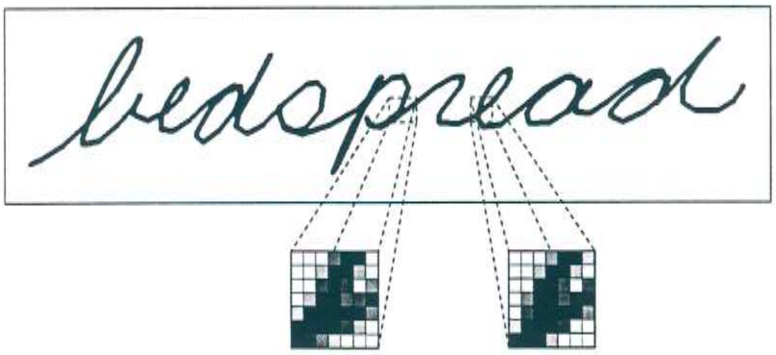

Figure 6.6: Similar bitmap representations in different contexts

Therefore, a feature which captures the local context of a node is added. For each branch originating in a node the type of the node (endpoint, node with certain edge degree, self-loop) located on the other end of the connection is noted. Figure 6.7 shows an example.

6.2.3 Node configuration

This simple feature tells the neural network the edge degree of a given node.

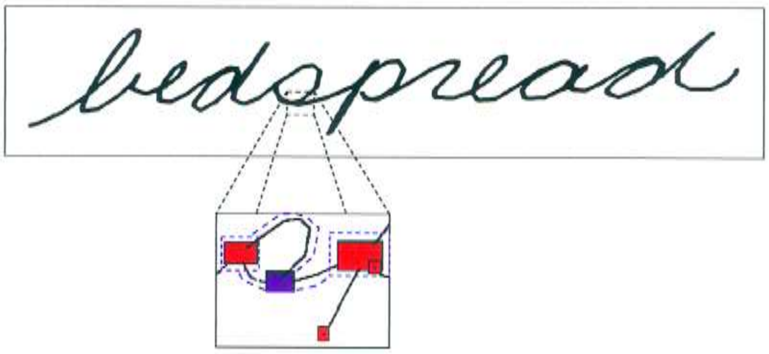

Figure 6.7: Local context feature. The nodes and edges belonging to the context of the highlighted node are connected by dashed lines.

6.2.4 History of Edge Usage

In the course of writing a word, every intersection is visited at least twice. Therefore, the correct outgoing edge in a given situation does not only depend on the incoming edge, but also on the history of encounters of the pen trajectory with the node so far. Figure 6.8 depicts an example of this situation. The highlighted node is first entered via edge 1. After leaving the node along the same edge, it is reentered by traversing edge 2. For the decision on the outgoing edge the temporal context is crucial. Without this context, the trajectory would most likely be continued via edge 1 and not via edge 0 as it should be.

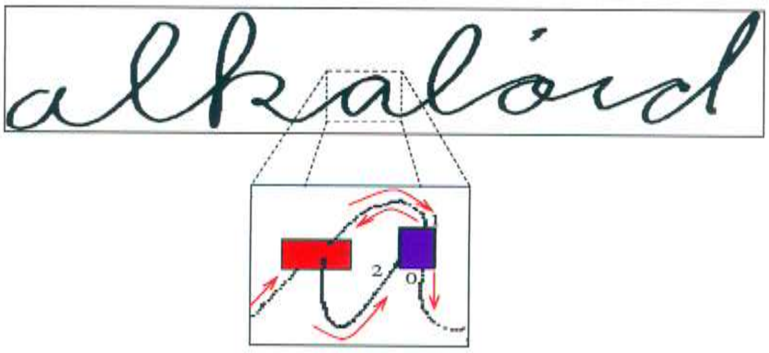

Figure 6.8: Edge usage of a node (with edges 0, 1 and 2). The highlighted node is first entered via edge 1.

In order to store this information the system uses counters for every edge of a given node. Experiments show that it is more beneficial in terms of recognition results to have separate counters for incoming and outgoing connections.

6.2.5 Energy Criterion

The last of the proposed features aims at capturing the energy necessary to connect a given incoming edge with a different outgoing edge. As mentioned in section 2.2 the stroke order is influenced by the attempt to minimize the energy required to produce the strokes. The standard way of quantifying this energy is to measure the angular deviation between the incoming edge and the possible outgoing edges (see figure 6.9).

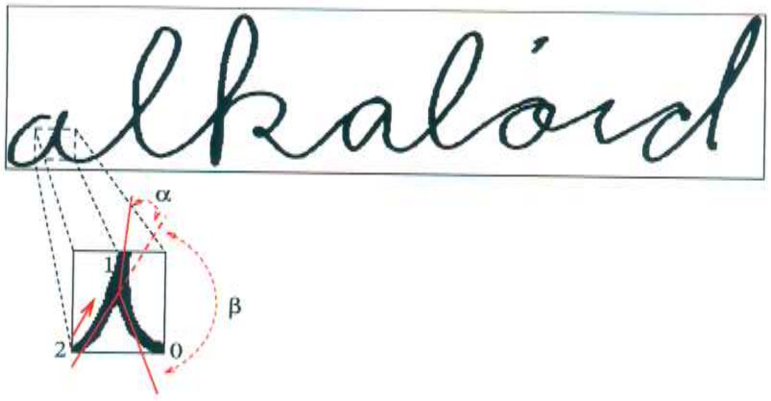

Figure 6.9: Standard way of measuring energy with angular deviation. The deviation between edges 2 and 1 is set to α, the one between edges 2 and 0 is approximated by β.

This method depends heavily on a correct placement of the control points for the calculation of the angles which is hard to guarantee. Therefore, an alternative approach is implemented in the proposed system. The energy that is needed to connect two edges of a node is usually defined by the curvature of the line through the node in question. It seems natural to measure the energy in the same way. This is done by fitting a Bézier curve through the two edges in question (incoming and outgoing edge) and using the curvature along the line as measurement for the energy (see figure 6.10).
Bézier curves of degree 3 which are used in the system are defined by four control points. These points should be close enough to the node to capture just the local situation without being located directly in the node. A polygonal approximation to the pen trajectory is used to find these positions[3]. The so-called Wall algorithm [WD84] maximizes the length of the approximating segments. These polygon edges are guaranteed to lie within a predefined distance of the curve. The first two control points of these line segments satisfy the conditions set above. Figure 6.11 shows the example from above with highlighted control points.
Bézier curves have a couple of properties which make them particularly useful for the

[3]This algorithm is applied in [LR95] to position the control points for the calculation of the angular deviation.

Due to the fact that only the control points P_0 and P_3 are interpolated, the optimal

6.3.2 Output Encoding

As described in chapter 2, a neural network is trained by providing the network with input patterns and desired output patterns. The system proposed in this thesis uses a 1–of–c coding for the network output with $c = 4$. The output classes correspond to the different edges of the generic node following the numbering scheme described above. Figure 6.13 illustrates this situation.

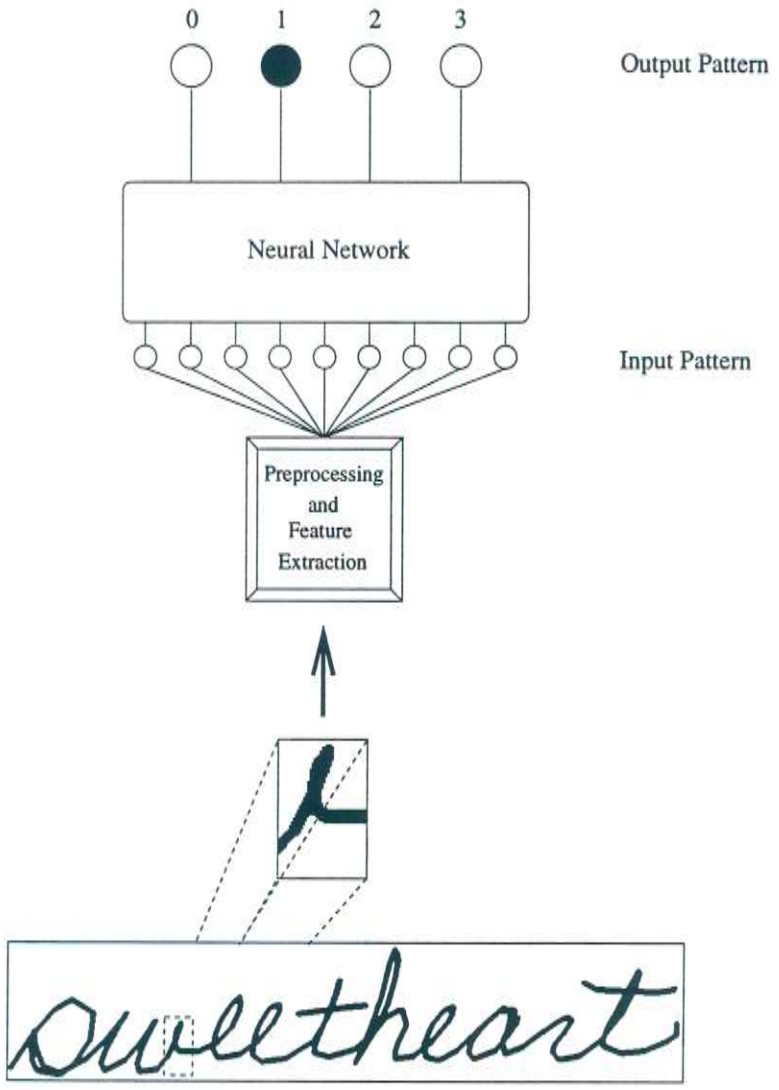

Figure 6.13: 1–of–c encoding for the output of the neural network

6.4 Training of the Neural Network

Due to the lack of generally available automatic procedures to determine the optimal structure of a neural network, a large number of training runs have been conducted to determine the parameters of the model. Unfortunately, this process does not guarantee that an optimal solution is found.

