Off-line Cursive Handwriting Recognition using Recurrent Neural Networks

Andrew William Senior

Trinity Hall,
Cambridge,
England.

September 1994

This thesis is submitted for consideration for the degree of Doctor of Philosophy at the University of Cambridge.

Summary

Computer handwriting recognition offers a new way of improving the human-computer interface and of enabling computers to read and process the many handwritten documents that must currently be processed manually. This thesis describes the design of a system that can transcribe handwritten documents.

First, a review of the aims and applications of computer handwriting recognition is presented, followed by a description of relevant psychological research. Previous researchers' approaches to the problems of off-line handwriting recognition are then described. A complete system for automatic, off-line recognition of handwriting is then detailed, which takes word images scanned from a handwritten page and produces word-level output. Methods for the normalization and representation of handwritten words are described, including a novel technique for detecting stroke-like features. Three probability estimation techniques are described, and their application to handwriting recognition investigated. The method of combining the probability estimates to choose the most likely word is described, and performance improvements are made by modelling the lengths of letters and the frequency of words in the corpus. The system is tested on a database of transcripts from a corpus of modern English and recognition results are shown. Recognition is described both with the search constrained to a fixed vocabulary and with an unlimited vocabulary.

The final chapter summarizes the system and highlights the advances made before assessing where future work is most likely to bring about improvements.

Key words

Off-line cursive script, handwriting recognition, OCR, recurrent neural networks, forward-backward algorithm, hidden Markov models, duration modelling.

Declaration

This thesis describes research carried out at Cambridge University Engineering Department between October 1991 and September 1994. It is the result of my own work and contains no work done in collaboration. The length of this thesis, including references and figure captions, is thirty-seven thousand words.

Acknowledgements

First of all, I would like to express my gratitude to the late Professor Frank Fallside, for supervising me during the first half of this thesis and for providing the original inspiration for this work. I am also indebted to Dr Tony Robinson who has supervised me admirably for the latter half of this thesis with enthusiastic guidance and support, particularly in the last few weeks.

I would like to thank everyone else in the Speech, Vision and Robotics group which Frank Fallside created. The group has been an ideal environment, both socially and technically, in which to conduct research. Those in the group who have helped in the creation of this thesis are too numerous to mention individually.

Special thanks must go to Chen Tham and Andy Piper for friendship; to Fuzzy who proof-read at such short notice, and particularly to Tim Jervis whose friendship has been invaluable in the last three years.

The former Science and Engineering Research Council is to be thanked for providing the financial support necessary for me to carry out this work. I would also like to thank everyone I met at Lexicus, IBM Hawthorne and AT&T Holmdel for recent fruitful discussions that have helped shape the writing of this thesis.

Finally, I would like to dedicate this thesis to my mother and to the memory of my father. My parents have always supported me and to them I owe everything .

Contents

Chapter 1

Introduction

By this art you may contemplate the variation of the 23 letters.
Robert Burton. The Anatomy of Melancholy.

The world is filling with computers. Whether we like it or not, they are becoming ubiquitous. As ever more people are forced into contact with computers and our dependence upon them continues to increase, it is essential that they become easier to use. As more of the world's information processing is done electronically, it becomes more important to make the transfer of information between people and machines simple and reliable.

One of the aspirations of the field of artificial intelligence, if one ignores for the time being the longer-term goals of analysing and emulating human intelligence, is simply to enable computers to accomplish tasks which are natural to people. Thus computers should be better able to interact with people and to act in human society in a less constrained manner than has previously been possible. These aims are reflected in the more modest attempt by the computer industry to make computers increasingly 'user friendly'. In this vein, computers have come out of laboratories and into homes and offices; we communicate with them using mice and keyboards rather than punched cards and toggle switches. Handwriting is a natural means of communication which nearly everyone learns at an early age.[1] Thus it would provide an easy way of interacting with a computer, requiring no special training to use effectively. A computer able to read handwriting would be able to process a host of data which at the moment is not accessible to computer manipulation.

After this argument, it seems surprising how little research there has been into the computer recognition of handwriting. One reason advanced is that the optimism about the capabilities of imminent speech recognition machines made people feel that other approaches were unnecessary. While some of the promises of speech recognition by machine have already been fulfilled, and researchers are still optimistic, some of the benefits have been slow to materialize and people have thought again about what is required of human-computer interfaces. Though speech is a very convenient form of commu-

[1] Downing and Leong (1982:p.299) quote an estimated world literacy rate of 71%. In those people coming into contact with computers, the figure must be higher.

nication, it is not always the most practical. In noisy environments, those where silence is important, or where a large number of people must work with computers, it is clear that voice input is not the best solution. Though computer professionals and secretaries would be loth to give up the convenience and speed of a keyboard, for those not familiar with keyboards, and for portable or occasional use, handwriting entry is clearly of practical value. This has lead to the growth in the last year or two of 'pen computing' — the use of computers which allow input from an electronic stylus (Geake 1992).

In addition to a potential mode of direct communication with computers, handwriting recognition is essential to automate the processing of a myriad of handwritten documents already in circulation. From cheques and letters to tax returns and market research surveys, handwriting recognition has a huge potential to improve efficiency and to obviate tedious transcription. As the Economist recently suggested, "Today's biggest prize in computer vision, however is text and handwriting...." (Browning 1992).

1.1 This thesis

This thesis investigates the use of handwriting recognition as a medium of communication between people and computers. After presenting a general overview of handwriting recognition, it focuses on the problem of reading handwritten documents. Later chapters present research carried out to develop a computer system which tackles this problem. The system has been described in earlier papers (Senior and Fallside 1993a; Senior 1993).

The thesis is divided into 9 chapters. This chapter describes the aim and contents of the thesis. The next chapter summarizes the aims and achievements of other work in the field of handwriting recognition and establishes a taxonomy of the field into which the original work of this thesis can be fitted. Applications for handwriting recognition are also examined. Chapter 3 studies work in the psychology of reading, to discover knowledge which can be put to use in the design of a machine handwriting recognition system.

Chapter 4 presents an overview of the handwriting recognition system that has been designed, and the following chapters describe the workings of individual parts of that system, including normalization and representation (Senior 1994); feature-finding (Senior and Fallside 1993b); probability estimation and language modelling. Each of these chapters includes details of experiments carried out to assess the performance of the techniques presented and a discussion of their validity.

The final chapter draws together the conclusions of the chapters about the handwriting system and summarizes what has been achieved in this programme of research. Further work which could be carried on from this thesis is also suggested.

1.2 Original contribution

This thesis describes a new, complete off-line handwriting recognition system. The major original contributions described in this thesis are as follows:

- The system applies a novel approach, using recurrent neural networks for probability estimation. While the recurrent neural network has previously been used for speech recognition, it has not before been applied to the recognition of handwriting.
- The training of a recurrent neural network with the forward-backward algorithm is described here for the first time.
- The psychology of reading literature is reviewed, showing how the study of human reading and writing gives an indication of the characteristics which might prove useful in a reading machine.
- The methods used here to normalize handwritten words are an original synthesis of new and established techniques. Previously published methods are compared and improved upon.
- Words are encoded in an original manner which is shown to be better than the common bit-map representation, and a novel method of feature detection, based upon the use of snakes is described.
- Chapter 8 investigates the use of duration modelling for off-line handwriting recognition and investigates the problems of out-of-vocabulary words with lexica of limited size.

1.3 Notation

Throughout this thesis, the distinction is made between a handwritten word, and the *idea* of that word. To make this distinction, the following typographical convention is employed. To represent a *handwritten* word or letter, the following font is used: '*abcdefghijklmnopqrstuvwxyz*'; and to denote the letters or words as *concepts* (McGraw *et al.* 1994), this font is used: '`abcdefghijklmnopqrstuvwxyz`'. The purpose of the system described here is to transcribe '*words*' into '`words`'. When the internal representation of the system is referred to (section 5.2), a single frame of data is shown thus: x_t; and the data representing a whole word are shown as x_0^T. The set of letters as concepts is denoted Λ and an arbitrary individual letter is shown Λ_i. The discrete probabilities used throughout are denoted P. These include the probability of one or several frames of data given that frame t is part of letter Λ_i — $P(\mathsf{x}_t|\Lambda_i)$ or $P(\mathsf{x}_0^t|\Lambda_i)$ respectively; the probability that frame t represents letter Λ_i given the data of the frame — $P(\Lambda_i|\mathsf{x}_t)$; and the probability of the jth element of a frame x_t, given that that frame represents letter Λ_i — $P((\mathsf{x}_t)_j|\Lambda_i)$.

Chapter 2

Handwriting recognition

> *... a vast population able to read but unable to distinguish what is worth reading.*
>
> G. M. Trevelyan. English Social History.

As computer power has increased over the years, and their range of applicability has similarly increased, one of the major goals of research into computers has been to make computers easier to communicate with and thus to make their benefits available to a much greater number of people. One of the major obstacles to the integration of computers as universal information processing systems is the fact that most useful business data is still stored on paper. Particularly when dealing with the general public, a huge amount of office paperwork is handwritten. Letters and faxes, as well as forms or annotations to printed documents, may be handwritten; in many situations it would be highly desirable to process the contents of these documents by machine, for which handwriting recognition is essential.

Similarly, computer user interfaces need to be improved to enable communication between computers and a wider class of users in a greater variety of circumstances. While ideas such as the mouse and touch-sensitive screens have been developed, and much work has been carried out into computer speech recognition, there is still much scope for making the interface more natural for users who are not familiar with computers. Handwriting ranks very highly as a way of communicating linguistic information in a way which is natural to very many people. Though speech recognition has been claimed as the panacea for user-interface problems, it has been slow to achieve its promise, particularly in noisy environments, and the limitations of speech recognition have become clearer as research has advanced.

In the last few years the field of handwriting recognition has become much more popular. Not only are more researchers trying to tackle the problems that it presents, but solutions to these problems are slowly becoming available and are actually being sold as useful products. Of late, pen computers have become available with handwriting recognition software for isolated characters and more recently for cursive script. Handwriting recognition systems have already started to be used for reading zip codes on envelopes and amounts on cheques.

Before describing a new handwriting recognition system in later chapters, it is worth presenting here the field of automatic handwriting recognition in its entirety. After describing a taxonomy of the field, applications envisaged for handwriting recognition systems are discussed and work by other authors is presented to demonstrate the approaches taken.

2.1 A taxonomy of handwriting recognition problems

Having established the need for automatic handwriting recognition in general, it is useful to examine the field more closely and to identify several areas with different applications and requiring different approaches. Though many techniques can be shared, the literature tends to divide into groups of researchers, each concentrating on a special area of handwriting recognition.

2.1.1 On-line versus off-line

The major division is between *on-line* and *off-line* systems. While other methods could be distinguished, handwriting recognition systems are generally polarized between those receiving their data directly from some sort of pen device attached to the computer, and those which recognize handwriting already present on a piece of paper — a handwriting equivalent of Optical Character Recognition (OCR) which is already widely used for reading printed matter. In the literature *dynamic* is sometimes used to mean on-line and *static* off-line. So far, the majority of systems have tackled the easier, on-line, problem where the time ordering of strokes is available as well as pen up/down information; overlapping strokes can easily be distinguished and stroke positions are accurately known. On the other hand, off-line systems have to cope with the vagaries of different pen types, wide strokes which overlap and a lack of ordering information. The growth of pen computing has seen much investment in on-line systems, and the difficulty of off-line recognition has deterred research until recently.

Since the on-line data from an electronic stylus are a one-dimensional stream of information, techniques from speech recognition have been successfully applied to this problem, including Hidden Markov Models (Bellegarda *et al.* 1994) and time-delay neural networks (Schenkel *et al.* 1994). The data from the tablet are usually (x, y) coordinates sampled at a constant frequency in time, though they are often re-parametrized to be equally-spaced, and represented in terms of arc-length, curvature, and angle, with information about whether the pen is touching the tablet. A particular problem of on-line recognition is how to handle *delayed strokes* — strokes which are written after the rest of the word, as in dotting '`i`'s and crossing '`t`'s. Some authors choose to manage without this extra data; Schenkel *et al.* record its existence as a 'hat' feature associated with the strokes over which the

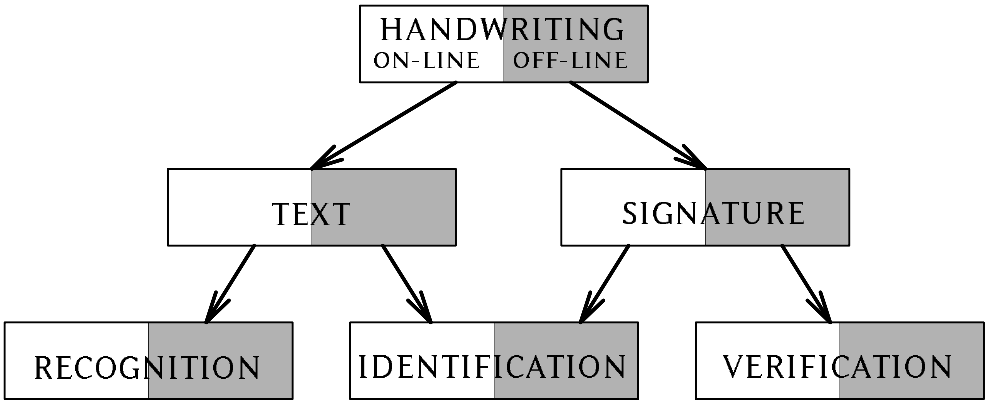


Figure 2.1: Subdivisions of machine handwriting recognition (after Plamondon and Lorette (1989)).

delayed strokes occur, and Bengio *et al.* (1994a) represent the surrounding visual context of all strokes so that the dot is seen above the cusp of the '*ı*'.

Although applications and techniques vary considerably, the general taxonomy of both off- and on-line handwriting analysis is similar; as is shown in figure 2.1 and described in the following section. While this thesis is concerned with off-line handwriting recognition, parallel work from on-line research is brought in throughout when there is a community of interests, such as in the modelling of handwriting production or in the application of probabilistic recognizers and grammatical constraints.

2.1.2 Author identification versus content determination

A second dichotomy in the field, orthogonal to the on-line/off-line division is according to the information to be extracted from the handwriting. From both on-line and off-line data, it may be necessary to determine the authorship of the writing, the content of what has been written, or both. In both cases, the effects of some variations should be ignored. To determine the authorship, differences in personal style should be highlighted, to capture what is characteristic about one person's writing (their *idioscript*). Conversely, to determine the content of the writing, the variations due to idioscript should be eliminated and ignored. These two requirements result in very different approaches. Techniques also differ depending on whether the author is to be recognized from a signature or from a piece of text.

If the author of a piece of text or signature must be determined, the distinction is made between verifying that the author is the claimed author (for instance in security or banking applications) or merely deciding between a pool of known authors, for instance in a writer-adaptive handwriting recognition system which uses different parameters for word recognition according to the author. The former is the more useful, but of course the harder, problem. Plamondon and Lorette (1989) give an overview of handwriting systems, and a thorough review of signature verification systems.

2.1.3 Writer independence

The whole field of handwriting recognition is similar to the already well-developed subject of automatic speech recognition, which is often classified along the lines of speaker dependence, vocabulary size and isolated word vs. continuous speech. Analogues to each of these exist in handwriting recognition, and are discussed in this and the following sections.

Handwriting styles are extremely diverse, depending both on the pattern used to teach handwriting to an individual and on the individual's idioscript (corresponding to spoken accents and idiolects). Because of this, it is more difficult to devise a system to recognize many peoples' handwriting than one which need only recognize the writing of a single author. Instead of creating a system which can recognize anybody's handwriting, the problem of multiple writers could be tackled by a system which is able to adapt to the current writer. Adaptation to the writer's style could be used when recognizing a lot of material by the same author, but would be of no use when identifying the city names on envelopes. Alternatively, many similar subsystems could be created, each recognizing one style of handwriting (or one individual's handwriting). Then a global system would select the subsystem which corresponded to a particular handwriting sample.

2.1.4 Vocabulary size

The task of recognizing words from a small lexicon is much easier than from a large lexicon (where words are more likely to be similar to each other). Thus, an important criterion in assessing system performance is the size of the lexicon used. The lexicon will depend on the application of the recognition system. For a general text transcription system, a lexicon of 60,000 words (the number of references in a medium-sized dictionary), would cover about 98% of occurrences, and for specific domains, such as reading cheque values in words, or postal towns from envelopes, the vocabulary can be much smaller. Alternatively, it may be necessary for the system to recognize non-words if the user is likely to write words not in the lexicon, such as abbreviations, foreign words or names. This issue is discussed again in section 8.4.

2.1.5 Isolated characters

Segmentation of continuous speech into its component words has been found to be very difficult since in natural speech words run together with no silence between. For simpler tasks the recognition is made easier by forcing the speaker to pause between words. Similarly, in *cursive* script it is hard to distinguish the boundaries between letters — the difference between '*ui*' and '*iu*' or between '*vv*' and '*w*' is very slight. The task can be simplified by forcing the writer to separate letters (*discrete* handwriting), to write in capitals or for the greatest clarity, to write clearly separated capitals in pre-printed boxes. When high reliability is required, the latter constraints may be unavoidable since they are already necessary to enable human readers to decipher responses on forms. A number of authors have investigated the problem of recognizing isolated characters (section 2.3.1), particularly for the problem of reading postal codes. Other authors have researched the recognition of discrete handwriting (*'hand print'* where lower-case letters are written but must be separate) or pure cursive script.

Similar constraints can be placed on cursive script, forcing the author to write each word in a separate box, or on a guide line. These constraints are mainly to encourage clarity since the word segmentation problem proves less difficult than segmentation into characters, and less strict constraints could still ensure high accuracy segmentation of a page into its component words. Other authors have described methods of segmenting pages into words and distinguishing between gaps in words and gaps between words (Srihari *et al.* 1993).

2.1.6 Optical character recognition

Off-line handwriting recognition has much in common with optical character recognition (OCR) — the reading of print by computer. This application received much attention during the 1980s and successful solutions have been found, with commercial packages available for microcomputers which can read type in a variety of fonts and in a certain amount of noise. The history and current status of OCR are reviewed by Mori *et al.* (1992) and Pavlidis (1993). In more difficult situations, these commercial packages are still not satisfactory. Authors describe problems working with unusual character sets and fonts, poor quality documents or documents in special formats (Bos and van der Moer 1993; McVeigh 1993). Indeed, it is not clear that OCR is economically viable in a great many cases when high accuracy is essential (Olsen 1993).

The reason why the success of OCR has not carried over into handwriting recognition is the great variability in handwriting. For type in a fixed font, all letters 'a' are produced from a single archetype, and thus are very similar on the page, only being corrupted by a relatively small amount of noise in forms such as blurring, merging and slight positional variations. The process of handwriting is much more variable in all of these processes and suffers from variations due to other effects such as co-articulation — the influence of one letter on another. Also, with type, the symbols are usually distinct (except certain ligatures, as 'fi', which can be learnt as a separate symbol) so the problem of segmentation is not present.

As a consequence of this the relatively simple techniques used in OCR, such as template matching, are inadequate when presented with the greater variability in handwriting so relatively little research in the OCR literature carries over to handwriting recognition.

2.2 Applications

This section reviews some of the more important applications that may be envisaged for off-line handwriting recognition. On-line recognition tends to be for data-entry to obviate a keyboard as in pen computers, but can also be used for special purposes such as using dynamic signatures to verify identity.

One potential application in the long term is in using off-line techniques for on-line handwriting recognition. Currently, off-line performance lags behind that of on-line recognition systems, but over the next few years, as the technology improves it is likely that methods for both types of handwriting recognition will converge, leading to more general systems and reduced development costs. This convergence can be seen in the model-based approaches now being used (Pettier and Camillerapp 1993; Doermann 1993), which interpret off-line handwriting as a path of ink laid down over time, rather than as an image to be analysed independently of its method of production. The data that can be derived by such algorithms is very similar to the data available to an on-line recognizer.

In the longer term though, it would seem that the convergence is likely to treat both off-line and on-line words as a two-dimensional image, and not as a one-dimensional stream of trajectory data. The reason for this can be seen by looking at the psychology of reading (chapter 3) — the way people read is by looking at an image, not by analysing the pen path used to produce the writing. Since this involves ignoring the time information, at first this seems to be a poor method of analysing on-line data. However, the information in handwriting is not transmitted in the timing of the pen trajectory. It does not matter whether the strokes of a word are written quickly or slowly, with changing speed, or even in random order, since it is the appearance of the finished word that matters. Thus, by discarding the time sequence, a source of mis-information is actually avoided. For instance, in current on-line systems, an '*o*' written clockwise must be recognized differently from an '*o*' written anticlockwise, for in the time sequence information, they appear different. Someone who writes '*lea*' may subsequently return to extend the final '*a*' stroke to make the word read '*led*', but this change would be lost on a machine relying on the time-ordering of strokes. An off-line approach ignores these factors and simply looks at the final position of the strokes, just as a human reader would. This approach also gives a satisfactory solution to the problem of delayed strokes (section 2.1.1). After these arguments, it may be seen that, while on-line recognition is better than off-line now, because the timing information generally is consistent, a good off-line approach might ultimately cope with a wider variety of variation. Conversely, the timing information is very useful when creating an author verification system —

on-line signatures are much harder to forge than off-line signatures, since the dynamics of strokes (with pen both up and down) are harder to forge than the finished appearance.

2.2.1 Cheques

One important commercial application for off-line cursive script is in the machine reading of bank cheques. While the amount in figures is easier to read, it should be checked that the amount in words is the same, and this can be used for confirmation where the numerical amount is unclear. Such a system would only need to have a small vocabulary (about thirty-five words). Given a system that achieved high accuracy without a lexicon, one could check that the payee corresponded to the account to be credited. Such a system might also include signature verification, bringing about an increase in security with the reduction in drudgery and time. Given the number of cheques passing through the banking system each day, a cheque reading system, even if only able to confidently verify half of the cheques, would save much labour on a tedious and unpleasant job. Cheques which could not be confidently verified by machine would still be processed manually, so accuracy would be maintained. The project supported by the French post office has the goal of achieving a 1 in 100,000 error rate from the combined recognition of literal and numerical amounts, but permitting 50% of cheques to be rejected for manual sorting (Leroux *et al.* 1991).

2.2.2 From postcodes to addresses

Off-line systems capable of recognizing isolated handwritten digits have already been created and installed in many post offices around the world, as part of automatic mail-sorting machines. Given a system to locate the postcode on an envelope (Wang and Srihari 1988; Martins and Allinson 1991; Palumbo *et al.* 1992) this can be read and used to direct mail automatically. Clearly certain countries such as the USA are at an advantage in having digit-only zip-codes and many researchers have already tackled this problem with reasonable success (section 2.3.1).

To process more mail automatically, systems must begin to use the information contained in the rest of the address. This allows the uncertainty in the postal code classification to be removed by comparing candidate zip codes with candidate addresses in a database of all address/zip code combinations, giving more high confidence classifications. Furthermore, for countries with limited resolution in the postcode, the address can be used to increase the resolution of sorting. U.S. postal service projects aim to use the address to determine an 11 digit *delivery point code* which specifies a single house even when only the five digit zip code was provided.

