# An On-Line Handwriting Recognizer with Fisher Matching, Hypotheses Propagation Network and Context Constraint Models

By

Jong Oh

A dissertation submitted in partial fulfillment of

the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

New York University

May 2001

_____

Davi Geiger

# DEDICATION

*To my late aunt Jung-Ja Cho*

*and my cousin Susan*

# ACKNOWLEDGEMENTS

I thank my advisor Davi Geiger for his support and helpful feed-backs on research. Especially his openness to new ideas was a great enabling factor in initiating this project. I also appreciate Mark Pipes, who was a visiting student from MIT during the last summer, for his coding help for experiments and his creative contribution in the idea of visual context modeling.

# ABSTRACT

We have developed an on-line handwriting recognition system. Our approach integrates local bottom-up constructs with a global top-down measure into a modular recognition engine. The bottom-up process uses local point features for hypothesizing character segmentations and the top-down part performs shape matching for evaluating the segmentations. The shape comparison, called Fisher segmental matching, is based on Fisher's linear discriminant analysis. The component character recognizer of the system uses two kinds of Fisher matching based on different representations and combines the information to form the multiple experts paradigm.

Along with an efficient ligature modeling, the segmentations and their character recognition scores are integrated into a recognition engine termed Hypotheses Propagation Network (HPN), which runs a variant of topological sort algorithm of graph search. The HPN improves on the conventional Hidden Markov Model and the Viterbi search by using the more robust mean-based scores for word level hypotheses and keeping multiple predecessors during the search.

We have also studied and implemented a geometric context modeling termed Visual Bigram Modeling that improves the accuracy of the system's performance by taking the geometric constraints into account, in which the component characters in a word can be formed in relation with the neighboring

characters. The result is a shape-oriented system, robust with respect to local and temporal features, modular in construction and has a rich range of opportunities for further extensions.

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# CHAPTER 1. Introduction

Handwriting has been a medium for communicating messages and ideas between people, across space and time, from the time when the first alphabet was invented a few thousand years ago. In ancient times, the capability of handwriting was a privilege available only to a small number of specially trained scribes and scholars, and practically all documents and publications had been handcrafted by them until the invention of the printing press. As the literacy rate improved, handwriting had served as a mainstay means of persistent communications until the advent of the typewriter. Nowadays, even with the modern technologies like word processors, fax machines and electronic mail, handwriting has survived as a useful and versatile communication method because of the ubiquity and the convenience of pen and paper in various everyday situations. In the future, handwriting may only thrive more because of the technological developments under way that intend to establish handwriting as a new mode for humans to communicate with computers.

Handwriting has long been studied by numerous disciplines for various different aspects and purposes, and they include experimental psychology, neuroscience, engineering, computer science, anthropology, education, forensic

science, etc ([57], [58], [68], [69], [75], [76], [77]). From the computer science perspective, the types of analyses involved are the recognition, the interpretation and the verification of handwriting. Handwriting recognition is the task of transcribing a language message represented in a spatial form of graphical marks, into a computer text, for example, a sequence of 8-bit ASCII characters. Handwriting interpretation is the task of determining the most likely meaning of a given body of handwriting, for example a mailing address written on an envelope. This can be regarded as a more general level of handwriting recognition and uses semantic context information to resolve the ambiguities arising from the multiple possible ways the input can be interpreted. Handwriting verification is the task of determining whether or not a given handwriting belongs to a particular person and can have use, for example in forensic investigation. Signature verification can be considered as a sub-field of handwriting verification and deals with a special type of handwriting, that is, people's signatures and has applications as a means of identification and security ([59]). Handwriting recognition tasks fall in two broad categories: one is on-line recognition and the other is off-line recognition.

## *1.1 On-Line Handwriting Recognition*

On-line handwriting recognition assumes that a transducer device is connected to the computer and is available to the user. The transducer converts the

user' s writing motion into a sequence of signals and sends the information to the computer. The most common form of the transducer is a tablet digitizer. A tablet consists of a plastic or electronic pen and a pressure or electrostatic-sensitive writing surface on which the user forms one' s handwriting with the pen. Sampling the movement of the pen-tip, the digitizer is able to detect information like *x* and *y* coordinates of a sampled point, the state of whether the pen touches the surface (pen-down) or not (pen-up). The information is sent to the connected computer for recognition processing (Figure 1). A "stroke" in on-line data is defined as a sequence of sampled points from the pen-down state to the pen-up state of the pen, and the completed writing of a word consists of a sequence of one or more strokes. A "digital ink" is the display of the strokes on the computer screen. By digital ink, the user can see what he or she writes on the tablet and it provides interactivity between the user and the computer. For example, by manipulating the digital ink, the user can correct or edit one' s writing in an interactive manner. One natural application of on-line handwriting recognition is as an alternative input method to the computer. In English, the size of the alphabet is relatively small and the language fits well for keyboard entry, but for a language like Chinese that has a much larger alphabet, using a keyboard is a non-trivial challenge. In addition, for the new trend of small form factor computers and devices used for mobile computing, carrying a keyboard, even in miniaturized from, is becoming less and

less an option. Another application is as a more natural and easier-to-use interface to the tasks involving complex formatting, like entering and editing equations, and drawing sketches and diagrams.

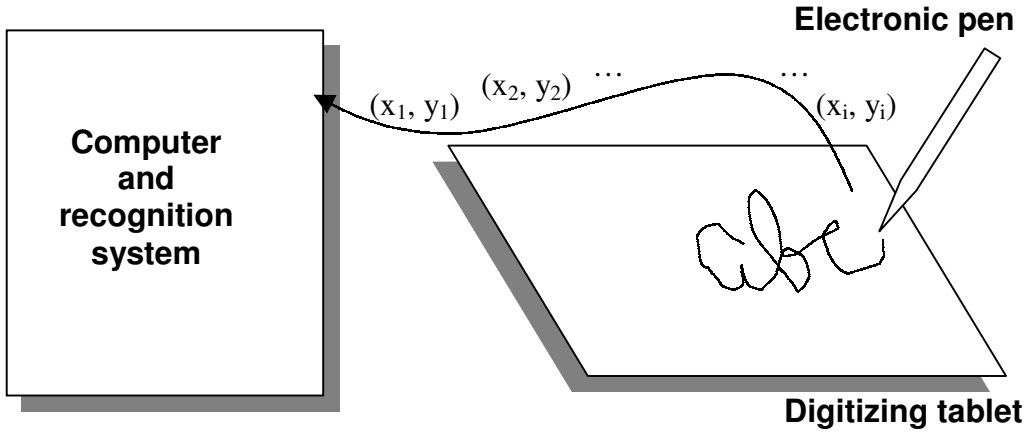



**Figure 1.** A tablet digitizer, input sampling and communication to the computer.

### *1.2 Off-Line Handwriting Recognition*

Off-line handwriting recognition can be regarded as an extended field of OCR (Optical Character Recognition) and lacks the interactive nature of on-line handwriting recognition provided by the digital ink. In OCR, machine-printed material is scanned into a computer file in two-dimensional image representation, while off-line handwriting recognition deals with hand-written messages instead of printed publications. Off-line data is two-dimensional in structure because of its

image representation and has a typical size of a few hundred kilobytes per word. Since an image has no granted provision to distinguish its foreground and background, the first step of an off-line recognition, called "thresholding" ([42], [52], [65]), is to separate the foreground pixels from the background in the input. Unlike on-line handwriting, a written image also has a line thickness whose width depends on the writing instrument used and the scanning process. Hence the next processing step is to apply a class of techniques called "thinning" or "skeletonization" ([37], [61]) which tries to shed out redundant foreground pixels from the input. These early preprocessing steps are necessary for off-line recognition but are in general expensive computationally and imperfect, and may introduce undesirable artifacts in the result, for example, "spurs" in the thinning process ([37], [61]). The applications of off-line handwriting recognition include

(a) (b)

**Figure 2.** (a) Original off-line image (b) the result with some spurs after applying thresholding and thinning.

reading handwritten mail addresses in automated postal sorting, reading bank check amounts, automatic processing of handwritten forms and interpretation of handwritten notes and manuscripts.

### *1.3 Comparisons of On-Line and Off-Line Recognition*

An aspect of on-line handwriting recognition that sets it apart from off-line handwriting recognition, OCR or other image recognition tasks, is the temporal input sequence information provided directly by the user. The digitizer naturally captures the temporal ordering information when it samples the points on the contour that the user is forming. Hence on-line data has one-dimensional structure and has a typical size of a few hundred bytes per word. This dynamic information provides clean foreground separation and perfect thinning, and the on-line recognition can bypass the preprocessings that are required by the off-line recognition processing. Also the difference in input representation leads to large difference in the size of the input data. As mentioned above, eon-line data, in general, is at least an order of magnitude more compact compared to off-line data because of the different dimensionalities in representation. The difference in the data size also results in substantial difference in the processing time.

Another important advantage of on-line data is that its sequence information makes the character boundary segmentation easier to do. After the preprocessing stage, most handwriting recognizers, whether on-line or off-line, try to break its input into intervals corresponding to hypothetical characters and apply an evaluation method to the intervals. The recognition performance of the system has a substantial dependence on the quality and the robustness of the character segmentation. Due to the cues available from the temporal ordering built into its input data, an on-line recognizer has a non-trivial advantage in generating segmentations reliably and efficiently. For example, when two neighboring characters overlap in the respective occupying regions, it is much harder for an

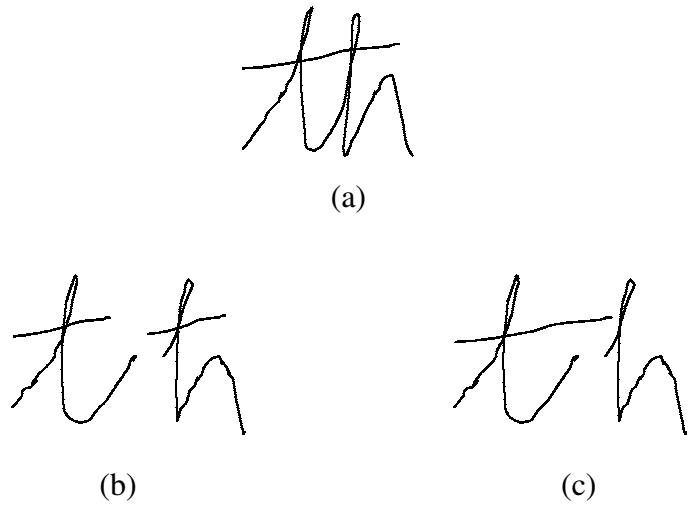



**Figure 3.** (a) A pair of characters "th" each of whose regions overlapping in space (b) spatial separation of "t" and "h" (c) temporal separation of "t" and "h"

off-line recognizer to segment them correctly because any simple geometric separation will contain a part of the other character (see Figure 3). For an on-line recognizer, the problem is easier to handle since the boundaries of the two characters may overlap spatially but never in time. Meanwhile, an advantage of off-line recognition' s image representation is that it is insensitive to variations in the ordering of the strokes contained in handwriting. See Figure 4. That is, the same handwriting may have been formed in many different orders of strokes, but the completed written image looks the same and has the same representation. This is not the case for on-line data since different orderings of the strokes will result in

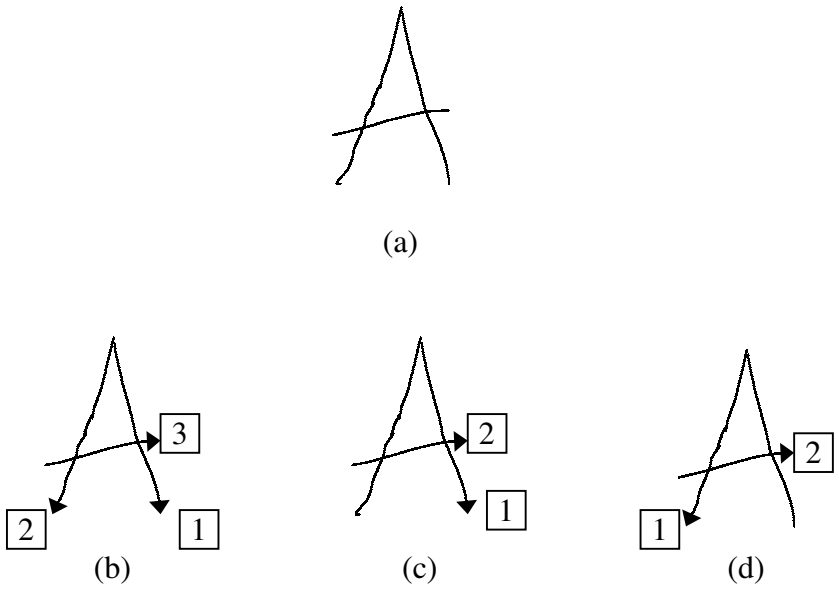



**Figure 4.** (a) A written image of "A" and the three different orders that it could have been written indicated in boxed numbers in (b), (c) and (d).

different representations even though the completed image is the same. Fortunately, each character class has certain regularity in stroke orderings so that the number of different stroke orders is not large in most cases. In overall comparison, the advantages of on-line handwriting recognition outweigh its disadvantages and on-line recognizers achieve consistently higher accuracy and run faster than the off-line recognizers do. Because of the benefits of on-line recognition, some efforts have studied the interchangeability of the representations ([14], [1], [49], [60]). The rationale is that if we have a means to convert off-line data to an on-line version and apply the on-line processing techniques, then we would achieve a level of performance comparable to on-line recognition, on the off-line data. In addition, we would have a unified paradigm that handles both types of recognition tasks using the same system. The essence of such a conversion would be the recovery of temporal ordering of the trajectories in the input image. However, the interchangeability has proven to be asymmetric: the conversion from on-line data to off-line version is not hard but the other direction has turned out to be difficult and has led to only limited success.

## *1.4 Constraints on Handwriting Style and Vocabulary*

In terms of the constraint on the input writing style, handwriting recognition, on-line or off-line, has three different modes supporting printed, cursive and mixed styles. See Figure 5. Printed style recognition is the easiest because each character of handwriting in such style has a clearer boundary with its neighbors. For example, the characters in printed style writing are usually separated by a pen-up signal in on-line recognition. The extra assumptions that we can take about printed style handwriting make the segmentation step easier to perform. In cursive script recognition, however, most of the component characters are connected to their neighbors by a kind of sub-stroke called "ligature" that is not part of any letters but a connecting pattern between two letters. In this situation, it

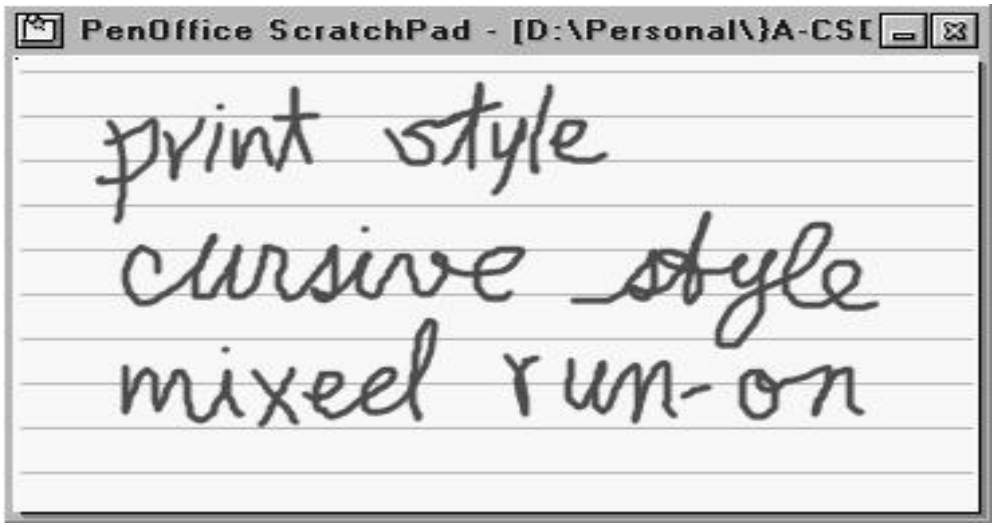



**Figure 5.** Three different handwriting styles.

is much more challenging to hypothesize about the character segmentation since there is scarcer information hinting at the likely segmentation boundaries. Printed style mode can be regarded as a subset of cursive mode recognition except for some idiosyncracies, and the mixed mode can be obtained as a by-product of obtaining both printed and cursive mode supports. Therefore the hardest problem is the cursive mode recognition.

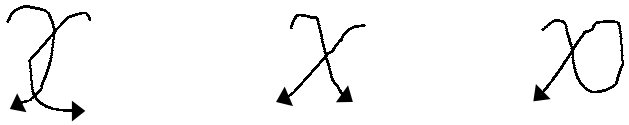
Another important constraint on handwriting recognition is the size of the vocabulary supported by the system. Almost all modern handwriting recognizers use a dictionary to improve the accuracy. The hypothetical characters identified by the system can easily be very ambiguous even to human eyes, when they are considered in isolation. Also many candidates that are mis-hypothesized by the system may look plausible as real characters when viewed in isolation from the context. Therefore the ambiguities are ever-present epiphenomena of the recognition process that the system needs to cope with and the neighboring context offers valuable information for the resolution. Other than the visual context, the lexicon provides the linguistic constraints that specify the legal strings permissible according to the vocabulary. By pruning the search space using the lexicon, many spurious hypotheses will not be generated and the system will need to handle a smaller number of legal candidates. This not only improves the accuracy of the system, but also speeds up the recognition performance due to the economy

achieved by the search focus. However, the size of lexicon has an inversely proportional relationship with the system' s accuracy because, with larger vocabulary, there will be more legal candidates generated and more room for ambiguities and mistakes. Therefore it is important for a handwriting recognizer to handle a large lexicon and scale up easily and robustly as more words are added to the dictionary.

### *1.5 Writer-Independence vs. Writer-Dependence*

Another taxonomy in handwriting recognition is the classes of writer-independent and writer-dependent systems. Writer-independence means that the system can handle the idiosyncrasies of multiple people' s writing styles, and a writer-dependent system is trained and optimized to recognize a single person' s writing. Within a character class, there can be more than one subclass of the class each of which has substantial difference from the others in visual shape. This subclass standing for a representative variability within a class is called "allograph." See Figure 6. Therefore each character class consists of one or more allographs and as the number of people increases, there will need to be more allographs covering the range of personal styles among them. Automatically identifying a good set of allographs is a challenging task and takes a huge number of samples for adequate construction. Also larger number of allographs means

(a)



(b)

**Figure 6.** (a) Two different allographs of "a" and (b) three allographs of "x."

more processing time and more opportunity for confusion. On the other hand, a writer-dependent system is trained with only one user and expectedly has less variability in the writing data, leading to smaller number of allographs and higher accuracy.

Most likely, writer-dependence may not be very meaningful to off-line recognition systems because of the nature of many of their applications like postal code sorting and recognizing the amounts of bank checks. In the case of on-line recognition, writer-dependence makes more sense since the system will typically serve as an input method to computers used personally by single user, that is, not intended to be shared by multiple users. Ideally such an on-line recognizer will

have a certain level of default writer-independence to start with. Thereafter, the system will be activated, either automatically or by the user, for personal training and the recognizer will learn the particularities of the owning user' s writing style, thereby picking up extra recognition accuracy. This process is termed "user-adaptation" and is an attractive and desirable property. The availability of user adaptation, however, is critically influenced by the kind of techniques used by the character recognition. Practically, user adaptation needs to be done incrementally with relatively small amounts of data in a short enough training time tolerable by the user. Therefore a data-hungry and time-intensive training paradigm like a neural network is not suitable for such purpose.