6.4.1 Single Network

As a first approach a single feedforward neural network was trained with standard backpropagation[4]. The best results could be obtained with a three layer network consisting of input, one hidden and output layer. The neurons in the hidden layer use a sigmoidal activation function, while the output of the units in the last layer is computed with the softmax function. The network is trained by minimizing the cross entropy error. As described in section 2.4.4, this is the optimal setup for the classification problem of this work. Table 6.2 summarizes the parameters used for the best network. Figure 6.14 shows classification and cross entropy error of this network. Based on the training and test sets described in section 4.2 the network achieves a classification accuracy of **93.4%**.

Parameter	**Value**
Input Layer [nodes]	92
Hidden Layer	200
Output Layer	100
momentum	0.2
learning rate	0.2
training iterations	50

Table 6.2: Parameters of the best single network

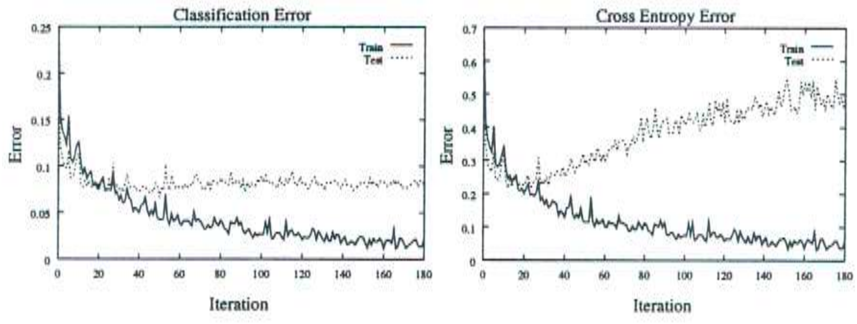

Figure 6.14: Error curves for a single network

[4]The implementation of the neural networks was taken from the JANUS speech recognizer, developed at the Interactive Systems Labs (see [Fri96]).

6.4.2 Multiple Networks

It is possible to identify a couple of typical connection patterns for the different node types (see figure 6.15). In order to correctly model the given data, separate networks for nodes of edge degree 3 (network A) and 4 (network B) have been trained. The small amount of training data available for nodes with higher edge degrees did not allow to train networks for those nodes. The parameters for the best networks are compiled in table 6.3. The corresponding error curves are shown in figure 6.16.

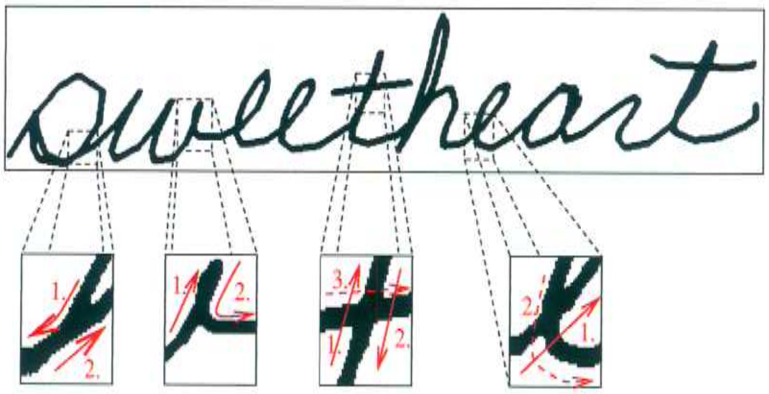

Figure 6.15: Connection patterns for different node types. The numbers indicate the order in which the pen passes through the intersection.

network A	
Parameter	**Value**
Input Layer [nodes]	92
Hidden Layer	120
Output Layer	80
momentum	0.1
learning rate	0.1
training iterations	25

network B	
Parameter	**Value**
Input Layer [nodes]	92
Hidden Layer	50
Output Layer	50
momentum	0.1
learning rate	0.1
training iterations	26

Table 6.3: Parameter values for the networks for nodes with edge degree 3 (network A) and edge degree 4 (network B)

The classification accuracies of network A and B are **94.01%** and **92.22%** respectively. This results in an overall accuracy of **93.8%**. A summarization of the different network performances can be found in table 6.4.

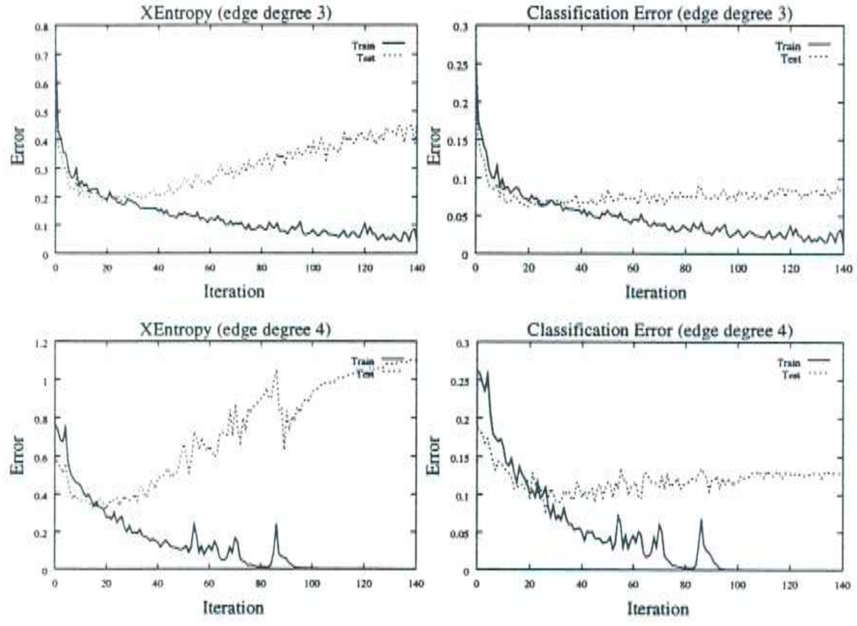

Figure 6.16: Error curves for networks A and B.

	classification accuracy	
single network		93.4%
network A	94.01%	**93.8%**
network B	92.22%	

Table 6.4: Classification accuracies of the different networks

6.5 Analysis of the Results

The results of the last section show that the neural networks are able to solve the classification problem defined in section 6.1. In order to identify the reasons behind the errors that the neural networks are still making, so-called *confusion matrices* are calculated for the different networks. In row i and column j of the matrix the cases labeled class j and classified by the network as belonging to class i are listed. Using the class definitions of section 6.1 the matrices distinguish classes 0 to 2 for the nodes of edge degree 3 and classes 0 to 3 for nodes of edge degree 4. The tables 6.5, 6.6 and 6.7 contain the confusion matrices for the single network and the separate networks A and B.

confusion matrix for single network (abs. freq.)				
	label			
classification	class 0	class 1	class 2	class 3
class 0	**1238**	34	22	2
class 1	38	**1050**	18	0
class 2	25	22	**260**	1
class 3	3	6	1	**11**

confusion matrix for single network (rel. freq.)				
	label			
classification	class 0	class 1	class 2	class 3
class 0	**95.5%**	2.6%	1.7%	0.2%
class 1	3.4%	**94.9%**	1.6%	0%
class 2	8.1%	7.1%	**84.4%**	0.3%
class 3	14.3%	28.6%	4.8%	**52.4%**

Table 6.5: Confusion matrix for single network

confusion matrix for network A (abs. freq.)			
	label		
classification	class 0	class 1	class 2
class 0	**1063**	74	27
class 1	47	**895**	10
class 2	28	38	**220**

confusion matrix for network A (rel. freq.)			
	label		
classification	class 0	class 1	class 2
class 0	**91.3%**	6.4%	2.3%
class 1	4.9&	**94%**	1.1%
class 2	9.8%	13.3%	**76.9%**

Table 6.6: Confusion matrix for network A

confusion matrix for network B (abs. freq.)				
	label			
classification	class 0	class 1	class 2	class 3
class 0	**130**	5	2	2
class 1	2	**151**	1	0
class 2	1	3	**16**	1
class 3	4	4	1	**11**

confusion matrix for network B (abs. freq.)				
	label			
classification	class 0	class 1	class 2	class 3
class 0	**93.5%**	3.6%	1.4%	1.4%
class 1	0.6%	**98.1%**	0.6%	0%
class 2	4.3%	13%	**69.6%**	4.3%
class 3	20%	20%	5%	**55%**

Table 6.7: Confusion matrix for network B

It can be concluded from these matrices that the networks have problems with the correct classification of samples belonging to classes 2 and 3. As the classification problem does not appear to be harder for those classes, it seems that insufficient training data is the reason behind this weakness. This conjecture is supported by the distribution of classes in the training set as shown in table 6.8.