Mail sorting can be seen as an ideal application for writer-independent handwriting recognition, since it has a wide variety of levels of difficulty, from isolated digits written at predetermined locations on an envelope, up to complete determination of an address without a postcode. Address recognition also admits of a certain amount of error while allowing a large rejection rate. Since there will always be some addresses that are illegible or incomprehensible to a machine, a 'don't know' answer can be given and the item sent to a bin for human sorting. Further, some mail is already misrouted, so the postal service is considered fallible and the consequent delays are already tolerated.

2.2.3 Form processing

Another major application which is now receiving attention is the automatic processing of forms. Forms are widely used to collect data from the general public. For anything more than the most simple information, for which check boxes can be used, replies are handwritten in spaces provided. Much of this information must be stored in databases and can be processed automatically once entered into the computer. Data entry is currently the bottle-neck in the process. Several authors have written systems to segment the handwritten data from the pre-printed form and then to transcribe the handwritten data. In some applications, this may be isolated capital letters written in boxes, but work is now moving on to hand print (Breuel 1994; Garris *et al.* 1994). Although forms must usually be hand printed to keep the writing as legible as possible, for human as well as machine processing, cursive recognition would still be useful for processing those forms that have mistakenly been filled out in cursive script.

2.2.4 Other applications

A variety of other office document processing systems using off-line handwriting recognition can easily be envisaged. Already many companies use electronic document processing systems which manipulate the scanned images of documents rather than the documents themselves. This is clearly a very data-intensive task, but one way of reducing the data storage is to extract the information and store text in ASCII (or perhaps in a richer format recording the style of writing). Documents would then be easily searchable and index construction would be made possible. Further possibilities exist in reading handwritten documents for the blind or in automatic reading of faxes. Faxed orders could be processed and dispatched automatically and standard enquiries replied to without human intervention. Other faxes could be fed directly into an electronic mail system, providing at the very least automatic notification of fax arrival by reading the cover sheet, if not the full text of the document.

Of course, the advantages of handwriting recognition are not restricted to English or to the Roman alphabet, though these have probably attracted most research. In the literature there is a wide range of papers describing handwriting recognition in a multitude of languages. The basic problems of handwriting recognition are common to all languages, but the diversity of scripts means that very different approaches may be used. For example, Japanese Kanji (Mori and Yokosawa 1988) and Chinese (Lu *et al.* 1991) characters are strongly stroke-based, and characters are easy to segment from one another, but characters are very complex and there are many classes to distinguish. Arabic and roman alphabets can be cursive, and Arabic and some Hebrew require accurate recognition of diacritic marks. Govindan and Shivaprasad (1990) cite many more languages.

2.3 Existing off-line handwriting recognition systems

This section reviews some of the off-line handwriting systems which have been detailed in print. To do this it is convenient to classify them, as described above, into isolated character and cursive script systems. Here only a brief overview of these systems is given. Specific details are provided in later chapters when particular issues are discussed.

2.3.1 Isolated characters or digits

Suen *et al.* (1980) provide a good review of handwriting recognition up to 1980, concentrating on isolated character recognition — which had been the focus of research until then. They describe a variety of feature based approaches and divide these into global features (templates or transformations such as Fourier, Walsh or Hadamard); point distributions (zoning, moments, n-tuples, characteristic loci and crossings and distances) and geometrical or topological features. The latter were, and have remained, the most popular techniques, and involve separate detectors for each of several types of features such as loops, curves, straight sections, endpoints, angles and intersections. For instance, Impedovo *et al.* (1990) use cross-points, end-points and bend-points as their features, coding these as to their location in three horizontal and three vertical zones within each character. The encoded characters are then identified using a decision tree classifier. Elliman and Banks (1991) also use features (end-point, junction, curve and loop) each of which is associated with a numerical quantity, such as curvature or length, before being decoded in a neural network (a feed-forward neural network or an adaptive feedback classifier).

Nellis and Stonham (1991) and Hepp (1991) both use sets of global morphological features created by separately examining the left, right, top and bottom edges of each character. The profile of the character from each edge is coded as a separate feature for classification by a neural network.

Le Cun *et al.* (1989) and Fukushima (1980) take the approach of feeding a normalized bitmap image of the character to be recognized into their networks (multi-layered perceptron and neocognitron respectively). Both these networks are constructed from layers of identical feature detectors, which become more specialized and less location specific deeper in the network, until the outputs of the final layer correspond to characters, independent of location in the image.

A host of other authors have tackled the problem of recognizing isolated digits or characters in the last few years (Hepp 1991; Idan and Chevalier 1991; Impedovo *et al.* 1990; Lanitis *et al.* 1993), particularly since the increasing availability of data has made this a standard test problem for testing pattern recognition methods (Simard *et al.* 1993; Hinton *et al.* 1992; Boser 1994). Isolated digit classifiers have now become so good that research is concentrating on reading whole zip codes where the digits are often touching (Fontaine and Shastri 1992; Kimura and Shridhar 1991; Matan *et al.* 1992), and finding optimal combinations of multiple classifiers now seems a more promising way of reducing error rates than finding better classifiers. Huang and Suen (1993) cite several papers taking this approach. Performance is now being limited by the number of digits which are entirely ambiguous and could not be confidently classified by human readers.

2.3.2 Off-line cursive script

The problem of off-line cursive script recognition has received little attention until recently, partly because of the difficulty of the problem, but also because of the lack of data. Simon (1992) and Suen *et al.* (1993) give brief reviews of script recognizers, but the best review is probably by Lecolinet and Baret (1994). Simon makes the distinction between the *segmentation* approach and the *global* approach, according to whether words are identified by recognizing individual letters or by recognizing words as a whole. In fact, very few authors take the latter strategy. Plessis *et al.* (1993) use a holistic match, but only to reduce the size of their lexicon before using a more detailed recognition method. Lecolinet and Crettez (1991) use the terms *explicit segmentation* and *implicit segmentation* according to whether an attempt is made to divide the word into separate characters and recognize these individually, or if the segmentation is a by-product of a recognition process working on a different unit of writing. Both approaches use strong evidence from well-written parts of words, together with a restricted lexicon, to recognize words which are partially badly written.

All the authors described below incorporate some form of preprocessing to normalize and clean the data. Some preprocessing methods are described in chapter 5. In each case, a recognition strategy then hypothesizes character or word identities, and because exact recognition is very difficult, all the approaches use a lexicon to constrain the responses to a known vocabulary.

Perhaps the most successful off-line handwriting recognition system is that of Kimura *et al.* (1993b,1993a) who have created a system for reading city or state names in addresses. These authors take a dual approach, with a first, quick classification to reduce the lexicon size, followed by a more accurate second classification using different techniques. The first stage finds a rough explicit segmentation and each segment is classified as a letter. The second stage finds a different explicit segmentation by splitting the word into disjoint boxes and joining the boxes together using dynamic programming to form complete characters. These are then passed to a character classifier. These authors report results of 91.5% recognition with a lexicon of 1000 words on the CEDAR database of words segmented from addresses in the U.S. mail (Hull 1993).

Cheriet and Suen's (1993) approach is also letter-based. However, their approach is to extract a number of key letters from each cursive word — particularly the initial letter and those clearly identifiable by ascenders, descenders or loops. For a small vocabulary task (reading cheques) as described in their paper, identifying these key letters might be sufficient to identify most words, but the authors propose their techniques as a way of filtering, to reduce the number of words in the lexicon of possible matches.

Papers by Srihari and Božinović (1987; Božinović and Srihari 1989) take an explicit segmentation approach, but here each segment need not correspond to a character. They find *presegmentation points* which include all the boundaries between characters, but also split some characters into two or more pieces. They then find features (16 in all, including dots, curves, strokes, loops and cusps) within the segments by a series of event detectors and use the features to construct letter hypotheses according to statistics of feature occurrences gathered during training. Words are hypothesized via a stack method, where the most likely prefixes are stored and expanded until the word end is reached. After the first iteration of this procedure, the stack contains all the hypotheses for the first letter in order of likelihood. The top (most likely) hypothesis is then expanded by looking at what letters could follow. The resultant two-letter sequences are put onto the stack, to be expanded when they are the most likely sequences. At the end of the word, the lexically correct word that is highest on the stack is chosen as the best match.

Srihari and Božinović conducted a number of experiments, using different writers and different lexica (780 and 7800 words). Testing on a single-author database of horizontal, non-slanting writing, a 77% recognition rate was obtained on the small lexicon, 48% on the large. A second single-author database yielded a 71% recognition rate on the smaller lexicon.

Yanikoglu and Sandon (1993) take a similar approach. They find possible character segmentation points and attempt to classify segments or groups of up to three segments with a neural network classifier trained on isolated letters. Incorrect segmentations tend to get lower classification scores than when a letter is correctly segmented, and when the scores are combined in a hidden Markov model, the best hypothesis for the groupings of segments and their identities is found. Results of 70% for single-author cursive word recognition are quoted for a lexicon of 30,000 words.

Edelman *et al.* (1990) have developed a handwriting reader which relies on the alignment of letter prototypes. Here, anchor points (*e.g.* endpoints; turning points at the top, bottom, left or right of a character) are found in the test word and these points are used to match the word against a set of prototype curves, coded as splines, which can be composed into lower-case characters. The system is hand-designed and is not trained automatically. Using a 30,000 word lexicon, these authors obtained an 81% recognition rate on the training set and around 50% on test sets by three authors. The stress of this system is on recognition without a lexicon, however, and recognition rates of 8–22% are given for three authors including the author whose writing was used to develop the system.

The problem of reading the amount on cheques (section 2.3.2) has been tackled by a number of authors in the problem posed by the French post office. The task here is to recognize amounts written (in words) on postal cheques and to use these to verify the amounts written in figures. Moreau *et al.* (1991) identify a few characteristics of the cursive words and match these to a set of reference words with Dynamic Programming. The identified words are used together with a grammar to verify the amount in figures. With a 60% rejection rate, the error rate achieved is 0.2%. Paquet and Lecourtier (1991) reduce each word to a series of curves which they match to examples in a lexicon. They achieve 60% correct on the 50% of words which are well-segmented and later (Paquet and Lecourtier 1993) achieve an error rate of 59% when rejecting 9.5% of words. Leroux *et al.* (1991) take two parallel approaches — one is to recognize the word as a whole, by finding a few features and comparing with reference words. The second is a letter-by-letter approach where the desire is to recognize only some of the letters, and to use this information to restrict the lexicon. Their system correctly identifies 62% of words. The system described by Simon (1992) achieves a 0.15% error rate with a reject rate of 24% using a 25 word vocabulary.

Chapter 3

Psychology of reading

There is an art of reading as well as an art of thinking and an art of writing.

D'Israeli.

Before attempting the machine recognition of handwriting, it is worthwhile considering the way that people read and write. Considering human reading may lead to an increased understanding of the transfer of information through the medium of handwriting, so that it can be seen which processes play a useful role, and which are merely epiphenomena. If it can be understood what information people use to recognize handwritten words, then a clue is found as to what features might be useful for a machine recognition system. Other features are likely to be poorly preserved since they play no useful role. Understanding handwriting production may similarly give insights as to which features of handwriting are representations of the information and which mere artefacts of the generation process.

A large body of psychological data has been gathered on the processes involved in reading type, some of which is applicable to cursive script. Taylor and Taylor (1983), Downing and Leong (1982) and Rayner and Pollatsek (1989) give thorough reviews of the psychology of reading. Most research so far has concentrated on reading individual letters or words out of context. It could be argued that this gives little indication of the processes occurring in normal reading where many words are visible and it is the text as a whole, not individual words, that is important. However results are hard to prove in such a natural environment with many variables, and it is only under restricted experimental conditions that hypotheses can be rigorously tested.

Research into reading, as in much of psychology, relies heavily on observing what errors are made under difficult conditions. One technique is the use of tachistoscopes to flash a word in front of a subject for a very short time followed by a patterned mask to inhibit iconic memory, which otherwise allows the subject to preserve an image of the word mentally for an uncontrolled period of time.

3.1 Reading by features

As will be seen later, many approaches to handwriting recognition rely on detecting features in the writing, such as the strokes which go to make up individual letters. Hubel and Wiesel (1962) describe the processes early in the visual cortex. The complex cells that they discovered code the presence of bars and edges and provide a compact representation of lines which is particularly appropriate to the representation of writing and print. A number of authors have sought to determine what higher-level representation might be used specifically for letters.

Bouma (1971) investigated the features which people use to recognize isolated characters by examining the confusions between letters presented either at a distance or for a short time, eccentrically in the subject's field of view. Bouma uses the errors made by subjects to identify groups of confusable, or 'psychologically close', letters. Bouma's classification is shown in table 3.1.

Outer contour	Bouma shape	Code	Letters
Short	inner parts and rectangular envelope	1	a s z x
	round envelope	2	e o c
	oblique outer parts	3	r v w
	vertical outer parts	4	n m u
Tall	ascending extensions	5	d h k b
	slenderness	6	t i l f
Projecting	descender	7	g j p q y

Table 3.1: Bouma shapes.

Shape type	Number of words sharing the same shape									
	1	2	3	4	5	6	7	8	9	10+
Outer contour	1389	250	102	45	23	20	16	9	7	36
Outer contour +initial	2301	313	77	34	14	7	2	3		
Bouma shape	3201	83	20	3	1					
Bouma shape + initial	3340	49	2							

Table 3.2: Word discrimination using word shape measures on the text of this thesis.

Using these classes, words can be encoded according to their shape, so '*keg*' would become 527, but so also would '*boy*' which is seen to be similar in shape. Taylor and Taylor used these Bouma shapes for a study on the text of their own book. Table 3.2 shows a similar experiment on the text of this thesis. The words are classified according to each of four shape description techniques, and the number of words of each shape is counted. The outer contour is a coarser coding than the Bouma shape, simply classifying letters as short, tall or projecting. The outer contour is enough to specify 1389 of the 3444 words uniquely, but there are 36 shapes shared by ten or more words each. If the first letter is known, the ambiguity is further reduced. The Bouma shape, having more classes than outer contour, gives more unique shapes — 3201 words are uniquely labelled.

This study shows that, in conjunction with a lexicon of permitted words, a few simple features can identify most words, without the need to recognize the individual letters. Haber and Haber (1981) have carried out similar work into the effectiveness of letter shape for reading, and also give a decision tree which might be used to distinguish the letters of the Helvetica font by observing only a limited set of features. Eldridge *et al.* (1984) investigate the variability of some handwriting features comparing variation in an individual's handwriting with that between individuals.

McGraw *et al.* (1994) further investigate the features that might be used in representing characters. Although their experiments are conducted with machine-generated letters made up of straight line segments, they investigate the recognition of letters at the limits of class-boundaries, so their work is of relevance to handwriting recognition. They suggest that letter recognition is carried out by finding word features that fill roles in internal models of letters. Thus a letter 'b' could be described as a loop with a short stroke above and to the left, or as a tall stroke with a curved section joined at the lower right. These authors do not consider the possibility of overlapping features which might characterize the letter as well if not better. For instance, a 'b' could also be described as a tall stroke overlapping a loop to the right. They make the important point that the higher-level features used for reading are not likely to simply arise bottom-up from the visual processing system, as Hubel and Wiesel cells do, but to be defined top-down depending on the classes to be distinguished. This depends in turn on the writing system to be read, just as when learning a new language the boundaries between phonemes have to be re-learnt according to the different distinctions and groupings made in that language.

Many studies have also been made into the processes involved in writing. If an accurate model of these processes can be found, then it could be used for representation of handwriting in a compact form, and for recognition. Alimi and Plamondon (1993) discuss a variety of models for handwriting generation, and Abbink *et al.* (1993) and Singer and Tishby (1993) have used the Hollerbach (1981) model for modelling handwriting for recognition. Singer and Tishby derive a very compact code which represents the handwriting but also allows the easy removal of slant, slope and other variation, making the writing more legible. Teulings (1994) discusses feature extraction from on-line cursive script. As yet these approaches have usually been applied to on-line script where the pen trajectory is accurately known. The static nature of off-line writing does not lend itself to these approaches, though Doermann (1993) shows that off-line script can be considered in this way. However it seems that, while compact representations can be found using the model-based approach, reading is a visual process and dynamic approaches will always fail to represent data such as the dots on a letter '`i`' appropriately, for here it is important *where* the dot occurs, not when or how.

3.2 Reading by letters and reading by words

One of the fundamental findings of reading research is the importance of recognition of words as single entities and not as the conjunction of their component characters. Taylor and Taylor cite work by Kolers & Magee, whose experiments involved training subjects on inverted text (where the letters are all upside-down). They trained two groups — one to read words and one to name letters — then each group was switched to the other task. No evidence was found that learning one task improved performance in the other, thus one may conclude that "relatively fluent reading requires familiarity with the shapes of words, but not with the letters in those words."(p.195)

Further evidence for reading by words rather than individual letters is given by the *word superiority effect*. This is the term used for the phenomenon that a letter is better recognized (more frequently recognized correctly when presentation time is short enough to induce errors) when presented as part of a word than when presented either on its own or surrounded by arbitrary characters in a non-word (for instance in Reicher's experiments described by Rayner and Pollatsek p.77).

It is interesting to note the work by Yamadori (1975) and Sasanuma (1984) which shows that damage to certain areas of the brains of Japanese readers can severely impair reading of Kana (syllabic) script whereas Kanji (morphemic) script is much less affected. This shows that different brain pathways must be used for the two script types and indicates that the mechanism of reading is more complex than it might at first appear. Downing and Leong discuss the possibility of phonological, visual or both pathways for indexing an internal lexicon, and the evidence seems to suggest that people use both a coding of the sounds of words and a coding of the visual image when recognizing words while reading.

Taylor and Taylor propose a reading mechanism with three paths:

Whole-word process This is a rapid process taking perhaps 50-100ms which is based only on the pattern of the word as a whole, or the first half-dozen letters of longer words.

Letter-based process From 50ms after a word is presented, the individual letter identities are becoming available. (This could be understood as a progressive increase in the frequency of the filter used as suggested in work by Marr (1982)). Outer letters are identified first, and may be used to adjust the first hypothesis of the whole-word process, or to generate a new one. These authors also suggest that word units (prefixes and suffixes) may be recognized as single items.

Scan-parse process This process is the slowest and uses the letter identities to produce a phonetic version of the written word, which can be used as additional evidence for the word identity.

3.3 Lexicon and context

Reading relies on the use of a lexicon of words. Words that are written unclearly can often only be identified because it is known that they must represent a real word, rather than one of the other letter strings that might be 'read into' the cursive word. The word of figure 3.1a could be interpreted in many ways, but a reader would generally opt for 'minimum' because that is a word. Psychological studies have verified the existence of some form of internal lexicon, though the form that this takes is unclear. Nevertheless the *lexical decision task* is an important tool in experiments. For this the experimenter measures the time taken to determine whether a string of letters is a word or not.

(a) (b)

Figure 3.1: Word ambiguity. (a) is identified by recognizing the two 'i's and knowing that the word must be in the lexicon. (b) is still ambiguous unless context is supplied.

Context is also significant. The correct interpretation of the word 'minimum' is made even more likely in a passage about optimization. (But another might be understood in the context of a discussion of non-words in the psychology of reading.) Context is also important in choosing between valid word hypotheses. The word in figure 3.1b could equally well be identified as 'clump' or 'dump' or even 'jump', and it is only from the meaning of surrounding words that the two can be distinguished. Grammar can be sufficient to distinguish ambiguous words, by determining from the surrounding context whether a word is a verb or a noun, or whether a verb is transitive or not. To implement this discrimination in an automatic system, some language model must be introduced to determine legitimate word sequences. Language models are discussed in section 8.4.

Context is important for the skilled reading of passages of text, but is not considered by Rayner and Pollatsek (p.62) to be an important influence on the reading of the words within that text. However, the results quoted by Edelman *et al.* (1990) show how difficult it can be to identify handwritten non-words, thus highlighting how important a restricted lexicon and context are.

> "In comparison, people recognize correctly 96.8% of handprinted characters [Neisser and Weene 1960], 95.6% of discretized handwriting [Suen 1983] and about 72% of cursive strings (see [Edelman 1988] appendix 1)."

Edelman's (1988) experiment consisted of presenting non-word cursive strings to four subjects. The subjects had to type their reading of the cursive string, with no time limit to the responses, and allowing multiple guesses. Edelman found the error rate consistent with the error rate for individual letters.

The problem of handwriting recognition is complicated by the fact that much handwriting is intended for use only by the author. When people speak, it is invariably with the purpose of being understood by someone else, and that person is there to query any ambiguities immediately, or to indicate if the speech is difficult to understand for whatever reason. There is feedback of any errors that are made, so behaviour can be corrected, with the aim of transferring information most effectively. On the other hand, writing is usually read much later than it is created, and this feedback loop does not exist. Writing not legible to others is easily accepted by an author who already knows what is written. Particularly if a writer is used to wordprocessing documents for consumption by others, notes written for personal use may be written in a way that other readers cannot understand. Words may simply become illegible mnemonics comprehensible only to the author who knows the context in which they were written. However, it is just such notes to one's self that pen computers are designed to store and, it is claimed, recognize — an exacting if not impossible task.

3.4 Summary

From the work that has been reviewed in this chapter, it is possible to extract a number of important principles which can be used for guidance in the design of a machine to read cursive script. While following psychological studies might not yield the easiest nor the best method of tackling this problem, being aware of how people read gives an indication of the operations of the best reading machine known. Those factors which are seen to be important are summarized below, and taken into consideration in the design of the handwriting recognizer in the subsequent chapters.

First, in the recognition of written forms, it seems that beyond the simple representational level of the Hubel and Wiesel cells, people recognize letters by observing higher-level features. Though the exact features are unknown, it seems that they correspond to such elements as loops, curved strokes and straight line segments. If these features are how information is conveyed between people in handwriting, then they would be a good choice of feature for a machine handwriting recognizer, as they are likely to be invariant between writers and under different conditions. Further, while people learn to read by recognizing individual letters, and this might be necessary for new or long words, skilled readers take in whole words at a time. It can also be seen that reading is made possible only by knowing that most words will fall into a prior vocabulary, and by using the context surrounding words to overcome ambiguity.

Chapter 4

Overview of the system

Polonius: What do you read my lord?
Hamlet: Words, words, words.

Shakespeare. Hamlet.

Having reviewed the literature, it is apparent that until recent years there has been a dearth of research and publications on the problems of off-line recognition, but that there is great potential for applying successful systems — particularly in the banking and postal fields. Recently the situation has changed, but there still remains a significant gap between the performance of research systems and the accuracy required for practical implementations.

To attempt to fill part of this gap, the system described in this thesis has been developed to carry out all the operations of off-line handwriting recognition, from scanning to producing a machine-readable document of recognized words. This chapter briefly describes the whole system and then details a number of issues relating to the complete design, including a description of the databases used for experiments. Subsequent chapters present the other aspects of the system in more detail.

4.1 Summary of parts

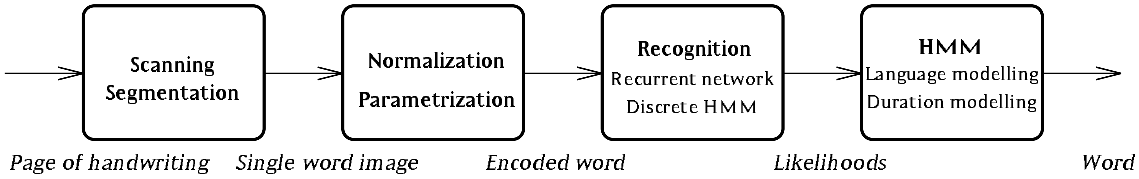


Figure 4.1: A schematic of the recognition system, showing the main processes which must be carried out to identify the words in a handwritten document.