## *1.6 Functional Topics of Handwriting Recognition*

We have identified below six functional topics that any word or higher-level handwriting recognizer, whether on-line or off-line, requires for performance. The data normalization is the front end of the system and performs noise suppression and regularizes the input data variability like size, translation, and rotation. The feature extraction computes the target features and plugs them in the internal representation of the system. The segmentation module generates the hypothetical character or sub-unit segmentations consumed by the recognition

engine. The component character recognizer evaluates a given segmentation into class-labeled scores determined by the recognizer' s internal metric. The recognition engine is the place where the various information from the rest of the system is all integrated to generate and propagate word level hypotheses. The

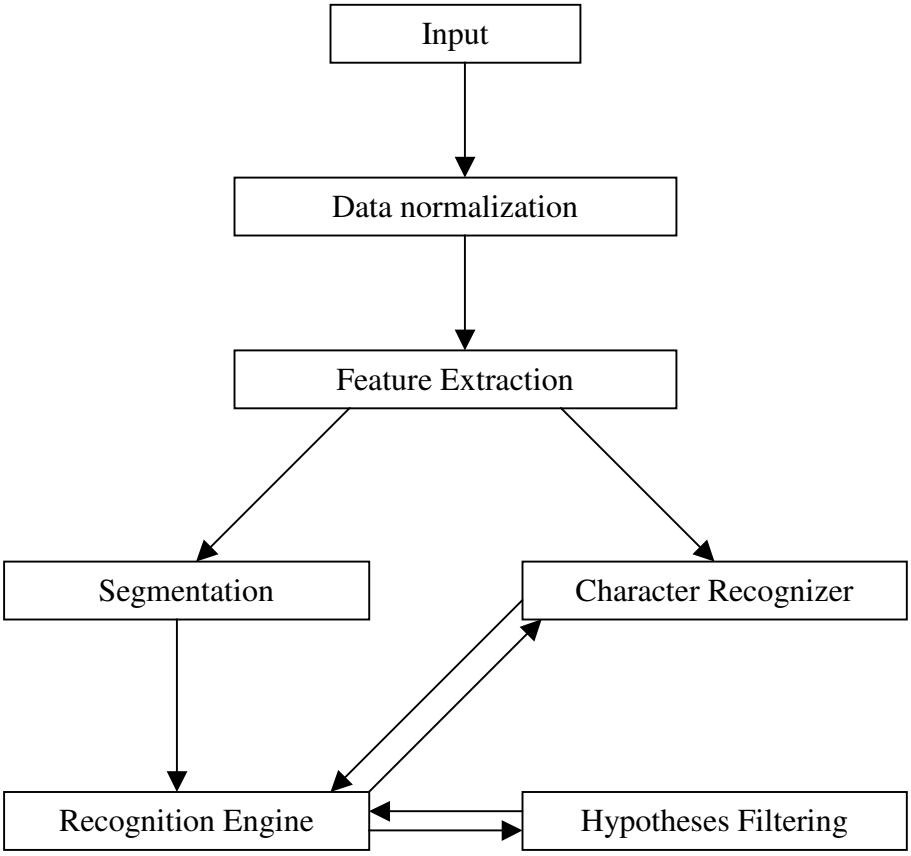



**Figure 7.** Block diagram showing the module entities and the flow of control and information between modules.

hypotheses filtering, or postprocessing, is used both to improve the accuracy and to speed up the recognition performance by controlling the amount of search performed by the recognition engine. Typically, the recognition engine generates large number of hypotheses, many of them spurious, so the filtering applies pre-arranged models either to eliminate or to rearrange the worthiness of the hypotheses. The block diagram of Figure 7 illustrates each module standing in relation to the entire system and the flow of control and information between the modules, which are actually representative of our system. The module entities and the actual control flow among the modules may depend on the specific approaches taken by a system. For example, by taking implicit segmentation strategy (see CHAPTER 4) there will be no segmentation module standing separately, and instead it will be merged with the recognition engine. Another example is that the feature extraction may have local focus as in our system so that the character recognizer needs to communicate with the module each time it is invoked.

## *1.7 Previous Works and Contributions of the Thesis*

The research focus in this thesis will be to develop a solution to the hardest problem in natural handwriting recognition: an on-line cursive script recognizer that has arbitrarily scalable vocabulary. Traditionally, most handwriting recognition researches have concentrated on the study of isolated character

recognition and relatively little effort has been spent on the rest of the process ([6], [13], [27], [39], [67], [71]). There may have been two reasons for this. Firstly the community may have over-relied on the potential contribution from the discriminative power of isolated level character recognizer. It is now being realized that the ambiguities and the spuriousness encountered during the recognition process are better and more naturally resolved by drawing relevant information from the context rather than trying to put the discriminative capacity of the character recognizer to the limit. Secondly, the community may have under-estimated the complexity of the string level recognition and tried to reuse the established standard like Hidden Markov Model. The character recognizer indeed plays an important role in the process, but the experiences in the field indicate that more orchestrated and higher level integration of diverse information from the rest of the system is in strong demand to accomplish higher performance and robustness. The center of such information integration is the recognition engine and ours is a general model geared for the stated focus. In addressing these issues, we will develop, demonstrate and claim the following contributions.

### 1.7.1 Fisher Discriminant Analysis Based Character Recognizer

Fisher discriminant analysis (FDA) ([15], [64]) is an improvement of more conventional linear projection methods like Principal Component Analysis (PCA).

A notable weakness of PCA is that the projection it performs scatters data in the projection space without considering of the class specific distribution structures. In contrast, the projection matrix of FDA is constructed by taking the class specific regularities into account. More specifically the FDA tries to maximize the between-class scatter while minimizing the within-class scatter in the projection space. The result is a clearer class boundaries and thus easier separation between the classes. While the principle has been known for decades, practical application dealing with high dimensional representation space had not been tried until recently when the face recognition community used it successfully ([3]).

A substantial advantage of using linear techniques like FDA is that the training is much faster and requires relatively smaller amount of training data compared with the more popular methods like neural networks and hidden Markov models described in CHAPTER 5. Therefore it has a potential to make user-tailored training feasible. The FDA will be used in this thesis as the base for the character recognizer for the first time for letter and word level performance in handwriting recognition.

### 1.7.2 Multiple Experts Fusion for the Character Recognizer

A problem with using the linear techniques for a character recognizer in string level recognition task has been that the character scores computed by such a

technique have a linear distribution so that it is not easy to determine which characters to exclude from the candidate list. Therefore such a recognizer needs to retain larger number of candidates for adequate accuracy. This is in turn slows down the recognition engine since it will generate more word level hypotheses.

We will address this issue by the use of multiple experts fusion paradigm. More specifically, the character recognizer in our system will use two FDA recognizers each working on different representation. The recognition results from the two FDA recognizers will be integrated at a fusion module that combines the information and produces the final recognition result. Conventional fusion approaches typically involve designing the topology of interconnections between the recognizers ([1], [28], [63]), therefore the overall mechanism is implicit. Our information fusion is unique in the sense that it has an explicit strategy of how to re-compute the new scores and thereby reshuffle the ranks of the final candidates. We will show how the fusion process proceeds and will demonstrate its effectiveness by the experimental result showing much reduced candidate set, also with improved accuracy.

### 1.7.3 Word-Level Hypothesis Evaluation Taking the Average of the Component Character Scores

The conventional method of evaluating string level hypothesis in handwriting recognition has been accumulating the confidence values assigned to each string component ([6], [13], [27], [39], [71]). The accumulation metric, however, has a weakness when the input contains a black-out interval, that is an ill-formed region for example by mistake. The accumulative scoring may also have the undesirable preference to longer or shorter (depending on the nature of the metric) hypotheses. This is one place the handwriting recognition community has not paid the due attention and the researchers have assumed the accumulation metric by default. We will take examples showing superior robustness of the components average based hypothesis metric, to the accumulation metric. The effectiveness of the new metric will be demonstrated when we will illustrate the experimental results on our recognition engine.

### 1.7.4 Hypotheses Propagation Network (HPN)

The standard Viterbi search runs optimally assuming that word evaluation metric is perfect. But the assumption is not practical in reality and other than the metric, there are many sources of noise like perturbations in data acquisition, in feature extraction, ambiguities and confusions in the character recognition process

and so on, all disturbing the Viterbi search. While this is a well-known problem in the statistical pattern recognition setting ([6], [13], [27], [39], [71]), no previous work has tried to address the soundness of the fundamental assumption or its remedy. Our strategy to cope with this problem is to allow more than one predecessor hypotheses at each propagation point in our recognition engine termed Hypotheses Propagation Network. The rationale is to compensate the expected sub-optimality due to the noises in the hypotheses evaluation by keeping more options open for determining the predecessors. The effectiveness of this approach over the conventional Viterbi search will be demonstrated when we illustrate the experimental result on word performances. The HPN also has modular design in the sense that various information coming from other parts of the system are handily integrated to make the management of the hypotheses propagation easier and efficient.

### 1.7.5 Feature-Link Code

Using our high-curvature based segmentation, we identify each segment between the two consecutive segmentation points as "feature-link." We have 24 convexity-directional pattern templates each having unique index. Using a template matching, we label each feature-link with an index of the most closely matching template. The result is a compact sub-character level representation of

the entire input in terms of the feature-link labels. This is different from the conventional directional code ([27], [71]) representation in two important ways. Firstly, the feature-link code can represent convexity and the direction code does not. Secondly, the direction code represents only equal length intervals while our code represents variable length intervals. Therefore the feature-link code is more natural and informed break-up of the input strokes than the conventional one. We will use the feature-link label representation for our efficient ligature modeling.

### 1.7.6 Ligature Filtering Modeling

Although the ligatures are big source of variability in the shapes of characters written continuously, not much effort has been expended on handling them. Most approaches simply ignored them ([7]) and others trying to deal with the ligatures mainly used hidden Markov models ([27], [35], [71]). In this thesis we propose a more efficient and intuitive modeling deriving from the feature-link coding mentioned above. We will demonstrate the effectiveness of our modeling by showing the word performance results with and without the modeling.

### 1.7.7 Visual Bigram Modeling (VBM)

VBM is a context constraint model drawing from the geometric properties of neighboring characters. This is another important opportunity promising substantial gain in system performance, yet has not been much explored in the field. The two previous efforts available from the literature have tried to either estimate and use ascender or descender sub-strokes of writing ([55]), or rely on the covariance matrix of the model parameters ([79]). We will show a new and more intuitive approach that does not need such information and the model parameters are learned by training on the data samples. Verifying the effectiveness of VBM, we will show a substantial increase in the word performance results when it is activated.

# CHAPTER 2. Data Acquisition and Normalization

As mentioned in CHAPTER 1, the tablet digitizer reports the sequence of ($x$, $y$) coordinate pairs to the recognition system, as the user writes on the tablet surface with the pen. The points represented by the coordinates come from the dynamic contour formed by the user writing, with a sampling frequency set up for the digitizer. The sampling rate is typically at least 100 Hz (i.e. one hundred points per second). An important information from the digitizing hardware other than the point coordinate pairs is the pen-down and the pen-up signals regarding when the user puts the pen on the writing surface and when one lifts up the pen, respectively. This information is used to determine where a stroke starts and ends and the overall handwriting data is represented as a sequence of strokes. Depending on the type of a digitizer it may be able to provide more information such as pen-pressure and pen-tilt, and with programming one can also compute the pen movement speed. Most systems including ours, however, use only the coordinates and pen-up/down signals.

The point coordinates provided by the digitizer are integer numbers with reference to the origin of the tablet's coordinate system, so its digital ink, when magnified, has jagged lines. Other than the limiting accuracy of the digitizer,

noise can originate from the digitizing process, hardware error, or erratic hand motion and pen-down indication. The same characters or words written by different users can vary greatly in size, shape and distortion. Even the same writer may write in substantially different ways depending on situations or over time. So the first task of a recognition system is to suppress noise and reduce the variability in the raw data for easier and standardized processing in later stages.

## *2.1 Gaussian Smoothing*

Smoothing is the technique to suppress the quantization noise of the point sampling, which averages an input point with its neighbors with some weighted mask. The primary purpose of smoothing at least in our system is to get more fine-grained real-number coordinates instead of the integer numbers in the raw data, so that the point curvatures can be computed reliably. Since our character segmentation strategy (CHAPTER 4) is based on determining the critical points by high curvature, it is important to computer the point curvatures accurately and reliably. The average can be a simple mean or can be obtained by applying a convolution kernel to a fixed size window centered at the point being processed. The most common form of such a kernel is a class of Gaussian distributions controlled by the size of the window ($\rho$) and the spread ($\sigma$) of the distribution.

For a contour $C =< p_1, p_2, \ \ldots \ , p_n >$, the Gaussian smoothing of $C$ transforms it to $C' =< p_1', p_2', \ \ldots \ , p_n' >$ where

$$p_i = (x_i, y_i), \quad p_i' = (x_i', y_i')$$

$$x_i' = \frac{\sum_{j=-\rho}^{j=\rho} x_{i+j} \cdot G(j)}{\sum_{j=-\rho}^{j=\rho} G(j)}, \qquad y_i' = \frac{\sum_{j=-\rho}^{j=\rho} y_{i+j} \cdot G(j)}{\sum_{j=-\rho}^{j=\rho} G(j)}$$

and the Gaussian mask $G(k)$, for $k = -\rho$ to $k = \rho$, is defined as

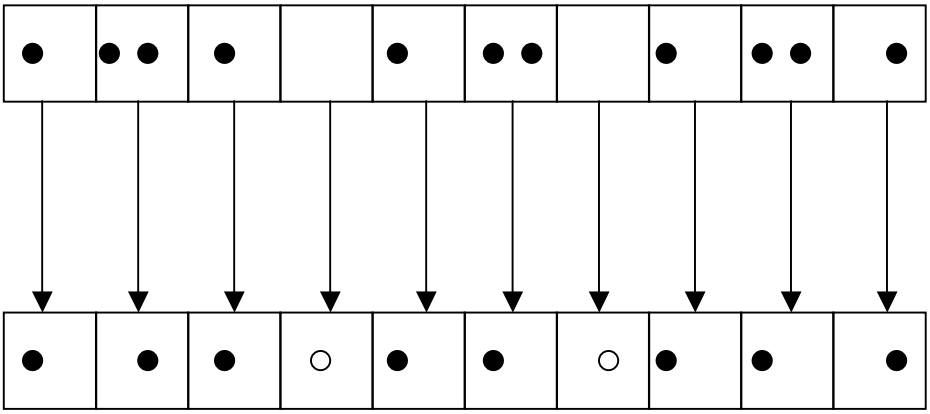
$$G(k) = \exp(\frac{-k^2}{2 \cdot \sigma^2}).$$

### *2.2 Global and Local Filtering*

Filtering eliminates duplicate data points and normalizes the irregularity in data point density caused by the relative speed of the user writing. For example, when the writing speed is slower in an interval, it will contain more points and when the speed is faster, the interval will have a sparser distribution of points. A common form of filtering is equi-distant resampling and forces a minimum Euclidean distance between two data points. This results in data points approximately equally spaced in distance. When scarcity of data points is the issue in a fast interval, an interpolation technique is used to fill the gaps. Usually this filtering step is performed only at the global level as part of the data

acquisition process. In our approach, we adopt two level filtering. The global level works the same as the conventional one. The local level filtering assumes and works on a writing interval or a character segment as input. A character hypothesis interval is a sequence of points contained in one or more consecutive sub-strokes since a character may span several strokes with the starting and ending parts covering possibly partial strokes. The character segmentation approach of our system generates a set of tentative segmentation points based on the curvatures. Which interval between the two segmentation points to try as hypothetical character, is managed and determined by the recognition engine and as a preparation of input to the component character recognizer, the local filtering is performed on the target interval. The resampling size of the local filtering is a fixed constant, and the spacing between two resampled points is dependent on the arc length of the given interval. Therefore, the local filtering generates as output a fixed size sequence of points regardless of how many points the input interval has or how long the interval is in terms of arc length. Because the size of the interval in terms of the contained points can vary greatly as the recognition engine tries from the minimum to the maximum sizes, it is possible that the input interval contains less points than the resampling size. In such case, an equi-distance interpolation is performed to make up for the missing points. (Figure 8)

(a)



(b)

**Figure 8.** Distribution of interval points in one dimension. Filled circles are original data points and empty circles are interpolated points (a) original distribution (b) distribution after the local filtering.

### *2.3 Translation and Scale Normalization*

The coordinate representation of the raw input is, of course, not translation invariant. To achieve the invariance of the coordinate values, we recompute the point positions with reference to a new standard origin. Which origin to take as the standard is facilitated by whether the system can hypothesize the definite boundaries of the characters, since then we can use a fixed boundary point as the origin. Depending on the segmentation methodology employed, however, this

information may not be available. For example, in the systems using implicit segmentation (see CHAPTER 4), the recognition engine works without the segmentation boundaries. Therefore such systems need to use some other representation, like directional code, instead of coordinates to obtain translation invariance. Once again, our adoption of compact set of the segmentation points come in handy to determine the origin. In our system, it is possible to compute the graphical bounding box of a writing interval hypothesized by the recognition engine as a possible character, since the interval has definite starting and ending points. The lower left corner of an interval is chosen as the new origin and the coordinates are recomputed in reference to it. That is, given an interval $I = <(x_1, y_1), (x_2, y_2), \quad \cdots \quad, (x_n, y_n), >$, the new origin is $O = (x', y')$ where $x' = \min\{x_i\}$ and $y' = \min\{y_i\}$, $i = 1 \ldots n$. Then the translation invariant version of the interval is

$$I' = <(x_1 - x', y_1 - y'), (x_2 - x', y_2 - y'), \quad \cdots \quad, (x_n - x', y_n - y'), > .$$

As with other visual recognition tasks, reducing the variability of the target objects in size is an important normalization step and this category of techniques try to adjust the scale of input to a standard size, and may be applied at the character level or to the entire word. However, more elaborate size normalization techniques requires extracting context features like the principal lines of

handwriting like the low and high baselines and the mid-line, consumes non-trivial amount of computation even before entering the main stage of the recognition process. Another weakness of the approaches depending on extracting such elaborate features is that they may not be stable in on-line handwriting data. For example, the tablet surface may be slippery and the user may form the writing hastily while in motion. In such situations, the user' s writing may have poorly aligned baseline or none at all in some cases. So either the user needs to be constrained to write with a clear baseline or relying on less-than-dependable feature is better avoided. The size normalization in our system is applied at the character level and achieved easily due to the local filtering approach we use. The output *S* of the local filtering applied to a character interval is a fixed length sequence of resampled points. After the translation normalization is applied to *S*, it becomes

$$S' = < (x_1, y_1), (x_2, y_2), \quad \cdots \quad , (x_n, y_n) > .$$

Then *S′* is transformed to an "Expanded Coordinate Vector" (ECV) by expanding the point coordinate pairs into vector form:

$$\mathrm{ECV}(\hat{S'}) = V = < x_1, y_1, x_2, y_2, \quad \cdots \quad , x_n, y_n > .$$

The extended coordinate vector *V* is in turn subjected to the vector magnitude normalization and becomes *U* such that |*U*| = 1.That is, $U = V / |V|$. Now *U* is the translation and scale invariant representation of the input character interval.

## 2.4 Other Normalizations

The remaining Euclidean geometric invariance not explicitly addressed in constructing the standard representation is rotation. "Deskewing" and "baseline correction" are two methods for the rotation normalization. Deskewing ([12], [44]) is a group of algorithms that minimize the variability in the slant of writing that can be applied either at isolated character or at entire word. Baseline correction ([1], [5]) tries to orient the baseline of writing horizontally. In our system, we have not addressed normalizing the rotation and there are two reasons. Firstly, we can assume that the intention of a user of a recognition system is to cooperate so as to get one's handwriting recognized, but not to put the system's capability to the test to the limit. So we can expect or ask the user's writing to be in a reasonable range of rotation for on-line recognition. Secondly, our character recognition technique itself is able to absorb the amount of variability in rotation typically seen in the writing formed on the limited space of a tablet.

# CHAPTER 3. Feature Extraction and Representation

With the normalized data, the next step is to extract prominent information from them and represent it in a standard form to be used by the recognition stage. Along with the segmentation strategy, selecting the data representation is one of the most fundamental decisions to make. The final choice may be influenced by which character recognition paradigm we will use, or with a fixed choice of representation it may influence the other way around. Depending on the scope or sight that a feature unit represents, there are 3 major categories of features.

## *3.1 Local Point Features*

The features in this group are the finest in grain and the examples are the point coordinates, the point curvature and the tangent angle of each point, etc. These are by themselves local features with a limited sight, arranged in sequence. For a character recognizer that builds up the evaluation of its input incrementally in sequence, the inflexible one-dimensional unfolding of inherently two-dimensional character data may lead to loss of spatial information ([51]). We may imagine a simple experiment in which a subject peeps through a hole moving

along the contour of a handwritten image and neither sees nor feels anything else. Intuitively, it will not be easy to make sense out of observing a long sequence of isolated or small groups of points even to human eyes, while seeing them at once would tell it all so vividly. But the local features are not without advantages. One is that they are easy to compute reliably. Another is their robustness due to their redundant and distributed nature.

### *3.2 Sub-character Primitive Features*

This group of features is extracted at a higher level than individual data points. Each character model is decomposed into a small number of subcharacter primitive units each of which represents a small salient sub-region of the character. The simplest kind is the directional code computed from equally spaced regions. That is, the character shape is divided into equally spaced intervals and the line orientation code corresponding to each interval is computed. Usually 8 or 16 directional code is used. More sophisticated methods compute a set of salient points according to their feature extraction scheme, and use the variable length intervals between the salient points, instead of the equal-spaced ones. The summary information computed from each interval may still be a directional code or a more elaborate feature. One novel approach is training the system to learn how to best divide into the primitives with integrated evaluation of each possible

feature options. For example, [27] trained Hidden Markov Models corresponding to each subcharacter primitive and set up a language grammar specifying the legal concatenation of the primitives to form a character. Another kind of feature is drawn from the topological characteristics of the character shape like corners, cusps, loops, and openings. One advantage of the subcharacter primitives is that they are more concise due to their encoded nature. Another is that they offer a higher level view because they represent a region not a point. One problem is that the primitive level feature extraction, depending on the granularity they work on, should be extremely robust to be effective since one mistake may make a big hole in the overall picture because of their regional representation nature. This is not easy to achieve especially in the setting of writer independent mode of recognition, in which a large variability in writing style exists and a system needs to deal with potentially missing or previously unseen features not covered in the training data.