Distribution of classes in the training set				
	class label			
node type	class 0	class 1	class 2	class 3
edge degree 3 (a)	9726	8050	2506	–
edge degree 3 (r)	47.9%	39.7%	12.4%	–
edge degree 4 (a)	1446	1504	314	338
edge degree 4 (r)	40.1%	41.8%	8.7%	9.4%

Table 6.8: Distribution of classes in the training set (a = absolute frequency, r = relative frequency

It is reasonable to expect that the classification results can be improved by adding more data to the classes in question.

6.6 Importance of the Inputs

In general, it is hard to find an optimal set of input features for a neural network. Therefore, the common practice is to use *a priori* knowledge of the given problem to define a number of features and determine the best subset by experiments. The feature set described earlier in this chapter is the result of this approach.
In order to get more inside into the relevance of the different features in the proposed feature set, their predictive importance is determined. This is done by measuring the classification error of networks obtained by omitting a certain feature from the input set. As the nodes with edge degree 3 represent the biggest subset in the training data, networks with the parameters of network A are trained. The resulting classification errors are compiled in table 6.9. Based on these results, the two most important features are the history and the bitmap representation.

feature set	**classification Error**
orig. set	5.99%
w/o history	**11.78%**
w/o bitmap	**10.57%**
w/o local	8.49%
w/o curvature	8.45%
w/o config	7.99%

Table 6.9: Classification errors of reduced feature sets

Figures 6.17 and 6.18 show the curves of classification and cross entropy error for the five different setups.

6.7 Contribution of this Thesis

This chapter described the feature representation developed as part of this thesis. Together with a neural network classifier the new representation has been proven to be highly effective. The results of this chapter show that an independent modeling of nodes with edge degrees 3 and 4 is beneficial for the performance of the neural networks. The proposed classifier achieves a classification accuracy of 93.8% on an independent test set.

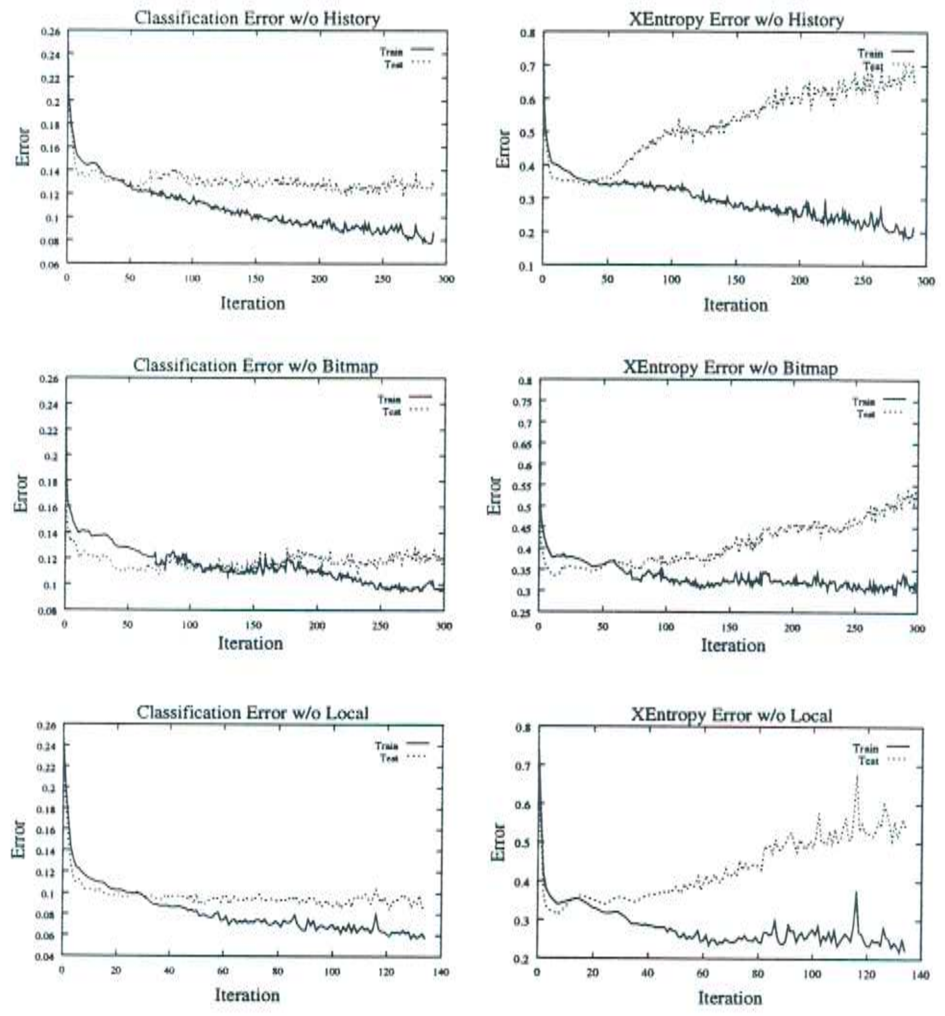

Figure 6.17: Error curves for reduced feature sets (1)

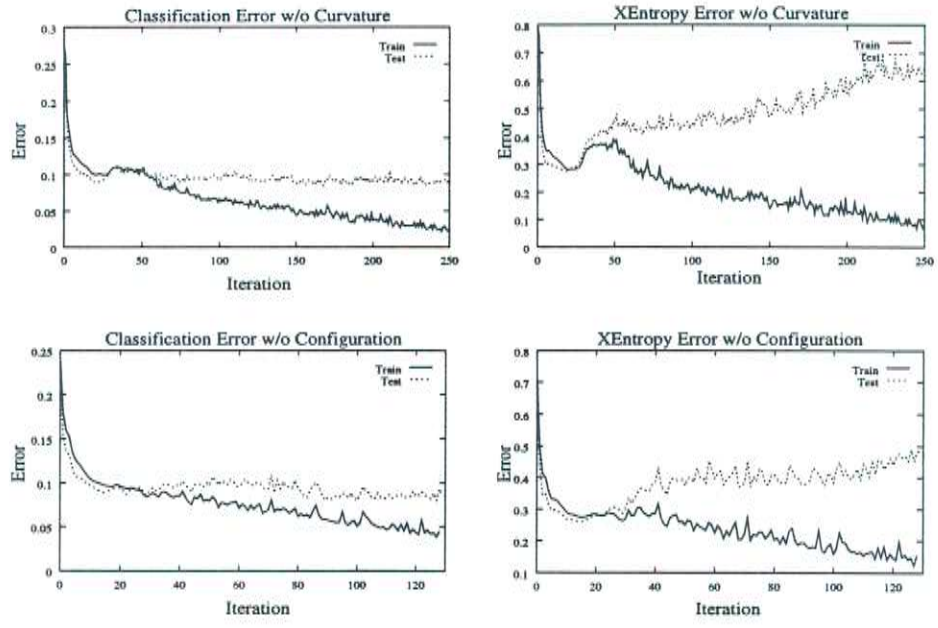

Figure 6.18: Error curves for reduced feature sets (2)

Chapter 7

Search

After introducing the algorithms for the extraction of the segment graph, the main task was identified as finding the best path through this graph (see chapter 6). Given the node models developed in the last chapter, one may think that the problem of the recovery of the writing trace can be solved by simply applying these models. The first section of this chapter explains why this is not the case. Afterwards the search approach implemented in the proposed system is presented and it is shown how to use the on-line handwriting recognizer **NPen^{++}** to find the best path through the segment graph.

7.1 Requirements for Search Strategy

The neural network model proposed in the last chapter only uses the *local* spatial and temporal context of a node. A search strategy solely based on the node models would most likely fail in finding a complete path through a given word.
The following list points out the problems that a search strategy for the whole word has to address:

- **Neural Networks**
  The networks presented in the last chapter achieve a high classification accuracy, but they still make mistakes. The search algorithm has to take that into account, possibly deviating from the top choice of the network.

- **Starting Points**
  The question of how to choose the right starting point has to be solved in the context of the whole word. Simply selecting the endpoint[1] closest to the upper left corner is not sufficient.

- **Endpoints**
  As the node models are only trained for intersection nodes, a strategy has to be developed on how to proceed in an endpoint.

[1] "Endpoint" refers to a node of the segment graph with edge degree 1. Depending on the current path this endpoint might be starting point, point in the middle or last point of the path.