The system described in this thesis can be conveniently divided into the same broad sections as are found in most other handwriting recognition systems, such as those described in chapter 2. The system begins with data acquisition and proceeds in a bottom-up manner, processing smaller amounts of data at successively higher levels of representation, to arrive at a word identity which can be output in ASCII code.

To capture data from a handwritten document, in general some sort of scanner is used rather than a camera, to ensure controlled conditions, especially of lighting. A variety of scanners is available, from hand-held units for reading a small amount of material, through flat-bed scanners and machines with sheet feed or page-turning, up to postal machines with a very fast throughput.

The scanned image must be *segmented* into separate words (section 4.2) and then a series of image processing operations is carried out to normalize the image, as described in the first half of chapter 5. The latter half of that chapter discusses the best way of representing the useful information contained in the image. That chapter and the next also discuss the derivation of handwritten features from the image, as a succinct way of describing the shape of the handwriting.

Chapter 7 then discusses how data probabilities can be estimated from the encoded feature information. Three different pattern recognition techniques are described together with the training method for each. From each of these the probabilities are combined in a hidden Markov model system (chapter 8) which finds the best choice of word for the observed data. This system allows the natural incorporation of prior information about the lengths of letters and about a restricted list of permitted words, about the grammar of a language and potentially even the semantic context of the writing.

4.2 Image acquisition and corpus choice

> *The success of any decipherment depends upon the existence and availability of adequate material. How much is needed depends upon the nature of the problem to be solved, the character of the material, and so forth.*
>
> John Chadwick. The Decipherment of Linear B.

The system is designed to process data captured from a scanner, but for research purposes it is convenient to work on a fixed database stored on disk for repeatability and speed. Ideally work would have been conducted using a standard database to produce results which would be easily comparable with the results quoted for other systems. In the speech recognition community the production of standard databases has made available large corpora of speech which individual institutions could not collect themselves. This has enabled reliable comparison between different recognition systems and encouraged competition, albeit tending to narrow the goals of research towards performing well on the standard tasks. However, at the start of this research there was no off-line cursive database available, so the only solution was to collect a new database. Subsequently the CEDAR database (Hull 1993) has been released, but it is designed specifically for the task of isolated word recognition from address blocks, and introduces a number of special problems which did not fall into the already wide scope of this research. These problems include having to deal with overlapping words and having to remove guide lines, envelope patterns and other clutter, though work has been done to remove much of this noise (Doermann 1993; Kimura *et al.* 1993b).

In the database collected for this research, words were written by a single author on a plain, white A4 sheet. The writer used a black fibre-tip pen which gave clear strokes with sharp edges, but the strokes are wide and overlap. The sheets, each containing 150–200 words, were then scanned on a flat-bed scanner at 300 dots per inch resolution, in 8 bits (256 levels of grey) to produce one file per page. Each page takes about 8Mbytes of storage in TIFF (Aldus and Microsoft 1988) format, when not compressed.

The next task is to segment each page into its component words. Under real conditions, this problem can be difficult. However, there exist published techniques for performing this operation (Garris *et al.* 1994; Yanikoglu and Sandon 1993) and it has not been studied in detail in this work. For this database each word was written within a wide border of white space to facilitate segmentation. The algorithm for segmentation is thus very simple, merely looking for blank horizontal lines to partition between lines of text. Within lines of text, the algorithm looks for long horizontal gaps between words. If the algorithm fails, the automatically determined bounding boxes around words can be manually adjusted using a graphical tool developed for the purpose. Figure 4.2 shows a section of a page of data with the automatic segmentation displayed. Words are automatically labelled by alignment with the machine readable file which was used to prompt the writer.

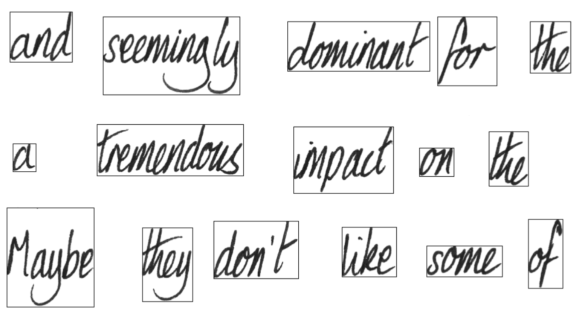


Figure 4.2: A section of a page of the database, showing the bounding boxes detected automatically.

Initial tests were carried out on a database of the numbers written out as words ('one' to 'nineteen', tens from 'twenty' to 'hundred', plus 'thousand', 'million' and 'zero'). These words were chosen because they form a corpus useful for an application such as cheque verification, but the small vocabulary enabled a reasonable study to be made in a short time and facilitated data collection. Ten exemplars of each of these words were taken: three to serve as a training set and four as test data (a test set of 124 images), plus a further three to be used as a validation set (see section 7.1.3).

Subsequently, a larger data set was created by the collection of transcripts of the Lancaster–Oslo/Bergen (LOB) corpus (Johansson *et al.* 1986). This is an extensive corpus of modern English collected from a wide variety of sources such as newspapers, novels and non-fiction books. The corpus as a whole contains a million words with a vocabulary of around 40,000 words. Writing out sentences from this corpus gives larger data sets permitting better training of the recognition system and laying the foundations for future work on language modelling to improve the results, based on work already conducted, for instance by Kuhn and de Mori (1990). The LOB handwritten database contains 2360 training images, 675 validation images and 1016 test images from words written by a single author. Initial transcriptions consisted entirely of lower case words, but subsequent additions to the database have included punctuation and capital letters. The vocabulary of the transcribed corpus is 1334 words, and results quoted use this lexicon size except where stated otherwise. The size of this database is sufficient for training for single-author recognition, but more data would be necessary to tackle the writer-independent task.

It is hoped that more standard databases of off-line data will become available as more research is conducted in the field. To encourage this and to encourage cross-testing on multiple data sets, the database described above has been made publicly available.[1]

4.3 A note on results

To provide a measure of the worth of each of the techniques presented, experiments are described throughout the thesis and the corresponding results are presented. Since there is usually no direct, objective measure of the effectiveness of one technique compared with another, two techniques are often compared by training a complete system for each of the possible conditions and testing on an unseen test set. The final results obtained are percentage error rates showing the proportion of words in the test-set incorrectly classified by the whole system. These error rates are used to compare two techniques or determine an optimum parameter value by holding all other variables constant. The standard experimental conditions for each part of the system are made clear in the following chapters as those parts are described (and are summarized in section 8.4.3), but many results are presented before the whole system has been explained in detail. For comparison, since the standard test vocabulary is 1334 words, random guessing would give a 99.9%

[1] A sample is available by anonymous ftp:
`ftp://svr-ftp.eng.cam.ac.uk/pub/data/handwriting_page_image.tar.gz`

error rate, and guessing the most likely word ('`the`') all the time would give a 93.2% error rate.

Because the training of recurrent networks is found to be dependent on initial conditions, results are subject to a certain amount of variation. Where possible, several networks have been trained under conditions identical except for the initial values of the weights. From these runs, an estimate $\hat{\mu}$ of the mean percentage error rate can be obtained, as can $\hat{\sigma}$, the standard error of the mean. However, the training of recurrent networks is very computationally intensive, so it has not been possible to train multiple networks for every experiment. In experiments where only one run has been carried out, standard errors estimated from multiple runs under similar conditions are quoted. Where two techniques are to be compared, statistical tests are carried out. The one-tailed Student's t-test is used for paired data, for instance when several networks are tested under two different conditions, to determine if the difference in the mean error rate is significant. The statistic of the test is denoted $T(\text{degrees of freedom})$ and the relevant tabulated value is shown as $t_{\text{significance}}(\text{degrees of freedom})$.

Training and testing times are quoted in the following chapters. For comparison purposes, all times are given as the equivalent for a Silicon Graphics R4400 Indigo with 150MHz clock. All times are approximate, and test times are given as the average time per test word over the whole test set.

4.4 The remaining chapters

The next chapter describes the techniques used to normalize the word image, and the coding schemes used to represent the data for recognition. Finally, it describes the simple features which can be extracted from the skeleton of a handwritten word. Chapter 6 describes a more complex technique which can be used to extract larger scale features. The recognition systems which operate on the encoded data to derive character probability estimates are described in chapter 7, and chapter 8 explains the system used to make the choice of the best word, given these estimates.

Finally, chapter 9 draws together the results of the previous chapters, makes an assessment of the whole system and points to the possibilities for further work building on that described in this thesis.

Chapter 5

Normalization and representation

L'écriture est la peinture de la voix.

Voltaire.

The system described in this work is designed to identify a handwritten word when presented with a scanned image. A system could be envisaged which identified the word directly from the image presented, but the task of the recognition system is greatly simplified by preprocessing the image, organizing the information and representing it in a more accessible manner. The processing to be carried out before recognition consists of two major parts — normalization and representation. The first of these attempts to remove variations in the images which do not affect the identity of the word, and the second then expresses the salient information contained in the image in a concise way, suitable for processing by a pattern recognition system. This chapter describes the normalization operations performed on each image by this system.

5.1 Normalization

Cursive script varies in many different ways. In addition to the peculiarities of an author's idioscript, which mean that one writer can be identified among thousands, there are the peculiarities of writing in different situations, with different media and for different purposes. In the recognition task to be solved here, all this variation is irrelevant and serves only to obscure the identities of the words, although in other applications, such as author verification, this 'noise' may be of most interest. One way of reducing the variation is to identify certain parameters of the handwriting that may vary to give a different appearance to a word. Then, a procedure must be determined to estimate each of these parameter values from the sample word (or several) and finally another procedure must be found to remove the effects of the parameter from the word. The most obvious parameters include the following:

Height The height of letters will vary between authors for the same task, and for a given author for different tasks (for instance dependent on the size of guidelines given, or the amount of text to be fitted into a space);

Slant The slant is the deviation of strokes from the vertical. This tends to be a writer-dependent parameter, but varies between words too;

Slope This is the angle of the base line of a word if it is not written horizontally. Even when given a horizontal guide line, authors will write all or some words with non-horizontal bases. Often this can be assumed to be straight, but in extreme cases curved, 'hill-and-dale' base lines may be observed (Srihari and Božinović 1987:p.229);

Stroke width This depends on such factors as the writing instrument used, the pressure applied and the angle of the writing instrument as well as the paper type;

Rotation If the page is skew in the scanner, then all the words will be rotated, by a process independent of slant and slope which are shear processes in the production of the handwriting. In this system though, rotation is assumed to be small and is removed by a combination of slant and slope-correction transforms.

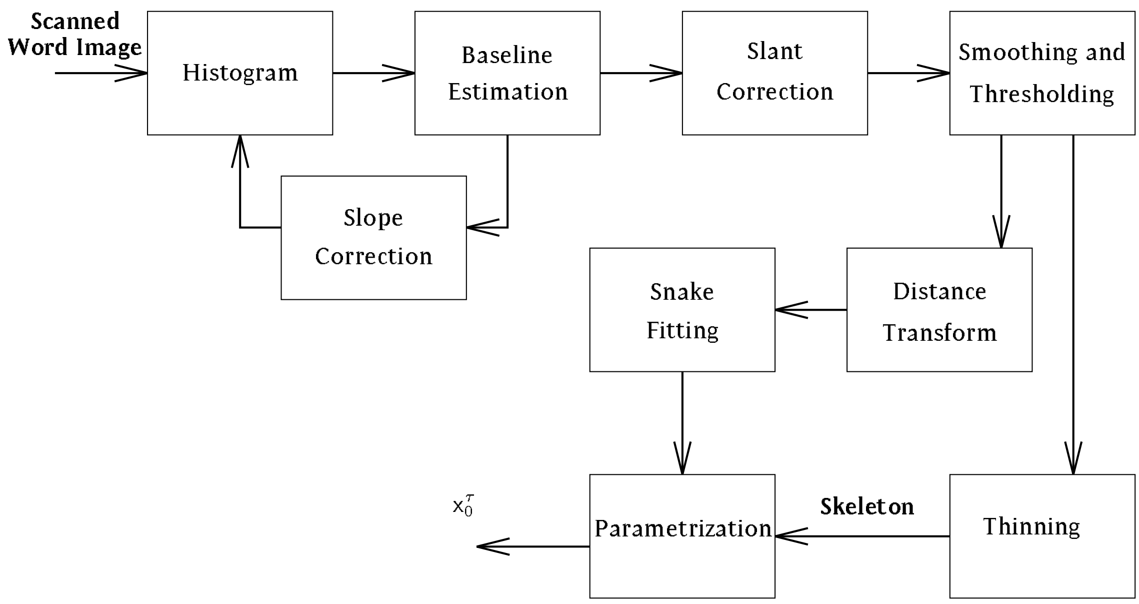


Figure 5.1: A schematic of the preprocessing operations needed to normalize the image before it is encoded.

The system described here incorporates normalization for each of these factors, reducing each image to one consisting of vertical letters of uniform height on a horizontal base line and made of one-pixel-wide strokes. Figure 5.1 shows a schematic of these normalization operations, which are explained in this chapter. The normalization process described in the following sections is illustrated for a sample word in figure 5.3.

5.1.1 Base line estimation and slope correction

The character height is determined by finding the intuitively important lines which are shown running along the top and bottom of lower case letters in figure 5.2 — the upper and lower base lines respectively (using the terminology of Srihari and Božinović), with a centre line between the two. With these lines, the ascenders and descenders which are used by human readers in determining word shape (section 3.1) can also be identified.

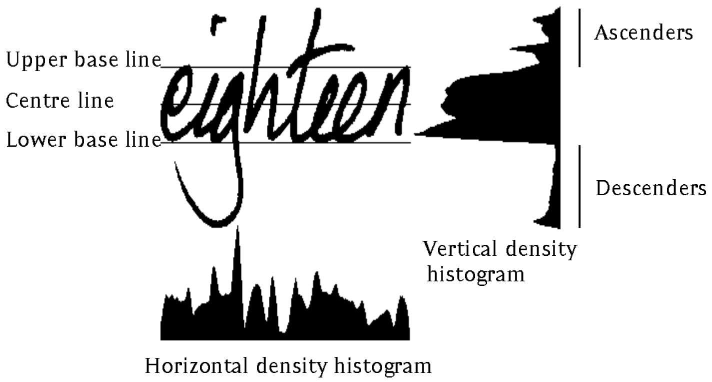
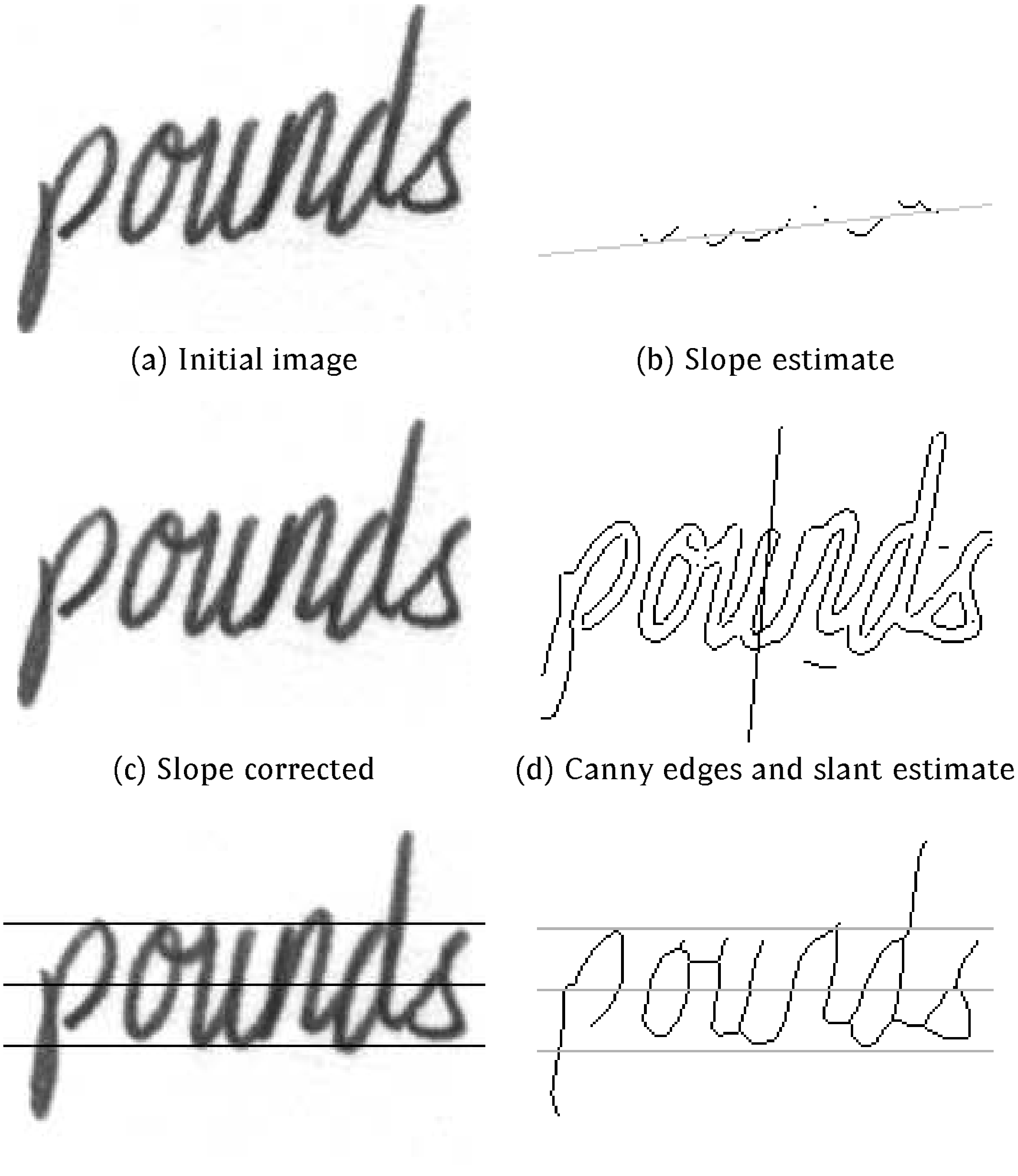


Figure 5.2: Histograms, centre line and base lines.

The heuristic used for base line estimation consists of the following steps:

1 Calculate the vertical density histogram by counting the number of black pixels in each horizontal line in the image. Vertical and horizontal density histograms are shown on the right and bottom edges of figure 5.2.
2 Reject the part of the image likely to be a hooked descender (as in the letters 'gqy'). Such a descender is indicated by a peak in the vertical density histogram. The minimum in the histogram above this point is found and the image is cleared from that point downwards.
3 Find the lowest remaining pixel in each vertical scan line.
4 Retain only the points around the minimum of each chain of pixels.
5 Find the line of best fit through these points (figure 5.3b).
6 Reject the outlying points and calculate the new line of best fit. This is now considered to be the base line of the character.

Given the estimate of the lower base line, the writing can be straightened to make the base line horizontal. This straightening is carried out by application of a shear transform parallel to the *y* axis (figure 5.3c). Slope correction



(a) Initial image

(b) Slope estimate

(c) Slope corrected

(d) Canny edges and slant estimate

(e) Slant corrected with base line estimates.

(f) Skeleton

Figure 5.3: Successive stages in the normalization.

can be carried out on whole lines to remove rotation in the scanned image or skewed writing, and then carried out on individual words to remove local transformations. Next, the height of the lower base line can be re-estimated, under the assumption that it is now horizontal. The upper line may be re-estimated using a similar procedure, though this is found to be less robust, because of the presence of 'ℓ' strokes, which are harder to separate from the body of text than are descenders, as Božinović and Srihari (1989) observe.

5.1.2 Slant correction

Božinović and Srihari (1989) detail a complex method for letter slant correction. This involves isolating areas of the text which are near-vertical strokes and estimating the slant of each of these. This procedure was found to be very sensitive to the thickness of the writing and is unreliable when the writing is thinner than expected. However, by making an estimate of the writing thickness from the distance transform (see section 6.1) and using an iterative technique, a more stable version of this algorithm has been developed.

Božinović and Srihari's algorithm commences by eliminating all horizontal rows in a word which contain horizontal strokes. These are identified as any rows which contain long runs of black pixels. The maximum number of consecutive black pixels which can be permitted before a line is eliminated is a parameter which must be specified. After each such row is eliminated, the remaining image is in horizontal strips, some of which are too narrow to use and are eliminated. (A second, less critical parameter is the smallest height of horizontal strip which can be used to estimate the slope.) The remaining strips are divided into boxes containing separate, near-vertical strokes in each of which the centroids of the upper and lower halves are determined, and the slant of the line between the two is calculated. Averaging the slants across all such strokes gives an estimate of the average overall slant of the word. The slant is corrected with a shear parallel to the x-axis. Figure 5.3e shows a slant-corrected word.

The modification which has been found to stabilize this algorithm is to split the word into strokes for a range of values of the run-length parameter and to use the value which gives the greatest number of boxes. It is under these conditions that the best slant estimates are obtained. A further refinement is to discard boxes in which the stroke fragments in the top and bottom sections are not connected and cannot be sensibly used to estimate the stroke slant.

In practice, despite the modifications, the algorithm was still sometimes found to give poor slope estimates, and an alternative technique was tried and found to be more reliable. This involves finding the edges of strokes, either by finding the contour of the thresholded image (Caesar *et al.* 1993a; Kimura *et al.* 1993a) or by using an edge detection filter. Both of these techniques gives a chain of connected pixels representing the edges of strokes. The orientations of edges which are close to the vertical are averaged to give an overall slant estimate. An estimate based on the Canny (1986) edge detector has been used in this system. It is found to tend to underestimate the slant as in figure 5.3d. Yanikoglu and Sandon (1993) find a similar estimate, using the mode slant found by edge operators within $\pm 30°$ of the vertical.

5.1.3 Smoothing and thinning

To remove noise from the image, either from the original document, from scanning defects, or from applying shear transforms to discrete images, it is useful to smooth the image. This is carried out by convolution with a 2-dimensional Gaussian filter. It has been found that there is little noise on a scanned image when using a black fibre-tip pen on plain white paper, but degradation from this ideal situation is possible from a large number of sources such as paper quality, age and condition; pen or pencil type; poor illumination when using a camera rather than a flat-bed scanner; and show-through from writing on the other side of a page.

Having normalized and smoothed the image, it is thresholded to leave every pixel black or white. Next an iterative, erosive thinning algorithm is applied to reduce the strokes in the writing to a width of one pixel so they can be followed later. This is the *skeleton* of the word shown in figure 5.3f. The algorithm used was that due to Davies (1990:p.153).