### *3.3 Annotated Image Features*

On-line handwriting input has one dimensional sequence representation in structure. As has been previously mentioned, converting off-line data to on-line version is hard to achieve reliably, but the other direction is relatively easier. Therefore, if a limited or refined form of image features can contribute substantially to the discriminative power, at least as a part of the overall feature set

used by the system, then it would be well worth a try. The advantages of converting on-line data to image representation include:

- Stroke order invariance: In an image, the information on which order a shape was formed is not apparent and not relevant. This is not a problem since we already have the data in an on-line version. Only one image model is needed for the same shape class. This is not the case for on-line data: even for the same shape class, multiple models are necessary, one for each substantially different stroke orders.
- Simpler size normalization: Depending on the level of elaboration, the topic of size normalization may become an involved topic. On the character level conversion from on-line to image, the size normalization is achieved easily by fitting the on-line data into the fixed size image, that is by contraction or expansion.
- Robustness: Image is a most natural, redundant, distributed and fully decoded type of representation. Therefore it fits well for a pattern recognition system like neural network, which works best on such a kind of representation. The robustness comes from the fact that such a recognition system learns and performs the feature extraction itself from the raw data, so that the choice and the development of features are automated instead of handcrafting that may be a fallible process.

- Higher level perspective: Image can naturally be made to represent a character or larger unit, so it provides even higher level of scope than the subcharacter primitive features.

The main disadvantage of the image representation is its large size, leading to more computing time and memory for processing. This is exacerbated by the fact that a string level recognizer needs to invoke the component character recognizer many times, say hundreds, before its recognition engine get through processing. The larger representation size also means the need of more training data because of the increased number of learnable parameters. To ameliorate the situation, on-line handwriting recognition systems whose data representation is based on image, use a scaled-down, low-resolution image and the loss of resolution is compensated by augmenting it with various on-line features. The capability of an on-line recognizer to work on off-line data representation like image is potentially important in the sense that it can naturally lead to bridging the gap between the two different modes of recognition. It has been mentioned before that the approaches that attempted to use on-line recognizer to handle off-line data, have had limited successes. The main reason is their dependence on the close reconstruction of the temporal information from the off-line data, which is a daunting task. With an on-line recognizer that can handle image representation,

the difficulty of the problem would be reduced substantially since such a recognizer will need only reasonable character segmentation, not an exact recovery of the temporal orderings.

### *3.4 The Features Used by the System*

We compute the local geometric features, namely the point curvatures and the point tangent angles. Firstly we compute the tangent angle at each point. This is approximated as the direction angle from the current point to the next point. That is, for the two consecutive points $p_i = (x_i, y_i)$ and $p_{i+1} = (x_{i+1}, y_{i+1})$, the tangent angle $\theta_i$ of $p_i$ is

$$\theta_i = \text{acos}\left( \frac{x_i - x_{i+1}}{\text{dist}(p_i, p_{i+1})} \right)$$

where dist(·) is the Euclidean distance between the two points. The curvature $k_i$ at $p_i$ is the then approximated as the amount of the direction angle change around point $p_i$. That is, $k_i$ is computed as the absolute amount of angle change from $\theta_{i-1}$ to $\theta_i$ multiplied by the sign that is plus if the angle change is clockwise, or minus if it is counter-clockwise. These features are local in nature and, as stated above, have potential weaknesses depending on the kind of character recognizer employed. Our segmentation approach and the character recognition paradigm

circumvent this problem. Using the tentative segmentation points generated by the segmentation module, our recognition engine generates evaluation intervals that are at the character level. So the point features are never considered individually or incrementally and they are always grouped at the character level. In addition, the Fisher discriminant analysis technique that our system adopts for character recognition, integrates all the component features together at once in evaluating the scores, so that small errors in feature extraction do not have ruinous impact on the overall evaluation as long as the rest of features are well aligned with the model parameters. In effect, this is equivalent to the recognizer having a higher, that is at the character, level perspective so it is not misled by a local fluctuation of feature variability. This is in contrast with the recognition method whose direction of data unfolding is less flexible like HMM. For example, since an HMM evaluates its input features incrementally in one direction, a feature error in the middle may hurt the score badly at the point and because of the multiplicative nature of HMM' s scoring represented usually by the accumulation of the log likelihood, the recovery from the damage is less likely even if the rest of the features align well. In other words, the locally mismatched feature will result in near zero local likelihood and since the HMM constructs the likelihood of a character hypothesis by multiplication of the local likelihoods, the final score will be very significantly low even if the later local probabilities are high. So our

approach combines the robustness of local features drawn from their redundancy and distribution, with the higher level contextual scope by a noise insensitive integration at the character level. Other than the basic features, we compute the signed or unsigned accumulation of the curvatures and the tangent angles from the starting point and these serve as more larger scale and accumulative features. We also perform a sub-stroke based feature extraction by assigning the type index to an interval between the tentative segmentation points. The type indices for the sub-stroke intervals are called Feature-Link Code (FLC) and are used in modeling the ligatures as hypotheses filtering scheme. Since it is performed after generating the hypothetical segmentation points, the description is delayed to CHAPTER 4.

# CHAPTER 4. Character Segmentation

Given a word or a larger unit of writing input, the recognition system needs to break it into more basic and smaller units, usually an individual character or even lower level sub-character primitives, and to hypothesize and evaluate the possibilities drawn from the input. The input breaking is called the character segmentation.

## *4.1 Analytic vs. Holistic Approach*

In analytic segmentation approach, the system generates hypothetical segmentations before the recognition process starts. Early handwriting recognition systems were heavily dependent on the segmentation strategies, their quality, efficiency and robustness, which are hard to achieve in perfection. In reaction to the situation, another philosophy, called "holistic approach", came to get the attention. In holistic approach of segmentation, only the global, Gestalt-like features accounting for the entire input, are extracted and evaluated ([11], [16], [17], [40], [54]). So in such an approach, the input is treated as an unbreakable whole and no segmentation is needed. One obvious limitation of this approach is

that it can support only a small limited vocabulary. For example, when applied at the word level, the system would need at least one feature model for every word in its vocabulary, and since there is no means to compare partial features incrementally, a given input would have to be compared against all the word models in the vocabulary. Hence as the size of the system vocabulary increases, it becomes less and less flexible to scale up in the amount of memory and processing time. Meanwhile, the analytic approach can support unlimited or very large vocabulary since what it recognizes is any legal combination of characters in the supported alphabets.

## *4.2 Explicit vs. Implicit Segmentation*

In more general visual recognition or pattern recognition problems, the segmentation of a given input into the target objects has been an important and difficult problem. There seems to be a cyclic dependency between the segmentation and the recognition problems: a perfect segmentation can be obtained as a by-product of the correct recognition that largely depends on the right segmentation. The same pattern repeats in the handwriting recognition task and, to be practical, the cycle needs to be broken somewhere. Depending on the choice of the break point, the analytic approach has two extreme forms: recognition-by-segmentation and segmentation-by-recognition.

### 4.2.1 Recognition by Segmentation

In the recognition-by-segmentation (or explicit segmentation) paradigm, the input is first broken down into hypothetical character segments using features like cusp, closure, estimated character widths, etc., and ordered and legal combinations of these segments were generated as possible strings of characters. This approach, adopted in the early systems ([25], [22], [43]), needs a complicated segmenter and, without careful control, can lead to intractable computation due to the large proliferation of generated candidates.

### 4.2.2 Segmentation by Recognition

At the opposite extreme to explicit segmentation is the segmentation-by-recognition (or implicit segmentation) paradigm. In this strategy, mostly based on path optimization framework like graph search, there is no complex segmenter that performs segmentation explicitly. Rather a best segmentation is obtained by extracting, after the recognition evaluation is complete, the path that represents the best candidate in terms of the metric applied to evaluate the paths ([73], [30], [26], [36], [27]). Or the system recognizes a word corresponding to the best-scoring path generated and evaluated by the system. This approach is attractive because it

bypasses the difficult and fallible segmentation step. The main disadvantage is that it requires a large amount of, although not intractable, computation. This stems from the fact that it lacks a segmenter that suggests the possible boundary points, so any input point is a potential segmentation point and should be treated accordingly, leading to exhaustive hypothesizing. Because this approach is based on dynamic programming of path optimization, the running time does not become intractable, but still takes too much computing to be practical.

### 4.2.3 Fuzzy Centering Segmentation

One reason why the segmentation step is important, is that the performance of the character recognizer relies on how well the input is segmented. Using implicit segmentation, the system actually does not really bypass the character segmentation altogether. Instead, it tries more exhaustive segmentation possibilities systematically so as to avoid intractable amount of computation. Since the character recognizer is typically trained with data that has regular starting and ending patterns, the thread of character recognition whose input aligns well with the character model, will have higher evaluation score. Such well-aligned (or well-segmented) intervals would constitute a component of winning overall segmentation. So the idea is to train the character recognizer with specially prepared data having fuzzy character boundaries. That is, the character samples

are extracted from the word contexts without crisp starting and ending portions, and contain parts of the neighboring character patterns. Therefore, a character recognizer trained in such way would be robust to contextual co-articulation problem and would not need crisp segmentation for good performance. Instead the recognizer would need well-centered input and the segmentation problem is transformed from finding the boundaries to finding the best centering positions. A system based on such a recognizer then will scan its input with the recognizer that sees fixed length frame as input, from start to end. After the scanning phase, the system will mark the positions in the input where the character recognizer' s response level has local maxima above certain threshold. Those maxima correspond to the centering points in the input, on which the character recognizer was at high activation level in terms of the recognition score. The system then proceeds to the rest of the processing with the segmentations guessed from the maxima points. Therefore, this segmentation approach avoids the explicit segmentation by changing the nature of the character recognizer, not by making the recognition engine absorb the segmentation problem. Because the input window length is fixed and there is no need to iterate to try different size input on the same starting point, this approach is more economical in the number of times the character recognizer needs to be invoked, than the regular implicit segmentation method. But the overall running time is still substantial because the

character recognizer would be invoked on every data point or at least on every few points, to make sure that the correct centering points are included in the set of points tried.

### *4.3 Curvature-Based Static/Dynamic Segmentation*

Another weakness of implicit or fuzzy boundary segmentation, other than the running time, is that it is no longer possible to apply a visual context modeling (see CHAPTER 7) used in later stage for filtering the hypotheses, since the character boundaries have been blurred. For example, in the ligature modeling and the visual bigram modeling (VBM), the geometric information from the context surrounding the current processing point is extracted including a possible sub-stroke between the characters, the relative size and positioning of the characters, and so on. Without hypothesizing about the definite boundary points, computing such information can not be made reliably. Hence, most successful approaches take a middle ground somewhere between the two extremes ([70], [79]) of pure explicit and pure implicit approaches.

Our segmentation strategy is a hybrid analytic approach that hypothesizes about the candidate segmentation points but the actual segmentations are handled dynamically by a rule built into the recognition engine, and the best global segmentation is naturally determined at the end. The generation of the candidate

**Figure 9.** A cursively written input word of "eye." The left side shows the high-curvature points (in filled dots). The right side shows the high-curvature points along with the augmentation points (in empty dots).

segmentation points will be based on detecting the high curvature points. The rationale is that the high curvature points, that is corner-points or turning-points, are the places where the information regarding the writing' s dynamics and geometry is most condensed, and therefore are natural candidates for character boundaries. The intuition is right for most cases, but the set of high curvature points is incomplete as full candidates since some real segmentation points can lie in the middle of a smooth interval whose points have only low curvatures. Hence, an augmentation is needed to make the candidate set a complete one, and we will resolve the issue by adding a middle point in a long enough (determined by a threshold) interval between two high curvature points (Figure 9).

In summary, our segmentation approach works like an implicit segmentation, but avoids exhaustive hypothesization because the character recognizer needs to be invoked on the set of tentative boundary points which is a

much smaller subset of the entire data points. Since the hypothesis characters intervals span only a few segmentation points, there is smaller number of character recognition threads on each character starting point. For example, the shortest character consists of the interval formed by 2 segmentation points and the longest 9 segmentation points, and therefore each successively longer interval spanning up to 9 segmentation points needs to be evaluated starting from a particular boundary point. The clearly separated character boundaries also make it amenable to extract useful context features as stated above. The next section defines and describes the sub-stroke level feature extraction drawing from the boundary information made available by our segmentation strategy.

## *4.4 The Feature-Link Coding*

### 4.4.1 The Feature-Links and Sub-Stroke Primitives

The segmentation points generated by the system break the entire input into a sequence of intervals each of which is determined by two consecutive segmentation points. That is, given a stroke $C =< p_1, p_2, \cdots, p_n >$, the segmentation module computes a sequence of segmentation points $S =< p_{k_1}, p_{k_2}, \cdots, p_{k_m} >$, where $p_1 = p_{k_1} < p_{k_2} < \cdots < p_{k_m} = p_n$. Then C is subdivided into the intervals $I_i =< p_{k_i}, p_{k_i+1}, p_{k_i+2}, \cdots, p_{k_{i+1}} >$ for $i$ = 1 to $m$-1. By

the nature of our segmentation points drawing from the level of the point curvatures, any interval so determined contains no corners or cusps and is smooth in shape since otherwise the system would have detected high curvature points inside the interval and it would have been further subdivided. We name these intervals "feature-links." The feature-links can be considered as sub-stroke or sub-character level primitives and exploiting their smoothness we may use a simple technique to classify them according to the line or the direction that the intervals form. This may sound similar to the conventional methods of computing the directional code representation from the input contour ([27], [71]). In comparison our situation has two notable differences from the typical direction coding schemes. Firstly, the traditional methods encode only the line direction. A feature-link, however, may have a substantial amount of convexity. Secondly, the usual direction code is constructed out of equally spaced intervals obtained by a resampling. But feature-links are irregular in length and may have large differences among them. Intuitively, the feature-links are more natural since they are not formed from uninformed artificial spacing and are based on the segmentation points that in turn are based on curvatures. Therefore forming a sub-stroke primitive using more consistent information like curvature can be considered to be more feature-worthy. The feature-links are also more economical in representation because a sequence of simple direction codes can be compacted

into a single code representing a feature-link. Conventional direction code schemes have to use short enough spacing in forming the direction intervals so that they are shaped as lines not curves, since such schemes do not incorporate the convexity. But the feature-link allows convexity and can represent longer length without loss of information.

### 4.4.2 Computing the Feature-Link Code

With the feature-links, we want to draw higher level information representing the line or curve formed by the component points of such intervals. To this end, we define 24 directional convex templates each representing the direction and the convexity of the corresponding categories of feature-links (see Figure 10). Given a sub-stroke $C =< p_1, p_2, \cdots, p_n >$, like feature-link, let' s define the "cross-length" of $C$ as the Euclidean distance between the first and the last points of $C$, that is $\text{cross\_length}(C) = \text{dist}(p_1, p_n)$. The templates have a standard scale in terms of the cross-length and are stored inside the system. Other than the visual scale, the feature-link templates are also normalized to have the same number $N$ of data points using the local filtering method described in CHAPTER 2.

With the feature-link templates set up, the next step is to assign a given feature-link the index that represents the feature-link template matching the input

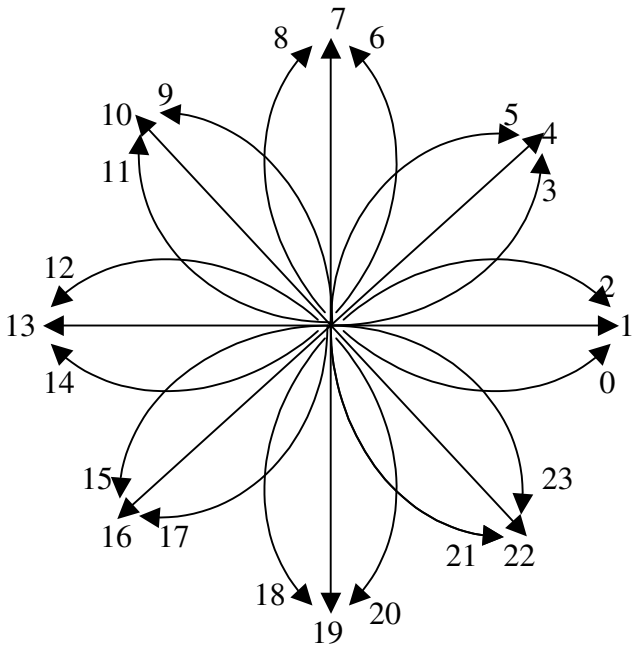



**Figure 10.** The 24 feature-link templates indexed from 0.

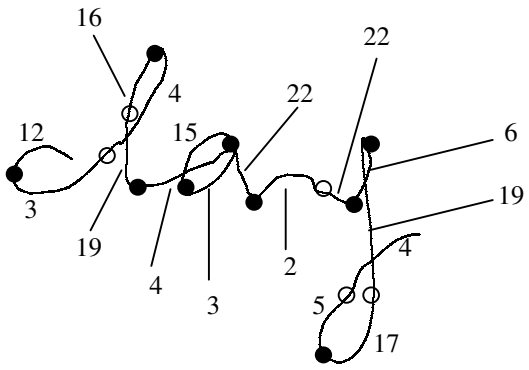



**Figure 11.** An example word "day," its feature-links and the corresponding feature-link code. The filled and empty circles are the segmentation points.

most closely. This task is performed by a template matching whose high level steps are as follows.

1. Given an input feature-link, normalize it to have the standard cross-length and the standard number $N$ of data points by local filtering. Let' s call the normalized input as $I$.
2. Determine the subset $T$ of templates that are the candidates to match with $I$.
3. For each template $T_i$ in $T$ do the following:
   3.1. Compute the amount of rotation angle $\theta(T_i, I)$ needed to align $T_i$ with $I$ in terms of the starting and the ending points.
   3.2. In the aligned state, compute the area $A(T_i, I)$ between $T_i$ and $I$.
   3.3. Compute the matching metric $E(T_i, I)$ by combining $\theta(T_i, I)$ and $A(T_i, I)$.
4. Output $k$ such that $k = \arg\min_i \{ E(T_i, I) \}$.

In step-2, each template is first translated to have the same starting position as that of the input $I$, and the distance between the ending points of the template and $I$ is computed. This distance gives a good hint at how much the template is off from $I$: the longer the distance, the more discrepancy between the directions of the template and $I$. Therefore the template matching does not need to consider all of

its feature-link templates and only a few of them that have end-point distances smaller than a threshold are selected as candidates for further processing.

In step-3.2, the area A($T_i$ , $I$) between $T_i$ and $I$ is actually an approximation of the gap between $T_i$ and $I$. Let $T_i =< t_1, t_2, \cdots, t_N >$ and $I =< x_1, x_2, \cdots, x_N >$. Note that both are normalized to have the same representation length $N$. Then A($T_i$ , $I$) is defined to be

$$\mathrm{A}(T_i, I) = \sum_{j=1}^{N} \mathrm{dist}\,(t_j, x_j)$$

where dist(·) is the Euclidean distance. This simple metric gives a reasonable approximation of the gap between a template and the input because the feature-links are by definition smooth intervals without sharp or complex curves.

In step-3.3, the system combines the rotation angle and the area to compute the matching score. The formula we use for determining the score is

$$\mathrm{E}(T_i, I) = \mathrm{A}(T_i, I)^3 \times \exp(c \cdot \theta(T_i, I))$$

where $c$ is a normalizing constant. The lower the value of E($T_i$ , $I$), the better $T_i$ matches $I$. The rationale is that the rotation angle is more critical in determining a match since it measures the level of directional alignment, and is given more rapidly growing functional. So as the value of the rotation angle grows, E($T_i$ , $I$) will be quickly dominated by $\theta$ ($T_i$ , $I$). If the rotation angles are within close range, then E($T_i$ , $I$) will be determined by the magnitude of , A($T_i$ , $I$). The scheme can be alluded to an energy minimization process. That is, we interpret the

rotation angle and the gap area as representing the amount of work to transform a template to the input. The larger the amount of work, the more energy it will consume for the transformation. Then the solution is the template that consumes the least amount of energy to transform itself to the given input.

For a feature-link *f*, the template matching outputs the index of the template that matches *f* the best. We call this index the "feature-link code" (or FLC for short) of *f*. Figure 11 shows an example word "day," the segmentation points identified on it, the feature-links and their FLC' s determined by the template matching. The FLC encoding may be used as a sub-character primitive level representation of handwriting and, if worked out adequately, may open a possibility for a large speed-up in the recognition time because of its very compact form. In this thesis, however, that potential is not pursed and instead will be mainly used for modeling the ligatures in CHAPTER 7. The problem with modeling the ligatures has been that it is not obvious how to categorize them and how to construct the contextual modeling efficiently in processing time and memory. The simplicity, robustness and the compactness of FLC representation will address the modeling issues nicely.