- **Global Optimization**
  The overall goal is to find a trace for the *whole* word which should be reflected in the optimization criteria of the search approach.

7.2 Search Engine

The most straightforward approach to the search problem is to follow every possible path through the graph. Due to the exponential growth of the resulting trees this exhaustive search is not practicable.
The search algorithm implemented in **NTime** is *guided* by the results of the neural network evaluation of the nodes without completely relying on it. The next sections highlight the main points of this approach, following the list of problems given above.

7.2.1 Starting Points

Figure 7.1 gives an example of a word where a simple heuristics picking the endpoint to the far left side would produce the wrong result.

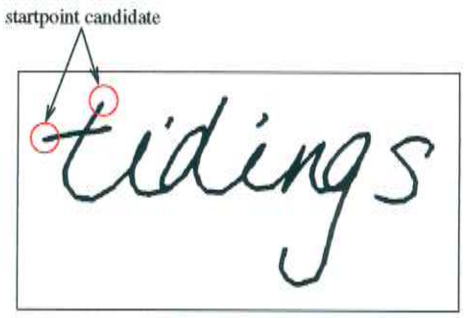

Figure 7.1: Word with two starting point candidates

As any kind of heuristics is likely to fail in certain situations, the system proposed in this thesis incorporates the search for the best starting point into the overall search by considering *every* endpoint of the given word as possible starting point. The path pruning which will be presented in section 7.2.5 quickly cuts off unlikely node sequences, so that the proposed approach stays computationally feasible. This method does not consider intersection nodes as starting points, hence assuming that the leading character of every word has a detectable endpoint.

7.2.2 Node Expansion

As explained above, the search strategy should not restrict the search space to the top choice of the neural network only. Therefore, the algorithm follows the best *two* paths suggested by the node model. The original path leading into the node is split into two paths which only differ by the last node. The probability of either alternative, as determined by the neural network, is stored as accumulated score in each path. Figure 7.2 depicts this procedure.

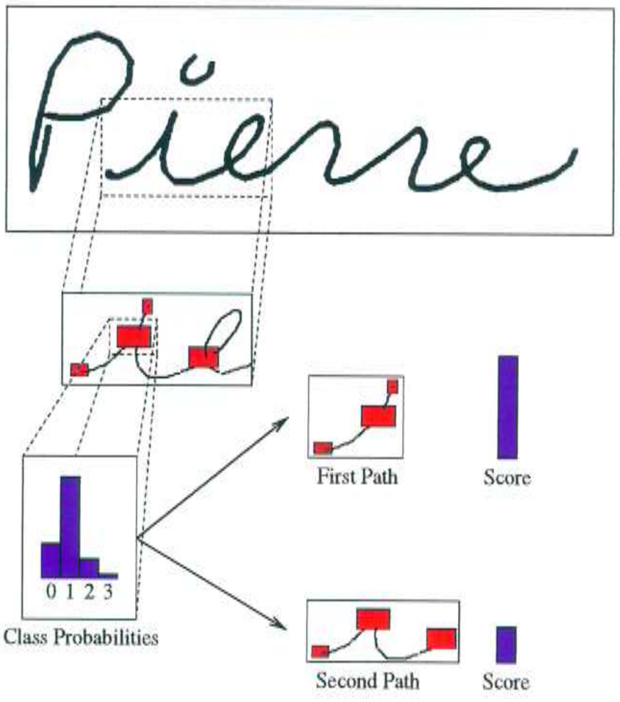

Figure 7.2: Expansion of a node in the segment graph. Following the class probabilities the path is expanded along edges 1 and 0.

This strategy still leads to an unbounded exponential growth in numbers of different paths which have to be followed in parallel.

7.2.3 Endpoint Handling

In the situation shown in figure 7.2, path 1 encounters an endpoint. In principle, there are there ways of proceeding out of an endpoint:

- **Retrace**
  Continue the path by retracing the edge taken into the endpoint back into the node.
- **Jump**
  Issue a pen-up and continue the path in another endpoint of the segment.
- **Terminate Path**
  End the current path in this endpoint.

In order to avoid making decisions based only on a local context, the search engine always considers the first two alternatives and produces new paths into every endpoint

which has not been visited and into the preceding node. The distance between the given endpoint and any target endpoint is stored with the resulting path and used during the pruning stage. Figure 7.3 shows an example of this approach.
A given path is terminated if the conditions for a continuation are not fulfilled, namely if there is no endpoint or node in which the path could be continued.

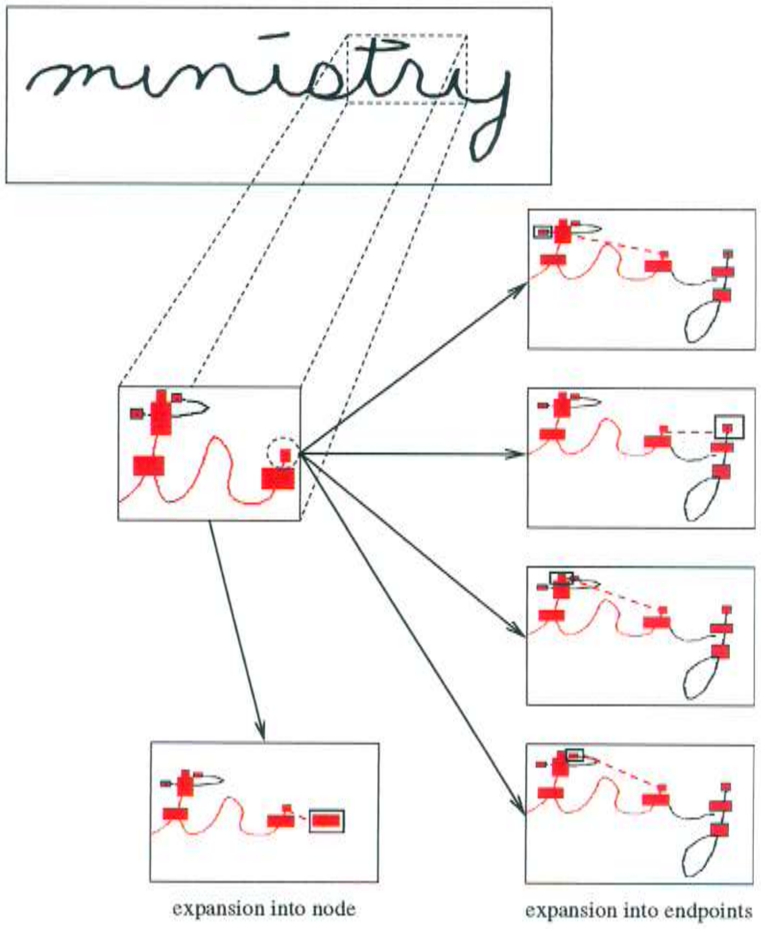

Figure 7.3: Handling of endpoints. The path ending in the endpoint highlighted in the center picture can be expanded into the endpoints shown on the right side or into the predecessor node as depicted in the bottom picture.

7.2.4 Search Procedure

Combining the points mentioned so far, the search algorithm consists of two steps:

```
1. Initializing a search path for every endpoint of the segment graph
2. until the termination condition is met do
       foreach active node $a_i$ do
           if $a_i$ is intersection node then
               expand original path into top two nodes as determined by the node
               model
           else if $a_i$ is endpoint node then
               expand original path into every unvisited endpoint and into preceding
               node if node has at least one unused edge
```

The expression "active nodes" refers to all nodes which are currently last node of a search path and therefore subject to expansion.

7.2.5 Path Pruning

Following the algorithm described in the last section, the number of active paths grows exponentially. A two stage pruning algorithm is used to keep this approach computationally feasible.

Consistency Check

In the first step, only paths which violate one of the following simple rules are cut off. A path is identified as *incorrect*, if:

- *it contains the same intersection node three times in a row*
  Explanation: This case only occurs if a loop is traced twice, which is not correct.
- *the same edge is taken three times in a row*
- *the same segment node appears more often than the type of the node allows*
  Explanation: Depending on the edge degree of a given node, a correct path will contain the node only a certain number of times. A specific endpoint for example will not be seen three times in a valid path.
- *the total distance bridged by jumps exceeds a multiple of the word length*
- *the search successively adds nodes without increasing the path length*
  Explanation: This occurs in paths which move "backwards", retracing parts of the graph which were already visited.

These rules are targeted at identifying *erroneous* paths which can be eliminated without loss of information. This technique already reduces the number of paths by 28% (see section 7.4).

Vertical Pruning

Unfortunately, the reduction obtained by performing the consistency check is not sufficient to keep the approach computationally feasible. If the number of paths remaining after the application of above stated rules exceeds a certain threshold, a second pruning step is applied. For the *vertical pruning*, a search beam of predefined width b is used. The currently active paths are ranked using the sum of two different scores and the top b paths are selected for further expansion.
In order to be able to compare the quality of different paths, two measures are used: *completion score* and *local penalty*.
Looking at traces through cursive words it is observable that all occurrences of a certain intersection node happen within a close temporal context. Figure 7.4 shows a typical example.