Skeletonization is a notoriously difficult problem to solve well, and many algorithms have been written, with a variety of properties. Lam *et al.* (1992) present a comprehensive review with 138 references. Despite this difficulty, because the skeleton is to be coarsely parametrized later, a simple algorithm was found to work well, and other algorithms that were tried (Zhang and Suen 1984; Arcelli and Sanniti di Baja 1985) did no better. There is scope for more work on identifying a suitable thinning algorithm for handwriting, but it would seem that a model-based method such as those of Pettier and Camillerapp (1993) and Doermann (1993), which use the knowledge that the image is made from a series of strokes, is the most promising approach. Ultimately what is required is a skeleton which represents the strokes perceived when a human reader observes a word. Such a skeleton is probably best found as that which approximates the path of the pen most closely (corresponding to the data received in an on-line system), and not an algorithm that best matches a human approximation to a pixel-based skeletonization algorithm as has been suggested (Plamondon *et al.* 1993). Experiments were carried out, matching skeletons of off-line images with the on-line data for the same writing, but it was found that from conventional tablets there is a large error (of the order of the stroke width) in the reported pen position when the pen angle varies. Without better hardware, this investigation could not be pursued.

It is worth noting that in the database collected here, the strokes tended to be wide, making skeletonization difficult. In many papers, (*e.g.* Caesar *et al.* 1993a) the stroke width is small, so skeletonization works well and both the skeleton and contour will give good approximations to the true pen path.

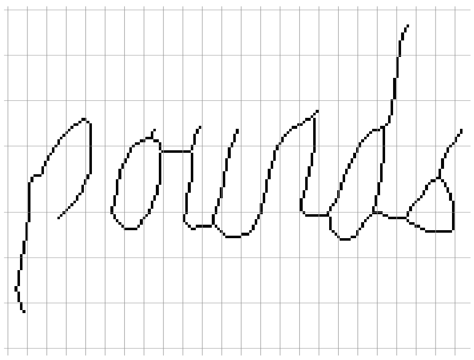
5.2 Parametrization

Now that the image has been reduced to a standard form, which highlights invariants of the words and suppresses spurious variations, the normalized image needs to be parametrized in an appropriate manner for input to the network which is to carry out the recognition process. From the original scanned image, which can take 8MB of storage space, all that is ultimately desired is the identity of the words on the page, an information content of the order of a few hundred bytes. One way of looking at recognition is as a process of information sifting with the ultimate aim of deriving the word identities. In order to process the data effectively with a recognition technique such as a connectionist network, they must be reduced in number and transformed into a form more appropriate than a grey scale image. Data representation is of prime importance in pattern recognition problems and can easily mean the difference between a particular method solving or failing to solve a problem. The problem of representation is discussed more generally by Marr (1982) and Winston (1984:ch.8). Speech is coded using techniques such as filters, cepstra, Mel scale binning and vector quantization before attempting recognition. These representations express the relevant information in a much more useful form than the original time-varying voltage measured by an analogue to digital converter attached to a microphone. Similarly, in script recognition, the useful, invariant information must be extracted from the written words while discarding the vast majority of redundant variation. The remainder of this chapter describes the processes used to reduce the amount of data used to describe a word, and deals with the problem of how the word should best be represented.

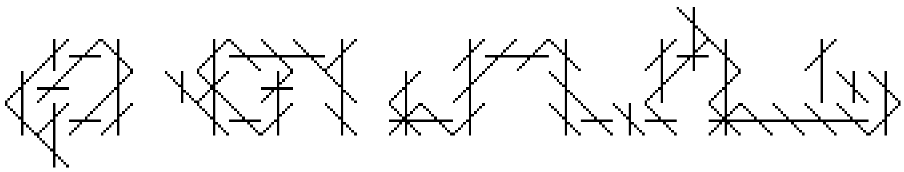
5.2.1 Skeleton coding

The main method of parametrization used is to code the skeleton of the word so that information about the lines in the skeleton is passed on to the recognition system. An alternative method, based on the grey-level image is described in section 5.2.3.

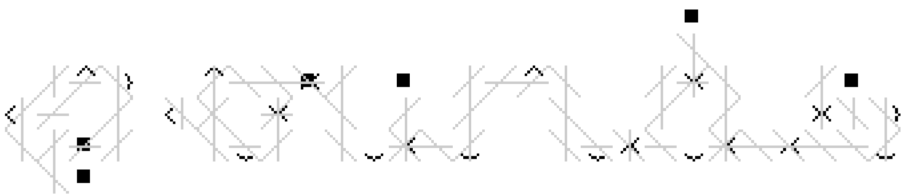
In the skeleton coding scheme, the area covered by the word is first divided into a grid of rectangles. (Figure 5.4a.) The vertical strips (*frames*)are of a fixed width for the whole word, a length determined by the height estimate of the character. Typically there are 6 frames in the horizontal space occupied by one character height. This assumes that the character height is proportional to the character width, which is a valid assumption for normal handwriting by a single author, but will not be as accurate for multiple writers.



(a) Skeleton with grid



(b) Parametrized line segment data



(c) Features superimposed on line segment data

Figure 5.4: Successive stages in the parametrization.

The vertical resolution of the grid is chosen so that the word is divided into seven regions, each of which can be identified as playing a definite, but distinct role in the representation of handwriting. The regions close to the upper and lower base lines identified in section 5.1.1 both contain most of the horizontal movements in a word, representing the turning points at the top and base of most small letters, and the ligatures between letters. These two regions also contain the end points of short strokes. The middle region between these two lines captures important information about the short strokes which make up the majority of handwriting, as well as containing the internal detail of the letters ‘e’ and ‘s’. The ascenders and descenders so important in the Bouma shape of a letter (section 3.1) are found in the regions above the half-line and below the base-line, and two more regions can be identified containing the endpoints or loops of ascenders and descenders.

A higher vertical resolution (16 regions) has been tried, but performance was slightly lower because generalization was impaired; the storage requirement of the training data also increased. There is a variable number of vertical frames in a word, with long words having more frames than short words, but a given character will always occupy approximately the same number. For each of these rectangles in the grid, four bins are allocated to represent different line angles (vertical, horizontal, and the lines 45 degrees from these). Within this framework, the lines of the skeleton image are ‘coarse coded’ as follows.

The one-pixel-wide lines of the skeleton are followed, and wherever the skeleton enters a new box in the grid, the section in the previous box is coded according to its angle. The box associated with this segment's (x, y, θ) values is now ‘filled’ (set to one). Segments which are not perfectly aligned with the angles of the bins contribute to the bins representing the two closest orientations. This representation can be seen to resemble the Hubel and Wiesel cells which code information early in the visual cortex. These are tuned to a particular spatial location and angle, but also respond to edges or bars with similar parameters. Caesar *et al.* (1993b) and Bengio *et al.* (1994a) use similar methods of representing off-line and on-line cursive script respectively. This provides the latter with a method for coding the spatial relationships of nearby strokes, and overcoming the problems of delayed strokes.

Figure 5.4b shows the input pattern schematically. Each line represents a full bin and its position and orientation correspond roughly to the position and orientation of the section of skeleton which gave rise to it. Because of the coarse coding, some line segments contribute to two bins and this is seen on the ‘ℓ’ stroke which is between the vertical and 45 degrees so both these lines are shown in the corresponding boxes in figure 5.4b.

Hereafter, the first frame of data in the representation of a word will be referred to as x_0 and the final frame x_τ. The frames $(\mathrm{x}_s, \ldots, \mathrm{x}_t)$ will be denoted x_s^t.

5.2.2 Non-uniform quantization

The above description coded all the frames to be of equal width, and the frames were chosen by blindly drawing a grid on the word image. The width of the frames was chosen in proportion to the character height. In practice though, character height and width vary independently from author to author, so it would be better if these scale factors could be estimated independently. Also, rather than blindly placing the frames, it would be better if they could be aligned more with the data. A single frame could then contain all of a vertical stroke, rather than strokes slightly off the vertical ending up in two adjacent frames.

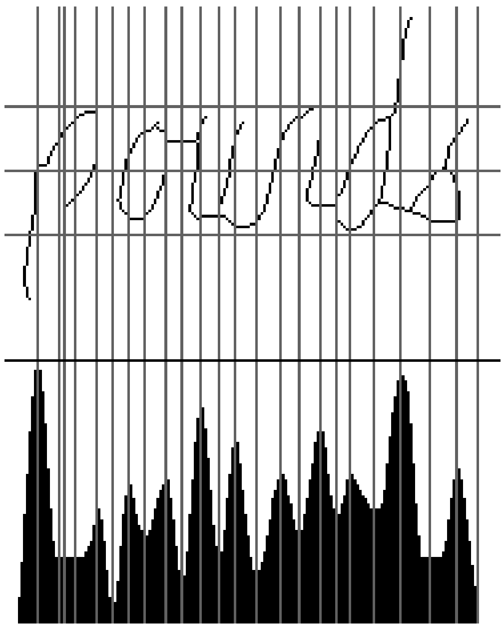


Figure 5.5: The non-uniform horizontal quantization scheme superimposed on the histogram of the original word and its skeleton.

To correct these two problems, a simple system has been devised, which is similar to the system used by Yanikoglu and Sandon (1993) for finding potential letter segmentation points. After the word has been normalized, but before thinning, the horizontal density histogram is calculated and smoothed. The maxima and minima of the smoothed density histogram are found, and frame boundaries are defined to be the midpoints between adjacent maximum/minimum pairs. Further frames are added where the maxima and minima are far apart, to ensure that the frames do not exceed a certain width (chosen according to the character height). Figure 5.5 shows the centres of segments found under this scheme. This quantization scheme is not completely robust, as small changes in the image can lead to different numbers of maxima and minima, despite the smoothing. A better scheme could perhaps be designed, but this one has improved results over the uniform quantization, as is shown in table 5.1.

Quantization method	Size of network	Error rate	
		$\hat{\mu}$	$\hat{\sigma}$
Uniform	80	15.6	0.72
Uniform	160	11.5	1.60
Non-uniform	80	13.3	1.60
Non-uniform	160	9.6	1.60

Table 5.1: Error rates for networks trained on data sampled by different quantization schemes. Results are shown for networks with different numbers of feedback units (section 7.1.3).

5.2.3 An alternative approach

Instead of coding the image in this complicated fashion, it may be asked whether it would not be much easier to simply present the recognition system with the image directly. This would reduce the amount of processing required, and skeletonization artefacts would not distort the data. The same normalization procedures must be carried out to give scale, slant and slope independence and the image must be sub-sampled to obtain a manageable amount of data. Here a vertical resolution of 32 pixels is used for coding letters with their descenders and ascenders. This makes each pixel approximately square when using the same horizontal quantization, and gives a similar number of bins to the skeleton coding. Figure 5.6 shows such an undersampled grey-level image. Each pixel is stored in 8 bits or 256 levels of grey.

Figure 5.6: The word 'pounds' undersampled.

The results obtained for this preprocessing technique are compared with the skeleton coding method in table 5.2. The skeleton coding gives a much lower error rate.

Representation	Error rate %	
	$\hat{\mu}$	$\hat{\sigma}$
Line segments	20.4	1.60
Undersample	31.7	0.84

Table 5.2: Error rates using line segment and undersampling preprocessing methods.

5.3 Finding handwriting features

The previous sections have described how the original word image can be normalized and encoded in a canonical form so that different images of the same word are encoded similarly. However, the coding only represented low-level information about the word, and coded it fairly coarsely to reduce the information burden. The performance of the recognizer can be improved by passing it more information about salient features in the word. Chapter 6 describes a method of finding large-scale features, but a number of useful features can be easily discerned from the processing that has already been performed on the writing.

Dots Dots above the letters '*i*' and '*j*' can be identified with a simple set of rules. Short, isolated strokes occurring on or above the half line are marked as '`i`' dots.

Junctions Junctions are easily found in the skeleton of the word, as points with more than two neighbours. Junctions indicate points where two strokes meet or cross.

End points End points are points in the skeleton with only one neighbour and mark the ends of strokes, though they can be produced as artefacts of the skeletonization algorithm.

Turning points Points when the direction of a skeleton segment changes from upward to downward are recorded as top turning points. Similarly left, right and bottom turning points can be found.

Loops Loops can be found from the skeleton or by performing a connected-component analysis on the original image, to find areas of background colour not connected to the region surrounding the word. A loop is coded by a number representing its area. A number of authors, including Srihari and Božinović (1987), use the topology of a word as a feature. However this is not always a good choice of invariant since extra loops can easily be formed, or loops that could be expected might not be fully closed. Ascenders can become loops, '`t`' strokes can join up with other letters to create a loop, and normally closed letters like '*a*' and '*o*' can be left open or filled in normal handwriting.

Each of these features can be encoded in a single bin but, while it is only useful to know whether a loop or dot is present in a particular frame, the positions of the endpoints, turning points and junctions are useful and they are recorded along with the angle bins for each horizontal strip. Thus instead of four angle bins at each vertical position, ten features are encoded, and an extra two features are associated with the whole frame. With seven horizontal bands, this increases the size of a frame from 28 bytes (7×4) to 72 ($7 \times (4 + 4 + 2) + 2$), but the additional information improves the network's performance. Some of these features are shown in figure 5.4c, superimposed on the line segment features. Endpoints are indicated by '■' shapes, turning points by '$<$' and junctions by '$\times$'. Table 5.3 shows the performance improvement obtained by adding these features to the representation.

Representation	Error rate %	
	$\hat{\mu}$	$\hat{\sigma}$
Line segments	20.4	1.60
Line segments with features	18.2	1.60

Table 5.3: Error rates using line segment coding method, with and without the skeleton features.

5.4 Summary

This chapter has described a variety of normalization methods for handwritten words and then described a coding scheme for those words. It has been shown that a coding based on extracting information from the skeleton is more effective than one based on the grey level of the image. Features have been extracted from the skeleton and are found to improve recognition further.

Chapter 6

Finding large-scale features with snakes

Let there be snakes! And snakes there were, are, will be. . .

Silvia Plath. Snakecharmer.

The previous chapter described a coding for handwritten words which records the location and orientation of the line segments in the skeleton. This coding was then extended to incorporate low-level features which could be easily identified. All of these features were simple and local — depending only on information from a small area of the image. However, in section 3.1 it was seen that the features generally held to be of most significance in reading were larger-scale, stroke-like features. It would be highly desirable if information about the presence of such features could be determined and concisely encoded for use in recognition.

A number of off-line handwriting recognition systems have used large-scale features for recognition, indeed some are based entirely on the use of such features. This chapter describes a new method of automatically finding a large class of stroke-like features in cursive words written with broad strokes. Before describing the method used in this system, it is worth looking at the methods that have been used by other authors.

Srihari and Božinović (1987) define their features with rules based on the contours of the word images. The features that are defined are short and long strokes, curve sections, loops and dots. These authors constructed their off-line data from on-line tracing information, which seems to have given smooth curves and narrow strokes. However, defining rules that will reliably pick out features when there is noise is extremely difficult, and relying on the contour means that features that run across intersections can not be detected.

Edelman *et al.* (1990) use a method similar to that described in this chapter to represent stroke-like features in on-line handwriting. They fit a number of prototype stroke features to the on-line handwritten string, and use the identities of the strokes that matched to find letter hypotheses and eventually word matches. The method is described as optical matching, but the data is again collected from a graphics tablet so the strokes are narrow and allow curve-fitting to the contour alone. Because stroke contours are smooth, enough strokes can be matched reliably along the length of the stroke sequence, and letter hypotheses can be proposed solely on the basis of these features.

6.1 Finding strokes

The problem with both of the above methods is that they require clean data and narrow strokes. They operate on the contour of the image, but the features that should be detected are the strokes, which are better characterized by the path of the pen centre than by either the left or right edge. Thus, this chapter describes a method of finding the centres of strokes regardless of the thickness of the stroke, the irregularities in the stroke edges, or the presence of overlapping strokes or edges.

The centres of strokes are those parts which are furthest from the edges, so a natural choice of representation to consider is the distance transform. This assigns a value, $D(x, y)$, to each pixel (x, y) in the thresholded image, which is the distance of that pixel from the nearest background pixel, zero if the pixel is itself part of the background. Thus circles in the image become cones in the distance transform, the transform increasing the further a point is from the edge, and strokes become ridges. Now detecting stroke centres becomes a problem of finding ridges in the distance transform. The method chosen to find these ridges is *snakes*.

6.2 Snakes

Snakes are deformable splines (smooth curve segments) placed in a potential field which translate and deform to reduce their potential energy. Traditionally they have been used to find edges in grey level images, by according low potentials to areas of high contrast so that the snake seeks to match its contours to high contrast edges. Such a use is seen in the original paper of Kass *et al.* (1987). Further uses have included tracking curve sections in video sequences (Cipolla and Blake 1990), and extraction of features from faces (Yuille *et al.* 1992). In the latter case, a parametric model was built for each of the features to be extracted (*e.g.* eyes, mouth) and these were fitted to real images. Leymarie (1990) uses snakes to find skeletons in much the same way as they are used here, attempting to find maxima of the distance transform. The remainder of this section describes in more detail the mechanism underlying the snakes' operation.

The shapes of snakes are governed by cubic B-splines (Pavlidis 1992). A series of N control points $\{\mathbf{p}_i : i = 0, \ldots, N-1\}$ is defined in a two-dimensional plane and the actual spline path generated is an interpolation of these points (figure 6.1), each point $\mathbf{x}(s)$, $s \in [0, N-1]$ on the path being a weighted sum of the nearest control points' positions. $B(s)$ is a polynomial function which determines how much weight is given to each control point, according to the parameter s which increases from one end of the curve to the other. The B-spline is forced to terminate at the end control points by generating 'phantom' control points $\mathbf{p}_{-1} = 2\mathbf{p}_0 - \mathbf{p}_1$, and $\mathbf{p}_N = 2\mathbf{p}_{N-1} - \mathbf{p}_{N-2}$.

$$\mathbf{x}(s) = \sum_{i=-1}^{N} B(s+2-i)\mathbf{p}_i \tag{6.1}$$

$$B(s) = \begin{cases} \frac{1}{6}s^3 & 0 \le s \le 1 \\ \frac{2}{3} - \frac{1}{2}(s-2)^3 - (s-2)^2 & 1 < s \le 2 \\ \frac{2}{3} + \frac{1}{2}(s-2)^3 - (s-2)^2 & 2 < s \le 3 \\ \frac{1}{6}(4-s)^3 & 3 < s \le 4 \\ 0 & elsewhere. \end{cases} \tag{6.2}$$

The spline shown in figure 6.1 has the minimum four control points. For more complex shapes, more control points can be added, but each point on the curve is only determined by the four nearest control points. Other (non-cubic) splines can be defined, interpolating more or fewer control points. The weighting polynomials ensure continuity and smoothness (C^2).

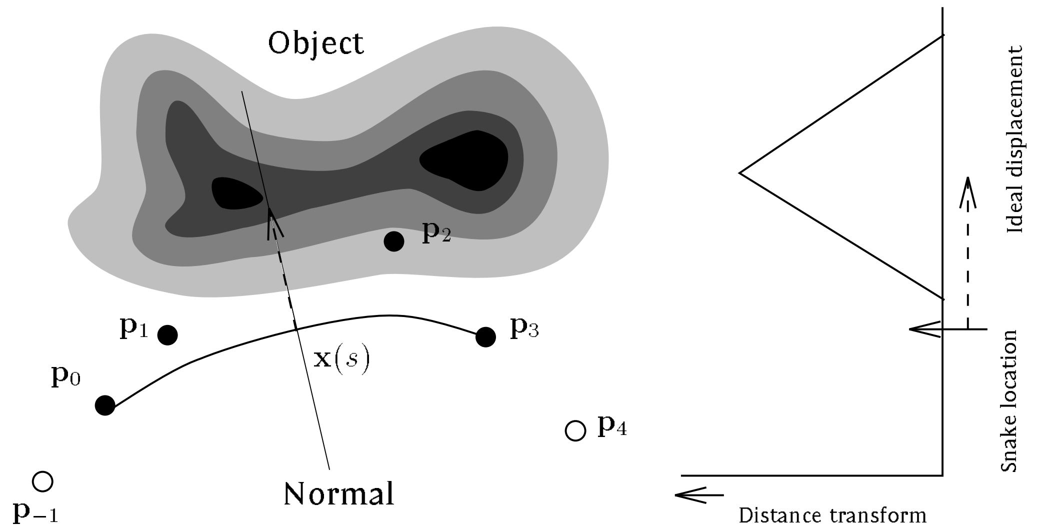


Figure 6.1: A snake with four control points and the distance transform along a normal.

Given the positions of the control points, the snake can now be located on an image. How it moves, according to the features in the image, must now be defined. A potential function $-f(x, y)$ is defined on the pixels $\{(x, y)\}$ where the snake is to be attracted to curves of high values in f. f might be intensity I, contrast $|\nabla I|^2$ or, as in this case, the distance transform $D(x, y)$. Here the city-block metric $D = |\Delta x| + |\Delta y|$ has been used for simplicity of computation.

The spline curves are sampled so that M samples are generated per unit in s. At each sample point s_k the normal to the curve is searched for the minimum of the potential function $-f$ within a certain distance on either side. The displacement of the minimum is recorded for each sampling point, and these displacements are then added to the control points to move the snake towards the local maxima. Since each sample point is a weighted sum of the nearest four control points:

$$\begin{aligned}\mathbf{x}(s_k) &= B(s_k+2-i)\mathbf{p}_i + B(s_k+1-i)\mathbf{p}_{i+1} + B(s_k-i)\mathbf{p}_{i+2} \\ &\quad + B(s_k-1-i)\mathbf{p}_{i+3},\end{aligned} \tag{6.3}$$

the displacement $\mathbf{d}(s)$ is distributed among these control points:

$$\mathbf{p}_i(t+1) = \mathbf{p}_i(t) + \frac{1}{M}\sum_k B(s_k+2-i)\mathbf{d}(s_k). \tag{6.4}$$

The new control points define a spline which lies closer to the lines of local maxima, and after two or three iterations a good match will be found if one is present in the search area around the snake's initial position.

6.3 Point distribution models and constraints

As defined above, these snakes do not serve the purpose of feature recognition. They are very flexible, so any snake can adapt to fit a wide range of feature shapes, even collapsing to a point in some potential wells. To compensate for this, Kass *et al.* define an internal energy based on the integral of first and second derivatives along the snake's length, to penalize high curvature. This general 'straightness' constraint suits the purposes of tracking edges in images, but to find features, the constraints need to be chosen so that the snake can only match features of a particular shape.

A number of models must be generated, each matching a particular feature, but able to match instances of that feature whole shapes vary somewhat. Cootes and Taylor (1992) describe 'Point Distribution Models' (PDMs) which they use as shape descriptors for various objects such as hearts in magnetic resonance images and resistors on images of circuit boards. The essence of the PDM is performing Principal Component analysis on the covariance matrix of the coordinates of the control points of a snake, and restricting the snake's shape to match shapes that have been seen in a training set.