# CHAPTER 5. Component Character Recognition

An analytic handwriting recognition system needs a recognizer working at the level of an individual character to classify and evaluate the segmentations given by the segmentation module. This recognizer is a pattern classifier that stores a model for each character class, which is the representative pattern derived from the shape characteristics of the characters in the class. The models are usually explicit or can be implicit as in the case of neural network based systems, and can be constructed as an encoding of a set of designed rules or can be trained on and learned from a large number of training samples. Given input is compared against each stored model using the metric specific to the system and is assigned a score for each interpreted class. To be useful, a character recognizer should be capable of absorbing the shape variability inside each class and also discriminative of the differences between the classes.

## *5.1 Character Recognition Paradigms*

In the last few decades, two distinct categories of character recognizers have been developed. One category is called the syntactic or structural recognition

and the other is statistical one. The statistical recognition methods are in turn divided into three prominent groups: explicitly statistical recognizers, artificial neural network based recognizers and those based on hidden Markov models.

### 5.1.1 Structural Character Recognition

This group of approaches tries to capture the shape characteristics of the character classes into a symbolic form, and establish a set of rules to manipulate and evaluate the symbolic features ([55], [19], [25], [33]). Due to the abstracted nature of its representation, these rule-based systems have the advantage of the economy of representation and the processing time if the size of the description rules can be kept small or moderate. One difficulty is that manually formulating a dependable set of classification rules that can account for a rich range of shape variability, is a daunting challenge. Another is a possible brittleness introduced by the summary nature of the representation. That is, the feature extraction is allowed little room for mistake since the recognizer' s decision will be based only on what the feature extractor provides. Therefore, misleading or missing information caused by the feature extraction will have a critical effect on the recognizer' s performance. Even if the approach is reformulated for training from samples, so that the rule-generation process is automated, the resulting rules tend to proliferate in a large number, reducing the manageability of the approach. Recently, however, this approach has gotten a momentum due to the development of

statistical training techniques in which a reasonable-sized stochastic description grammar is learned from annotated training samples informing which rule is applicable.

### 5.1.2 Neural Network Based Character Recognition

Multi-layer artificial neural network is a general learning paradigm that, with adequate network architectures and a large enough set of training data, can learn arbitrarily complex decision boundaries. The classification behavior of a neural network is fully determined by the statistical characteristics of the training data, with a given structure of the network ([8]). Three major variations of the neural network have been applied to the handwriting recognition field: the standard fully-connected back propagation perceptrons, Kohonen's self-organizing feature map (SOFM) approach and the time-delay neural networks (TDNN).

In general, the capacity of a learning system is determined by the number of learnable parameters in the system. The larger capacity, however, requires a larger amount of training data for proper training, which in turn means longer training time and the need for more effort to ascertain the desired statistical properties of the data. Therefore, it is desirable that the capacity of a system can be tuned to an optimal size by choosing the right set of learnable parameters, for better and faster training with a smaller training data set. This design is limited in

a fully connected perceptron since every unit or neuron is connected with each other without choice, and the approach has not been very successful.

The SOFM networks transform the input vector into a fixed dimensional (usually below three dimension) discrete map subject to a topological (neighborhood preserving) constraint ([34]). The technique allows automatic detection of shape prototypes from a large number of character samples, and is analogous to k-means or hierarchical clustering. The quantization effect of SOFM output is usually used by a postprocessor that tries to interpret the patterns of SOFM output ([41]).

The convolutional neural networks (CNN) are the networks whose connection structure is designed on the basis of the evidences from neural science that the neural connection patterns do not have to be exhaustive and in many cases localized connections are desirable and effective to represent specialized and limited scope functions ([29], [20]). The locally connected neurons at a layer are then integrated together at the next layer to form higher level features. In this scheme, a unit of a network layer is connected to only a limited field of units in the previous layer, and the connection weights are replicated and shared by some other units distributed over the same layer. The overall connections consist of a set of such replicated connection weight patterns. In essence, those replicated and distributed connections perform localized feature extraction whose output is

integrated, at the next layer, into a larger scope feature ([38]). The TDNN is the one-dimensional version of general CNN, sharing weights along a single temporal dimension, and is used for space-time representation of handwriting signals. The approach is known to provide a useful degree of invariance to spatial and temporal distortions because of the distributed nature of the feature extraction in the system ([66], [23]).

### 5.1.3 Hidden Markov Model Based Character Recognition

A Hidden Markov Model (HMM) is a doubly stochastic process in which the underlying process is hidden from observation and the observable process is determined by the underlying process ([62]). The underlying process is characterized by a conditional state transition probability distribution, where the state is hidden from observation and depends on the previous states (the Markov assumption). The observable process is characterized by a symbol emission probability distribution, where a current symbol depends either on the current state transition or on the current state. Because of its inherently temporal nature, HMM' s have been successfully employed in speech recognition field where the acoustic data has strictly one-dimensional properties. Recent efforts have tried to extend the same approach in the handwriting recognition field but so far the success has been limited ([27], [45], [48]). One important reason is that despite

some similarities, handwriting data is inherently two-dimensional and a one-dimensional data unfolding in an inflexible way as taken by a conventional HMM leads to loss of information and integrity in the data. For example, speech data is completely ordered by time and its recognition systems can rely on this assumption. However, in handwriting data, even if it is represented in one-dimension, the component parts can be out of order in time. An example is the delayed dots and strokes that can happen in the formation of characters like "i," "j," "t," "x," etc.

## *5.2 Linear Projection Methods*

The category of explicitly statistical character recognition is derived more or less from the linear projection methods and the hierarchical or partitional cluster analyses ([53], [24]). The features are summarized into a fixed length vector that maps to a data point in high-dimensional representation space. Therefore, similar patterns having similar vector representation will wind up clustering closely each other as points in the high-dimensional space. There are various possible distance metrics, measuring the closeness of the two points in such space. The most common is the Euclidean distance. In this setting, the training and learning corresponds to finding a set of hyper-planes that best separate the classes into different regions of the space. In case there is no a priori knowledge about the

class labels, the clustering analysis methods provide automatic ways to construct out of data a set of clusters corresponding to categories or taxonomies. If we can keep all the training samples, then the classification can be done simply by identifying the class of the sample in the representation space that is closest to the input. Because of the large amount of memory needed to keep many samples, usual solution is keeping only the centroids of the classes, that is, the class means of the training samples in each classes. The prominent property of the linear projection methods is the reduction of the problem' s dimensionality and they try to attain the economy of features by reducing the dimensionality of the original representation space with as small discriminative compromises as possible. The projection can visualize the data set in the projection space if a sufficiently small output dimensionality is chosen.

The Principal Component Analysis (PCA) technique is one of the most widely known projection methods. The essence of PCA is the construction of the projection matrix that defines the linear mapping from the original space to the projected feature space ([78], [74]). Let $\mathbf{V} = \{\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, \cdots, \mathbf{v}_M\}$ be the set of $n$ dimensional data vectors. Let $\mathbf{W} \in \mathbf{R}^{n \times m}$ be a projection matrix of orthonormal columns and consider the linear transformations defined by **W**:

$$\mathbf{p}_k = \mathbf{W}^{\mathrm{T}} \mathbf{v}_k \quad k = 1, 2, \ldots, M \ .$$

Hence $\mathbf{p}_k$ is the feature vector in the projection space. The dimensionality *m* of the projection space needs to be meaningfully smaller than *n*, the original dimensionality. The covariance matrix of a set of data vectors is a measure of how the data are distributed or scattered in the representation space. Let $\boldsymbol{\mu} = 1/M \cdot \sum_{i=1}^{M} \mathbf{v}_i$. That is $\boldsymbol{\mu}$ is the mean of **V**, the global center. Then the covariance matrix **K** of **V** is defined as

$$\mathbf{K} = \sum_{i=1}^{M} (\mathbf{v}_i - \boldsymbol{\mu})\,(\mathbf{v}_i - \boldsymbol{\mu})^{\mathrm{T}}.$$

For the projected data $\mathbf{P} = \{\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, \cdots, \mathbf{p}_M\}$, its covariance matrix $\mathbf{K}'$ is

$$\begin{aligned}
\mathbf{K}' &= \sum_{i=1}^{M} (\mathbf{p}_i - \boldsymbol{\mu}')\,(\mathbf{p}_i - \boldsymbol{\mu}')^{\mathrm{T}} \\
&= \sum_{i=1}^{M} (\mathbf{W}^{\mathrm{T}}\mathbf{x}_i - \mathbf{W}^{\mathrm{T}}\boldsymbol{\mu})\,(\mathbf{W}^{\mathrm{T}}\mathbf{x}_i - \mathbf{W}^{\mathrm{T}}\boldsymbol{\mu})^{\mathrm{T}} \\
&= \sum_{i=1}^{M} (\mathbf{W}^{\mathrm{T}}\mathbf{x}_i - \mathbf{W}^{\mathrm{T}}\boldsymbol{\mu})\,(\mathbf{x}_i^{\mathrm{T}}\mathbf{W} - \boldsymbol{\mu}^{\mathrm{T}}\mathbf{W}) \\
&= \sum_{i=1}^{M} \mathbf{W}^{\mathrm{T}}(\mathbf{x}_i - \boldsymbol{\mu})\,(\mathbf{x}_i^{\mathrm{T}} - \boldsymbol{\mu}^{\mathrm{T}})\mathbf{W} \\
&= \mathbf{W}^{\mathrm{T}} \left( \sum_{i=1}^{M} (\mathbf{x}_i - \boldsymbol{\mu})\,(\mathbf{x}_i^{\mathrm{T}} - \boldsymbol{\mu}^{\mathrm{T}}) \right) \mathbf{W} \\
&= \mathbf{W}^{\mathrm{T}} \left( \sum_{i=1}^{M} (\mathbf{x}_i - \boldsymbol{\mu})\,(\mathbf{x}_i - \boldsymbol{\mu})^{\mathrm{T}} \right) \mathbf{W} \\
&= \mathbf{W}^{\mathrm{T}}\mathbf{K}\,\mathbf{W}
\end{aligned}$$

where $\boldsymbol{\mu}' = \mathbf{W}^{\mathrm{T}}\boldsymbol{\mu}$ is the global mean of the projected vectors. Hence $\mathbf{K}' = \mathbf{W}^{\mathrm{T}}\mathbf{K}\,\mathbf{W}$. Then PCA constructs the projection matrix $\mathbf{Z}$ that maximizes the determinant of $\mathbf{K}'$ . That is,

$$\mathbf{Z} = \arg\max_{\mathbf{W}} \left| \mathbf{K}' \right| = \arg\max_{\mathbf{W}} \left| \mathbf{W}^{\mathrm{T}}\mathbf{K}\,\mathbf{W} \right|.$$

$\mathbf{Z}$ can be constructed by finding the $n$ solution eigenvalues $\{\lambda_1, \lambda_2, \ldots, \lambda_n\}$ and their corresponding eigenvectors $\{\mathbf{z}_1, \mathbf{z}_2, \mathbf{z}_3, \cdots, \mathbf{z}_n\}$, of the eigenvalue equation

$$\mathbf{z} = \lambda\,\mathbf{K}\,\mathbf{z}.$$

The economy draws from the fact that not all eigenvectors are essential and typically only a fraction of all eigenvectors has enough representative-ness. The feature-worthiness of an eigenvector is determined by the size of its related eigenvalue and the technique retains a set of $n$ eigenvectors whose eigenvalues are among the largest $n$. Let $\lambda = < \lambda_{s_1}, \lambda_{s_2}, \cdots, \lambda_{s_n} >$ such that $\lambda_{s_1} \geq \lambda_{s_2} \geq \cdots \geq \lambda_{s_n}$. Then

$$\mathbf{Z} = < \mathbf{z}_{s_1}, \mathbf{z}_{s_2}, \cdots, \mathbf{z}_{s_m} >$$

that is, the columns of $\mathbf{Z}$ can be constructed as the eigenvectors corresponding to the largest $m$ eigenvalues of the problem $\mathbf{z} = \lambda\,\mathbf{K}\,\mathbf{z}$. The $m$ eigenvectors are interpreted as the $m$ most significant direction of the region formed by the data population and as describing the distribution of the data. PCA also maximizes the scatter of all data points in the projection space and in general has the effect of

widening the gap between the class boundaries, therefore facilitating the discrimination among the classes (see Figure 12).

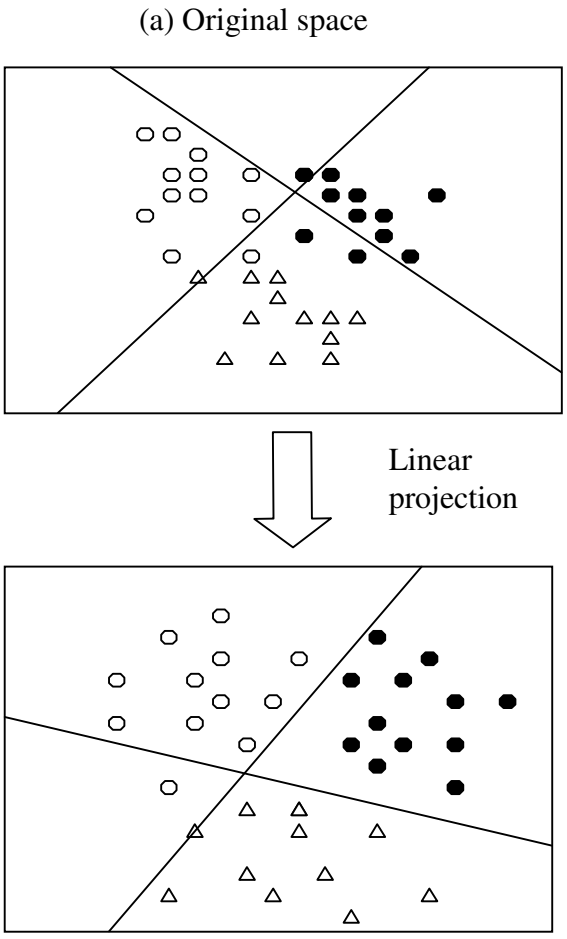



**Figure 12.** (a) Hypothetical three class clusters in the original space and their best separating hyper-planes. (b) The clusters projected to the projection space, the scattering effect and the separation between the classes.

### *5.3 Fisher's Segmental Matching*

A substantial advantage of using linear techniques like FDA is that the training is much faster and requires relatively smaller amount of training data compared with the more popular methods like neural networks and hidden Markov models described previously. Therefore it has a potential to make a user-tailored training feasible. The problem with the PCA is there is no provision built into the linear projection that can take class-specific regularity into account. The projection matrix is constructed with reference to the single global mean and the scattering effect gained in the projection space is indiscriminate of the classes. That is, the projection widens the between-class scatter, but it also widens the scatter within the classes and this is not a desirable effect for classification purposes. For example, in the face recognition task under large variation of lighting, the PCA would wind up with the projection matrix in which the projection columns with the largest eigenvalues tend to represent the variation in lighting. The reasons are that the dominant pixel value changes in such task would come from the lighting condition, and that the PCA would draw the globally more significant feature first, in this case the lighting level. Therefore the PCA projection would put the lighting level before the differences among the facial patterns of different people, in terms of the feature worthiness. This is clearly misleading information in many situations.

Fisher' s linear discriminant analysis [15], [64]) is one of the linear techniques, which projects the input onto a lower-dimensional subspace. It was successfully used in [3] for improving the performance of a face recognition task under extensive variation of lighting conditions and facial expressions. One major reason for using linear projection methods in the face recognition community is to reduce the huge number of dimensions typically involved in face images. The salient feature of Fisher projection is that it tries to maximize the between-class scatter in relation with the within-class scatter, by taking class-specific regularities into account in its construction. This contrasts with PCA where the global scatter is maximized, indiscriminate of the classes. This point is demonstrated by a simple two-class experiment in [3] where the PCA partially mixes up the two classes in the projection space while the Fisher projection yields a clean-cut separation.

### 5.3.1 Construction of the Fisher Projection Matrix

Suppose that we have the number $C$ of classes $V_1, V_2, V_3, \cdots, V_C$ and each class $V_i$ has a population of $N_i$ vectors $V_i = \{\mathbf{v}_1^i, \mathbf{v}_2^i, \quad \cdots \quad, \mathbf{v}_{N_i}^i\}$. Each data vector is assumed to have $n$ dimensions. The Fisher analysis considers two kinds of scatter matrices: one for between-class distribution and the other for within-class distribution. The between-class scatter $\mathbf{\Phi}_B$ is defined as

$$\mathbf{\Phi}_B = \sum_{i=1}^{C} N_i \, (\boldsymbol{\mu}_i - \boldsymbol{\mu}) \, (\boldsymbol{\mu}_i - \boldsymbol{\mu})^{\mathrm{T}}$$

where $\boldsymbol{\mu}_i$ is the centroid of the class $V_i$ and $\boldsymbol{\mu}$ is the global centroid. The within-class scatter $\mathbf{\Phi}_W$ is defined as

$$\mathbf{\Phi}_W = \sum_{i=1}^{C} \sum_{j=1}^{N_i} \; (\mathbf{v}_j^i - \boldsymbol{\mu}_i) \, (\mathbf{v}_j^i - \boldsymbol{\mu}_i)^{\mathrm{T}} \; .$$

Therefore the Fisher technique use the class label information to describe the two comparative kinds of distributions. Given a projection matrix $\mathbf{W}$ (of size $n$ by $m$) and its linear transformation $\mathbf{p} = \mathbf{W}^{\mathrm{T}} \mathbf{v}$, the between-class scatter in the projection space is

$$\begin{aligned}
\mathbf{\Psi}_B &= \sum_{i=1}^{C} N_i \, (\boldsymbol{\mu}_i' - \boldsymbol{\mu}') \, (\boldsymbol{\mu}_i' - \boldsymbol{\mu}')^{\mathrm{T}} \\
&= \sum_{i=1}^{C} N_i \, (\mathbf{W}^{\mathrm{T}} \boldsymbol{\mu}_i - \mathbf{W}^{\mathrm{T}} \boldsymbol{\mu}) \, (\mathbf{W}^{\mathrm{T}} \boldsymbol{\mu}_i - \mathbf{W}^{\mathrm{T}} \boldsymbol{\mu})^{\mathrm{T}} \\
&= \sum_{i=1}^{C} N_i \, (\mathbf{W}^{\mathrm{T}} \boldsymbol{\mu}_i - \mathbf{W}^{\mathrm{T}} \boldsymbol{\mu}) \, (\boldsymbol{\mu}_i^{\mathrm{T}} \, \mathbf{W} - \boldsymbol{\mu}^{\mathrm{T}} \mathbf{W}) \\
&= \sum_{i=1}^{C} \mathbf{W}^{\mathrm{T}} \, N_i \, (\boldsymbol{\mu}_i - \boldsymbol{\mu}) \, (\boldsymbol{\mu}_i^{\mathrm{T}} - \boldsymbol{\mu}^{\mathrm{T}}) \, \mathbf{W} \\
&= \mathbf{W}^{\mathrm{T}} \left( \sum_{i=1}^{C} N_i \, (\boldsymbol{\mu}_i - \boldsymbol{\mu}) \, (\boldsymbol{\mu}_i - \boldsymbol{\mu})^{\mathrm{T}} \right) \mathbf{W} \\
&= \mathbf{W}^{\mathrm{T}} \, \mathbf{\Phi}_B \, \mathbf{W}
\end{aligned}$$

where $\boldsymbol{\mu}'_i$ and $\boldsymbol{\mu}'$ are the class centroid and the global centroid in the projection space respectively. Similarly $\mathbf{\Psi}_W = \mathbf{W}^{\mathrm{T}} \, \mathbf{\Phi}_W \, \mathbf{W}$ is the within-class scatter in the

projection space. The desired goal is to select a **W'** maximizing the between-class scatter while minimizing the within-class scatter in the projection space so that the widened gaps between the class boundaries lead to better class separability. The problem can formulated as solving the following equation

$$\begin{aligned} \mathbf{W}' &= \arg\max_{\mathbf{W}} \left\{ \frac{\left| \mathbf{\Psi}_B \right|}{\left| \mathbf{\Psi}_W \right|} \right\} \\ &= \arg\max_{\mathbf{W}} \left\{ \frac{\left| \mathbf{W}^{\mathrm{T}} \, \mathbf{\Phi}_B \, \mathbf{W} \right|}{\left| \mathbf{W}^{\mathrm{T}} \, \mathbf{\Phi}_W \, \mathbf{W} \right|} \right\} \end{aligned}$$

The construction of $\mathbf{W}'$ can be implemented by solving the generalized eigenvalue equation

$$\mathbf{\Phi}_B \mathbf{w} = \lambda \mathbf{\Phi}_W \, \mathbf{w}$$

and computing the *m* largest eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_m$ of the problem and their corresponding eigenvectors $\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_m$. Then the $\mathbf{w}_i'$ s form the orthonormal columns of the target projection matrix, that is

$$\mathbf{W}' = [\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_m].$$

### 5.3.2 Training and Character Recognition

Computing the Fisher projection matrix with a set of character data vectors is equivalent to training for character recognition using a metric in the Fisher

projection space. About the representation feature for character samples, there are various choices as was described in CHAPTER 3. As a baseline representation, we take a fixed length ECV (expanded coordinate vector, see CHAPTER 2) of the character samples. That is, given a sample $S$, $\mathrm{ECV}_N(S) = < x_1, y_1, x_2, y_2, \cdots, x_N, y_N >$ for a constant $N$. The expanded coordinate values $x_i$'s and $y_i$'s of $\mathrm{ECV}_N(S)$ are the resampling results of the local filtering after applying the data normalization steps of CHAPTER 2 to the original data. The training data are compiled by computing $\mathrm{ECV}_N(\cdot)$ for each character sample. Then the data go through the Fisher analysis and the projection matrix is constructed. In the meantime, the training process also computes the model centroid $\mathbf{m}_i$ for each class $i$. The class vectors are normalized to be unit vectors for the character matching later. Given a vector $\mathbf{V}$, let $\mathrm{F}(\mathbf{V})$ be the vector in projection space mapped by the projection matrix that has been trained by the Fisher analysis. Given an input ECV $\mathbf{y}$, its Fisher matching score (FMScore) for a class $c$ is defined as

$$\mathrm{FMScore}(\mathbf{y}, c) = \frac{2 - \mathrm{dist}(\mathrm{F}(\mathbf{m}_c), \mathrm{F}(\mathbf{y}))}{2}$$

where $\mathrm{dist}(\cdot)$ is the Euclidean distance. So $\mathrm{FMScore}(\cdot)$ ranges from 0 to 1 since the model centroid and the input are normalized to unit vectors beforehand. The score

of 1 corresponds to the perfect match and 0 the complete mismatch. In summary, the training steps are

1. Compile the training data in the standard representation.
2. Compute the model centroids and store them in the standard representation.
3. Compute the Fisher projection matrix and store it.