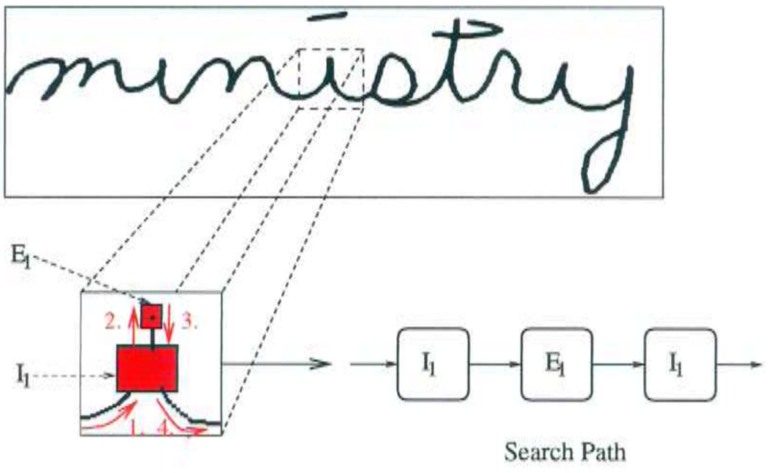

Figure 7.4: Motivation for completion score

The intersection node I_1 appears exactly twice in the whole path, only separated by the endpoint E_1. This property is exploited by the *completion score*. The score is calculated for every active path by following the path back into the root and scoring each intersection node with edge degree 3 depending on its compliance with the above scheme. The scope of this score is restricted as nodes of edge degree higher than 3 are not always traced in such a way. The intersection node of the horizontal "t"-bar with the vertical "t" line for example is likely to appear at completely different points in the search path as the "t"-bar is often written after the word is finished.
The local *penalty score* summarizes a number of general rules of handwriting which hold in most but not in every place. Disregard of these rules is therefore only recorded in the penalty score of a given path and the path is not immediately pruned. The rules and the corresponding penalty cases are:

- *The trajectory of a word usually starts in the upper left corner*
  The distance to the endpoint closest to the upper left corner is used as penalty score for all paths which do not start in this point.

- *The overall writing direction is from left to right*
  A penalty is added if paths are expanded in the opposite direction.
- *There are only a small number of pen-ups in cursive words*
  For each pen-up a value proportional to the distance between the endpoint where the pen was lifted and the endpoint where the pen was put down is added to the penalty score.
- *In cursive words, pen-ups usually take place at the end of a segment*
  For pen-ups occurring in the middle of the word with the subsequent pen-down point lying to the right, a value proportional to the distance between these two endpoints is added to the penalty score.

7.2.6 Objective Function

The *objective function* implements an optimization criterion for search paths. It incorporates all available information including the two scores introduced in the last section. More specifically it uses:

- *completion score S*
- *penalty score Pen*
- *probability score P*
  This is the accumulated score from the neural network evaluation.
- *path length L*

For a given search path $path_i$ the value under the objective function is calculated as:

$$f(path_i) = \alpha_1 * S(path_i) + \alpha_2 * Pen(path_i) + \alpha_3 * P(path_i) + \alpha_4 * L(path_i)$$

The parameters $\alpha_1, \ldots, \alpha_4$ are determined experimentally.
The ranking produced by applying this function is also used to terminate the search. The straightforward approach of searching until every node in the given segment is included in a path can lead to infinite search runs in case a certain node can not be reached. In order to avoid this problem, the listing produced by the objective function is monitored. If the best path remains unchanged over a number of consecutive search steps the search is terminated. Alternatively, the search is aborted if the length of the longest path does not change over a long time.

7.3 Selection of Best Path

One of the goals of the **NTime** system is to use the on-line handwriting recognizer **NPen**$^{++}$ to recognize the coordinate sequence produced by **NTime**. It is possible to simply select the best path emerging from the search procedure as path to be returned by the system. As **NPen**$^{++}$ recognizes input sequences using the model it built during training, this approach only succeeds if the sequences emitted by **NTime** are in compliance with the model. Without retraining of the handwriting recognizer with data produced by **NTime** the results are likely to be poor. Therefore **NPen**$^{++}$ is used to *determine* the best sequence.
The best n paths as calculated by the search engine of **NTime** are given to **NPen**$^{++}$. In case the underlying word consists of more than one segment, the best paths of every segment are combined to sequences. As the input data is restricted to cursive words, the number of segments occurring in a word is either 1 or 2. The score which **NPen**$^{++}$ produces along with the recognition result can be used, after normalization with the length of the recognized word, to rank the different hypotheses. The sequence with the highest score is selected, using **NPen**$^{++}$ as *model* for handwritten words. Figure 7.5 depicts this approach.

7.4 Evaluation

Table 7.1 summarizes some figures of the search approach described in the last sections. The results were obtained on the data set introduced in section 4.2.

Category	Value
nbr paths	34 356.3
nbr paths pruned	9 609.8
nbr search steps	31.8
length best path	28.7

Table 7.1: Evaluation of the search engine

The average number of paths produced during the different search runs was 34 356.3. Approximately 28% or 9 609.8 of these paths were cut off by the consistency check. The search beam applied during the vertical pruning was set to a width of 800 paths. This value was found to produce the best results. The search performed 31.8 iteration steps on average with the length of the best path being 28.7. The three additional steps can be attributed to the termination criterion.

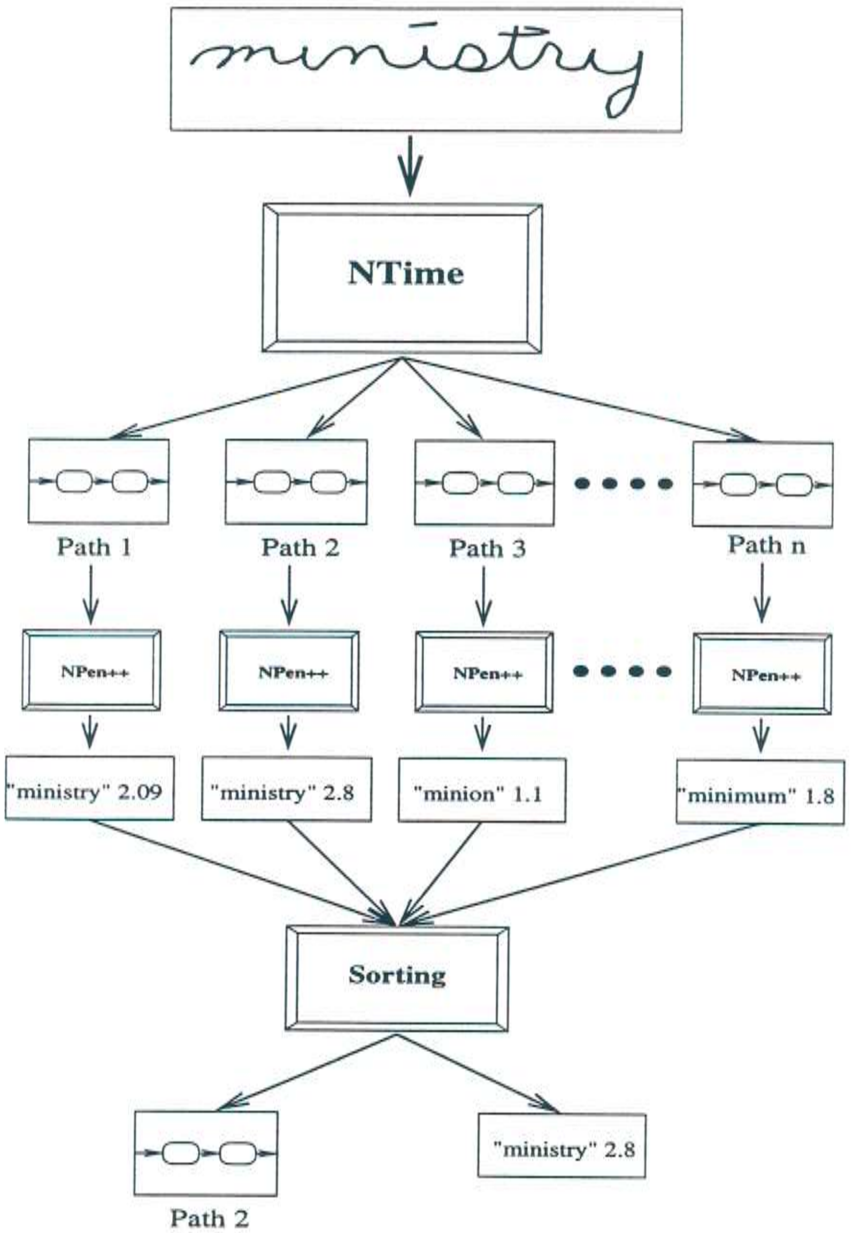

Figure 7.5: Selection of the best path. The numbers behind the hypotheses are scores produced by NPen^{++}.