If a snake with n control points is placed on K examples of a particular feature, for instance the short vertical stroke of an '*ı*', the positions of the control points can be recorded and statistics gathered. If the kth example feature has position $\mathbf{s}_k = (\mathbf{p}_{k,0}, \ldots, \mathbf{p}_{k,n-1})^{\mathrm{T}}$ the centroid of that example can be found:

$$\bar{\mathbf{p}}_k = \frac{\sum_i \mathbf{p}_{k,i}}{n}. \tag{6.5}$$

The mean displacement of each point from the centroid can be calculated by subtracting the centroids and averaging:

$$\delta\mathbf{s}_k = (\mathbf{p}_{k,0} - \bar{\mathbf{p}}_k, \ldots, \mathbf{p}_{k,n-1} - \bar{\mathbf{p}}_k)^{\mathrm{T}} \tag{6.6}$$

$$\overline{\delta\mathrm{s}} = \frac{\sum_k \delta\mathbf{s}_k}{K}. \tag{6.7}$$

$\overline{\delta\mathrm{s}}$ is the mean shape of the feature and represents a typical example. If the deviation of a particular example from the mean shape of a feature is found:

$$\mathbf{\Delta}\mathrm{s}_k = \delta\mathrm{s}_k - \overline{\delta\mathrm{s}}, \tag{6.8}$$

it can be considered as a vector of $2n$ coordinates and the $2n \times 2n$ covariance matrix $\mathbf{\Sigma}$ of the shapes can be found:

$$\Sigma = \frac{\sum_k \mathbf{\Delta}\mathrm{s}_k \mathbf{\Delta}\mathrm{s}_k^{\mathrm{T}}}{K}. \tag{6.9}$$

Principal Component Analysis can be carried out to determine the modes of variation in the system. This is done by diagonalization of the covariance matrix. Each eigenvector shows a correlation in the variation of the point coordinates — a 'mode' of variation in which the points concerned have linearly related displacements. The eigenvalues give the extent of variation in the direction of the corresponding eigenvector, so the largest eigenvalue's eigenvector captures most of the variation in the model shape. These modes are strikingly demonstrated in Cootes *et al.*'s (1992) resistor model where the first few modes correspond to natural physical parameters such as the position of the resistor on its wire, the bend of the wire, and the shape of the resistor body. Figure 6.2 shows the major modes of variation of two feature models.

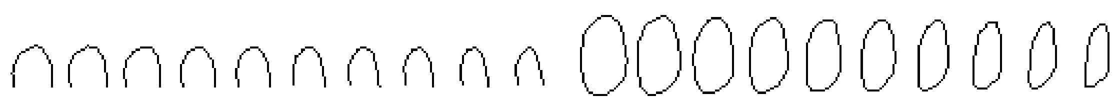


Figure 6.2: Snake models for 'n' and 'o' features showing the major mode of variation within $\pm 1.5\sigma$ of the mean.

Having determined these modes of variation, they can be used to constrain the variation of a snake. Having worked out the new position of a snake with no constraints, from one iteration of the techniques of section 6.2, the centroid of the snake is calculated from the new control point coordinate vector. Transforming this difference into the coordinate frame of the principal components gives the deviation from the mean in each direction. Variation in the minor modes is suppressed since this represents deviation from the space of typical stroke shapes. The Mahalanobis distance $d^2(\mathbf{\Delta}\mathrm{s}) = \mathbf{\Delta}\mathrm{s}^{\mathrm{T}}\Sigma^{-1}\mathbf{\Delta}\mathrm{s}$ shows how much the snake deviates from the model. This distance scales down variation along the principal axes, giving a measure of how many standard deviations the snake lies from the mean, assuming that deviations of snakes from the mean are distributed as a Gaussian ellipsoid. If the distance is too great, it can be reduced by scaling down all components of the deviation. The constrained deviation is then transformed back to the original coordinates, and added to the centroid to generate a new snake which will have a shape similar to those observed in the training set.

Because the displacement to find the distance transform maxima and the application of the constraints are two separate processes, and because the image space is quantized, it is possible that the snake enters a cycle of displacing onto the maximum and being constrained to its original position. The snake thus never reaches a stable position. To avoid this case, the fitting process is stopped after a maximum of 10 iterations, though a match is usually found after just 2 or 3 iterations.

Lanitis (1992) and Lanitis *et al.* (1993) have investigated the use of these models for isolated character recognition for postcode reading. Here a model is produced for each of 36 alphanumeric characters and these models are matched to pre-segmented images of handwritten characters from a postcode database. Each model is compared with each image, and the best match is chosen. These authors do not use the distance transform for the match, but instead rely on the skeleton, which can often be distorted away from the actual strokes at intersections.

6.4 Training feature models

In this work the ideas of splines and principal component analysis in the form of point distribution models have been linked together to form constrained B-spline models of features of handwritten letters.

One model is constructed for each feature to be recognized. In initial studies these features have been: '*n*' hump; '*u*' trough, which also models ligatures; '*i*' stroke (found in many letters including '*u*' and '*n*'); '*t*' cross-stroke; ascender; descender and '*o*' shape. Each of these features can be modelled by a single spline, though other models such as '*x*' may be constructed by joining more than one. Each model contains the mean displacement $\overline{\delta \mathrm{s}}$ of each of the spline control points; the permitted relative variations in these point positions, given by the covariance matrix Σ; and the mean and variance of the observed y co-ordinate of the centroids $\bar{\mathrm{p}}_k$, to record how high in a word the feature occurs. The preprocessor determines character size, so the coordinates are normalized to be independent of the writing size.

Initially a seed model is generated by hand to describe the general characteristics of the feature:

- The number of points needed to model the feature. For a small, straight feature, only four points may be necessary. For a longer line or a curve, six are found to be adequate, but for an 'o' or 's' feature, eight points are required to represent the shape.

- The feature topology (loop or line) and the interconnection of the splines (whether they form an 'x' or whether a loop has a tail or not).
- The position of the feature in a character — whether the feature is in an ascender, a descender or in the middle section of lower case letters.
- The initial shape of the feature.

The seed models are now matched to instances of the features in images of handwritten words. Initially this can be by pointing out feature instances manually, and allowing the seed model to deform without constraint from the mean to match the stroke. When the potential minimum has been found, the snake's shape is added in to the statistics of observed shapes. When a good model has been found, this procedure can be automated so that the features in a word are found automatically. The automatic feature spotting is used both to train the models and subsequently to spot the features used in the recognizer.

6.5 Finding feature matches

Having created a model for each of the features to be found, the next step is to find all occurrences of each feature in the word. The methods described above will find a feature match if one lies close to the starting position of the snake, so snakes must be placed at regular intervals along the word to detect all the features present. A snake, whose shape is initially the mean shape for the model, is placed at the left edge of the word, and permitted to deform to match the distance transform potential, but with the deformation being constrained to lie within κ standard deviations of the mean shape — so the shape will always be similar to shapes already taken by that feature before. (For κ, a value of 1 has been used here.) A best match given the constraints is found by iterating for a limited number of times or until the snake ceases to move. Should the snake move above or below the band where it is normally found, for instance a '4' stroke feature matching the top of an 'r', then it is rejected. Otherwise, the degree of match between the snake and the image is determined.

The degree of match, M, is defined as the difference of two components, representing the degree of support that the data provides for the model and the amount of deformation of the model required to fit the data. The support is the sum of two components: the sum of the distance transform along the length of the snake plus an extra weight, w, for all points that are not background points, and the deformation is measured with the Mahalanobis distance $d(\mathbf{\Delta}\mathrm{s})$ of the match shape from the mean shape of the feature.

$$M = \sum_k f(\mathbf{x}(s_k)) + w_k - d(\mathbf{\Delta}\mathbf{s}) \qquad (6.10)$$

$$\text{where } w_k = \begin{cases} w & \text{if } f(\mathbf{x}(s_k)) \neq 0, \\ 0 & otherwise. \end{cases} \tag{6.11}$$

Snakes with scores greater than a threshold are accepted as feature matches, and the remainder are rejected. The extra weight acts as a penalty for the model crossing areas that are not strokes. Its value is determined empirically (typically 7) and the value of the threshold is adjusted in accordance with this value and the mean value of the distance transform. This makes the matching process independent of the width of the strokes since thick strokes give ridges with higher distance transform values than thin strokes. The mean value of the distance transform is also used to indicate the stroke width in the modified slant detection algorithm, and to give the spatial frequency parameter for the Canny edge detector (section 5.1.2).

This is in contrast to the measure of fit used by Lanitis, who adds two components — the amount of data modelled by the snake and a penalty for the amount of data which the snake fails to model. This is to prevent, for example, an '$\mathcal{L}$' model being matched to a '$\mathcal{B}$'. If the unmodelled data were not taken into account, the '$\mathcal{L}$' model might appear to match the '$\mathcal{B}$' along its whole length. Since only a small part of each image is to be matched at a time, such a measure would be inappropriate here.

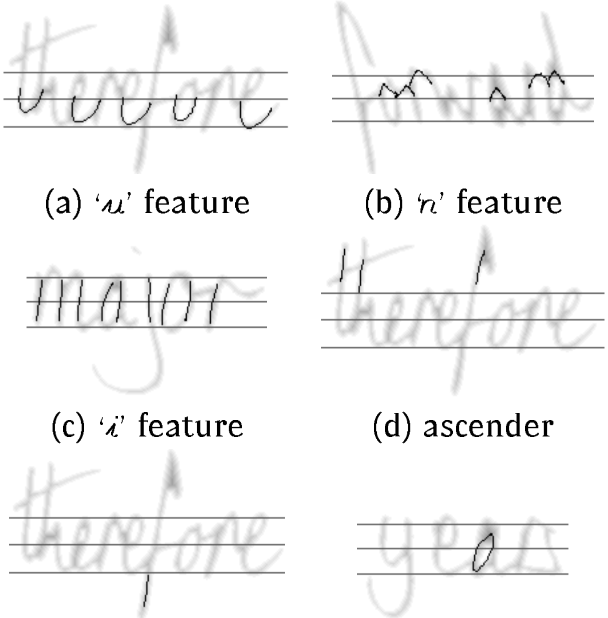
After each match, the shape and height of the snake is re-initialized to the mean and is displaced to the right by half its width, where the procedure is repeated until the whole word has been searched for that feature. In this way, each feature is matched across the whole of each word in the training set. It is possible that two successive placements of a snake will converge to the same feature, but multiple matches of this sort can be rejected on the basis of the x co-ordinates of the centroids being very close. Figure 6.3 shows all the matches for the features used in a variety of words.

For this application, the feature matches must be coded in the same preprocessing format described in the previous chapter. In this case, one more byte per snake model is allocated in each frame, and whenever a feature match is found this is recorded in the appropriate place in the frame which corresponds to the centroid of the matching model. In fact one model might span several frames, but the match is only recorded in the central frame.

Method	Error rate (%)	
	$\hat{\mu}$	$\hat{\sigma}$
With snake information	15.6	0.72
Without	18.2	1.60

Table 6.1: Error rates with and without including snake information.

Table 6.1 shows the improvement gained using snake features in addition to the basic skeleton coding of chapter 5. Adding the features into the



(a) ‘u’ feature (b) ‘n’ feature

(c) ‘i’ feature (d) ascender

(e) descender (f) ‘o’ feature

Figure 6.3: Different features found automatically in several words.

representation reduces the system error rate.

6.6 Discussion

The speed of the preprocessing algorithms has not been discussed so far, since the system described here has been designed for flexibility in comparing alternative algorithms rather than for maximum speed. In particular, a large number of intensive raster operations are carried out, which could be combined for greater speed. The speed of preprocessing in the current system is approximately one second per word. This could easily be considerably reduced by optimizing the program, and many of the operations should be easily parallelizable for an application requiring high speed.

It is difficult to have many features since with wide strokes, features tend to overlap in their roles and match the same parts of words. For example if one were to train a ‘c’ shape, it would be likely to match ‘i’ strokes too. For the same reason, it is found that maintaining a useful degree of flexibility in the constraints on an ‘o’ feature to make it fit a wide variety of ‘o’s means that it is also flexible enough to collapse and match ‘i’ strokes. Further individual constraints could be imposed, in the manner of Yuille *et al.* (1992), but would mean losing the simplicity of this system. If the matching and constraints could be made more reliable, it would be desirable to make a more complete set of snake features that would provide a complete cover of the word image, accounting for all the ink. Such a coding could be used as a complete representation of the word, much more compactly than the skeleton representation. Then, as with Edelman *et al.*'s system, recognition could be based on this representation alone.

Alternatively, character models could be developed from multiple snakes, giving matches for whole characters within a cursive string as those of Lanitis do for isolated capital letters. Hinton *et al.* (1992) also use spline models for entire characters. They model the ink of digit images as being generated by Gaussian sources distributed along a spline whose shape matches that of the character. They use probabilistic methods to define an energy measure which is minimized to adapt their models to the data. While the method is attractive, the authors admit that it is slow, and has not proven to match other approaches. Such whole character models could also be adapted to multiple positions in a cursive word to find reliable character matches, either for preliminary lexicon reduction as done by Cheriet and Suen (1993) or as an additional source of knowledge for any recognition system.

Chapter 7

Recognition methods

> *...in learning to read we were satisfied when we knew the letters of the alphabet, which are very few, in all their recurring sizes and combinations; not slighting them as unimportant whether they occupy a space large or small, but everywhere eager to make them out; and not thinking ourselves perfect in the art of reading until we recognize them wherever they are found.*
>
> Plato. The Republic.

The next stage in the process of deducing word identities from handwriting is to recognize what is represented by the frames of data created in the previous chapter. A variety of pattern recognition methods is available, and many have been used for handwriting recognition by other authors. Here three techniques are presented which calculate an estimate of the probability of any given frame being part of the representation of a given letter. How these probabilities are combined together to find the most likely word is explained in the next chapter; this chapter simply describes how these probability estimates can be derived.

There are several established methods of estimating a sequence of probabilities from a sequence of data. The speech recognition community has been finding solutions to this problem for some time, and their solutions are applicable to the problem of handwriting recognition. From the literature, three main methods emerge. Hidden Markov models have become the most widely used approach to modelling speech (*e.g.* Woodland *et al.* 1994). Feed-forward neural networks have been used by several authors, including Bourlard and Morgan (1993), and recurrent neural networks have also been successful in this field (Robinson 1994).

Other authors have used these approaches to on-line recognition, estimating probabilities for short sections of the input data. Among these are the hidden Markov models of Bellegarda *et al.* (1994), Nag *et al.* (1986) and Starner *et al.* (1994). The latter have obtained good results simply using a speech recognition system with handwritten data. Time-delay neural networks (TDNNs), a form of feed-forward network, are used by Schenkel *et al.* (1994) and Manke and Bodenhausen (1994).

These methods are also applicable to off-line handwriting, though there is no longer a readily apparent time-ordering of information. Instead the x-axis is divided up to give successive frames, processed left-to-right in the same way as scanning processes of reading. Caesar *et al.* (1993b) and Gilloux *et al.* (1993) use hidden Markov models for off-line recognition, though the latter use a sparse x-ordered series of large-scale features, unlike the representation with many parallel features per frame that is used here. Breuel (1994) uses a feed-forward network for classifying off-line handprinted strings.

In this work, all three of these methods have been investigated as methods of estimating the data likelihoods $P(\mathrm{x}_0^\tau|\Lambda_i)$ which are used to find word likelihoods in the next chapter. The remainder of this chapter describes each model, though intensive study was not made of TDNNs because they did not perform as well as the recurrent networks in early trials.

7.1 Recurrent networks

This section describes the recurrent error propagation network which has been used as one of the probability distribution estimators for the handwriting recognition system. Recurrent networks have been successfully applied to speech recognition (Robinson 1994) but have not previously been used for handwriting recognition, on-line or off-line. Here the time axis is replaced by the horizontal displacement through the word, frames representing not a speech signal over time, but successive vertical strips from a word, working left to right. A recurrent network is well suited to the recognition of patterns occurring in a time-series because the same processing is performed on each section of the input stream. Thus a letter 'a' can be recognized by the same process, wherever it occurs in a word. In addition, internal 'state' units are available to encode multi-frame context information so letters spread over several frames can be recognized.

Recurrent networks are a type of connectionist (often termed 'neural') network; that is to say they are composed of a large number of simple processing units with many interconnecting links. Each unit merely outputs a function of the weighted sum of its inputs, but the usefulness of such networks resides in the existence of training algorithms which can, by repeated presentation of training examples, adjust the weights to converge towards a desired function approximation. In this case the network is taught to recognize letters and the functions to be approximated are letter probability distributions $P(\Lambda_i|\mathrm{x}_0^t)$.

The recurrent network architecture used here is a single layer of standard perceptrons with nonlinear activation functions, as described by Rumelhart *et al.* (1986). The output o_i of a unit is a function of the inputs a_j and the network parameters, which are the weights of the links w_{ij} with a bias b_i:

$$o_i = f_i(\{\sigma_j\}), \tag{7.1}$$

$$\sigma_i = b_i + \sum a_j w_{ij}. \tag{7.2}$$

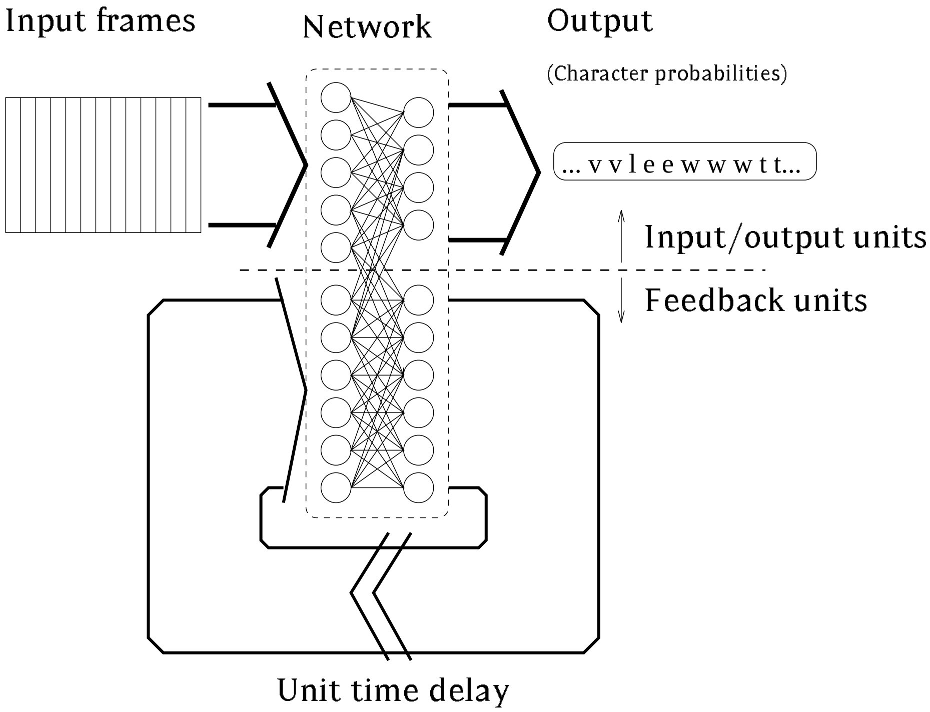


Figure 7.1: A schematic of the recurrent error propagation network. For clarity only a few of the units and links are shown.

The network is fully connected — that is, each input is connected to every output. However, some of the input units receive no external input and are connected one-to-one to corresponding output units through a unit time-delay (figure 7.1). The remaining input units accept a single frame of parametrized input and the remaining 26 output units estimate letter probabilities for the 26 character classes. The feedback units have a standard sigmoid activation function $f(\sigma_i) = (1 + e^{-\sigma_i})^{-1}$, but the character outputs have a 'softmax' activation function $f_i(\{\sigma_j\}) = \frac{e^{\sigma_i}}{\sum_j e^{\sigma_j}}$ (section 7.1.1).

During recognition ('forward propagation'), the first frame is presented at the input and the feedback units are initialized to activations of 0.5. The outputs are calculated from equations 7.1 and 7.2 and the output letter probabilities are read off from the outputs. In the next iteration, the outputs of the feedback units are copied to the feedback inputs, and the next frame presented to the inputs. Outputs are again calculated, and the cycle is repeated for each frame of input, with a probability distribution being generated for each frame.

It can be shown (Bourlard and Morgan 1993:p.118) that when the global minimum of the network is reached, assuming that the network has enough parameters and the training scheme can find the global minimum, the network outputs will approximate the posterior probabilities $P(\Lambda_i|\mathsf{x}_0^t)$. It will be seen later (chapter 8) how these probabilities can be combined to obtain word likelihood estimates in a Markov model framework. This framework makes use of the data likelihoods $P(\mathsf{x}_t|\Lambda_i)$ which can be approximated by assuming that the current character class is conditionally independent of the previous frames, given the current frame. (*i.e.* that $P(\Lambda_i|\mathsf{x}_t) \approx P(\Lambda_i|\mathsf{x}_0^t)$ which is a standard assumption made by researchers using hidden Markov models to model handwriting). Then the following equation can be used (Bourlard and Morgan 1993):

$$P(\mathsf{x}_t|\Lambda_i) \quad \propto \quad \frac{P(\Lambda_i|\mathsf{x}_t)}{P(\Lambda_i)}. \tag{7.3}$$

The assumptions used in making this approximation are explained further in the next chapter.

To allow the network to assimilate context information, several frames of data are passed through the network before the probabilities for the first frame are read off, previous output probabilities being discarded. This input/output latency is maintained throughout the input sequence, with extra, empty frames of inputs being presented at the end to give probability distributions for the last frames of true inputs. A latency of two frames has been found to be most satisfactory in experiments to date. A longer latency to incorporate whole letters in the context would be ideal, but learning long term dependencies in recurrent networks is not easy (Bengio *et al.* 1994b) because of the number of layers through which errors must be propagated, and a compromise is used.

7.1.1 Training

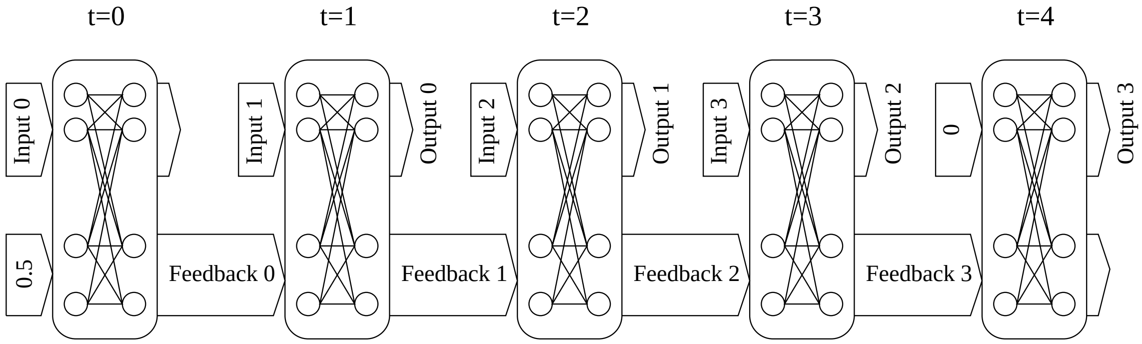


Figure 7.2: A network 'unfolded' for training after forward propagation on four frames of data. An input/output latency (section 7.1) of one frame is shown, so the first outputs are discarded and the last frame input is all zeros. The feedback units are initialized to 0.5 as described in section 7.1.4.

Training the network requires 'unfolding' it in time. During training on a word, the frames of data are input and propagated forward, as for recognition, but the inputs, outputs and feedback activations for each frame are stored. At the end of a word, errors in the network's output are propagated back using the generalized delta rule (Rumelhart *et al.* 1986), and changes to the network weights are calculated. The network at successive time steps is treated as adjacent layers of a multi-layer network (figure 7.2). This process is generally known as 'back-propagation through time'. After processing $(\tau+1)$ frames of data with an input/output latency, the network is equivalent to a $(\tau + 1 + \mathrm{latency})$ layer network. Readers are referred to Rumelhart *et al.* (1986) and Robinson (1994) for a detailed description of the basic training procedure.