The high level description of the character recognition is

1. Give an input y, convert it into the standard representation **y**.
2. For each class $c$, compute the score $f_c$ = FMScore(**y**, $c$) and produce the pair $<c, f_c>$.
3. Sort the $<c, f_c>$ pairs, in decreasing order on $f_c$, into the list $L$.
4. Return the list $L$.

In case the character recognizer works as a standalone system, as is for an isolated letter-by-letter recognizer, the system can just return the pair $<c, f_c>$ or the index $c$ such that $f_c$ is the maximum, instead of the steps 3 and 4 above. But for a character recognizer as a component of a word or higher level recognition system, such curtailed output is not enough since the system will need fuller information for generating and managing the string hypotheses.

## 5.4 Experimental Results on Basic Representations

We used two basic representations for testing the performance of the Fisher character recognition. One is the ECV representation described in the previous section and the other is point tangent based, called tangential feature vector (TFV for short). In more detail, let $S =< p_1, p_2, \ldots, p_N >$ be the normalized local filtering result of an input character. Then

$$\begin{aligned} \text{TFV}(S) \quad = \quad & < \tan(p_1), \theta(p_1), U(p_1), \tan(p_2), \theta(p_2), U(p_2), \\ & \ldots \quad , \tan(p_N), \theta(p_N), U(p_N) > \end{aligned}$$

where $\tan(p_i)$ is the tangent of the point $p_i$. $\theta(p_i)$ is the accumulation of the signed tangent angle values up to the point $p_i$ and $U(p_i)$ is the unsigned accumulation of the tangent angles up to the point $p_i$. The same Fisher training procedure works on the TFV representation. For training, 1040 lowercase cursive letter samples were used. For testing the performance, a set of 520 lowercase cursive letters was used, which is disjoint from the training set. The test has two parts: one is self-test and the other is disjoint-test. The self-test is performed on the training data and measures the level of learning that took place. The disjoint-test is done on the test data set that is disjoint from the training data set, and measures the generalization capability acquired by the system through the training. The top choice accuracy is tested in each character class and the global average performance is also measured. See Table 1 and Table 2.

If the character recognizer is a standalone system, that is, works only as an isolated character level recognition, then the only criterion that matters would be the top choice accuracy since such task would be one-time invocation process. For a character recognizer that works as a component of word recognition system, just reporting the top choice would not be enough. Instead, an output of a set of candidate characters with the corresponding confidence values, which are worth considering is more desirable and in most cases is required, for the recognition engine to work robustly at word level. To this end, the Fisher character matching produces the set of candidates by keeping only the character classes that have its Fisher matching score within the upper 40% of the top score. The numbers labeled "average candidate set size" in Table 1 and Table 2 are the global average sizes of all the character candidate sets produced during the tests. The candidate set size has significant impact both on the speed and the accuracy of the system performance. The recognition engine generates and grows word hypotheses by combining the character hypotheses and therefore a small increase in the number of character candidates will turn into much larger number of word level hypotheses. This not only slows down the system' s working, but also

| Class | Self-test | Disjoint-test |
|---|---|---|
| a | 97.5% | 95.0% |
| b | 100.0% | 100.0% |
| c | 95.0% | 100.0% |
| d | 100.0% | 100.0% |
| e | 92.5% | 85.0% |
| f | 92.5% | 95.0% |
| g | 97.5% | 95.0% |
| h | 97.5% | 100.0% |
| i | 85.0% | 90.0% |
| j | 100.0% | 80.0% |
| k | 95.0% | 85.0% |
| l | 100.0% | 90.0% |
| m | 92.5% | 90.0% |
| n | 92.5% | 95.0% |
| o | 95.0% | 100.0% |
| p | 97.5% | 100.0% |
| q | 82.5% | 45.0% |
| r | 100.0% | 100.0% |
| s | 90.0% | 95.0% |
| t | 100.0% | 94.7% |
| u | 85.0% | 95.0% |
| v | 100.0% | 75.0% |
| w | 90.0% | 95.0% |
| x | 100.0% | 100.0% |
| y | 92.5% | 85.0% |
| z | 90.0% | 95.0% |
| Global average performance | 94.62% | 91.54% |
| Average candidate set size | 11.87 | 13.57 |

**Table 1.** Character recognition tests on ECV representation.

| Class | Self-test | Disjoint-test |
|---|---|---|
| a | 92.5% | 95.0% |
| b | 95.0% | 100.0% |
| c | 87.5% | 90.0% |
| d | 100.0% | 100.0% |
| e | 82.5% | 80.0% |
| f | 90.0% | 70.0% |
| g | 100.0% | 100.0% |
| h | 95.0% | 90.0% |
| i | 60.0% | 60.0% |
| j | 100.0% | 100.0% |
| k | 92.5% | 85.0% |
| l | 92.5% | 85.0% |
| m | 95.0% | 95.0% |
| n | 95.0% | 90.0% |
| o | 97.5% | 100.0% |
| p | 100.0% | 95.0% |
| q | 97.5% | 100.0% |
| r | 90.0% | 85.0% |
| s | 97.5% | 100.0% |
| t | 95.0% | 52.6% |
| u | 85.0% | 85.0% |
| v | 92.5% | 75.0% |
| w | 90.0% | 95.0% |
| x | 92.5% | 85.0% |
| y | 90.0% | 65.0% |
| z | 95.0% | 95.0% |
| Global average performance | 92.31% | 87.5% |
| Average candidate set size | 16.57 | 17.72 |

**Table 2.** Character recognition tests on TFV representation.

introduces many spurious ambiguities leading to more misjudgments on the part of the recognition engine. As seen in the Table 1 and Table 2, the size number is not small for the Fisher character recognizers based on the ECV and the TFV features and they keep on average about half of all the classes as candidates. The problem stems from the nature of Fisher discriminant analysis or other linear analysis techniques, that the evaluation scores they generate have linearity in their distribution. For example, Figure 13 shows an example distribution of the sorted scores of the candidate characters produced by ECV based Fisher recognizer with

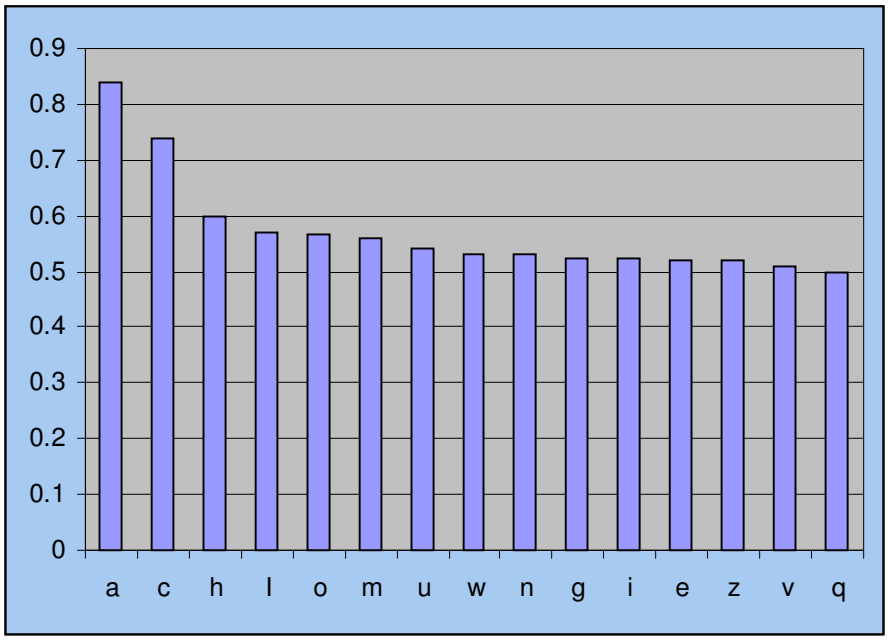



**Figure 13.** The distribution of the candidate characters and their scores produced by ECV based recognizer with an input of "a."

an input of "a." Therefore the distribution does not have a structure that is easy to distinguish candidates and non-candidates and for reasonable robustness the system needs to keep not a small number of candidates. This drawback will be addressed in the next section with a new recognizer that integrates multiple representations.

### *5.5 Multiple Experts Fusion*

The rationale behind the paradigm of multiple experts fusion is that rather than searching for a single omnipotent representation, a group of different and simpler features compensating each others' weaknesses while retaining one' s own strength, sounds more natural and effective. Conventional fusion approaches typically involve designing the topology of interconnections between the recognizers ([1], [28], [63]), therefore the overall mechanism is implicit. Our information fusion is unique in the sense that it has an explicit strategy that compares the consistency of the two interpretations coming from each recognizer, thereby re-computing the scores and the ranks of the candidate characters. The new character recognizer implementing the fusion will not only improve in accuracy, but will also achieve the desirable property of more selective response pattern mentioned in the previous section. To this end we will use ECV and TFV as the two different base representations. Let ECV-FCR be the Fisher character

recognizer based on ECV representation and TFV-FCR be that on TFV representation. The outputs of the ECV-FCR and TFV-FCR will be integrated at the module called Fisher Fusion Module that outputs the final result. See Figure 14.

Given a character segment input, the ECV and the TFV vectors are extracted and they are fed to the corresponding base recognizers. In each base recognizer, after the Fisher matching the candidates that have scores within the top 50% from the top score will be filtered and are output in a sorted list. Let these lists be $L_{\text{ECV}}$ and $L_{\text{TFV}}$ respectively for ECV and TFV representations. $L_{\text{ECV}}$ and $L_{\text{TFV}}$ may have different elements and different lengths. As seen in Table 1 and Table 2 ECV-FCR performs more reliably, so we take $L_{\text{ECV}}$ as the main information and the list $L_{\text{TFV}}$ is compared with it to proceed with the fusion process. For each element $<c, f_c>$ pair in $L_{\text{ECV}}$, where $c$ is a class label and $f_c$ is its Fisher matching score, we try to find $<c, g_c>$ in $L_{\text{TFV}}$. If $<c, g_c>$ pair is not found in $L_{\text{TFV}}$ then it means TFV-FCR missed the class $c$ as a candidate, so $f_c$ is penalized heavily by reducing it to 50%. Otherwise, we reevaluate $f_c$ as follows. Let $\text{top}_{\text{TFV}}$ be the top score of $L_{\text{TFV}}$, then we compute the ratio of $g_c$ with $\text{top}_{\text{TFV}}$. That is, $f_c$ is updated into $f_c'$ as below

$$f_c' = f_c \cdot \frac{g_c}{\text{top}_{\text{TFV}}}.$$

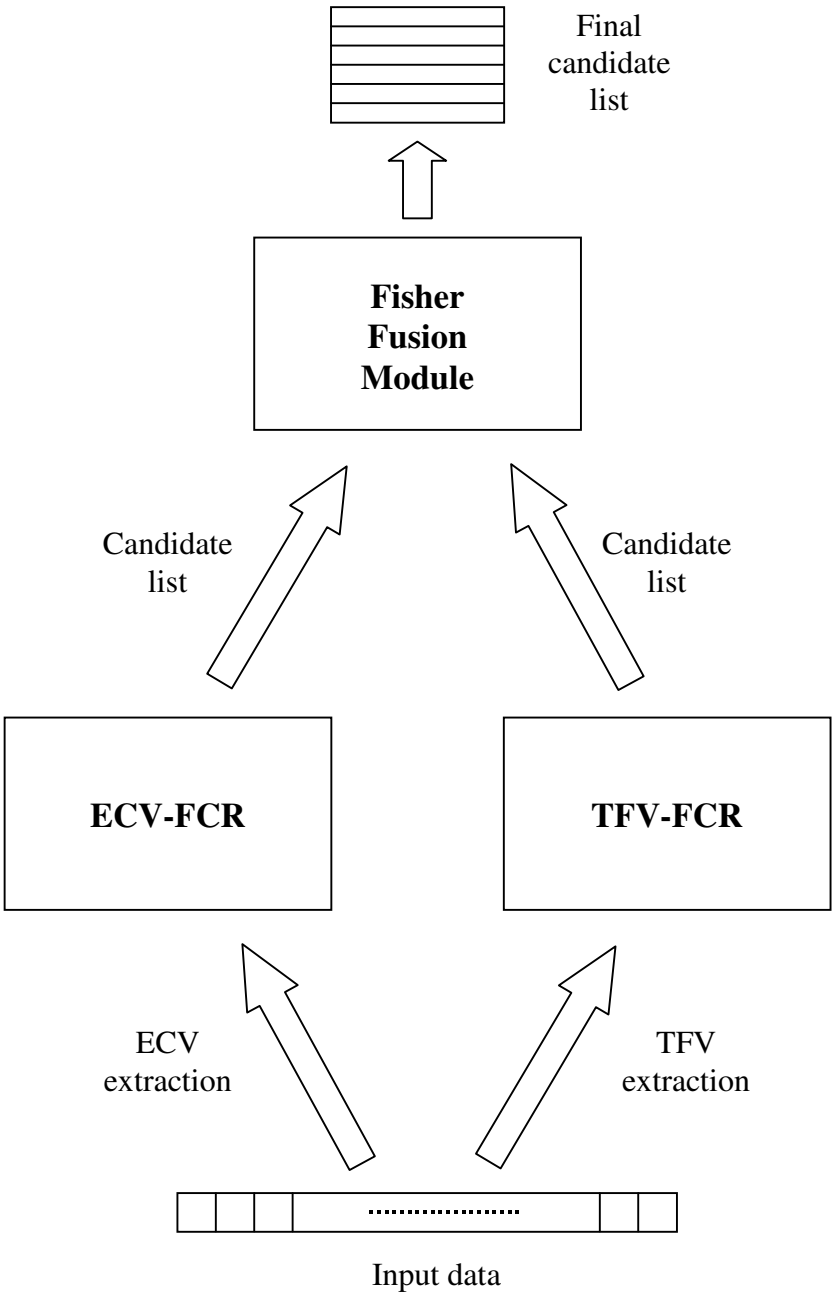



**Figure 14.** Overall architecture of feature-fusion character recognizer.

The updated pair $<c, f_c'>$ is sorted into the final list $F$. At the end, $F$ is pruned to have the items whose score is within the top 40% of its top score. The steps at the Fisher fusion module are summarized below

1. Inputs are $L_{\text{ECV}}$ and $L_{\text{TFV}}$.
2. For each $<c, f_c>$ in $L_{\text{ECV}}$
   .2.1. Find $<c, g_c>$ in $L_{\text{TFV}}$.
   .2.2. Compute $f_c'$ and sort $<c, f_c'>$ pair into the list $F$.
3. Prune out items in $F$ whose score is below 60% from the top score.
4. Return $F$.

### *5.6 Experimental Results on the Fusion Matching*

Table 3 shows the character recognition test results using the combined representation fusion matching. Table 4 compares the recognition rates with those of the base representations. For the disjoint-test, the error rate reduction rates from the ECV and the TFV representations are 12.5% and 40.8% respectively. More significant improvements were obtained in reducing the size of the output candidate list. Now the new fusion matching produces the average number of candidates of 3.68 for the disjoint-test. This is a large improvement from the average number of candidates of 13.58 and 17.73 generated by the ECV and the TFV matchings respectively, for the same test (Table 5). Therefore we have

obtain a target character recognizer that has more selective response pattern with better accuracy. The fusion matching will run a little slower because it consists of the two component matchings on base representations. But the compact candidate set will more than compensate the extra cost since the recognition engine will generate and handle significantly smaller number of word level hypotheses, and the system performance will speed up overall.

| Class | Self-test | Disjoint-test |
|---|---|---|
| a | 97.5% | 100.0% |
| b | 100.0% | 100.0% |
| c | 95.0% | 100.0% |
| d | 100.0% | 100.0% |
| e | 90.0% | 85.0% |
| f | 92.5% | 90.0% |
| g | 100.0% | 100.0% |
| h | 100.0% | 95.0% |
| i | 87.5% | 85.0% |
| j | 100.0% | 100.0% |
| k | 97.5% | 85.0% |
| l | 97.5% | 85.0% |
| m | 97.5% | 95.0% |
| n | 97.5% | 95.0% |
| o | 95.0% | 100.0% |
| p | 100.0% | 100.0% |
| q | 92.5% | 70.0% |
| r | 100.0% | 100.0% |
| s | 97.5% | 100.0% |
| t | 100.0% | 94.7% |
| u | 92.5% | 95.0% |
| v | 100.0% | 80.0% |
| w | 92.5% | 95.0% |
| x | 97.5% | 100.0% |
| y | 97.5% | 80.0% |
| z | 95.0% | 95.0% |
| Global average performance | 96.63% | 92.5% |
| Average candidate set size | 3.16 | 3.67 |

**Table 3.** Character recognition tests on combined representation fusion matching.

| Self-test | | | |
|---|---|---|---|
| Representation | Recognition rate | Error rate | Fusion's error reduction rate |
| Fusion | 96.6% | 3.4% | |
| ECV | 94.6% | 5.4% | 37.4% |
| TFV | 92.3% | 7.7% | 56.2% |
| Disjoint-test | | | |
| Representation | Recognition rate | Error rate | Fusion's error reduction rate |
| Fusion | 92.6% | 7.4% | |
| ECV | 91.5% | 8.5% | 12.5% |
| TFV | 87.5% | 12.5% | 40.8% |

**Table 4.** The fusion matching' s characterecogniton performance and comparison with that of ECV and TFV representations.

| Self-test | | |
|---|---|---|
| Representation | Average candidate set size | Candidate number reduction rate of fusion matching |
| Fusion | 3.16 | |
| ECV | 11.87 | 73.4% |
| TFV | 16.57 | 80.9% |
| Disjoint-test | | |
| Representation | Average candidate set size | Candidate number reduction rate of fusion matching |
| Fusion | 3.68 | |
| ECV | 13.58 | 72.9% |
| TFV | 17.73 | 79.2% |

**Table 5.** The fusion matching' s reduction effect on the size of candidate set compared with the matchings of ECV and TFV.

# CHAPTER 6. Word Recognition Engine

Unlike a standalone character level recognizer, the word level recognition task faces two big challenges: the segmentation and the management of the word level hypotheses. The recognition engine is the place where the issues are addressed and to this end the various information from the rest of the system is integrated and utilized. In our system the segmentation is partially addressed in the preprocessing step by generating the tentative segmentation points. The segmentation hypothesis points are just tentative possibilities and how to actually try the permissible character segments is handled dynamically in the recognition engine. Although the recognition engine is where the word level recognition really takes place and is the center of global information fusion, relatively little effort has been expended on an improvement of it, and the conventional approach is using the Hidden Markov Model The conventional method of evaluating string level hypothesis in handwriting recognition has been accumulating the confidence values assigned to each string component ([6], [13], [27], [39], [71]).