Chapter 8

System Evaluation

There are a number of ways to evaluate the system presented in this work. One possible approach is to measure the correspondence of the sequences produced by **NTime** with the on-line data used to create the bitmap representations of the words. However, following this idea it is hard to distinguish between variances in writing style and errors in the sequences produced by the system. Therefore, the recognition accuracy of **NPen**$^{++}$ is used as performance measure.
This chapter presents the results obtained by using the coordinate sequences returned by the system proposed in this thesis as input to **NPen**$^{++}$. The errors made by **NTime** are examined and a detailed error analysis is given. The chapter concludes with some examples of traces derived by the system.

8.1 Recognition Accuracy

The test data is used in three different variations in order to determine the system performance. The set S_1 contains the original on-line data as collected for the handwriting recognizer. S_2 combines the coordinate sequences of the test words obtained by applying the preprocessing steps to the bitmap representation and using the correct labels extracted from the on-line data. Finally, the third set, S_3, consists of the sequences produced by **NTime**. The accuracy is measured for three different dictionary sizes: 120, 1 000 and 5 000 (denoted as $Dict_1$, $Dict_2$ and $Dict_3$)[1]. The first dictionary contains only the words occurring in the test set. The recognition accuracies resulting from the test runs are shown in table 8.1 and figure 8.1.
The recognition accuracy of the handwriting recognizer on the data produced by the proposed system is on average 12.2% below the accuracy achieved on on-line data. Out of this amount 1.1% are already lost when using data set S_2 instead of set S_1. Possible reasons for this loss are preprocessing errors and the fact that the sequences resulting from this process are slightly different from those **NPen**$^{++}$ was trained with.

[1]The dictionaries are build out of a 50 000 word dictionary from the Wall Street Journal corpus.

Recognition accuracies			
	$Dict_1$	$Dict_2$	$Dict_3$
S_1	95%	90.8%	80%
S_2	95%	89.1%	78.3%
S_3	**84%**	**77.3%**	**67.8%**

Table 8.1: Recognition accuracies of the different data sets using the dictionaries $Dict_1$, $Dict_2$ and $Dict_3$

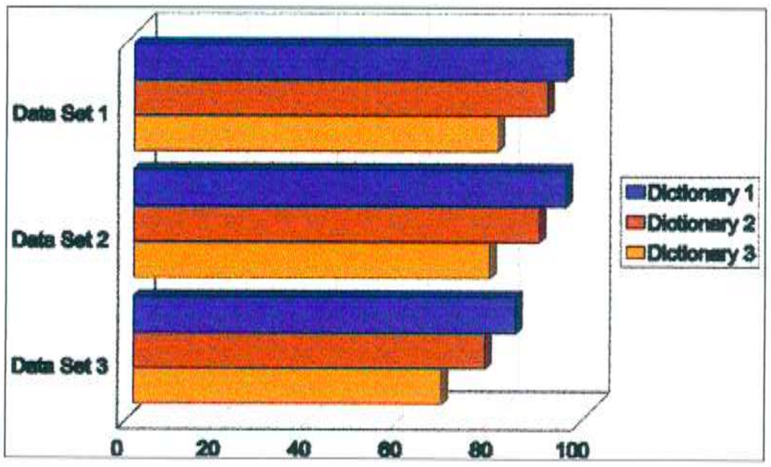

Figure 8.1: Recognition accuracies for different dictionaries

8.2 Error Analysis

In order to get more insight into the nature of the errors **NTime** is making, a detailed analysis was conducted. The errors can be classified into the following categories:

- *Assumption Error*
  Errors occurring due to assumptions made within the system (no direction changes within an edge, at least one endpoint detectable for the leading character).
- *Bad Preprocessing*
  Preprocessing fails to extract the segment graph correctly. This usually results in additional intersection nodes which highly disturb the structure of the graph.
- *Problems with character "t"*
  The calculated sequence branches off into an edge of the "t"-bar during the tracing of the body of "t".
- *Parts left out*
  Edges are left out when the corresponding node is first visited. This leads to more errors as the search tries to include the missing parts later on.
- *One character error*
  One error occurring in one character of the word with the rest of the sequence being correct.
- *Wrong Jump*
  Pen is lifted (and put down) in the wrong position

Figures 8.2 to 8.4 depict some of the error categories. The results of the analysis are summarized in table 8.2 and figure 8.5[2].

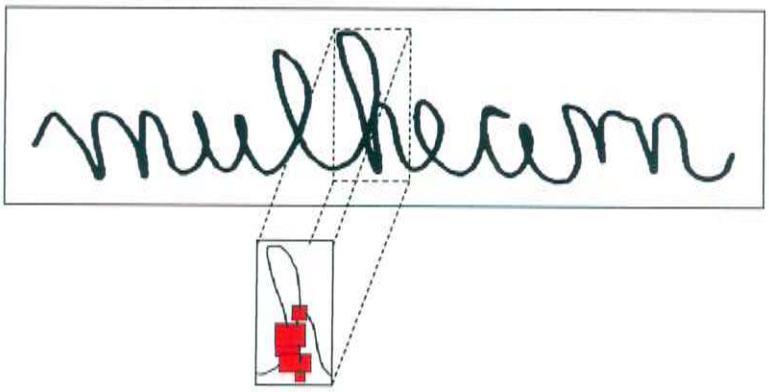

Figure 8.2: Error resulting from bad preprocessing. The segment graph extraction places three nodes on top of each other.

[2]The analysis is based on the manual examination of 85 cases.

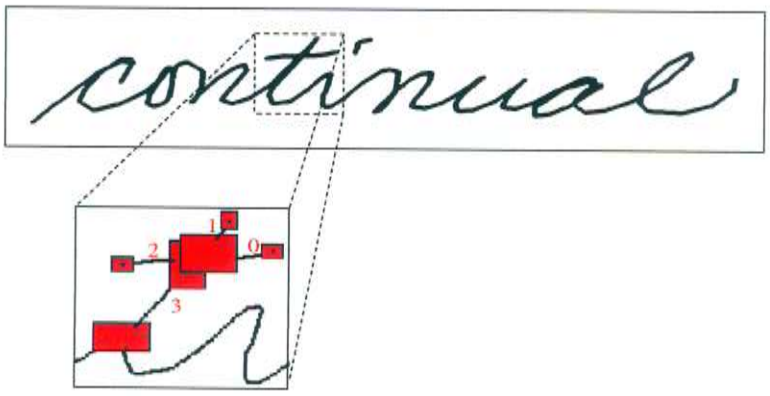

Figure 8.3: Error in character "t". The t-bar is traced before the body of the "t". This results in the erroneous edge sequence 3→2→2→0, where the correct sequence would be 3→1→1→3 and 2→0 some time afterwards.

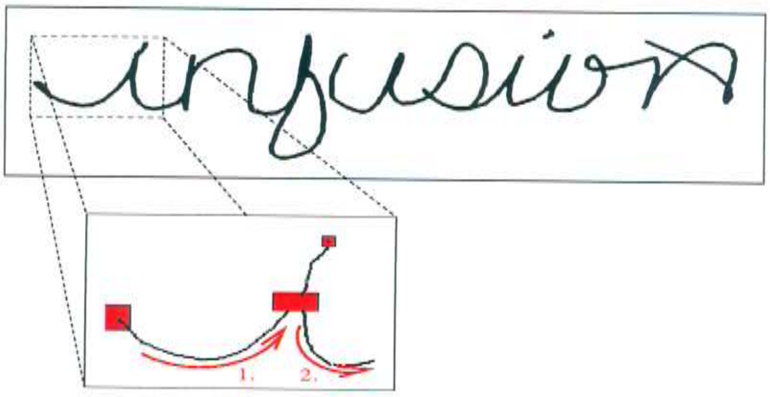

Figure 8.4: Edge is left out from sequence

In the following list, possibilities to reduce the different errors are discussed:

- For most of the *assumption errors* an immediate solution is not obvious. It is especially not clear how to handle directional changes occurring between two nodes. However, the errors emerging from missing endpoints in the leading character can, most likely, be avoided by an additional heuristics targeted at this problem.
- The errors related to *bad preprocessing* can be tackled by a more robust preprocessing. Ideas for this task are mentioned in section 5.5.
- The category of *one character errors* subsumes small irregularities occurring in an otherwise correct sequence. It is reasonable to believe that the robustness of **NPen^{++}** towards these variations can be increased by training the recognizer on data produced by **NTime**.