It is widely recognized that this back-propagation algorithm can be improved in a variety of ways, to speed convergence and to make convergence to a good local minimum more likely. In addition to the incorporation of a momentum term in the weight update formulae, two such improvements have been used in this work, namely Jacobs' delta bar-delta update rule (Jacobs 1988) and Bridle's (1990) *softmax*. The former provides for individual learning rates for each weight which adapt according to the signs of successive weight changes. The latter provides a different transfer function on the output units of the network, ensuring that the outputs are between 0 and 1 and sum to 1 (as is desirable since they are treated as probabilities). This also trains the network according to a relative entropy (between the output and target probability distributions) error criterion instead of the least-squares error measure more commonly used in back-propagation networks. Because of difficulties in training stability, modifications to the delta bar-delta rule suggested by Robinson and Fallside (1991) were incorporated and gave much improved convergence. These changes use multiplicative learning rate changes and prevent the learning rates from deviating too far from the mean. For this work an additional measure was taken, of zeroing momentum terms when the mean output/target relative entropy over the training set increased.

Training times for neural networks can be very long. In this instance training takes several days on a fast computer. (More than 3 days of CPU time for an 80-unit network.) In addition to the methods described above, a number of other ways to improve training speed have been explored. The most significant is to choose an efficient training schedule. This specifies how many patterns should be presented to the network before each weight update. Initially the weight updates from different patterns will tend to be in roughly the same direction, as the network moves to an appropriate region in weight space. Later the updates from different patterns will be in different directions, and the updates need to be smoothed to find the best displacement for the whole training set. Thus, at the start of training, weights can be updated on a per-pattern basis ('on-line' or 'stochastic' training), but for fine-tuning near the end of training, weight updates should be averaged over a larger set of data.

In this application, a number of simple schedules have been tested, with the best being to start by updating on a small number of words, typically a batch of four words or about 80 frames. Then, whenever the mean relative entropy increases, the batch size is doubled, with a corresponding cut in the step size parameter. This continues up to a limit of 1024 words per batch (roughly a third of the training set). The momentum factor also controls this smoothing, but no schedule based on changing this parameter was found to be as good. This is, however, the method preferred by Robinson (1994) who increases the momentum parameter (the degree of smoothing) over time. Bourlard and Morgan (1993) also prefer on-line training. The choice is perhaps largely to do with the size of the training set. Although the handwriting database was large (56,000 frames), it was feasible to calculate a weight update based on a third of the training set, which is impossible for the much larger speech databases. The presentation of all the training examples to the network is called an epoch. The number of weight updates per epoch decreases to three during training.

The Quickprop weight update scheme (Fahlman 1988) was also tried. This approximates the error surface as a quadratic, with diagonal covariance, and uses quadratic interpolation to predict the minimum in each dimension. This is effective for small dataset problems, where weight updates are always based on the whole dataset, so a good estimate of the true error surface can be obtained. The method did not perform well with the on-line training used here, as the shape of the error surface is different for each batch of data.

7.1.2 Network targets

For training, a target value must be given, against which the network output can be compared in order to compute the error in the outputs and the weight updates. The target value is given in the form of a label for each frame of the training data, indicating the correct class — the class for which the network output should be one, all others being zero. With the data collected here it is a relatively simple matter to associate the word label with each word image (section 4.2). However, the labelling of individual frames with the corresponding class is not as easy, and some thought must be given to this problem. Unlike the segmentation problem of most handwriting systems (section 2.3.2), this is not the problem of determining where the test word image must be split to separate its component letters, but that of assigning a letter label to each of the frames of a training word. This is only for training purposes, and need not be carried out on test words. In new data, this frame/letter correspondence is not trivially determined; it can only be truly carried out by accurate recognition — a catch 22 situation. For some problems, such as speech recognition, people have resorted to hand-labelling data to give an initial training set. This has been avoided here by using a 'bootstrap' scheme which derives an approximate segmentation from a very naive technique. This segmentation is good enough to train the network to a point where its own segmentations are more accurate. Hand segmentation would be more accurate still, so might give improved results, but would require a large amount of tedious work, for little or no gain.

The scheme used initially is an 'equal length' scheme, where each letter in any word is assumed (though this is clearly inaccurate) to occupy the same number of frames of input. Thus, in an n letter word which takes $\tau+1$ frames, the first $\frac{\tau+1}{n}$ frames are labelled with the first letter of the word. In '*noun*', for example, one quarter of the frames are assumed to belong to each letter.

This can be made slightly more accurate by recognizing that '*w*' and '*m*' are longer than other letters and '*i*' and '*ℓ*' are shorter. Letters in these classes are given relative lengths of 3 and 1 respectively, compared to 2 for other letters. The frames are then labelled in proportion to the relative lengths of the letters in the word. Thus, in the word '*wig*', the first half of the frames would be considered to represent the '*w*', the next sixth the '*i*' and the remaining third the '*g*'. It is this segmentation that gives the targets which the recurrent network is trained to reproduce. The targets are set to one for the correct class and zero for all other classes.

These targets are only used for preliminary training. Re-estimated targets are used to achieve greater performance. The re-estimation process will be described in chapter 8.

7.1.3 Generalization

A problem with network training is to obtain the optimum solution to the trade-off between training and generalization. This well-known problem can perhaps best be seen by considering the problem of curve-fitting to n data points. An $(n-1)$th order polynomial can be found to perfectly interpolate any such set, but if there is any noise in the data, the values on the curve between will correspond badly to the values of any subsequently observed data-points. The curve is over-fitted, and generalization is poor. Similarly, in training a recurrent network, given enough time and computing power it should be possible to train a large enough network to match the desired targets arbitrarily closely. However, such a network will give poor generalization and make poor predictions for inputs other than those included in the training set.

One way of maintaining good generalization is to make sure that the network size is right for the size of the problem. In this case the number of parameters is kept down and the order of the model is chosen to be appropriate to the task to be solved (*e.g.* fitting a straight line to the n data points when a linear effect is being modelled). For complex problems the size of the network for optimum generalization is difficult to determine, though individual authors have found rules-of-thumb relating the number of training examples to the number of free parameters to be trained (Bourlard and Morgan 1993:p.234). In practice, for a specific problem, trial-and-error is often used. Methods whereby the network is grown or pruned to the right size have also been developed.

An alternative is to use a network known to be at least large enough for the problem, but to prevent over-training within that network. Possible techniques include weight decay and adding noise to weights, but the method used here is *early-stopping* which can be implemented without changing the training procedure and has the advantage of limiting training according to the same performance criterion (word error rate) as will ultimately be used for testing the network. If a network is trained on a dataset, it is found that, during training, the error rate when tested on an independent validation set will fall as a solution is learnt, and then begin to rise as generalization is impaired by over-training. If training is stopped at the minimum of the validation error, optimum recognition on an independent test set will be obtained. This method has been widely used in the neural-network community, and is particularly appropriate for large dataset tasks. Bourlard and Morgan (1993) have used a similar method for large-vocabulary speech recognition.

To determine the best time to stop training, the training set is partitioned into separate training and validation sets. After training the network for a short time, the network's performance is tested on the validation set. This train and validate cycle is repeated every epoch until the error rate on the validation set starts to increase, indicating that the network is starting to become over-trained. The *stopping criterion* is a heuristic based on the observation of validation word error rate over time. The criterion used here is to stop when the validation error rate is above the minimum observed during training for more than twelve epochs, or the same without a decrease in the mean relative entropy. After finishing training, the network with the lowest error rate is reloaded, and tested on the training set which consists of data not previously presented to the network.

Number of units	Error rate (%)		Epochs	Time per epoch (s)
	Fixed target	Retraining		
0	49.0	40.9	75	1230
2	41.1	34.0	171	1250
4	29.3	26.2	141	1250
10	23.1	22.3	133	1280
20	21.6	19.1	132	1270
40	21.4	16.3	181	1450
80	16.9	15.6	132	2100
160	14.8	12.2	115	4900
320	13.5	9.6	116	14000

Table 7.1: Error rates for networks with different numbers of feedback units.

Table 7.1 and figure 7.3 show the error rates for units with different numbers of hidden units. Results are quoted before and after re-training with re-estimated targets, a process explained in section 8.3. Performance can be seen to improve steadily as the number of units increases. Thus it can

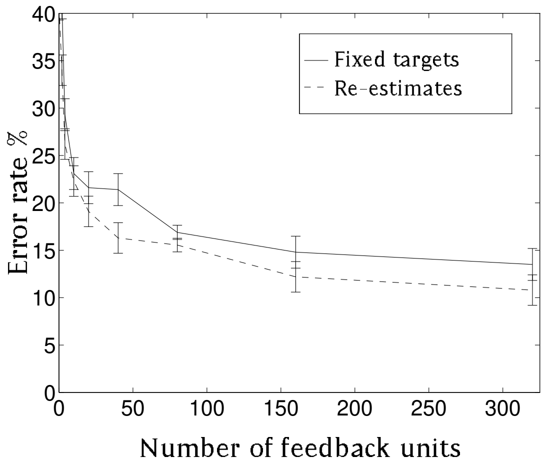


Figure 7.3: Test error rates against number of feedback units, showing error bars (one standard deviation). The lower curve shows the error after retraining with the Baum-Welch re-alignment.

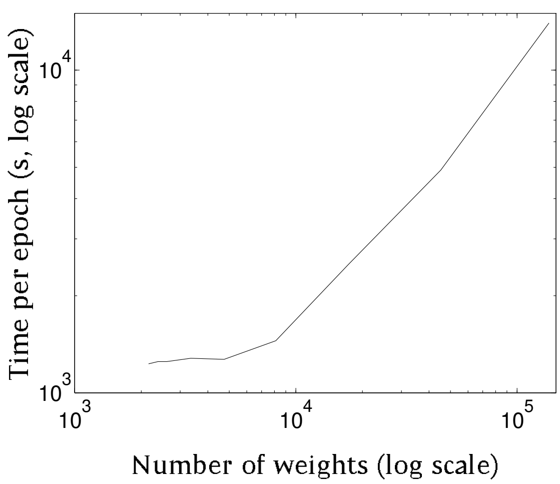


Figure 7.4: Approximate average training time against number of network weights (log-log scale).

be seen that early stopping ensures that generalization does not suffer when the network size is increased. In fact the increased capacity of more feedback units allows the network to perform better. Because of the increased training time associated with larger networks, no network above 320 feedback units has been trained, though it is likely that the recognition rate would be still higher. The time estimates are seen to come from a constant term (because of overheads and of cross-validation testing) plus a term proportional to the number of weights (proportional to the square of the number of feedback units), which becomes significant only with 40 or more feedback units (figure 7.4).

It will be seen from the high values for the standard errors of the mean error rates quoted that the final solutions obtained are dependent on the initial conditions (the random weights given to the network prior to training). It can easily be seen that there are many global minima (any permutation of the feedback units gives an identical solution) and it is not surprising that a different solution is found each time, the local minima found in weight space corresponding to networks giving different performances. This is a problem that might be solved with more data or by better training, for instance by finding a better training schedule.

In summary, while a satisfactory method of training has been found, which reaches good solutions, there is scope for speed improvement. This scope exists both in finding better training schedules within the space of solutions tried already, and in trying more complex update techniques. The ensemble of training methods currently used resembles those arrived at by Bourlard

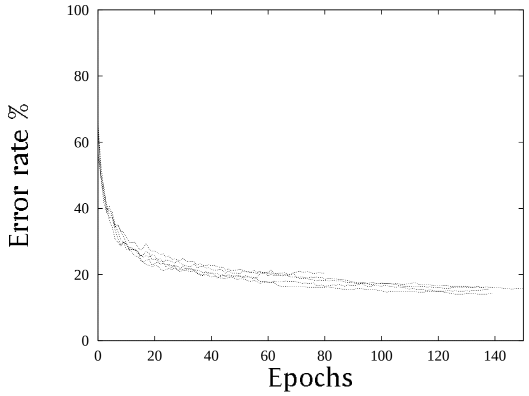


Figure 7.5: Validation error rate against number of training epochs for five networks under the same conditions, but different initial weights.

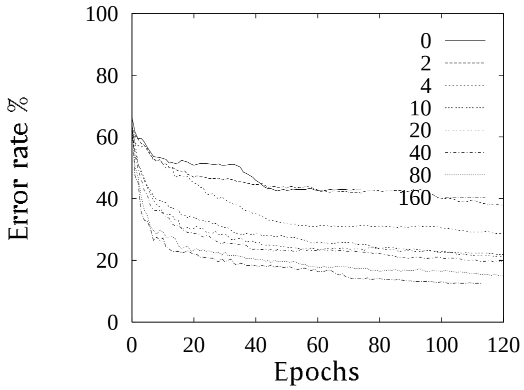


Figure 7.6: Percentage recognition error rate versus number of training epochs for networks with different numbers of feedback units.

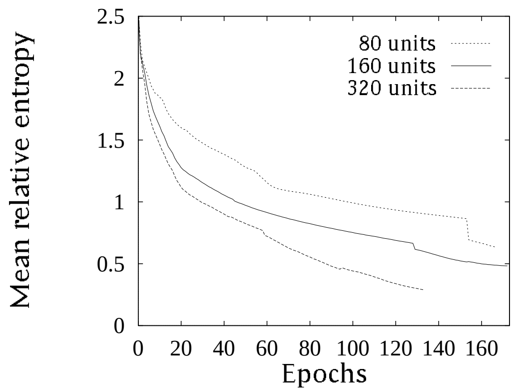


Figure 7.7: Average relative entropy of the training set outputs and targets against number of training epochs.

and Morgan (1993) and Robinson (1994), but differs in a number of details.

7.1.4 Understanding the network

One of the great problems with neural networks in general, and recurrent networks in particular, has been the lack of understanding of how the networks operate. It is not always well understood to which problems they are best suited, or how best to use them on problems to which they are appropriate. Neural networks have been studied in greater depth in recent years, though the high dimensionality of interesting problems makes analysis difficult. While 'gradient descent on the error surface' is often talked about, it is only for a trivial neural network with two weights that this surface can be plotted, and for higher dimensions it becomes difficult to calculate, let alone visualize. Recurrent networks are harder still to understand, since the dimensionality is much higher — outputs are dependent on the inputs, not only of the current frame (and for the handwriting recognition networks discussed here, there are about 80 inputs), but also of all the preceding frames.

Robinson (1989) and Pearlmutter (1990) have previously studied the operation of recurrent networks under certain conditions. In order to discover how the recurrent network is operating in this task, a graphical interface to the network has been constructed, enabling inputs, activations and weights to be examined. The remainder of this section discusses some of the understanding that has been reached as to the internal representation of data in the network.

A first experiment to demonstrate the network's operation is to pass a single word through the net and to observe the outputs. Figure 7.8 shows an example of the word '*fortunate*' being presented to the network. The horizontal traces show the activations of the output units against time. Since the outputs of the network are constrained by the softmax function to sum to one, most of the outputs are seen to be always close to zero, with only one or two rising to a significant value at any time. The activities during the first two frames (before the first vertical line) are always ignored in the training and testing of the network because of the input/output latency. Subsequent frames see the probabilities for '`f`', '`o`', '`r`' and so on increasing, with a small amount of activity in other letters. Note that the valley between the '*u*' and '*n*' is confused with a '`v`', and that the '*t*' is partially confused with an '`l`', but these confusions are eliminated by the duration modelling (discussed in chapter 8.2) and the requirement that the word should be in the lexicon. The vertical lines represent the letter boundaries of the forced alignment (section 8.3) from the Viterbi decoder.

Consider now the weights within the network. Initially networks were randomly initialized with weights of zero mean and small variance. However, after training, all the weights from any feedback unit to the same unit for the next time-step were found to be positive, with strong connections. (For a typical network they have mean 2.6 and standard deviation 0.6.) Connections

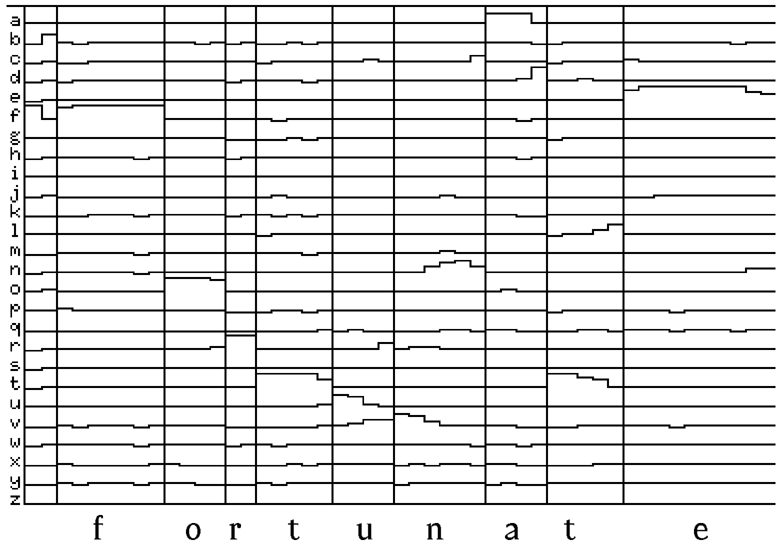


Figure 7.8: The system recognizing the word 'fortunate'. The activations of the output units are plotted against the number of frames processed. Class boundaries found by Viterbi forced alignment are shown with the associated class labels (section 8.3).

to other feedback units vary greatly, with a slightly negative mean (*e.g.* mean -0.4, standard deviation 1.2). This indicates that the network is learning the intuitive mechanism of having the feedback units preserve their state, except when influenced by inputs and other feedback units. Since the network solutions seem to favour this state-preservation, better solutions might be found more quickly by choosing an initial weight distribution which preserves state. This can be calculated as follows.

If the feedback units are assumed to have a mean activation $a_j = 0.5$ (corresponding to a weighted sum of inputs $\sigma_i = 0$, since the sigmoid activation function, for which $f(0) = 0.5$, is used for the feedback units), then

$$\sigma_i = b_i + \sum_j a_j w_{ij} \approx b_i + 0.5 w_{ii}$$

if the other weights have a zero mean. For steady-state conditions, $\sigma_i = 0$, so $b_i = -0.5 w_{ii}$. Now, for an activation $a_i = 0.5 + \delta a_i$,

$$\sigma_i = b_i + a_i w_{ii}.$$

Since $a_i = f(\sigma_i)$, for small δa_i:

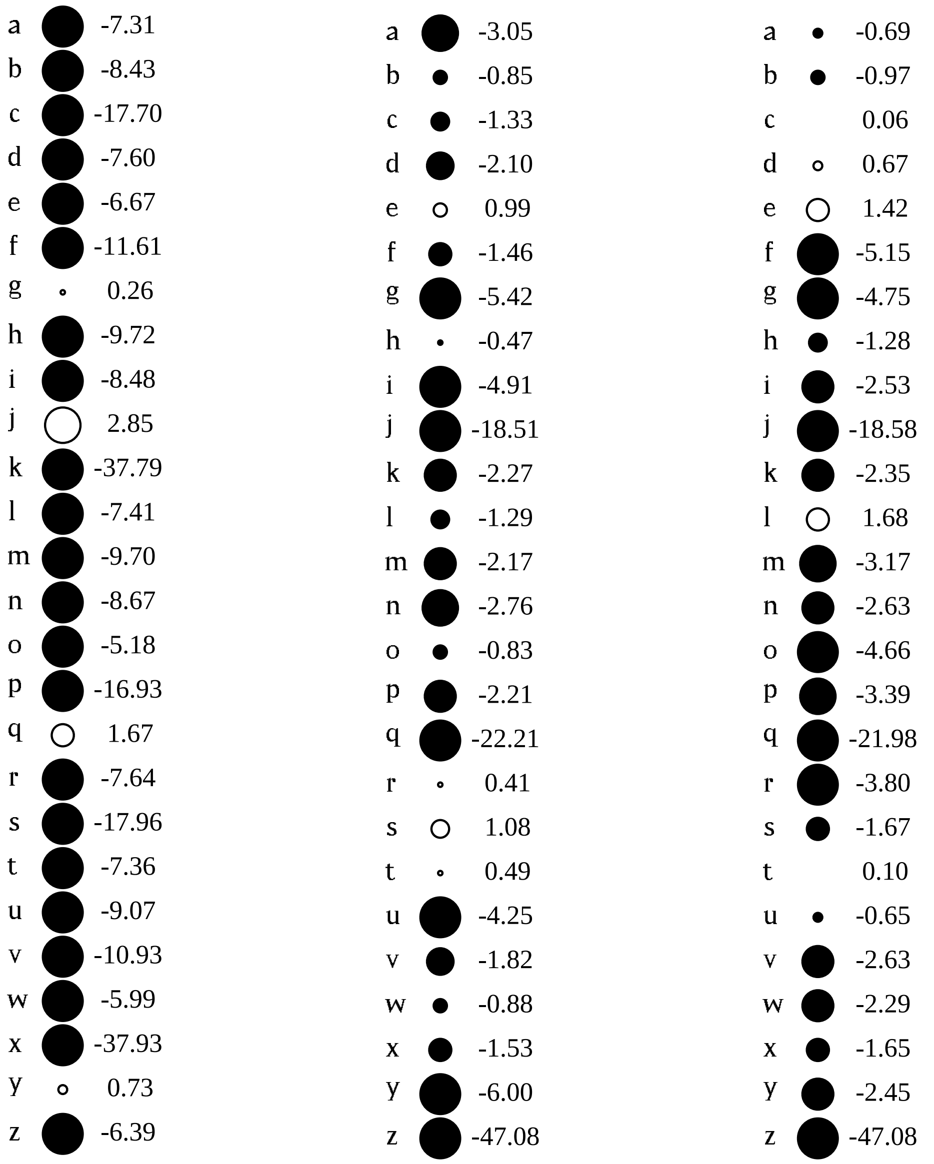
$$\delta a_i = f(a_i) - 0.5 \approx \delta a_i w_{ii} f'(0).$$

For the sigmoid, $f'(0) = 0.25$ so the state is stable when $w_{ii} = 4, b_i = -2$. Priming the network connections to these values gives faster training and a greater recognition accuracy after training. The final values of these links are much higher (mean 4.6, standard deviation 0.5), revealing that priming the network weights puts the network into useful areas of weight space that were not explored while training un-primed networks. It also confirms the usefulness of feedback connections which preserve the feedback units' state.

Examining other connections within the network, it is seen that very few weights from input to output units are positive. This is to be expected, since a single frame of input is itself ambiguous and does not give a strong indication as to the character of the frame two time-steps previously (which direct links would indicate, since outputs refer to the frames input two time-steps previously). One notable exception to this is the letter 'q' which has strong links from the units representing lines in the lowest part of a word. This is because 'q' is written with a descender to the right of (delayed with respect to) the body of the letter. Figure 7.9a shows the links from one input unit in the lowest part of the word. All the letters with descenders to the right are activated by this input unit, while other outputs are inhibited. Because some information is transferred by the direct input-to-output connections, it has been found that a network with these connections performs better than one which does not.

In a recurrent network, the most important aspect to understand is the feedback units. In this handwriting problem, they need to represent the features presented at the input during the last few time-steps so that a classification of the current frame can be made according to the context, since an individual frame is ambiguous. However, the way this information is encoded is not readily apparent. As was noted earlier, each unit has a strong feedback connection to itself to maintain the state over time. Otherwise, few links from the feedback units are found to be strongly positive.