### 6.1 Conventional Word Hypothesis Evaluation and Its Problems

In handwriting recognition context, the HMM's were originally meant the models for character or sub-character units. The HMM, however, can be extended to work at the word level by concatenating the character level HMM's according to a given grammar that specifies legal combinations. Then the standard Viterbi search process constructs the possible paths aligning with the input and the best path can be found by minimizing or maximizing on the path scores. Let $X =< x_1, x_2, \ \ldots \ , x_T >$ be an input sequence and consider a possible partition of $X$

$$\tau = X(\tau_1), X(\tau_2), \ \ldots \ , X(\tau_K)$$

where $0 \le \tau_1 \le \tau_2 \le \ \cdots \ \le \tau_K = T$ and $X(\tau_i) =< x_{\tau_{i-1}+1}, x_{\tau_{i-1}+2}, \ \cdots \ , x_{\tau_i} >$. Each segment $X(\tau_i)$ in the partition can be interpreted as modeled by a character HMM whose index is the value of the variable $h_i$. A path in this context is then defined as $H = <h_1, h_2, \ \ldots \ , h_K>$. Then estimating the best path $\overline{H}$ can be formulated as

$$\Pr(X \mid \overline{H}) = \max_{H,\tau} \left\{ \Pr(X(\tau_1) \cdots X(\tau_K) \mid h_1 \cdots h_K) \right\}$$

or, if we assume the conditional independence among the segments

$$\Pr(X \mid \overline{H}) = \max_{H,\tau} \left\{ \prod_{j=1}^{K} \Pr(X(\tau_j) \mid h_j) \right\}.$$

Therefore the problem can be recast as a global level HMM of lower level, that is character level, models, and becomes amenable to dynamic programming similarly as in original HMM ([71]). The probability quantity can be replaced with other kind of confidence metric *C* related with other kind of model *m* than HMM with a path now defined as $M = <m_1, m_2, \ldots, m_K>$:

$$C(X \mid \overline{M}) = \max_{M,\tau} \left\{ \prod_{j=1}^{K} C(X(\tau_j) \mid m_j) \right\}$$

One problem of this approach is that the multiplication of the more general component confidence values may not be natural if it is not drawn from real statistics. For example, let' s suppose the confidence is a score evaluated from a

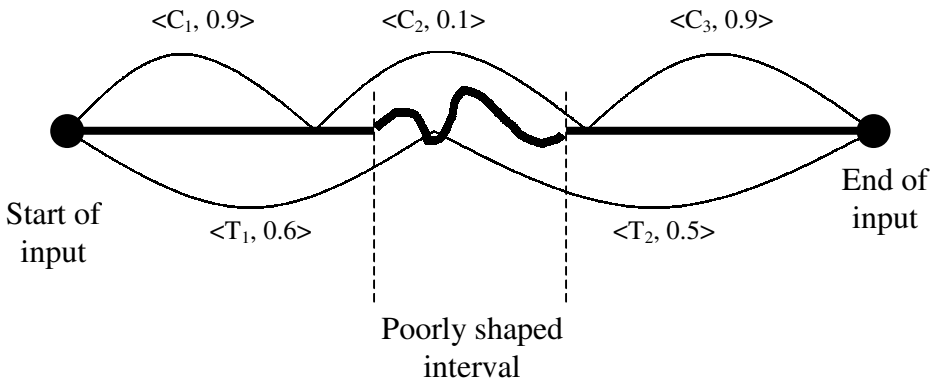



**Figure 15.** A hypothetical word input whose middle part is poorly shaped, and the arcs representing the segments identified by the system. The pair <*s*, *n*> on each arc is the interpretation of the corresponding segment, where *s* is the character label and *n* the character recognition score.

shape matching. In this case, it is not clear what the multiplicative path score may mean. Rather the average of the component scores may make more sense, because it is clearly a measure of how well the component characters are shaped overall. The other problem is the brittleness that may be introduced by the multiplicative nature of the path score. For example, suppose that one component character was poorly shaped and scored low. In this case it is possible that the correct hypothesis may not survive the propagation or may break down at the poorly formed character. In Figure 15, let' s suppose that the string $C_1C_2C_3$ is the correct interpretation of the input. Then its related multiplicative path score is $0.9 \times 0.1 \times 0.9 = 0.081$ and the string will be pushed away by the wrong interpretation $T_1T_2$ having the path score of $0.6 \times 0.5 = 0.3$. Also the multiplicative path score has the effect of preferring shorter interpretations to longer ones because an interpretation will have its confidence value reduced multiplicatively as it concatenates a character and grows in length.

## *6.2 Word Recognition as a Graph Search Problem*

A word level hypothesis is essentially a string of character hypotheses and its confidence score. A character hypothesis consists of the interpreted character class label, the evaluation score computed by the character recognizer, and other information like its starting and ending points, the bounding box, etc. In our

system the score of a word hypothesis is not the conventional accumulation of the log-likelihood of the character confidences. Instead it is taken as the average of the scores of the component character hypotheses. Therefore our word hypothesis score is a measure of how well its contained characters are shaped overall in relation with the Fisher character models. The robustness of the new path scoring can be shown by retaking the example of the previous section drawn from Figure 15. The averaging method gives the path score of (0.9+0.1+0.9) / 3 = 0.63 to the correct hypothesis $C_1C_2C_3$, which is higher than 0.55 assigned to the wrong hypothesis $T_1T_2$. With the segmentation points and the rule to form the character segments, the structure of the recognition engine can be formulated as a graph. The segmentation points along with the starting and the ending points of the input can be regarded as the set of vertices. The starting point corresponds to the special vertex *S* and ending point to the special vertex *T*. A directed edge $s \rightarrow t$ between the two nodes *s* and *t* can be regarded as representing the segment starting from the segmentation point represented by *s*, to the segmentation point represented by *t*. Which edges are present in the graph is determined by the rule in the recognition engine. Conceptually this graph is called the "segmentation graph" of a given input because it describes the structure of the segmentation possibilities. See Figure 16. Note that the graph is a DAG (directed acyclic graph) since the edges are directed forward and never go back in time. Next the segmentation graph is

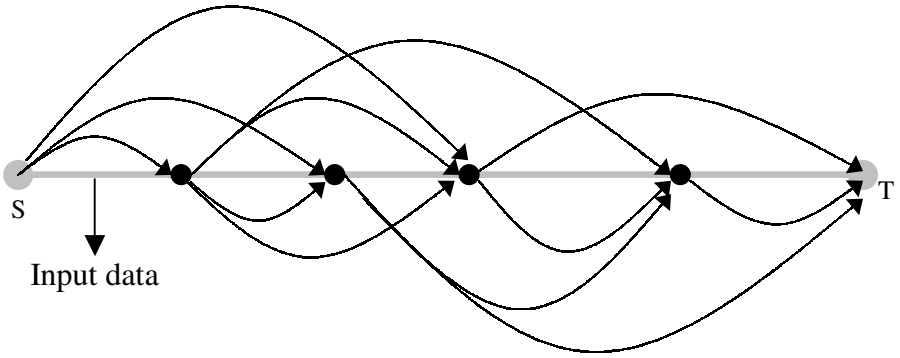



**Figure 16.** The segmentation graph of an input. The segmentation points are indicated by the dots over data, which are also the vertices of the graph along with S and T. The edges installed by the generation rule are indicated by the arcs with arrows.

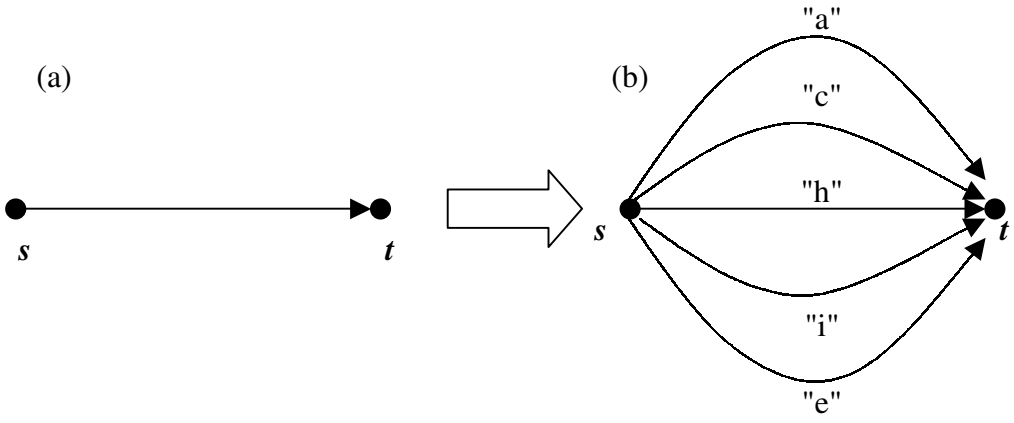



**Figure 17.** (a) An edge from the segmentation graph (b) the same edge expanded into multiple interpreted edges in the interpretation graph.

expanded into a graph called "interpretation graph." In the interpretation graph, each unlabeled directed edge in the segmentation graph is interpreted, by invoking the character recognizer with the corresponding segment as input, and is replaced by a set of edges having the same source and destination vertices, but now being labeled with the character class index and the evaluation score (see Figure 17).

In this setting the word recognition problem can be transformed into finding the optimal path from the starting vertex *S* to the ending vertex *T* in the interpretation graph ([39]). The score of a path is a function of the scores of the edges contained in the path. Traditionally the path score function is the summation of the log-likelihood of the individual edge score. The path score function in our system, however, is the average of the component character scores for the reasons explained the previous section. In addition, we use a variant of Viterbi search in which more than one predecessors are kept in a list at each propagation points. If we have an oracle metric that can evaluate word hypotheses with infinite precision, then we would not need to care but use the standard Viterbi search which runs optimally assuming that word evaluation metric is correct. In practice, the reality is less than ideal especially in statistical pattern recognition context and there are many sources of noise like perturbations in data, in feature extraction, ambiguities and confusions in the character recognition process, and so on. Therefore the optimality of the Viterbi search can not be expected because the

hypotheses evaluation metric itself is imperfect. Our strategy to cope with this problem is to retain more than one predecessor hypotheses at each propagation point in our recognition engine.

## *6.3 Hypotheses Propagation Network*

The recognition engine of our system, named "Hypotheses Propagation Network" (HPN), is a two-dimensional lattice in structure and implements the interpretation graph and the search algorithm on it described in the previous section. One dimension of HPN is the time in terms of the segmentation points: the first segmentation point is the time-1, the second segmentation point the time-2, and so on. The other dimension ranges over the indices of the character classes (see Figure 18). The intersections of HPN lattice correspond to a node $N(t, m)$, where $t$ is a time and $m$ is the index of a character class. Given two nodes $N(t', m')$ and $N(t, m)$ where $t' < t$, the edge $N(t', m') \rightarrow N(t, m)$ corresponds to the segment from $t'$ to $t$ in the input that is interpreted as the class $m$, having $m'$ as the predecessor.

### 6.3.1 The HPN Search

In HPN, the interpreted edges are constructed dynamically by the segment generation rule. At each processing time $t$, the HPN looks back in time and ranges

over the look-back windows of sizes from 1 to $w$ that is the maximum size. Therefore the $i$-th look-back window $W_i$ starts at time $t$-$i$ and ends at $t$ (see Figure 19). For each $W_i$, the HPN sends the feature vector extracted from the corresponding data segment to the component character recognizer which in turn returns the list of candidates $\{m_1, m_2, \cdots, m_k\}$. For each of the candidate $m_j$, the HPN iterates over the nodes $N$ ($t$-$i$, $m'$) and considers whether or not to place the edge $N$ ($t$-$i$, $m'$) $\rightarrow N$ ($t$, $m_j$) in the graph. The decision is based on the information coming from various hypothesis-filtering models. One example is the use of lexicon or dictionary. That is, if the string corresponding to a hypothesis that has been propagated to the node $N$ ($t$-$i$, $m'$) forms a legal prefix of the lexicon if $m_j$ is concatenated to it, then the new prefix is legal and the edge is permissible.

As stated above, the main difference of HPN search from other dynamic programming search techniques is the use of multiple predecessors. At each node $N$ ($t$, $m$), the HPN stores the list of word level hypotheses $H$ ($t$, $m$) each of whose elements is a hypothesis that ends at time $t$ with the character of class $m$ as the last character of its string. For an edge $N$ ($t_1$, $m'$) $\rightarrow N$ ($t_2$, $m$), the HPN iterates on each element of $H$ ($t_1$, $m'$) and computes a new hypothesis with the score of the edge

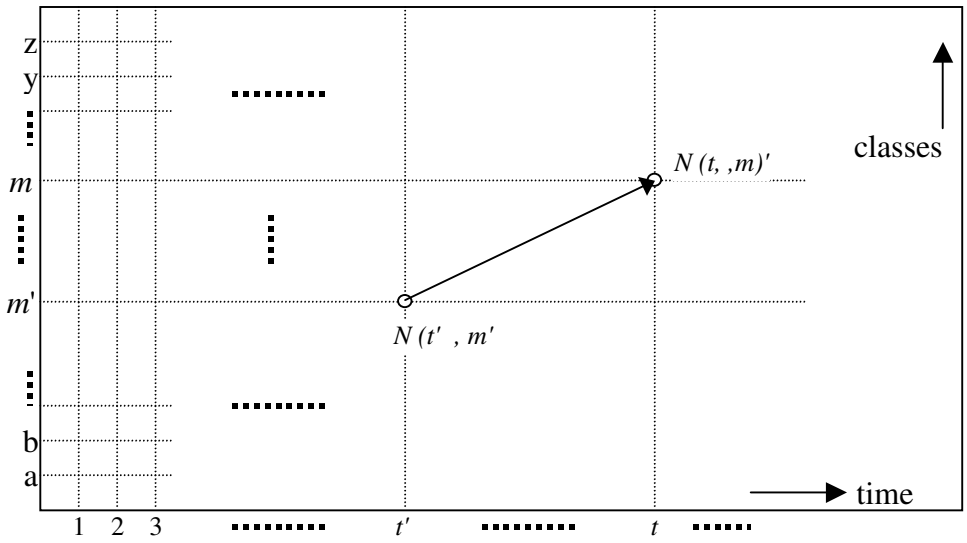



**Figure 18.** The lattice structure of HPN and an example of edge from the node $N(t', m')$ to node $N(t, m)$.

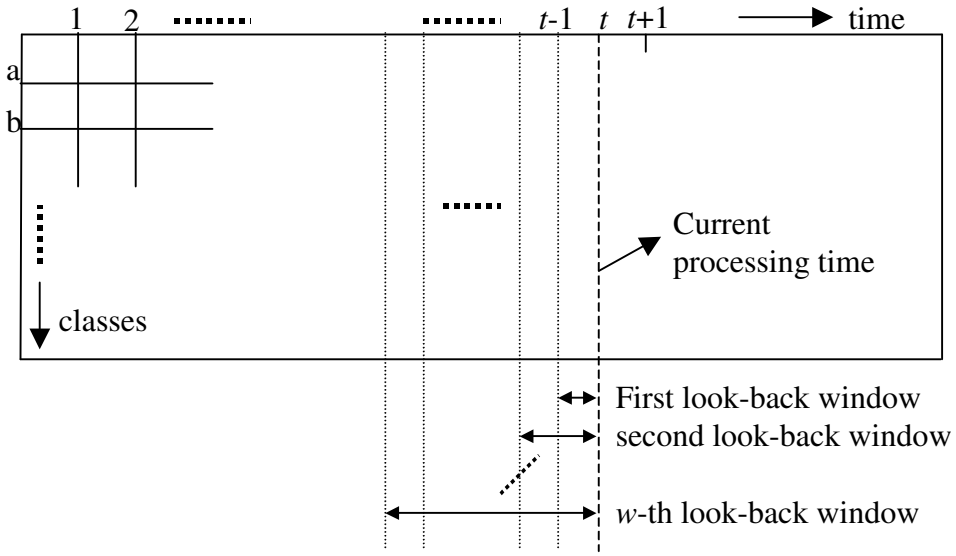



**Figure 19.** The look-back windows of HPN, from the current processing time.

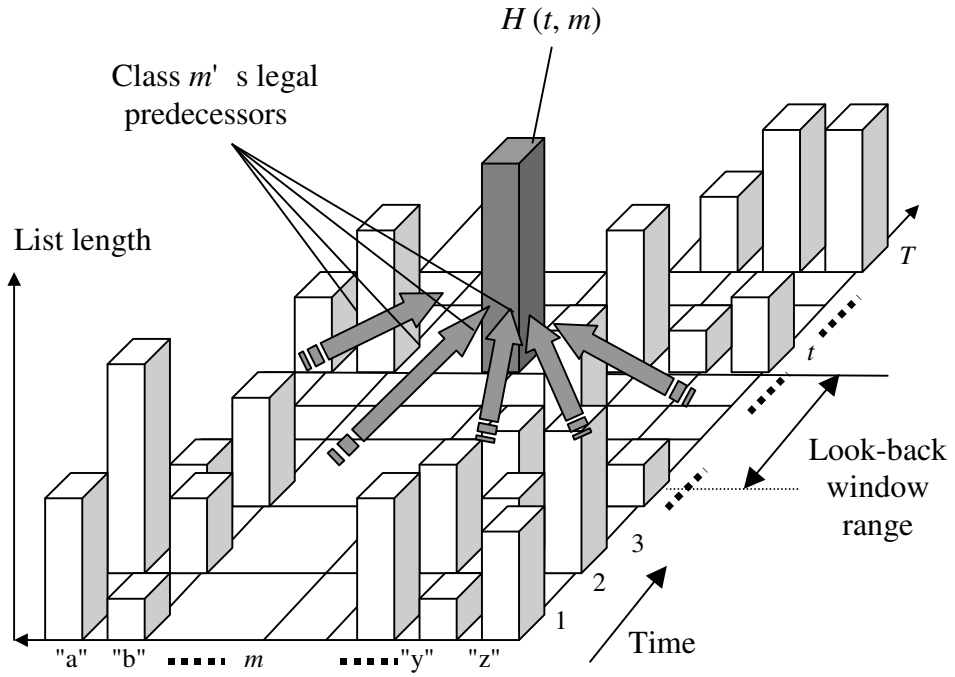



**Figure 20.** Snapshot of HPN search building the hypothesis list $H(t, m)$ for time $t$ and the class $m$.

and the class label $m$ and inserts it into the list $H(t_2, m)$. See Figure 20. After the last time $T$, the lists $H(T, *)$ are merged into a single sorted list $H$ that is the sequence of candidate words recognized by the system, ordered according to the confidence values.

### 6.3.2 Pruning on the Hypothesis List

Now allowing multiple hypotheses ending on the same propagation node, with no limit on the length of the lists $H(t, m)$ can lead to intractable computation since the hypothesis growing and propagating process would encounter exponentially growing number of predecessor as the time proceeds. The hypotheses filtering models of the next chapter are ways to prune the search by blocking the propagation of the hypotheses that are detected as not consistent with the various context information. On the HPN itself, the search structure is taken similar to the beam search. On one level, all $H(t, m)$ are restricted to have at most $C$ number of items. At the other level, the maximum number of hypotheses at each time $t$ is restricted to $U$. This is achieved by keeping at most the top $U$ hypotheses at time $t$, in terms of their hypothesis scores, that are distributed over the $H(t, *)$'s. The numbers $C$ and $U$ are among the system parameters controlling the accuracy and the running speed.

## *6.4 Experimental Results*

The word recognition tests were performed in 4 different settings on a set of 100 words collected by the author and the results are summarized in the following tables. All tests were run with a lexicon of 450 words and the ligature modeling described in CHAPTER 7.2. Table 6 shows the results of the tests using

the ECV for representation and the two different recognition engine searches, that is HPN search and the Viterbi. The system outputs a list of word candidates sorted according to the confidence values assigned to each candidate. In our setting the rank of the list starts from 0, not 1, so the top choice has rank-0. The performance is taken as correct only if the target word is the same as the top choice. The table also shows the distribution of the ranks of the mis-recognized or non-recognized words. Non-recognition means that the target word is not included in the candidate list. The average rank in the table means the mean value of the ranks of the mis-recognized words.

| ECV matching | HPN search | Viterbi search |
|---|---|---|
| Recognition rate | 85% | 73% |
| Rank distribution of mis/non-recognized words | | |
| rank-1 | 7 | 3 |
| rank-2 | 3 | 1 |
| rank-3 | 1 | 0 |
| out of rank | 4 | 23 |
| Average rank | 1.45 | 1.25 |

**Table 6.** Word recognition performances on the ECV representation and its Fisher matching with different recognition engine searches.

From the table, it is clear that our strategy to keep multiple predecessors at each propagation points is more robust than the standard Viterbi search. Most of the mistakes it makes are non-recognition, not mis-recognition, which means that

the correct hypothesis did not survive to propagate to reach the end time in many cases. The point that this situation can be remedied by keeping more opportunities open in the form of having multiple predecessors at each propagation points as is done in the HPN, is verified.