Error type	Percentage
Assumption Error	7.1%
Bad Preprocessing	15.5%
Character "t"	13.1%
One Character Error	16.6%
Others	1.2%
Parts left out	**22.6%**
Wrong Jump	**23.8%**

Table 8.2: Error analysis with the two most frequent cases highlighted

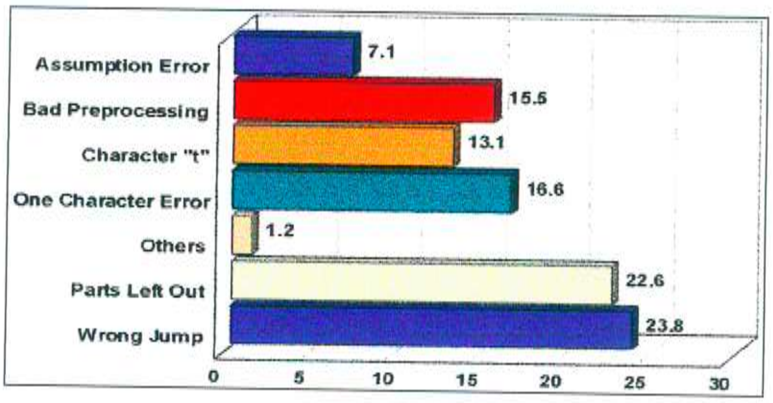

Figure 8.5: Results of the error analysis

- The errors related to the *character "t"* and to *left out parts* are orthogonal to each other. The tracing of "t" demands for relaxed optimization criteria which allow the completion of the missing edges belonging to the "t"-bar after the rest of the word is written. In cases where parts of the graph were left out a more restrictive optimization would be better. It is not clear how this problem can be overcome effectively. Possible solutions include the restriction of pen-ups (as "t"-bars disturb the handwriting recognizer anyhow) and the extension of the optimization criteria.

- Errors emerging from *wrong jumps* also indicate that pen-ups should be more restricted.

8.3 Contribution of this Thesis

This chapter proposed a new heuristic search algorithm for the determination of the best path through the segment graph of a given word. The approach *directly* incorporates the search for the best starting point therefore avoiding the use of a separate procedure. With the help of effective pruning algorithms the described method stays computationally feasible. When applied on the sequences produced by the **NTime** system, the on-line handwriting recognizer **NPen**$^{++}$ achieves recognition accuracies of 84%, 77.3% and 67.8% with dictionary sizes of 120, 1 000 and 5 000 words, respectively.

8.4 Examples

This section provides some examples of traces found by the system.

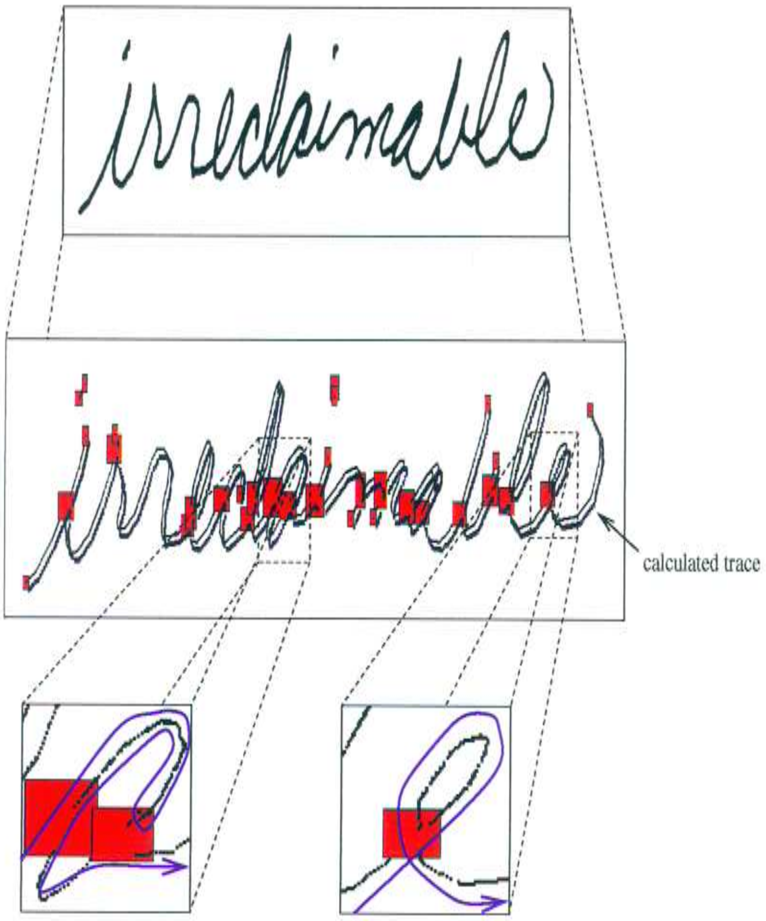

Figure 8.6: Example showing the correct tracing of characters "a" and "e"

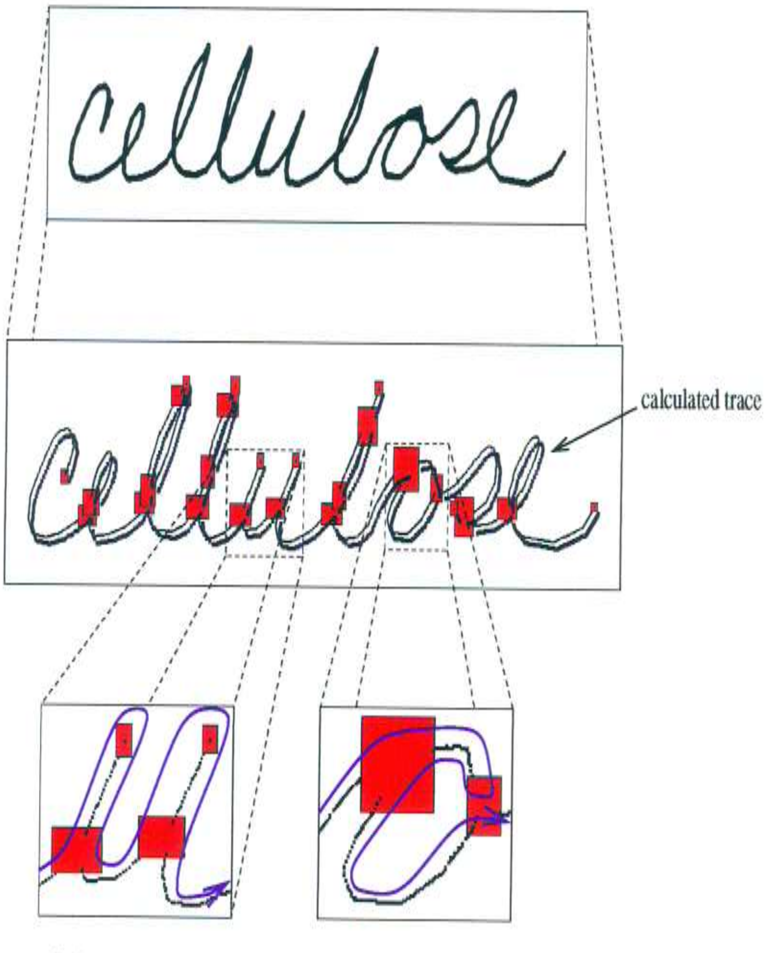

Figure 8.7: Example showing the correct tracing of characters "u" and "o"

Chapter 9

Summary and Future Work

9.1 Summary

The previous chapters proposed a system for the model-based extraction of dynamic writing information from static handwriting. The results presented in this thesis support the following hypotheses:

- **The segment graph is an appropriate representation for handwritten words.**
  Based on bitmap data produced from on-line sequences fast preprocessing algorithms are used for the extraction of contour and segment graph. The segment graph is an abstract representation of the underlying bitmap which provides a maximum of information for subsequent parts of the system.

- **Neural networks are excellent models for nodes in the segment graph.**
  Using the segment graph the core problem of the task can be identified as the determination of the outgoing edge of a node when given the node and the incoming edge. This classification problem is solved using feedforward neural networks. With the feature representation developed in this thesis, the neural networks achieve a classification accuracy of 93.8%.

- **Search on the whole segment graph is computationally feasible**
  Due to the use of a two stage pruning procedure the search on the whole segment graph is feasible. The algorithm determines a set of path hypotheses of maximum length and picks the best one using the handwriting recognizer **NPen^{++}**.

- **The proposed system is able to produce on-line data out of off-line data.**
  The quality of the on-line data generated by the proposed system is measured by using the on-line handwriting recognizer **NPen^{++}**. The results are recognition accuracies of 84%, 77.3% and 67.8% when running **NPen^{++}** with dictionary sizes of 120, 1 000 and 5 000 words, respectively. The corresponding recognition

rates for the original on-line sequences are 95%, 90.8% and 80%, which averages to a loss of 12.2% in accuracy when using the sequences calculated by **NTime**.

9.2 Future Work

Throughout this report a number of possible extensions of the methods currently used within the system were indicated. The following list stresses the most important points.

- *Extension of the scope of the input data*
  The system is currently limited to cursive words. It is planned to extend the scope of the input data by including printed and mixed style words. While the preprocessing is relatively unaffected by the change of the input, the neural networks have to be retrained with the new data. The fact that pen-ups are occurring more frequently within printed words should be reflected by an additional output class for the neural networks.

- *Different neural network architectures*
  The classification problem of finding the outgoing edge of a node given the input edge was solved by applying standard backpropagation feedforward networks. This leaves room for investigations of different network architectures possibly with different feature representations.