If a network with very few units is examined, it is easier to understand the role of the feedback units. Figure 7.9 b,c shows the connections from the only two feedback units in a small network to the outputs. It is noticeable that the connections reflect the frequencies of the letters in the training set. Very rare letters such as 'q' and 'z' have very strong negative connections. Because of their rarity, these letters generate very little error signal, so it is inappropriate for the scarce resources to be used modelling these letters. On the other hand, the letters 'edlrst' have positive connections from the feedback units since these are common. The two most common letters ('et') are modelled by both feedback units. Figure 7.10 shows the output probabilities for the word '*speaking*', which shows the effect of this. The letters 'se' are well-defined, though not as clearly as with the 80 unit network (figure 7.8). There are noticeable peaks in the output traces of these two letters, but the other letters show no marked deviation from zero. It can also be seen that through the direct input-output connections, the descender is identified as belonging to either a 'g' or a 'y', though the network does not have the modelling capacity to distinguish the two. Despite the uncertainty of the network during most of the frames, the correct word is still chosen from the lexicon.



(a) (b) (c)

Figure 7.9: Connection strengths to the outputs in a recurrent network. Circles are white for positive weights, black for negative. Larger magnitudes are represented by larger radii. (a) shows the connections from a descender input unit in a 60-unit network. (b) and (c) are the connections from the only two feedback units in a small network.

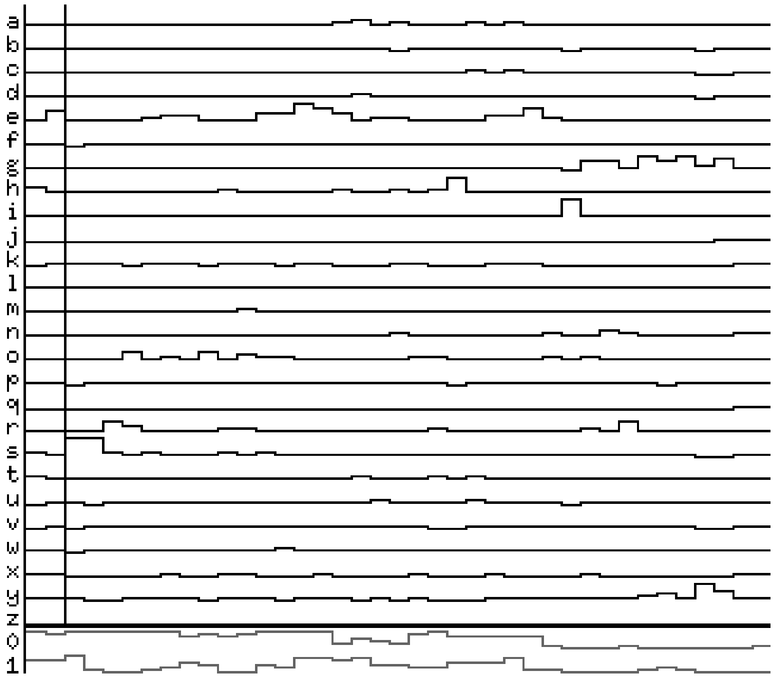


Figure 7.10: The two unit network recognizing the word 'speak-ing'. No class boundaries are shown because the 2-unit network re-estimates are inaccurate.

The role of the feedback units can also be verified by examining their activations when presented with word data. The units are generally seen to have high activations when the relevant letters are present at the input, and low otherwise, though the correlation is far from perfect. In figure 7.10 the activation of feedback unit zero is high during the 's' and 'e', though unit 1 does not go high during the 'e' as might be expected. The biases to the output units are found to reflect the variation in class frequencies, but this correlation is not as strong as suggested by the experience of Bourlard and Morgan (1993:p.127). Examining a network with four units, one of the feedback units is found to have negative connections to all the outputs except 'i', and to receive strong positive input from the input unit representing the dot feature. This representation allows the network to remember the presence of an `i` dot during the latency period.

Another way of investigating the network's behaviour under controlled conditions is to feed a null input into the network. A data file where all frames are entirely zero is constructed, and presented to a trained network. The unforced output for a sample network with 60 feedback units is shown in figure 7.11. It can be seen that the output and feedback units go through several cycles before reaching a steady state with all the units in saturation. Examining a network with but one hidden unit shows that the network dynamics are, understandably, simpler. The outputs are all monotonic, and

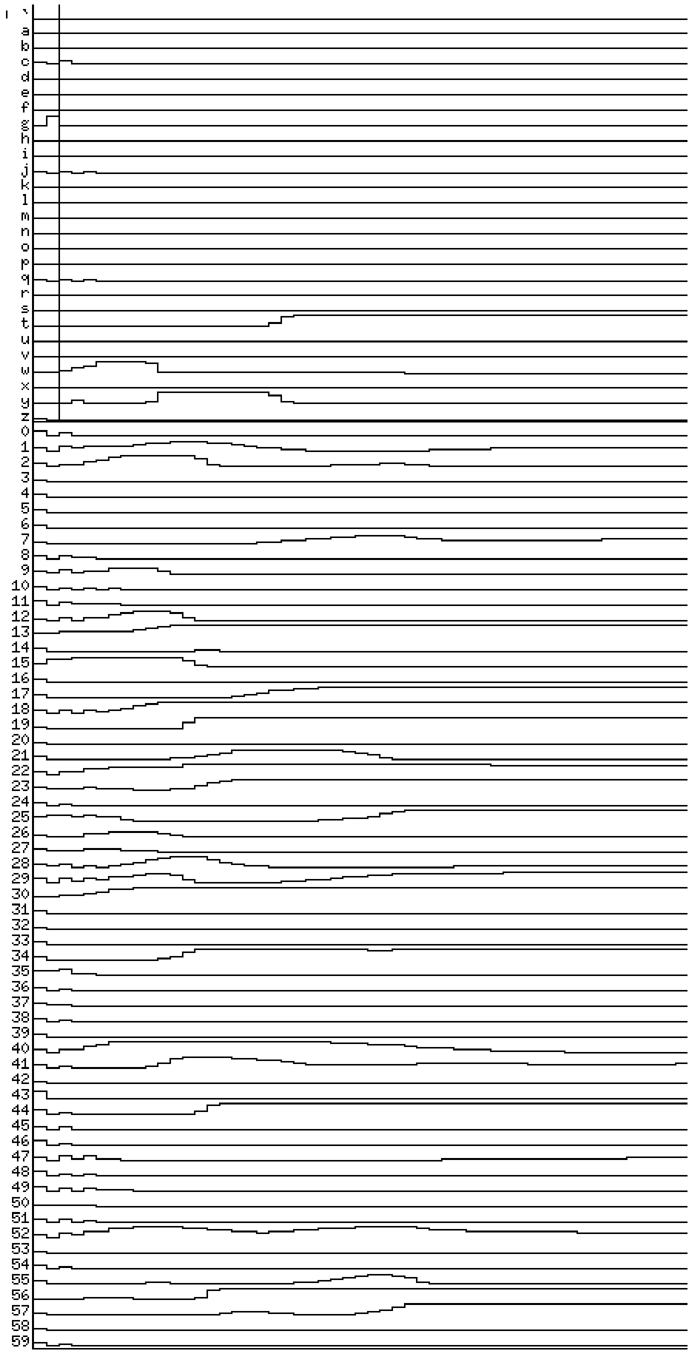


Figure 7.11: The network outputs for unforced inputs.

reach a steady state after a few frames. As networks with more and more feedback units are tested, the behaviour becomes more complex, until with a 160-unit network, no steady state is achieved after 130 frames. The network appears to be entering limit cycles, exhibiting dynamic behaviour with no active inputs.

7.2 Time-delay neural networks

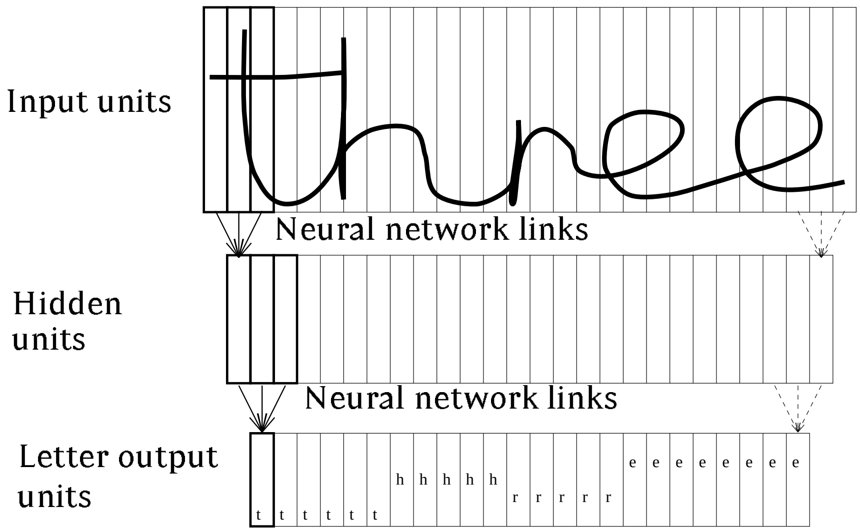


Figure 7.12: A schematic of the time delay neural network, showing a single hidden layer.

Time-delay neural networks (TDNNs) are a method of applying a simple forward-propagation neural network to a sequence of frames of data to arrive at a sequence of probability estimates. A TDNN is represented in figure 7.12. A layer of perceptrons, as used in the recurrent network, takes a small group of input frames (three in the diagram) and calculates the activations of a corresponding hidden frame with equations 7.1 and 7.2. The receptive field of the perceptrons is then shifted to the right, and another hidden frame calculated. This process can be repeated for all the frames. At the same time, a second layer of perceptron units takes a group of hidden frames and for each of these calculates an output probability distribution with softmax units, just as for the recurrent network. Thus, for each input frame a corresponding output distribution is calculated. Since the same perceptrons operate on each section of the input, the TDNN is good at position-invariant pattern recognition. It has a fixed window of context which is the number of input frames on which each output depends. The length of this window (five frames in the diagram) is determined by the receptive fields of the perceptrons. This makes the TDNN good for recognition of patterns with limited context, when the extent of this context is known, but longer-term dependencies can not be learnt. Because of the rigid hierarchy of the input and hidden units, dependencies of variable length are hard to learn. Each perceptron can only associate features which are a fixed distance apart. The recurrent network, on the other hand, stores all context in the hidden units which are available at every time step. If the context is of variable length, the feedback units will vary slowly and the correlation between two features can be detected at an arbitrary delay.

It is believed to be for this reason that TDNNs did not perform well on this handwriting recognition task. They were also found to be unwieldy since the architecture of a TDNN is specified by a large number of parameters. The number of hidden layers must be specified, as well as the number of units in each and the size of each receptive field. A further parameter that can be controlled is the number of frames shifted between successive operations of each of the sets of perceptrons. Finding a good set of values for all these parameters requires a long search, whereas the recurrent network has a single such parameter — the number of feedback units (section 7.1.3). Because of this poor initial performance, TDNNs were not investigated further, and no results are presented for them here.

7.3 Discrete probability estimation

This section describes the third technique investigated for probability estimation. This involves computing a number of integer-valued indices from each frame and using these to look up probability values in pre-computed tables. When combined with the hidden Markov models (HMMs) described in the next chapter, the system is a conventional discrete HMM since this is the usual method of calculating probabilities for a discrete HMM. By contrast, the recurrent network and HMM together would be termed a hybrid system.

The probabilities that must be estimated are the likelihoods $P(\mathsf{x}_t|\Lambda_i)$ — the probability of a frame of data being generated, given the identity of the letter. Since the data are represented as about 80 features, each coded as a byte (256 possible values), to store the probability of each possible co-occurrence would require $256^{80} \times 26$ probabilities to be stored and estimated. This is clearly computationally impractical and would require infeasible quantities of data to give estimates of the probabilities. Parametric distributions could be used, which calculate these probabilities as functions of a smaller number of parameters, but the numbers are still impractical, and the re-estimation more difficult. Two methods are used to simplify the estimation.

7.3.1 A simple system

First, since the units mostly record simply the presence or absence of a feature, even for the skeleton where the coarse coding does give values between 0 and 1, the most important information is whether a line segment is present or not. The inputs are thus re-quantized to be binary-valued (or some other number of values much less than 256). Secondly, the features are assumed to be independent. Thus the probability of the co-occurrence of all the features in a frame is simply the product of the occurrence of the individual features.

$$P(\mathsf{x}_t|\Lambda_i) \approx \prod_j P((\mathsf{x}_t)_j|\Lambda_i) \tag{7.4}$$

Now only $80 \times 2 \times 26$ probabilities need to be stored or, since the pairs must sum to one, only 80×26.

The assumption of independence in the occurrence of features in the input is clearly inaccurate since, for example, the occurrence of a vertical stroke in one box is highly correlated with the occurrence of a vertical stroke in the box below. In practice, the assumption is far too strong, and the performance of the HMM system is much worse than that of the recurrent network (an error rate greater than 50%). The following section describes a system which obviates the independence assumption, and gives better recognition results.

7.3.2 Vector quantization

Vector quantization (VQ) is a method of characterizing each frame by a single number, or code $c(\mathsf{x}_t)$. The quantization process is designed so that similar frames are all coded as the same number. Then, instead of estimating the probability of all the features in a frame given the character class, it is only the probability of the code given the character class that must be estimated: $P(\mathsf{x}_t|\Lambda_i) \approx P(c(\mathsf{x}_t)|\Lambda_i)$.

In vector quantization, each frame is considered as a vector in a metric space with as many dimensions as there are elements in the frame. Quantization determines a *codebook* of code vectors c_i in this space. Each frame x_t is then coded according to the nearest code vector: $c(\mathsf{x}_t) = \mathrm{argmin}_i \|\mathsf{c}_i - \mathsf{x}_t\|^2$. In the subsequent training, it is these codes that are the features, and it is the probability of a code being part of a given letter that must be estimated.

Before being able to estimate the probabilities, the code vectors must be determined. To be representative, they must be well distributed in the space of vectors actually produced by the preprocessing system, and each should represent a typical group of vectors which can be considered to be similar. The groups of equivalent vectors are assumed to be those close to one another in the metric space, and the code vectors are determined by a clustering algorithm which finds these clusters in the training vectors. Each code vector is then the centroid of a cluster of training vectors. A number of algorithms exist for carrying out this clustering, and a number are reviewed by Gray

(1984). The method used here is by Linde *et al.* (1980). It produces a set of coding vectors given a training set of vectors output by the preprocessor — the same training set which, when coded by the quantizer is used to estimate the code probabilities which are stored in the tables. In brief, the algorithm works in the following manner:

1. Seed the quantizer with one classification vector — the centroid of the training set.

2. Split each classification vector to give two, perturbing each slightly. This has the effect of dividing the original cluster with a hyperplane perpendicularly bisecting the line joining the two new centres. If the perturbation is sufficiently small, the other class allocations will be unaffected. Perturbation along the line joining the centroid to the origin was found to work just as quickly as perturbation along the axis with the greatest in-cluster variance.

3. Classify each of the training vectors by assigning it to the nearest classification vector.

4. Move each classification vector to the centroid of the training vectors which were nearest to it.

5. Go to step 3 unless the classifications are the same as in the last iteration.

6. Go to step 2 until the desired number of classification vectors is obtained.

For step 3, a distance metric must be specified. As a first approximation the Euclidean distance was used. This is reasonable since all the inputs are constrained to fall in the same $[0, 1]$ interval. This distance will reduce to the Hamming distance when all the vectors are binary valued. An alternative which has also been tested is the Mahalanobis distance (already seen in chapter 6), where the distance between two points x and y is given by:

$$\|x - y\|^2 = (x - y)^T \Sigma^{-1} (x - y), \tag{7.5}$$

where Σ is the covariance matrix of the training vectors.

The Mahalanobis distance is derived from the assumption that the distribution of vectors is elliptically Gaussian, which is clearly not true here. Nevertheless, it allows correlations between vector elements to be taken into account when finding the distance between two vectors. A better metric, based on knowledge of the origin of the data and the fact that the data are largely binary-valued could probably be found. This would model the correlations between features better and result in more representative clusters.

Better results might be obtained from quantizing with such clusters. However, hand-crafting a metric would be a complex procedure, and the Mahalanobis metric is the most complex metric investigated here.

A further issue in designing a VQ-HMM system, is the optimum number of clusters to choose. This involves striking a balance between an over-trained system which does not generalize well and one which has a low discriminative power. Results are given for a variety of numbers of clusters and the optimum value chosen.

7.3.3 Training

Discrete probability estimation requires the tables of probabilities to be filled with the estimate of $P(c_i|\Lambda_j)$ for each of the codes c_i and letters Λ_j. After vector quantizing the corpus and labelling each frame with the automatic segmentation procedure, the number of times code c_i is part of letter Λ_j is counted over the whole training corpus. Dividing by the number of frames representing Λ_j gives an estimate of the emission probabilities $P(c_i|\Lambda_j)$. By re-aligning with the Baum-Welch procedure of chapter 8, the probabilities can be re-estimated and the recognition rate improved slightly. For this HMM framework, the Baum-Welch procedure is very fast, since the maximization step of the Expectation-Maximization algorithm, of which this is an example, consists only of taking the frequency counts rather than doing gradient descent as with the recurrent networks — a notoriously time-consuming problem.

Recognition rates for the HMM system with Euclidean and Mahalanobis distances are shown in tables 7.2, 7.4 and 7.3. The numbers of clusters are powers of two in the first table, since at each iteration of the splitting algorithm the number of clusters is doubled. In the other tables, the number of clusters is lower because during the splitting some clusters have been found to be empty and the corresponding centroids discarded.

Clusters	Error rate (%)
256	24.1
512	20.6
1024	22.9
2048	28.1
4096	93.5

Table 7.2: Error rates for the hidden Markov model system with Euclidean distance vector quantization.

Clusters	Error rate (%)
256	25.9
509	22.0
1006	21.0
1979	23.9
3796	40.6

Table 7.3: Error rates for the hidden Markov model system using diagonal covariance Mahalanobis distance vector quantization.

Clusters	Error rate (%)
254	26.6
505	24.5
1001	22.4

Table 7.4: Error rates for the hidden Markov model system using Mahalanobis distance vector quantization.

From tables 7.2 and 7.3 it can be seen that increasing the number of clusters up to 512 increases the discriminative performance of the system, so the error rate falls. Beyond this, the generalization fails and performance falls off rapidly. By 4000 clusters the system fails completely. The diagonal Mahalanobis distance method gives slightly, but not significantly worse results, and the full-covariance Mahalanobis distance gives worse results again. The full-covariance matrix codebook is prohibitively expensive, computationally, to work out for larger numbers of centroids. The lack of improvement is due to the unusual distribution of the inputs which are nearly always zero, and often one. The Mahalanobis distance is intended for modelling distributions which are Gaussian distributed, an assumption not true here.

7.3.4 Discussion

The best of these discrete probability estimators has 512×26 parameters — the same as a recurrent network with 64 feedback units. A network with 60 feedback units achieves a 14.5% error rate. It can thus be seen that the pure HMM system does not perform as well as the hybrid recurrent network/HMM system. While this shows that the recurrent network is a more practical solution to the problem of modelling the graphic data, it does not argue absolutely against the use of hidden Markov models. While much of the work of this thesis is equally applicable to both systems, more time has been spent perfecting the recurrent network system than investigating improvements in the pure HMM approach. It is undoubtedly true that with further investigation the HMM system could be improved. There is a set of standard techniques that could be taken from speech HMMs and applied to this system, which could reasonably be expected to give better performance. These include giving different states within a letter separate probability distributions, and producing context-dependent models which would be able to model the coarticulation between adjacent letters — most particularly the ligatures which vary with different contexts. However, similar methods might also be applied to the hybrid system.

7.4 Summary

This chapter has presented three methods of probability estimation which can be used for the problem of off-line handwriting recognition, and has discussed some of the issues involved in using them. The training of the models has also been discussed and recognition results presented. The recurrent networks were found to perform better than both the discrete hidden Markov model and the time delay neural network. Training the recurrent networks is very time-consuming, but a number of methods have been used which reduce the training time, including weight initialization, Jacobs' weight-update scheme, and a training schedule which changes the size of weight-update batches during training.

The next chapter completes the description of the system by explaining how the probability estimates are used for word recognition.

Chapter 8

Hidden Markov modelling

The reading is right which requires so many words to prove it wrong.
Samuel Johnson.

The previous chapter described methods of modelling the graphical data of a handwritten word. Each method gave an estimate of the likelihood $P(\mathsf{x}_t|\Lambda_i)$ for each frame of input x_t and for each character class Λ_i (of 26). This chapter deals with the process of deriving the best word choice from a sequence of these frame probability distributions by the use of hidden Markov models. The methods described here apply equally to the pure discrete HMM and to the recurrent network hybrid system, but tests are described for the hybrid system since it was found to be more effective. For the time being, the system is assumed to have a known vocabulary and it is assumed that any word presented to it will be in that vocabulary.

8.1 A basic hidden Markov model

Because the data are noisy or ambiguous, the output of the whole system should be a probability distribution across the words in the lexicon, being the probability for any word that it was the one originally written. Normally the probability should be close to one for one word, and close to zero for the others, but where there is ambiguity, error or poor data, the distribution might be more uniform. For instance, for the ambiguous word of figure 3.1b, high, roughly equal probabilities would be expected for the three words '`clump`', '`jump`' and '`dump`', with a lower probability for '`lump`' and small probabilities for other words. The probability distribution to be determined is $P(W|\mathsf{x}_0^\tau)$ across all words W in the lexicon $\mathcal{L}$, given the input data x_0^τ.

The individual frame probabilities are combined to produce word probabilities using a hidden Markov model (HMM). A good tutorial article on HMMs is that of Rabiner and Juang (1986). A separate HMM is created for each word in the known lexicon, with one state representing each letter. Figure 8.1 shows a model for the word '`one`'. If there are N states, the set of states is $Q = \{q_r : r = 0, \ldots, N-1\}$, corresponding to the letters

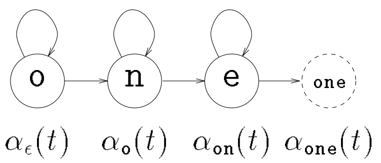


Figure 8.1: A simple Markov model for the word 'one' with one state per letter.

$\mathtt{L}(q_r)$. The Markov model represents a process by which the writing could have been generated. Each circle in the diagram represents a state of the model. At time $t = 0$ the model is in state q_0, corresponding to the beginning of the word. At each time step $t = 0, \ldots, \tau$, a state transition is made, following one of the arrows in the diagram. This means that either the next state is entered, or a self-transition is made and the state at the subsequent time step is the same as the current state. The state at time t is written S_t. In general a hidden Markov model can allow transitions between any pair of states, but in handwriting, the order of the letters is known and no letters can be missed out, so the model is made more restrictive. To use the model, transition probabilities are assigned to each of the permitted transitions and are assumed to be independent of the time: $a_{p,r} = P(S_{t+1} = q_r | S_t = q_p)$, $a_{p,r} = 0$ except when $r = p$ or $r = p + 1$. For the model to be a true Markov model, all the transition probabilities are dependent solely on the current state. By this process, a state sequence $S = (S_0, \ldots, S_\tau)$ is arrived at, which records the state at each time step. A typical state sequence might be $S = (q_0, q_0, q_0, q_0, q_1, q_1, q_1, q_2, q_2, q_2, q_2, q_2)$ corresponding to the letter sequence $\mathtt{L}(S) = (\mathtt{o,o,o,o,n,n,n,e,e,e,e,e})$. The model is a *hidden* Markov model because S is not directly observable, only inferred. It is only the frames of data that are observed.