Table 7 shows the same kind of tests but done on the fusion representation and its Fisher matching, instead of the ECV. The fusion representation with the HPN search improved the recognition rate by 3% or reduced the error rate by 20%, over the ECV-HPN combination. Also there are no non-recognized words in Fusion-HPN case. This is attributable to the more discriminative property of the fusion matching obtained by tuning the character scores with the information from the different representations. Here again, there is a big gap in the performance between the HPN and the Viterbi searches.

| Fusion matching | HPN search | Viterbi search |
|---|---|---|
| Recognition rate | 88% | 68% |
| Rank distribution of mis/non-recognized words | | |
| rank-1 | 8 | 2 |
| rank-2 | 1 | 0 |
| rank-3 | 2 | 0 |
| rank-4 | 1 | 0 |
| out of rank | 0 | 30 |
| Average rank | 1.67 | 1.0 |

**Table 7.** Word recognition performance on the fusion representation and its Fisher matching with different recognition engine searches.

# CHAPTER 7. Hypotheses Filtering Models

With no contextual analyses and relying only on the scores assigned by the component character recognizer, the recognition engine is bound to generate a large number of hypotheses, many of them spurious, because it lacks higher level contextual sight. For example, some English characters are inherently ambiguous: "o" and "0," "1" and "I," "1" (one) and "l" (lowercase "L"), to name a few. In many cases, the ambiguities of such characters can only be resolved by taking the given context into account. With a contextual perspective, however, many of the generated hypotheses may not make sense any more and may be eliminated from further propagation. The economy of hypothesis propagation obtained by such filtering substantially contributes to the performance of the system.

## *7.1 Dynamic Lexicon*

One of the most common forms of context information is the use of a lexicon, that is a dictionary of permissible words. The lexicon filtering is usually applied either after the generation of word candidates or during the propagation process but the former approach can little contribute in pruning the search space.

In still another approach, the lexicon is dynamically reduced by an early-recognition stage using an approximate but fast recognizer-like method ([67]). The purpose of the early-recognition is to determine quickly a small set of plausible candidate words, thereby reducing the size of the lexicon. The rest of the system then focuses only on the reduced lexicon. A statistical *n*-gram modeling of character sequences is also used and makes sense if the system tries to recognize out-of-lexicon words. In our system, the dictionary is organized into the compact "trie" data structure and is looked up dynamically when the HPN tries to propagate a word hypothesis. A non-leaf node of the lexicon trie corresponds to a legal prefix of the system, that is a proper prefix of a full word. The full words are represented by the leaves of the trie. Each word level hypothesis has a pointer set to the node of the trie that corresponds to the prefix string that the hypothesis represents. So, when the HPN processes a hypothesis *h* ending at an HPN node for propagation with an edge interpreted as a character class *x*, the HPN looks up the trie node of *h*. If the node has *x* as a successor, then it means that *h* can be extended to *h'* having *x* as the last character. This way, the system dynamically prevents a hypothesis from being extended to a non-permissible string and all the word level hypotheses generated and propagated are limited to legal prefixes of the lexicon.

## 7.2 Ligature Modeling

Much of the variability of a character written in a cursive script comes from how it connects with the surrounding characters, that is by the ligatures. This phenomenon is similar to what is called co-articulation in speech recognition, in which a phoneme has greater variation in pattern around the border with the neighboring phonemes. Ligatures are not necessary but ever present in continuous cursive writing due to physical constraints in and psychological need of fast writing. Traditionally, most approaches simply ignored them ([7]), some others trying to deal with the ligatures mainly used hidden Markov models ([27], [71]). Still others used very expensive methodology ([35]), for example by using the classes of continuously written character pairs, thereby proliferating the system's classes in large number. The rationale of our modeling the ligatures is that there are certain regularities in their formation, and they can be used to measure a well-formedness of a hypothesis. By appropriately modeling away these dummy bridges, we can expect more regularity in the shapes of characters. The difficulty of modeling ligatures, however, is that they are context sensitive and taking the full contexts into account leads to a proliferation of models. For example, a straightforward ligature modeling of 26 lowercase English letters by HMM will wind up with $26 \times 26 = 676$ models. The large number of models inevitably increases the amount of computing time and storage. Depending on a language

and its structure, more efficient modeling is possible (for example, Korean). In [71], a context-free grammar was defined to model the way the ligatures can be formed between the component sub-character units in Korean language that has a concise syntax specifying how the sub-character units can be combined to form a legal character, and HMM's were trained to actually model each type of ligature. But for English there are few alphabetic constraints on the formation of ligatures that could be exploited for more concise modeling.

### 7.2.1 Using Feature-Link Code to Model Ligatures

The feature-link code (FLC) was introduced in CHAPTER 4.4 and its computation was also described there. In Figure 10, the 24 convexity-directional feature-link templates were shown along with the indices assigned to the individual templates. In Figure 11, example handwriting of the word "day," its feature-link intervals and the FLC's were demonstrated. From the example, it is clear that the FLC is a compact representation to describe a smooth contour interval. In our system, the feature-link interval is the smallest unit along which the entire input is broken down. A character segment is always formed as a consecutive sequence of feature-link intervals. Therefore a ligature, if present, is also a sequence of feature-link intervals between two character segments. In our system, we use the feature-link intervals to hypothesize about the ligatures. A

ligature segment, however, does not need to contain many feature-link intervals in normal handwriting, because, unlike the characters, it serves merely as a bridge and its shape does not have a complex structure. Except for mis-formed or ill-formed handwriting, it is a smooth interval with certain degree of convexity. This sounds the same as the definition of the feature-link interval and through the FLC we have an efficient way to categorize a ligature hypothesized between two characters. Since the FLC is computed into a table at the pre-preprocessing step, the consumer (that is, the HPN) just needs to look into the table.

For filtering purpose, we set up two lookup matrices IsRequired[$c_1$, $c_2$] and IsLegal[$c_1$, $l$, $c_2$] where $c_1$ and $c_2$ range over character class indices and $l$ is an FLC. IsRequired[$c_1$, $c_2$] has value TRUE if a ligature is required between the two character classes $c_1$ and $c_2$ in continuous writing, and FALSE otherwise. IsLegal[$c_1$, $l$, $c_2$] is set to TRUE if $l$ is a permissible ligature between $c_1$ and $c_2$, and FLASE otherwise. The quantity IsRequired[$c_1$, $c_2$] is used when the segments of $c_1$ and $c_2$ touch each other and therefore no ligature is present. If IsRequired[$c_1$, $c_2$] is TRUE in this case, then it means that $c_1$ followed by $c_2$ is mis-hypothesized since they require a ligature in continuous formation. When two characters $c_1$ and $c_2$ are separated by an interval labeled with FLC $l$, the matrix entry IsLegal[$c_1$, $l$, $c_2$] is looked up by the system to check whether the ligature is permissibly formed. Therefore if a hypothesis that is being considered by the HPN for extension with

such context, is not consistent with the ligature models, such an instance is blocked from further propagation.

With the models explained, the question is how we determine what value to enter into the matrix entries. Ideally, training may do this from the samples and in such case we may use the observed probability quantity instead of Boolean values. But because of the difficulty in obtaining large amount of labeled ligature data, we used hand-designed code instead. The design was meant to be tolerant so that a ligature type between two characters, that is FLC, was made legal unless it is clearly nonsensical. The result is collapsing the large modeling, both in time and storage, down to moderate-sized matrices and one or two matrix-entry lookups.

### 7.2.2 Examples and Experimental Results

Figure 21 shows two examples where the matrices IsRequired[$c_1$, $c_2$] and IsLegal[$c_1$, $l$, $c_2$] are used. Part (a) of the figure shows the incorrect segmentation of "g" into "o" and "j." The string "oj" when written continuously, needs a ligature between the two characters since without one it should look as a shape of "g." Therefore the value of IsRequired["o", "j"] is TRUE. Because the segmentation provides no room for a ligature, the system can detect that "oj" is a wrong interpretation. The next example in Figure 21 is how the quantity IsLegal[$c_1$, $l$, $c_2$] is used. Part (c) of the figure shows a wrong segmentation of "pie" into "jie."

However the direction and the convexity of the hypothetical ligature between "j" and "i" is clearly not a possible pattern under a normal condition, so the system will find FALSE value at the entry IsLegal["j", *l*, "i"] where *l* is the FLC of the ligature.

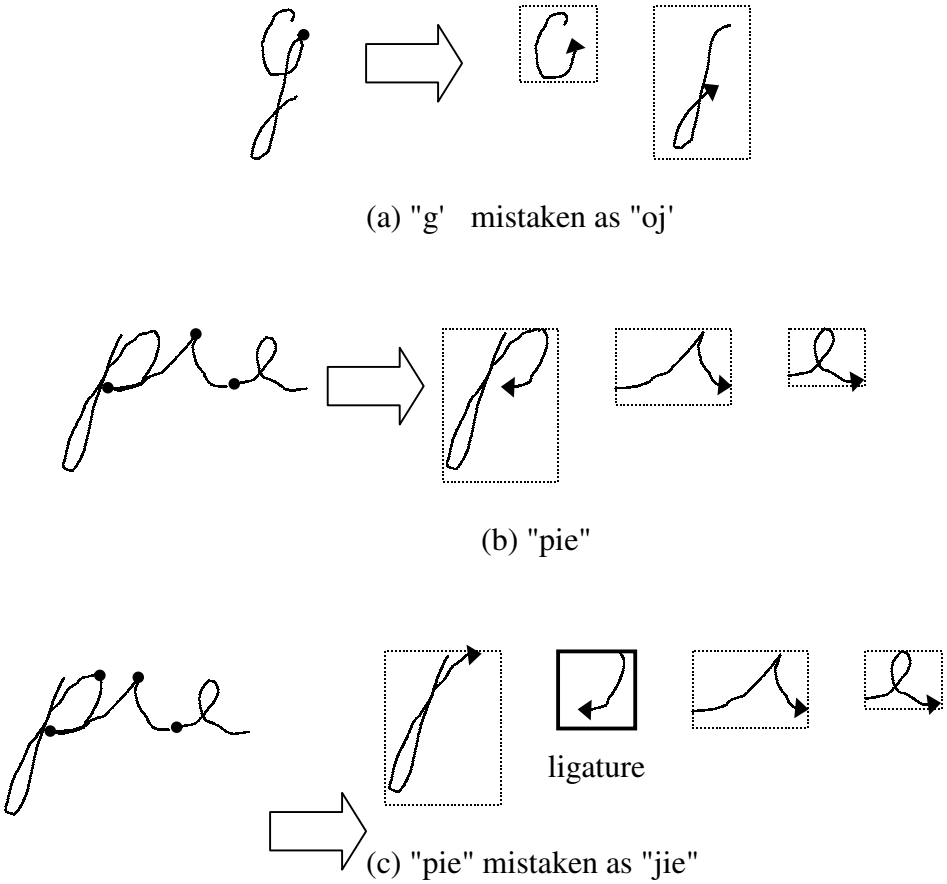



**Figure 21.** Incorrect and correct character segmentations and the ligature modeling. The segment boundaries are indicated as dots. (a) Wrong segmentation of "g" into "o" and "j." This case is rejected by ligature modeling because continuously written "oj" requires a ligature between the segments. (b) Correct segmentation of "pie." (c) Wrong segmentation of "pie" into "jie." The hypothesized ligature is not consistent as a ligature between "j" and "i." Therefore it will not be taken as permissible.

In CHAPTER 6.4, all tests were run with the ligature modeling turned on. To see how the absence of ligature modeling impacts the word recognition performance, a test using the fusion representation and the HPN search was run with the ligature modeling turned off. Table 8 compares the test result with that of the test with the modeling. When the models are turned off, the recognition rate dropped by 19%. A large part of the errors are non-recognition, meaning that the extra confusions caused by no modeling, pushed the correct word hypotheses out of the predecessor lists somewhere during the propagation. The rank distribution of the no modeling case also shows the negative impact on the mis-recognition cases in which the ranks of the targets are scattered wider from the top.

| Ligature modeling | Yes | No |
|---|---|---|
| Recognition rate | 88% | 69% |
| Rank distribution of mis/non-recognized words | | |
| rank-1 | 8 | 10 |
| rank-2 | 1 | 5 |
| rank-3 | 2 | 1 |
| rank-4 | 1 | 2 |
| rank-5 | 0 | 1 |
| rank-6 | 0 | 1 |
| out of rank | 0 | 11 |
| Average rank | 1.67 | 2.1 |

**Table 8.** The comparison of word recognition performances with and without the ligature modeling. Both tests were done using the fusion representation and the HPN search.

### *7.3 Visual Bigram Modeling*

The ligature modeling is an example of hypothesis filtering using visual context. It modeled the range of permissible connecting patterns between the character hypotheses. In the visual bigram modeling we compare the geometric characteristics of the character hypotheses directly to check if it is consistent with the context drawn from the pair of hypotheses. That is, the variability of relative geometric information like the relative size and positioning of a character unit in comparison to its neighbors, is modeled and the system evaluates the fitness of a hypothesis according to the models. An isolated character can be highly ambiguous while its identity can be more evident when put in a context. For example, the word "go" can be confused with "90" when the system considers each individual character. But if the relative size and the positioning of the second

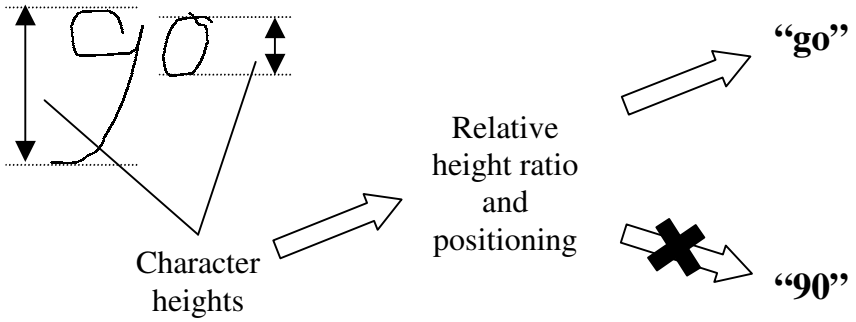



**Figure 22.** The example characters that are ambiguous individually but are obvious when seen in the context.

character hypothesis are taken into account, in relation with the first character hypothesis, it becomes clearer that the "go" is the more likely interpretation (Figure 22). Some studies ([35], [21]) tried to incorporate this modeling into the character recognizer, that is, by training the recognizer with a data set of all possible character pairs. This approach, however, has the problem in that it requires a huge amount of training samples and much larger number of class models. Other efforts available from the literature have tried to either identify and use ascender or descender sub-strokes of writing ([55]), or rely on the covariance matrix of the model parameters ([79]). We will show a new and more intuitive approach that does not need such information and the model parameters are learned by training on the data samples.

### 7.3.1 Modeling Visual Bigram Information

When the feature vector is extracted by the local filtering for a character segment, it also computes the bounding box of the segment. Here a visual bigram $<c_1, c_2>$ is a pair of two consecutive character hypotheses $c_1$ and $c_2$, along with the information of their respective bounding boxes. Given $<c_1, c_2>$, let

- $\text{top}_i$ be the top-most $y$-coordinate of the bounding box of $c_i$
- $\text{bottom}_i$ be the bottom-most $y$-coordinate of the bounding box of $c_i$
- $h_i = \text{top}_i - \text{bottom}_i$, that is the height of the bounding box of $c_i$

for $i \in \{1, 2\}$. The combined height of $<c_1, c_2>$ is, $H(<c_1, c_2>) = \max(top_1, top_2) - \min(bottom_1, bottom_2)$. Next, let us define the three functions of $<c_1, c_2>$ as follows:

- Height difference ratio: $HDR(<c_1, c_2>) = \frac{h_1 - h_2}{H(<c_1, c_2>)}$
- Top difference ratio: $TDR(<c_1, c_2>) = \frac{top_1 - top_2}{H(<c_1, c_2>)}$
- Bottom difference ratio: $BDR(<c_1, c_2>) = \frac{bottom_1 - bottom_2}{H(<c_1, c_2>)}$.

Suppose that we have a model $M_H$ that measures the fitness of the height difference ratio of an input class $<c_1, c_2>$. For lowercase English alphabet, each $c_i$ ranges over 26 letter classes, and a straightforward approach would take 26×26 = 676 bigram classes. The information that we want for modeling, however, is the relative size and the positioning between the characters. Therefore by categorizing the letters into groups according to this criterion, the number of needed bigram classes can be greatly reduced. To this end, let us consider the two kinds of sub-strokes: ascender and descender. An ascender is a sub-stroke running beyond the upper-baseline of the lowercase letter and a descender is a sub-stroke running below the lower-baseline of the lowercase letter. All lowercase letters can be divided into three groups: one having ascenders, another having descenders and

lastly the one having none. Table 9 shows the three categories and their member letter classes.

| Ascender or descender | Type name | Members |
|---|---|---|
| Ascender | A | b, d, f, h, k, l, t |
| Descender | D | f, g, j, p, q, y, z |
| None | N | a, c, e, i, m, n, o, r, s, u, v, w, x |

**Table 9.** Three categories of lowercase letters according to the presence of ascender or descender sub-strokes.

The 26 letter classes have been reduced to the 3 classes of A, D, and N representing the ascender group, the descender group and the none-group respectively. Hence we only need to deal with the 9 bigram classes of {A, D, N}×{A, D, N}: <A, A>, <A, D>, <A, N>, <D, A>, <D, D>, <D, N>, <N, A>, <N, D>, and <N, N>. Now the model $M_H$ can be set up for 9 bigram classes as above, instead of 676. The computing steps of $M_H(<c_1, c_2>)$ for the class $<l_1, l_2>$. is

1. Compute HDR($<c_1, c_2>$).
2. Compare HDR($<c_1, c_2>$) with the parameters of $<l_1, l_2>$.
3. Return the confidence value of HDR($<c_1, c_2>$) interpreted as that of $<l_1, l_2>$.

Suppose that we also have the models $M_T$, $M_B$ set up similarly as $M_H$, but measuring the top-difference ratio and the bottom-difference ratio respectively. Then what we desire from modeling the visual bigram information can be summarized in the form

$$\text{VBScore}(<c_1, c_2>) = k_H \cdot M_H(<c_1, c_2>) + k_T \cdot M_T(<c_1, c_2>) + k_B \cdot M_B(<c_1, c_2>)$$

where $k_H$, $k_T$ and $k_B$ are coefficients or weights to the corresponding models.

### 7.3.2 Training the VBM

In computation of the model score $M(<c_1, c_2>)$ for a model M that is one of either $M_H$, $M_T$, or $M_B$, the comparison of the related difference ratio of $<c_1, c_2>$ and the model's parameter is actually a look-up into a table holding the distribution histogram of the ratio. Training the VBM parameters is essentially constructing the model distributions from visual bigram samples. For example, let us consider the training of height difference ratio (HDR) model for the bigram class <A, A>. We collect the set of the HDR values for <A, A> class samples. Let $S = <s_1, s_2, \ldots, s_k>$ be the sorted list of the HDR values. Then the interval $[s_1, s_k]$ is divided into $N$ equal length sub-intervals. A bin is set up for each such sub-interval, to count the number of $s_i$ values that fall inside the sub-interval. Therefore, after finishing counting, the sequence of the bins forms the histogram of the distribution of HDR values. The histogram is then Gaussian-smoothed. After training, the quantity in

a histogram slot represents a likelihood of the HDR values that fall within the slot. The training of the rest of the models $M_T$ and $M_B$ proceeds similarly. The process is iterated on each of the 9 bigram classes and the system will wind up with 27 histograms in total. With the histogram set up for a model M, the steps of computing the model score on input $<c_1, c_2>$ are as follows: (Let $HIST_M[*]$ be the histogram of M)

1. Compute the model's difference ratio $R$ from $<c_1, c_2>$.
2. Compute the histogram slot index $k$ from $R$: $k = \lfloor (R - \min) / \text{slotsize} \rfloor$, where min is the lower-bound of the first histogram slot interval and slot-size is the interval length of the histogram slots.
3. Return $HIST_M[k]$.

### 7.3.3 Computing the Class Model Coefficients

Each model of a specific bigram class, has its coefficient that is used in the computation of the target confidence level of an input as shown at the end of CHAPTER 7.3.1:

$$\text{VBScore}(<c_1, c_2>) = k_H \cdot M_H(<c_1, c_2>) + k_T \cdot M_T(<c_1, c_2>) + k_B \cdot M_B(<c_1, c_2>).$$

Intuitively, the coefficient of a model measures the amount of contribution that the information coming form the model gives, in determining the fitness of the input with the suggested class interpretation. In our approach for determining the

coefficients we compute the amount of discrepancy of the distribution compared with the same kind of models of different classes. If the discrepancy of a model is larger than those of other models of the same class, then it means that the model is worth larger coefficient because it provides more information in measuring the fitness of the input. Given two bigram classes $T_1$, $T_2$ and a model M, let D(M, $T_1$, $T_2$) be the distribution discrepancy of $T_1$'s M from $T_2$'s M. The computing steps of D(M, $T_1$, $T_2$) is

1. Align M's scales of distribution of $T_1$ and $T_2$ by extending the histograms $HIST_{T1}$ and $HIST_{T2}$ to $HIST'_{T1}$ and $HIST'_{T2}$, so that the latter two have the same range of intervals.
2. Compute the discrepancy DSUM as follows by iterating over *i*

   2.1. $$DSUM := \begin{cases} HIST'_{T_2}[i] - HIST'_{T_1}[i] & \text{if } HIST'_{T_1}[i] < HIST'_{T_2}[i] \\ HIST'_{T_1}[i] - HIST'_{T_2}[i] & \text{else} \end{cases}$$

3. Return DSUM.

Step 1 is necessary since $T_1$ and $T_2$ have, in most cases, different distributions corresponding to different real value intervals. For a given class T, let $D_M(T)$ be the sum of D(M, T, *)'s over all other classes. This process repeats on other models of T. Let $D_H(T)$, $D_T(T)$ and $D_B(T)$ be the three discrepancy quantities for the T's models $M_H$, $M_T$ and $M_B$ respectively. Then the coefficients for the class T is computed as

- $k_H = D_H(T) / (D_H(T) + D_T(T) + D_B(T))$
- $k_T = D_T(T) / (D_H(T) + D_T(T) + D_B(T))$
- $k_B = D_B(T) / (D_H(T) + D_T(T) + D_B(T))$.