- *Training of handwriting recognizer* **NPen^{++}**
  It has to be assumed that sequences recorded from a digitizer tablet and sequences produced by the system proposed in this thesis are always going to look different. In order to increase the recognition accuracy of **NPen^{++}** on data converted from bitmaps, the recognizer has to be retrained on this data. It is reasonable to believe that this is also going to increase the tolerance of the handwriting recognizer towards small variances in the sequences generated by **NTime**.

- *Graph algorithms for search*
  Some of the work reviewed in chapter 3 successfully uses graph algorithms. It is planned to investigate possibilities to incorporate standard graph algorithms into the search engine.

List of Figures

List of Tables

Bibliography

[AHD96] I.S.I. Abuhaiba, M.J.J Holt, and S. Datta. Processing of binary images of handwritten text documents. *Pattern Recognition*, 29(7):1161–1177, 1996.

[BASS97] H. Bunke, R. Ammann, M. Schenkel, and R. Seiler. Recovery of temporal information of cursively handwritten words for on-line recognition. In *Proceedings of the International Conference on Document Analysis and Recognition*, pages 931–935, 1997.

[BCCM93] G. Boccignone, A. Chianese, L.P. Cordella, and A. Marcelli. Recovering dynamic information from static handwriting. *Pattern Recognition*, 26(1):409–418, 1993.

[Bis96] C.M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, Oxford, England, 1996.

[BNNB94] J.B. Bellegarda, D. Nahamoo, K.S. Nathan, and E.J. Bellegarda. Supervised hidden markov modeling for on-line handwriting recognition. In *Proceedings of the International Conference on Accustics, Speech and Signal Processing*, volume 5, pages 149–152, 1994.

[Cap84] D.W. Capson. An improved algorithm for the sequential extraction of boundaries from a raster scan. *Computer Vision, Graphics and Image Processing*, 28(1):109–125, October 1984.

[CP92] C. Chouinard and R. Plamondon. Thinning and segmenting handwritten characters by line following. *Machine Vision and Applications*, (7):185–197, 1992.

[DR93] D.S. Doermann and A. Rosenfeld. The interpretation and reconstruction of interfering strokes. In *International Workshop on Frontiers in Handwriting Recognition*, pages 41–50, 1993.

[DR95] D.S. Doermann and A. Rosenfeld. Recovery of temporal information from static images of handwriting. *Internation Journal of Computer Vision*, 15:143–164, 1995.

[Far93] G.E. Farin. *Curves and Surfaces for Computer Aided Geometric Design.* Academic Press, 3rd edition, 1993.

[FKH+76] Y. Fujimoto, S. Kadota, S. Hayashi, M. Yakamoto, S. Yajima, and M. Yasuda. Recognition of handprinted characters by nonlinear elastic matching. In *Proceedings of the International Conference on Pattern Recognition*, pages 113–118, 1976.

[Fre83] J.J. Freyd. Representing the dynamics of a static form. *Memory and Cognition*, (4):342–346, 1983.

[Fri96] J. Fritsch. Modular neural networks for speech recognition. Technical Report CMU-CS-96-203, Carnegie Mellon University, August 1996.

[FvFH90] J.L. Foley, A. van Dam, S.K. Feiner, and J.F. Hughes. *Computer Graphics: Principles and Practice.* Addison-Wesley, 2nd edition, 1990.

[Gro97] R. Groß. Run-on recognition in an on-line handwriting recognition system. Project Report, University of Karlsruhe, June 1997.

[HKP91] J. Hertz, A. Krogh, and R.G. Palmer. *Introduction to the Theory of Neural Computation.* Addison-Wesley, 1991.

[Hol81] J.M. Hollerbach. An oscillation theory of handwriting. *Biological Cybernetics*, 39:139–156, 1981.

[Hue97] W. Huerst. Repair in on-line handwriting recognition. Master's thesis, University of Karlsruhe, 1997.

[HW92] P. Haffner and A. Waibel. Multi-state time delay neural network for continous speech recognition. In *Advances in Neural Information Processing Systems (NIPS-4).* Morgan Kaufman, 1992.

[HY95] T. Huang and M. Yasuhara. A total stroke slalom method for searching for the optimal drawing order of off-line handwriting. In *Proceedings of the International Conference on Systems, Man and Cybernetics*, volume 3, pages 2789–2794. IEEE, 1995.

[Kas95] R.H. Kassel. *A Comparison of Approaches to On-line Handwritten Character Recognition.* PhD thesis, Massachusetts Institute of Technology, June 1995.

[LLS92] L. Lam, S.-W. Lee, and Y. Suen. Thinning methodologies - a comprehensive survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(9):869–885, September 1992.

[LP92] S. Lee and J.C. Pan. Offline tracing and representation of signatures. *IEEE Transactions on Systems, Man, and Cyneretics*, 22(4):755–771, 1992.

[LR95] B. Lüdemann-Ravit. Extraktion von temporaler Information mit Hilfe eines symbolischen Lernverfahrens in der Offline-Handschriftenerkennung. Master's thesis, University of Karlsruhe, November 1995.

[MFW94] S. Manke, M. Finke, and A. Waibel. Combining bitmaps with dynamic writing information for on-line handwriting recognition. In *Proceedings of the International Conference on Pattern Recognition*, 1994.

[MFW95] S. Manke, M. Finke, and A. Waibel. NPen^{++}: A writer independent, large vocabulary on-line cursive handwriting recognition system. In *Proceedings of the International Conference on Document Analysis and Recognition*. IEEE Computer Society, 1995.

[MFW96] S. Manke, M. Finke, and A. Waibel. A fast search technique for large vocabulary on-line handwriting recognition. In *International Workshop on Frontiers in Handwriting Recognition*, Colchester, England, 1996.

[Mit97] T. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[MOD85] E. Mandler, R. Oed, and W. Doster. Experiments in on-line script recognition. In *Image Analysis: Proceedings of the 4th Scandinavian Conference*, pages 77–86, 1985.

[O'G88a] L. O'Gorman. An analysis of feature detectability from curvature estimation. *Computer Vision and Pattern Recognition*, pages 235–240, June 1988.

[O'G88b] L. O'Gorman. Curvilinear feature detection from curvature estimation. In *International Conference on Pattern Recognition*, pages 1116–1116, 1988.

[OI92] R. Ogniewicz and M. Ilg. Voronoi skeletons: Theory and application. In *Proc. of Computer Vision and Pattern Recognition (CVPR)*, pages 63–69. IEEE, 1992.

[Pla89] R. Plamondon. A handwriting model based on differential geometry. In R. Plamondon, C. Y. Suen, and M. Simner, editors, *Computer Recognition and Human Production of Handwriting*, pages 179–192. World Scientific, 1989.

[Pla95] R. Plamondon. A delta-lognormal model for handwriting generation. In *Proceedings of the Seventh Biennal Conference of the International Graphonomics Society*, pages 126–127, 1995.

[Qui93] J.R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, Inc., San Mateo, California, 1993.

[Rab89] L.R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proc. IEEE*, 77(2):257–286, February 1989.

[Sen94] A. W. Senior. *Off-line Cursive Handwriting Recognition using Recurrent Neural Networks*. PhD thesis, University of Cambridge, September 1994.

[SGH94] M. Schenkel, I. Guyon, and D. Henderson. On-line cursive script recognition using time delay neural networks and hidden markov models. In *Proceedings of the International Conference on Accustics, Speech and Signal Processing*, volume 2, pages 637–640, 1994.

[SLN+93] C.Y. Suen, R. Legault, C. Nadal, M. Cheriet, and L. Lam. Building a new generation of handwriting recognition systems. *Pattern Recognition Letters*, 14:303–315, April 1993.

[SMSC94] T. Starner, J. Makhoul, R. Schwartz, and G. Chou. On-line cursive handwriting recognition using speech recognition methods. In *Proceedings of the International Conference on Accustics, Speech and Signal Processing*, volume 5, pages 125–128, 1994.

[ST94] Y. Singer and N. Tishby. Decoding cursive scripts. In J. D. Cowan, G. Tesauro, and J. Alspector, editors, *Advances in Neural Information Processing Systems 6. Proceedings of the 1993 Conference*, pages 833–840, San Francisco, CA, 1994. Morgan Kaufmann.

[TSW90] C.C. Tappert, C.Y. Suen, and T. Wakahara. The state of the art in on-line handwriting recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12:787–808, August 1990.

[WD84] K. Wall and P.-E. Danielsson. A fast sequential method for polygonal approximation of digitized curves. *Computer Vision, Graphics and Image Processing*, 28:220–227, 1984.

[WHH+89] A. Waibel, T. Hanazawa, G. Hinton, K. Shinao, and K. Lang. Phoneme recognition using time-delay neural networks. *IEEE Transactions on Acoustics, Speech and Signal Processing*, March 1989.

[Zel94] A. Zell. *Simulation Neuronaler Netze*. Addison-Wesley, 1994.