The three coefficients are computed for each of the 9 bigram classes, so the total of 27 coefficients are computed for each model of each bigram class, using the above procedure.

### 7.3.4 Experimental Result

To see the effectiveness of VBM on the word recognition performance, a test using the fusion representation and the HPN search was run with the VBM obtained by training on 240 visual bigram samples. Table 10 compares the test result with that of the test without the modeling. This time, the VBM filtering was applied after the HPN finishes its search. That is, for each the final candidate word of the system, the word' s VBM score was multiplied by the standard word score to get a new score. The candidates were re-sorted according to the new score into the new list. Therefore the new list is a re-ordering of the standard output list of the system, with the items reshuffled in reflection of the VBM fitness information. As seen in the table, the recognition rate was increased from 88% to 93%, an improvement of 5%.

| Using VBM | Yes | No |
|---|---|---|
| Recognition rate | 93% | 88% |
| Rank distribution of mis/non-recognized words | | |
| rank-1 | 4 | 8 |
| rank-2 | 1 | 1 |
| rank-3 | 1 | 2 |
| rank-4 | 1 | 1 |
| rank-5 | 0 | 0 |
| rank-6 | 0 | 0 |
| out of rank | 0 | 0 |
| Average rank | 1.85 | 1.67 |

**Table 10.** Comparison of word recognition performances with and without the VBM. Both tests were done using the fusion representation and the HPN search.

# CONCLUSION

Research in on-line handwriting recognition started in the early sixties, as the first generation of tablet digitizers became available. Since then, the limitations in hardware like less than accurate and unreliable digitizers and the amount of computing required to meet the need of demanding recognition task, along with premature state of understanding and technical advancement, have not allowed until recently a practical solution of on-line handwriting recognition with unconstrained style and vocabulary. The research focus in this thesis has been to develop a solution to the most fundamental problems in natural handwriting recognition: an on-line cursive script recognizer that has arbitrarily scalable vocabulary. In doing so, we have addressed and contributed the following points:

- Improved character recognizer using FDA and multiple experts fusion paradigm, having the desirable selective response pattern and better accuracy.
- New and more robust measure of word hypothesis evaluation that takes the mean of each component character scores, instead of the conventional non-decreasing accumulative scores.

- A general recognition engine (HPN) having multiple predecessors thereby absorbing the potential loss of optimality in the standard Viterbi search.
- Design and computation of the compact FLC representation and its use for efficient ligature modeling.
- Design and training of the efficient and effective geometric context modeling in the form of VBM.

Our approach of integrating the local features into a larger context perspective, more specifically at the character level, facilitates more robustness by avoiding the sensitivity to local fluctuation of feature variability. This has been possible since the features are not evaluated incrementally according to the temporal order, but are rather evaluated at once at the character unit. More generally our approach allows rearrangement of the order of the features inside a character segment in a standard way. This would be hard to do in a more sequence-bound approach for example a baseline HMM. The potential advantage is that such rearrangement of input can be used for writing-order-independent recognition, so that it would become more shape-based and would work more like humans are able to do. Furthermore, the same methodology could be adapted to off-line recognition task since the system would not have needs for ordering information other than for segmentation purpose. Our overall modular design also allows incremental

evolution of the system since each functional modules can be studied and improved in separation from the whole system and re-integration of the re-worked module can be done with little modification to the rest of the system.

# REFERENCES


[1] Fevzi Alimoglu and Ethem Alpaydin, "Combining Multiple Representations and Classifiers for Pen-based Handwritten Digit Recognition," *Proc. of the 4th International Conference Document Analysis and Recognition* (ICDAR ' 97), 1997.

[2] H. S. M. Beigi, K. Nathan, G. J. Clary and J. Subrahmonia, "Size Normalization in On-Line Unconstrained Handwriting Recognition, " in *Proc. IEEE Int' l Conf. Acoustics, Speech and Signal Processing*, pp. 169-172, Adelaide, Australia, Apr. 1994.

[3] Peter N. Belhumeur, J. P. Hespanha, and David J. Kriegman, "Eigenfaces vs. Fisher Faces: Recognition Using Class Specific Linear Projection," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 19, No. 7, 1997.

[4] Richard Bellman, "On a Routing Problem," Quarterly of Applied Mathematics, vol. 16, no. 1, pp. 87-90, 1958.

[5] Y. Bengio and Y. LeCun, "Word Normalization for On-Line Handwritten Word Recognition," in *Proc. 12th Int' l. Conf. Pattern Recognition*, vol. 2, pp.409-413, Jerusalem, Oct. 1994.

[6] Y. Bengio and Y. LeCun, C. Nohl and C. Burges, "Lerec: A Neural Network/HMM Hybrid for On-Line Handwriting Recognition," *Neural Computation*, vol. 7, no. 5, 1995.

[7] S. Bercu and G. Lorette, "On-Line Handwritten Word Recognition: An Approach Based on Hidden Markov Models," *Proc. Int'l Workshop on Frontiers of Handwriting Recognition*, '' pp. 385-390, Buffalo, N. Y., 1993.

[8] C. M. Bishop, *Neural Networks for Pattern Recognition*, Clarendon Press, 1995.

[9] G. Boccignone, A. Chianese, L. T. Cordelia and A. Marcelli, "Recovering Dynamic Information for Static Handwriting," *Pattern Recognition*, vol. 26, pp. 409-419, 1993.

[10] R. Bozinovic and S. N. Srihari, "Off-Line Cursive Script Recognition," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 11, no. 1, pp. 68-83, 1989.

[11] M. K. Brown and S. Ganapathy, "Cursive Script Recognition," in *Proc. Int' l. Conf. Cybernetics and Society*, 1980.

[12] D. J. Burr, "Designing a Handwriting Reader," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 5, pp. 554-559, 1983.

[13] M. Y. Chen, A. Kundu and J. Zhou, "Off-Line Handwritten Word Recognition Using a Hidden Markov Model Type Stochastic Network," ," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 16, no. 5, pp. 481-496, 1994.

[14] D. S. Doermann and A. Rosenfeld, "Recovery of Temporal Information form Static Images of Handwriting," *Int' l J. Computer Vision*, vol. 15, pp. 143-164, 1995.

[15] *Pattern Classification and Scene Analysis*, Richard O. Duda, Peter E. Hart, New York, Wiley, 1993.

[16] L. Duneau and B. Dorizzi, "On-Line Cursive Script Recognition: A User-Adaptive System for Word Recognition," *Pattern Recognition*, vol. 29, no. 12, pp.1981-1994, Dec. 1996.

[17] C. E. Dunn and P. S. P. Wang, "Character Segmentation Techniques for Handwritten Test - A Survey," *Proc. 11th Int' l. Conf. Pattern Recognition*, vol. 2, pp. 577-580, The Hague, Netherlands, 1992.

[18] Restor Ford Jr., D. R. Fulkerson, "Flows and Networks," Princeton University Press, 1962.

[19] L. S. Frishkopf and L. D. Harmon, "Machine Reading of Cursive Script," in C. Cherry, Ed., *Information Theory (4th London Symp.)*, London, England: Butterworths, pp. 300-316, 1961.

[20] K. Fukushima and S. Miyake, "Neocognitron: A New Algorithm for Pattern Recognition Tolerant of Deformations and Shifts in Position," *Pattern Recognition*, vol. 15, pp. 455-469, 1982.

[21] P. D. Gader, M. Mohammed and J. H. Chiang, "Handwritten Word Recognition with Character and Inter Character Neural Networks," *IEEE Trans. System, Man and Cybernetics*, vol. 27, no. 1, pp. 158-164, 1997.

[22] G. Gaillat, "An On-Line Recognizer with Learning Capabilities," *in Proc. 2nd Int' l. Joint Conf. Pattern Recognition*, pp. 305-306, Aug. 1974.

[23] I. Guyon, P. Albrecht, Y. LeCun, J. S. Denker and W. Hubbard, "Design of a Neural Network Character Recognizer for a Touch Terminal," *Pattern Recognition*, vol. 24, no. 2, pp. 105-119, 1991.

[24] I. Guyon, I. Poujaud, L. Personnaz, G. Dreyfus, J. Denker, Y. LeCun, "Comparing Different Neural Net Architectures for Classifying Handwritten Digits," in *Proc. Int' l. Joint Conf. Neural Networks*, Washington DC, vol. 2, pp. 127-132, 1989.

[25] S. Hanaki, T. Temma and H. Yoshida, "An On-Line Recognition of Handprinted Kanji Characters," *Pattern Recognition*, vol. 12, pp. 421-429, 1908

[26] C. A. Higgins and D. M. Ford, "On-Line Recognition of Connected Handwriting by Segmentation and Template Matching," *11th Int' l. Conf. Pattern Recognition*, vol. 2, p. 200, Aug. 1992.

[27] J. Hu, M. K. Brown and W. Turin, "HMM Based On-Line Handwriting Recognition," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 18, no. 10, pp. 1039-1044, Oct. 1996.

[28] Y. S. Huang and C. Y. Suen, "Combination of Multiple Experts for the Recognition of Unconstrained Handwritten Numerals," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. 17, No. 1, Jan. 1995.

[29] D. H. Hubel and T. N. Wiesel, "Receptive Fields, Binocular Interaction and Functional Architecture in the Cat' s Visual Cortex," *J. Physiology*, London, vol. 160, pp. 106-154, 1962.

[30] J. J. Hull and S. N. Srihari, "A Computational Approach to Visual Word Recognition: Hypothesis Generation and Testing," *Int' l. Conf. Computer Vision and Pattern Recognition*, pp 156-161, Jun. 1986.

[31] *Advances in Handwriting and Drawing: A Multidisciplinary Approach*. C. Faure, P. J. G. Keuss, G. Lorette, and A. Vinter, eds., Paris; Europia, 1994.

[32] I. Karls, G. Maderlechner, V. Pflug, S. Baumann, A. Weigel and A. Dengel, "Segmentation and Recognition of Cursive Handwriting With Improved Structured Lexica," *Proc. Int' l Workshop on Frontiers in Handwriting Recognition*, pp. 437-442, Buffalo, N. Y., 1993.

[33] D. D. Kerrick and A. C. Bovik, "Microprocessor-Based Recognition of Handprinted Characters from a Tablet Input," *Pattern Recognition*, vol. 21, pp.525-537, 1988.

[34] T. Kohonen, "The Self-Organizing maps," *Proc. IEEE*, vol. 78, pp. 1464-1480, 1990.

[35] A. Kosmala, J. Rottland and C. Rigoll, "Improved On-Line Handwriting Recognition Using Context Dependent Hidden Markov Models," *4th Int' l. Conf. Document Analysis and Recognition*, pp.641-644, 1995

[36] *A*. Kundu, Yang He and P. Bahl, "Recognition of Handwritten Words: First and Second Order Hidden Markov Model Based Approach, " *Pattern Recognition*, vol. 22, no. 3, p. 283, 1989.

[37] L. Lam, S. W. Lee and C. Y. Suen, "Thinning methodologies: A Comprehensive Survey," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 14, pp.869-885, 1992.

[38] Y. LeCun, "Generalization and Network Design Strategies," in R. Pfeifer, Z. Schreter, F. Fogelman and L. Steels, eds., *Connectionism in Perspective*, Zurich: Switzerland, Elsevier.

[39] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-Based Learning Applied to Document Recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278-2324, Nov. 1998.

[40] M. Leroux, J-C. Salome and J. Badard, "Recognition of Cursive Script Words in A Small Lexicon," in *Proc. Int' l. Conf. Document Analysis and Recognition*, pp. 774, Saint Malo, France, Sep. 1991.

[41] C. Y. Liou and H. C. Yang, "Handprinted Character Recognition Based on Spatial Topology Distance Measurement," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 18, no. 9, pp. 941-944, Sep. 1996.

[42] Y. Liu and S. N. Srihari, "Document Image Binarization Based on Texture Features," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 19, no. 5, pp. 1-5, May 1997.

[43] W. W. Loy and I. D. Landau, "An On-Line Procedure for Recognition of Handprinted Alphanumeric Characters," *IEEE Trans. Pattern Recognition and Machine Intelligence*, vol. 4, pp. 422-427, Jul. 1982.

[44] F. J. Maarse and A. J. W. M. Thomassen, "Produced and Perceived Writing Slant: Difference Between Up and Down Strokes," *Acta Psychologica*, vol. 54, pp.131-147, 1983.

[45] J. Makhoul, T. Starner, R. Schartz and G. Lou, "On-Line Cursive Handwriting Recognition Using Speech Recognition Models,", *Proc. IEEE Int' l Conf. Acoustics, Speech and Signal Processing*, vol. 5, pp. 125-128, Adelaide, Australia, 1994.

[46] S. Manke, M. Finke and A. Waibel, "Combining Bitmaps with Dynamic Writing Information for On-Line Handwriting Recognition," *in Proc. 12th Int' l. Conf. Pattern Recognition*, Jerusalem, Oct. 1994.

[47] M. Mohammed and P. Gader, "Handwritten Word Recognition Using Segmentation-Free Hidden Markov Modeling and Segmentation-Based Dynamic Programming Techniques," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 18, no. 5, pp. 548-554, May 1996.

[48] K. S. Nathan, H. S. M. Beigi, J. Subrahmonia, G. J. Clary and H. Maruyama, "Realtime On-Line Unconstrained Handwriting Recognition Using Statistical Methods," *Proc. IEEE Int' l Conf. Acoustics, Speech and Signal Processing* pp. 2619-2622, Detroit, 1995.

[49] H. Nishida, "An Approach to Integration of Off-Line and On-Line Recognition of Handwriting," *Pattern Recognition Letters*, vol. 16, no. 11, pp. 1213-219, Nov. 1995.

[50] F. Nouboud and R. Plamondon, "On-Line Recognition of Handprinted Characters: Survey and Beta Tests," *Pattern Recognition*, vol. 25, no. 9, pp. 1031-1044, 1990.

[51] Jong Oh and Davi Geiger, "An On-Line Handwriting Recognition System Using Fisher Segmental Matching and Hypotheses Propagation Network," in *Proc. Int' l. Conf. Computer Vision and Pattern Recognition ' 00*, vol. 2, pp. 343-348, Hilton Head, South Carolina, Jun. 2000.

[52] N. Otsu, "A Threshold Selection Method from Gray-Scale Histogram," *IEEE trans. Systems, Men and Cybernetics*, vol. 8, pp. 62-66, 1978.

[53] R. Otto, "Construction of Quadratic Polynomial Classifiers," *in Proc. Int' l. Conf. Pattern Recognition ' 76*, pp. 161-165, 1976.

[54] T. Paquet and Y. Lecourtier, "Handwriting Recognition: Application on Bank Cheques," in *Proc. Int' l. Conf. Document Analysis and Recognition*, p. 749, Saint Malo, France, Sep. 1991.

[55] M. Parizeau and R. Plamondon, "Allograph Adjacency Constraints for Cursive Script Recognition," *3rd Int' l. Workshop on Frontiers in Handwriting Recognition*, pp. 252-261, 1993.

[56] M. Parizeau and R. Plamondon, "A Fuzzy Syntactical Approach to Allograph Modeling for Cursive Script Recognition," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vo. 17. No. 7, ppl. 702-712, Jul. 1995.

[57] *Pattern Recognition,* special issue on handwriting processing and recognition, R. Plamondon, ed., 1993.

[58] *Computer Processing of Handwriting*. R. Plamondon and G. Leedham, eds., Singapore: World Scientific, 1990.

[59] R. Plamondon and G. Lorette, "Automatic Signature Verification and Writer Identification - The State of the Art," *Pattern Recognition*, vol. 22, no. 2, pp. 107-131, 1989.

[60] R. Plamondon and C. M. Privitera, "The Segmentation of Cursive Handwriting: An Approach Based on Off-Line Recovery of the Motor-Temporal Information," *IEEE Trans. Image Processing*, vol. 8, no. 1, pp. 80-91, Jan. 1999.

[61] R. Plamondon, C. Y. Suen, M. Bourdeau and C. Barriere, "Methodologies for Evaluating Thinning Algorithms for Character Recognition," *Int' l J. Pattern Recognition and Artificial intelligence*, special issue on thinning algorithms, vol. 7, no. 5, pp.1247-1270, 1993.

[62] I. R. Rabiner, "Tutorial on Hidden Markov Model and Selected Applications in Speech Recognition," *Proc. IEEE*, vol. 77, no.2, pp. 257-285, 1989.

[63] A. F. R. Rahman and M. C. Fairhurst, "Introducing New Multiple Expert Decision Combination Topologies: A Case Study using Recognition of Handwritten Characters," *Proc. of the 4th International Conference Document Analysis and Recognition* (ICDAR ' 97), 1997.

[64] *Pattern Recognition and Neural Networks*, B. D. Ripley, Cambridge, New York; Cambridge University Press, 1996.

[65] P. K. Sahoo, S. Soltani, A. K. C. Wong and Y. C. Chen, "A Survey of Thresholding Techniques," *Computer Vision, Graphics and Image Processing*, vol.. 41, pp.233-260, 1988.

[66] M. Schenkel, I. Guyon and D. Henderson, "On-Line Cursive Script Recognition Using Time-Delay Neural Networks and Hidden Markov Models," *Machine Vision and Applications*, vol. 8, no. 4, pp. 215-223, 1995.

[67] G. Seni, R. K. Srihari, and N. Nasrabadi, "Large Vocabulary Recognition of On-Line Handwritten Cursive Words," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 18, no. 7, Jul. 1996.

[68] *Forensic, Developmental and Neuropsychological Aspects of Handwriting*, special issue, J. Forensic Document Examination, M. Simner, W. Hulstijn, and P. Girouard, eds., 1994.

[69] *Handwriting and Drawing Research: Basic and Applied Issues*. M. L. Simner, C. G. Leedham and A. J. W. M. Thomassen, eds., Amsterdam: IOS Press, 1996.

[70] B. K. Sin, J. Y. Ha, S. C. Oh and J. H. Kim, "Network-Based Approach to On-Line Cursive Script Recognition," *IEEE Trans. Systems, Man and Cybernetics*, vol. 29, no. 2, pp. 321-328, Apr. 1999.

[71] B. K. Sin and J. H. Kim, "Ligature Modeling for On-Line Cursive Script Recognition," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 19, no. 6, Jun. 1997.

[72] C. Sung-Bae and J. H. Kim, "Combining Multiple Neural Networks by Fuzzy Integral for Robust Classification," *IEEE Trans. Systems, Man and Cybernetics*, vol. 25, no. 2, pp. 380-384, 1995

[73] C. C. Tappert, "Cursive Script Recognition by Elastic Matching*," IBM J. Res. Development*, vol. 26, pp. 765-771, 1982.

[74] M. Turk, A. Pentland, "Face Recognition Using Eigenfaces," *in Proc. Int' l. Conf. Computer Vision and Pattern Recognition*, pp. 586-591, 1991.

[75] "Neuromotor Control in Handwriting and Drawing," *Acta Psychologica*, G. P. Van Galen and P. Morasso, eds., vol. 100, nos. 1-2, p. 236, 1998.

[76] Handwriting: Issues of Psychomotor Control and Cognitive Models," *Acta Pychologica*, G. P. Van Galen and G. E. Stelmach, eds., special volume, Amsterdam: North Holland, 1993.

[77] *Development of Graphic Skills: Research, Perspectives and Educational Implications*. J. Wan, A. M. Wing and N. Sovik, eds., London: Academic Press, 1991.

[78] *S*. Watanabe, "Karhunen-Loeve Expansion and Factor Analysis: Theoretical Remarks and Applications," in *Proc. 4th Prague Conf. on Information Theory*, 1965.

[79] I. S. Yaeger, B. J. Webb and R. F. Lyon, "Combining Neural Networks and Context Driven Search for On-Line Printed Handwriting Recognition in the NEWTON," *AI Magazine*, vol. 19, no. 1, pp. 73-89, 1998.