In the generation process which is to be modelled, the system produces a frame of graphic data x_t at each time step. The data are part of the graphic representation of the letter signified by the current state. The data are assumed to occur according to a probability distribution $P(\mathsf{x}_t | \mathtt{L}(S_t))$, which is estimated by the recognition system of chapter 7. With this information an expression can be derived for the probability of a word, given a particular observation sequence x_0^τ.

The posterior probability of a word W can be rewritten using Bayes' rule:

$$P(W|\mathsf{x}_0^\tau) = \frac{P(\mathsf{x}_0^\tau|W)P(W)}{P(\mathsf{x}_0^\tau)}, \tag{8.1}$$

where $P(W)$ is the prior probability of the word occurring, which is discussed in section 8.4.2. The probability $P(\mathsf{x}_0^\tau)$ of the data occurring is unknown, but assuming that the word is in the lexicon $\mathcal{L}$, the probabilities must sum to one, and can be normalized:

$$\sum_{W \in \mathcal{L}} P(W|\mathsf{x}_0^\tau) = 1 \tag{8.2}$$

$$P(W|\mathsf{x}_0^\tau) = \frac{P(\mathsf{x}_0^\tau|W)P(W)}{\sum_{W\in\mathcal{L}} P(\mathsf{x}_0^\tau|W)P(W)}. \tag{8.3}$$

There are many state sequences representing any given word. Writing

$$\mathcal{S}(W) = \{S, \text{ such that } S \text{ represents } W\}, \tag{8.4}$$

then

$$P(\mathsf{x}_0^\tau|W) = \sum_{S\in\mathcal{S}(W)} P(\mathsf{x}_0^\tau|S)P(S), \tag{8.5}$$

where the state sequence probability $P(S)$ is the product of the initial distribution, $\pi_r = P(S_0 = q_r)$, and the subsequent transition probabilities:

$$P(S) = \pi_{S_0} \prod_{t=0}^{\tau-1} a_{S_t,S_{t+1}}. \tag{8.6}$$

Here $\pi_r = 0$ for all states except the first ($\pi_0 = 1$), so the model is constrained to start with the first letter. Now, by Bayes' rule

$$\begin{aligned} P(\mathsf{x}_0^\tau|S) &= P(\mathsf{x}_0|S)P(\mathsf{x}_1^\tau|S,\mathsf{x}_0) & (8.7)\\ &= P(\mathsf{x}_0|S)\prod_{t=1}^{\tau} P(\mathsf{x}_t|S,\mathsf{x}_0^{t-1}). & (8.8)\end{aligned}$$

If it is assumed that the emission probability is dependent solely on the class that the current state represents, this reduces to:

$$P(\mathsf{x}_0^\tau|S) = \prod_{t=0}^{\tau} P(\mathsf{x}_t|\mathsf{L}(S_t)), \tag{8.9}$$

which involves the terms $P(\mathsf{x}_t|\mathsf{L}(S_t))$ stored in the tables of chapter 7. A weaker assumption is that the emission probability is conditionally independent of preceding or following states, given the current state:

$$\begin{aligned} P(\mathsf{x}_0^\tau|S) &= \prod_{t=0}^{\tau} P(\mathsf{x}_t|S_t,\mathsf{x}_0^{t-1}) & (8.10)\\ &= \prod_{t=0}^{\tau} P(\mathsf{x}_t|\mathsf{L}(S_t),\mathsf{x}_0^{t-1}) & (8.11)\end{aligned}$$

where, by further applications of Bayes' rule, it can be seen that:

$$P(\mathsf{x}_t|\mathsf{L}(S_t),\mathsf{x}_0^{t-1}) = \frac{P(\mathsf{L}(S_t)|\mathsf{x}_0^t)P(\mathsf{x}_0^t)}{P(\mathsf{L}(S_t),\mathsf{x}_0^{t-1})}. \tag{8.12}$$

Now $P(\mathsf{L}(S_t)|\mathsf{x}_0^t)$ is exactly the posterior probability estimated by the recurrent network. $P(\mathsf{x}_0^t)$ is the probability of the first few frames of data, which is the same for all words. $P(\mathsf{L}(S_t),\mathsf{x}_0^{t-1})$ is assumed to be proportional to

$P(\mathtt{L}(S_t))$, the prior probability of a frame belonging to the class $\mathtt{L}(S_t)$. This assumption is clearly incorrect, but is found to work in practice. This probability can be estimated by counting the number of frames in each class according to the labels of the training set.

Thus there are two expressions for the likelihood $L(W|\mathsf{x}_0^\tau)$ of a word, which can be normalized to give word probabilities:

$$P(W|\mathsf{x}_0^\tau) = \frac{L(W|\mathsf{x}_0^\tau)}{\sum_W L(W|\mathsf{x}_0^\tau)} \tag{8.13}$$

$$L(W|\mathsf{x}_0^\tau) \approx P(W) \sum_{S\in\mathcal{S}(W)} \left(\prod_{t=0}^{\tau} P(\mathsf{x}_t|\mathtt{L}(S_t))\right)\left(\pi_{S_0}\prod_{t=0}^{\tau-1} a_{S_t,S_{t+1}}\right) \tag{8.14}$$

$$L(W|\mathsf{x}_0^\tau) \approx P(W) \sum_{S\in\mathcal{S}(W)} \left(\prod_{t=0}^{\tau} \frac{P(\mathtt{L}(S_t)|\mathsf{x}_0^t)}{P(\mathtt{L}(S_t))}\right)\left(\pi_{S_0}\prod_{t=0}^{\tau-1} a_{S_t,S_{t+1}}\right). \tag{8.15}$$

Equation 8.14 is used for the table look-up system and equation 8.15 is used for the recurrent network. For simplicity, the likelihoods $P(\mathsf{x}_t|\mathtt{L}(S_t))$ are used henceforth, but the scaled likelihoods $\frac{P(\mathtt{L}(S_t)|\mathsf{x}_0^t)}{P(\mathtt{L}(S_t))}$ are to be understood when the equations are applied to the recurrent network system.

These expressions can be calculated efficiently using the principle of Dynamic programming, in an array structure representing the states of the Markov model. In this model, each state is accorded a probability $\alpha_r(t)$, which is the probability of being in state r after t frames have been observed.Thus $\alpha_r(0) = \pi_r$ the initial distribution.

As successive frames of data are fed into the recognizer, and character probabilities are generated, the Markov model forward probabilities are calculated recursively by the formula:

$$\alpha_r(t+1) = \sum_p \alpha_p(t) P(\mathsf{x}_t|\mathtt{L}(q_p)) a_{p,r} \tag{8.16}$$

until all have been processed. At this point the final state (dashed in figure 8.1) contains $P(\mathsf{x}_0^\tau|W) = \alpha_n(\tau+1)$, the likelihood that the data x_0^τ represented the word of this model. By choosing the maximum of the likelihoods, $\mathrm{argmax}_W L(W|\mathsf{x}_0^\tau)$, if the models are good, a good estimate of the identity of the original word is obtained.

All of these probabilities are stored and multiplied in the log domain for speed and numerical accuracy. Multiplications become additions in the log domain. Probability additions can be calculated by using the identity

$$\log(a+b) \equiv \log a + \log(1+\exp(\log b - \log a)), \tag{8.17}$$

and deriving the second term from a look-up table, as described by Brown (1987).

8.1.1 Labelling

It will be recalled from chapter 4 that the database consists of both upper and lower case letters as well as punctuation. In fact the punctuation is excluded in the segmentation process, so only word images are passed to the preprocessing system, and no recognition of punctuation is carried out. If this were desired, a separate system for recognizing punctuation marks would be necessary. As punctuation marks appear in isolation and are largely defined by location, the recurrent network apparatus would be inappropriate. A much simpler system could be used, perhaps based on rules for the location and contour shape of punctuation marks.

The system described here gives a distribution across the 26 letter categories, and makes no distinction between upper and lower case letters. An 'a' and an 'A' are both labelled the same, and the network is trained to give the same output for either. There are not enough examples of capital letters in the database to train a network with separate output classes for both upper and lower case letters, since capitals only occur at the beginning of a few words and in a few acronyms. Indeed, the current system recognizes capital letters poorly, but since they are generally only initial letters, recognition is still possible based on the remaining letters and the constraints of the limited vocabulary. Testing a 160-unit network with a grammar gave an 8.8% error rate, but among words with capitals the error rate was 35%. The average rank in the lexicon of incorrectly recognized words with capitals was 96, compared to 15 for incorrect words without capitals. More data with capital letters would improve the recognition rate on capital letters, bringing down the overall error rate.

If more data were available, and distinction between upper and lower case were required, the network could be given 52 outputs to represent the upper and lower case letters. However, it might be better (because the network size would be kept down) to keep just 26 output categories, and have a separate unit indicating the case of the letter. Such a unit would give an independent probability, with a sigmoid output (equivalent to the two-class softmax). When using such a system, the hidden Markov models would need to be adapted to account for the separate classes and, according to the task, models with initial capitals, full capitals or even mixed case words could be permitted.

Some systems (Schenkel *et al.* 1994) have a 'noise' output class to allow the network to indicate that the inputs do not correspond to any of the letter classes. Such a class could be used in this system to represent poor writing or the ligatures between letters, but the implementation would be difficult since there is no noise or ligature class in the labelling of the training data. Since the system accepts cursive and discrete writing, the data would need to be hand-labelled to indicate the presence of ligatures. If such hand-labelling were done, then an optional ligature model could be inserted between the letter models of each word. A noise model could be placed in parallel with the letter models to allow letters to be skipped when there was something illegible in the input. Since few frames contain only ligatures, and the data used here were clean, these ideas were not implemented.

8.1.2 Decoding

In practice, most of the state sequences S are highly improbable, and sequences such as $\mathtt{L}(S) = (\mathtt{o,o,o,o,o,o,o,o,o,o,n,e})$ are going to contribute little to the probability of the word. In fact, in most cases, it can be said that there will be a small number of similar state sequences which are much more likely than all the others. Also, the single most likely sequence, S^*, will be similar to all of these, and can be considered to be representative. Thus, a good approximation to equation 8.5 is:

$$P(\mathsf{x}_0^\tau|W) \quad \propto \quad P(\mathsf{x}_0^\tau|S^*)P(S^*). \tag{8.18}$$

Carrying out decoding on only the most likely state sequence is called *Viterbi decoding*. In this case, the decoding is simpler. A different set of likelihoods, α', is stored:

$$\alpha'_r(t+1) \quad = \quad \max_p \alpha'_p(t) P(\mathsf{x}_t|\mathtt{L}(q_p))a_{p,r}. \tag{8.19}$$

These likelihoods can be computed more quickly than the full probabilities, α, and are found to give better results for this handwriting recognition system ($T(2) = 9.72, t_{.99}(2) = 6.96$). Comparative results are given in table 8.1.

Decoding method	Error rate (%)		Decoding time per word (s)
	$\hat{\mu}$	$\hat{\sigma}$	
Viterbi	17.0	0.68	1.32
Full	20.4	0.82	1.65

Table 8.1: A comparison of error rates and decoding times for five 80-unit networks trained on Viterbi segmentations, and tested with Viterbi or full decoding.

8.2 Duration modelling

This section investigates how the transition probabilities $a_{p,r}$ in equation 8.6 can be chosen so that words are modelled as well as possible, and to give optimum recognition performance. As a first approximation, it could be said that all state sequences are equally likely, and so all the transition probabilities could be made identical ($a_{p,p} = a_{p,p+1} = \frac{1}{2}\ \forall p$). Since a fixed number of frames is being decoded, any state sequence would have probability $P(S) = (\frac{1}{2})^{(\tau+1)}$.

In this case the state sequence probability has no effect on the recognition, and the word probabilities depend entirely on the observed data, taking no account of whether the state sequence is reasonable for the word.

Practically, though, a number of improvements can be made to the transition probabilities to make the Markov models model the true durations of letters much better. Hochberg (1992) has used similar techniques for the modelling of HMM state durations in speech recognition. In the simple, one-state-per-letter model of figure 8.1, the transition probabilities for dwelling in a given state or exiting to the next state in a word (p and $q = 1 - p$ respectively) can be adjusted. The obvious choice is to arrange for the expected duration of the model to be equal to the mean observed duration of a letter: $q = 1/d_{\mathrm{av}}$. In fact, in such a simple model, this will merely tend to favour long or short words depending on whether $p > q$ or not, because for a word of λ letters, $P(S) = p^{(\tau+1)}(\frac{q}{p})^{\lambda}$. Adjusting the mean length of each model individually gives improved modelling, but to start to obtain accurate models of the lengths of letters, the duration distribution needs to be examined.

The duration distribution specifies the probabilities $P(n)\ \forall n > 0$ of remaining in the state for n frames. The duration distribution of the simple model of figure 8.1 is geometric, as in the solid line of figure 8.2.

$$P(n) \quad = \quad p^{n-1}q \tag{8.20}$$

This does not match the duration distributions found in practice (shown in the dotted line of the figure). Better performance (ultimately in terms of reduced error rates) is to be expected if $P(S)$ can be modelled more accurately.

8.2.1 Enforcing a minimum duration

It is found that poor modelling often results from passing through a model in a single time step, when the data match the current model very badly, though a single letter is very rarely contained in a single frame of data. Although the probability of such a short duration will be very low, this can be outweighed by the increase in the data probability. To avoid this problem, a minimum duration $d_{\min} \geq 1$ is enforced. This forces $P(n) = 0$ for $n < d_{\min}$.

Several methods have been used to choose the minimum duration of a letter model. The first is to choose $d_{\min}$ to be the smallest duration observed in the training set, but this is subject to noise, particularly since the durations are determined automatically. A better method seems to be to choose $d_{\min} = d_{\mathrm{av}}/2$, though other similar methods work just as well.

The simplest method of implementing a minimum duration is to repeat each of the states in a given model, as shown in figure 8.3. The graphic data probabilities are the same for all the states in each class (*i.e.* the emission probabilities are *tied*). The operations for calculating the likelihoods α are exactly the same, but there are twice as many. When Viterbi decoding, this results in a minimum duration $d_{\min}$, longer durations having probabilities

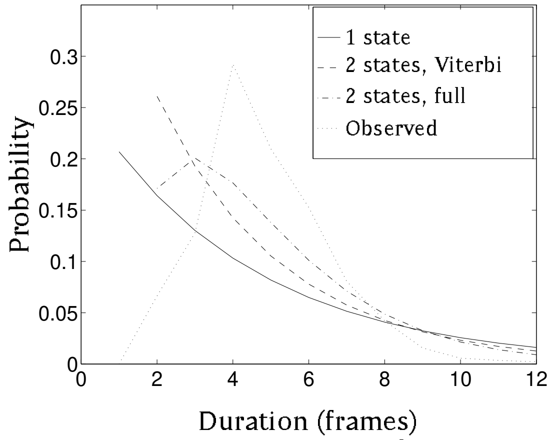


Figure 8.2: Probability distributions for the simple Markov models, compared with observed '*e*' durations.

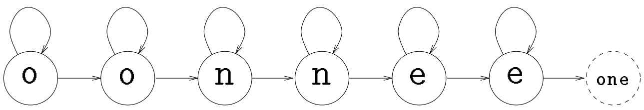


Figure 8.3: A Markov model for the word 'one' with two states per letter.

given by the geometric distribution. The probability of remaining in such a model for n frames is given by:

$$P(n) = \begin{cases} p^{n-d_{\min}} q^{d_{\min}} & n \geq d_{\min} \\ 0 & \text{otherwise} \end{cases} \tag{8.21}$$

$$q = \frac{1}{d_{\text{av}} - d_{\min} + 1} \tag{8.22}$$

where d_{av} is the average duration of a letter determined from the training set. In fact these are likelihoods, and the normalized probabilities are

$$P(n) = \begin{cases} p^{n-d_{\min}} q & n \geq d_{\min} \\ 0 & \text{otherwise.} \end{cases} \tag{8.23}$$

Robinson (1994), for example, uses geometric distribution models of this form to enforce minimum phone durations in speech recognition.

When doing full (as opposed to Viterbi) decoding, where multiple paths are permitted, the distribution given by this model is no longer geometric, but

$$P(n) = \begin{cases} C^{n-1}_{d_{\min}-1} p^{n-d_{\min}} q^{d_{\min}} & n \geq d_{\min} \\ 0 & \text{otherwise} \end{cases} \tag{8.24}$$

$$q = \frac{d_{\min}}{d_{\text{av}}}. \tag{8.25}$$

This distribution is closer to the observed distribution (figure 8.2), but by better modelling of the whole of the probability distribution, the performance can be increased still further.

8.2.2 Parametric distributions

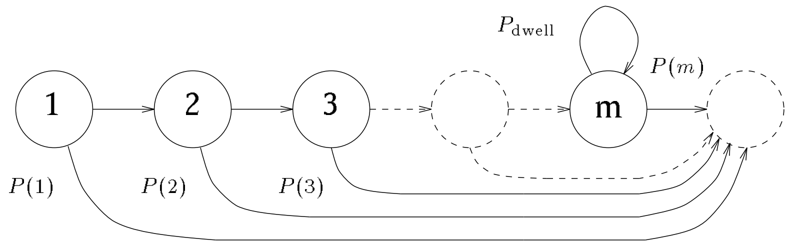


Figure 8.4: A complex duration model with m states for one letter.

More detailed modelling of the duration probability distribution can be accomplished with a more complex model, shown in figure 8.4. Here, each letter is represented by m states. The first $m-1$ states correspond to letter durations of from 1 to $m-1$ frames. From each of these states, the only permitted transitions are onto the next state of the same letter or onto the first state of the next letter. The transitions to the next letter are thus labelled with the duration probabilities $P(n)$. The final state has a dwell loop which gives the distribution a geometric tail. The probability p_{dwell} is adjusted to make the exit probabilities sum to one:

$$\sum_{n=1}^{m-1} P(n) + \sum_{n=m}^{\infty} P(m) P_{\text{dwell}}^{n-m} \quad = \quad 1. \tag{8.26}$$

The remaining transitions are given probability one. While this makes the sum of the probabilities at any node not equal to one, the sum of the probabilities of transition out of the model is one, so the duration of the model is described by a probability distribution. In fact, by normalizing appropriately, the same model duration distribution can be maintained while making the sum of probabilities at each state equal to one, but the form described here is clearer.

The more states in the model, the more accurately a given probability distribution can be modelled. With m states the model is perfect up to $n = m$, and follows a geometric distribution thereafter. However, the decoding time is proportional to the number of states, so the length of the model must be chosen from a trade-off between accuracy and speed. Table 8.2 shows the effect that model length has on recognition accuracy.

The duration distribution could be made to follow exactly the observed duration histogram from the training data. Without large quantities of data,

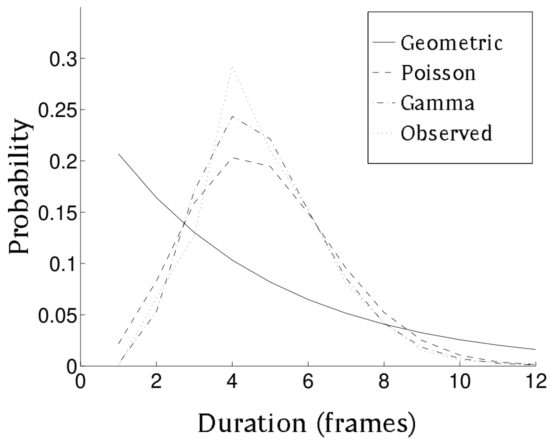


Figure 8.5: Probability distributions for three duration models, compared with the histogram of observed '$\mathfrak{a}$' durations.

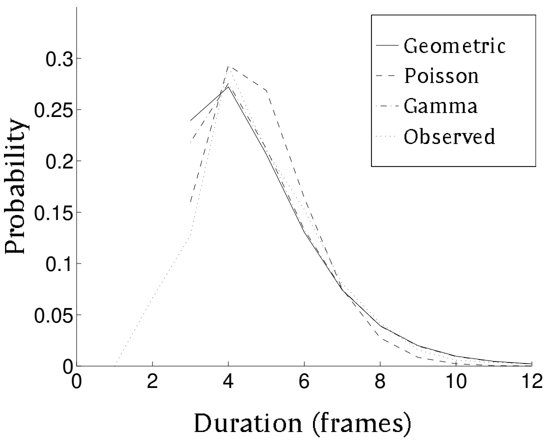


Figure 8.6: The same with a forced minimum duration of 3 frames.

however, these distributions are noisy, so a parametric probability distribution is used which fits the observed histogram well. In this work, three duration models have been investigated — based on the geometric, Poisson and gamma distributions. In each of these cases, the parametric distribution is used to calculate the probability of being in a letter model for a given number of frames. Each of these distributions can be shifted to impose a minimum duration $d_{\min} \geq 1$.

The Poisson distribution

Even for the case $d_{\min} = 1$, the Poisson distribution is shifted, since for the true Poisson distribution, $P(0) \neq 0$.

$$P(n) = \begin{cases} \frac{e^{-\lambda}\lambda^{n-d_{\min}}}{(n-d_{\min})!} & n \geq d_{\min} \\ 0 & \text{otherwise} \end{cases} \quad (8.27)$$

$$\lambda = d_{\mathrm{av}} - d_{\min}. \quad (8.28)$$

Schenkel *et al.* (1994) have recently used the Poisson distribution for duration modelling in on-line handwriting.

The gamma distribution

This distribution is parametrized by two parameters η and ν which determine the mean and variance. The values of η and ν are set according to the method of moments:

$$\eta = \frac{\mu - d_{\min} + 1}{\sigma^2} \tag{8.29}$$

$$\nu = (\mu - d_{\min} + 1)^2 \sigma^2 \tag{8.30}$$

$$P(n) = \begin{cases} \frac{\eta^{\nu}(n+1-d_{\min})^{\nu-1} e^{-\eta(n+1-d_{\min})}}{\Gamma(\nu)} & n \geq d_{\min} \\ 0 & \text{otherwise.} \end{cases} \tag{8.31}$$

8.2.3 Results

Sample error rates and recognition times are shown in table 8.2. It can be seen that enforcing a minimum duration of 2 in the geometric model reduces the error rate, but further increases impair the performance. Both of the complex duration models perform better than the geometric distribution models, and the gamma distribution performs better than the Poisson ($T(34) = 4.49, t_{.999}(34) < 3.14$). Modelling longer durations more accurately by adding states improves the performance but the returns diminish and the computation time increases. Comparing the 2 and 8 state gamma distributions shows a significant reduction in error rate ($T(4) = 3.28, t_{.975}(4) = 2.78$), but comparing 8 and 12 state gamma distributions does not ($T(4) = 0.16$). The 8 state gamma distribution is used in other experiments throughout this thesis.

One specific way in which the better modelling is manifested is in distinguishing between single and double letters. In the geometric model, for a given set of data, there is no difference between the probabilities for the models ‘reed’ and ‘red’ for example if the duration of the ‘e’ is longer than the minimum duration of the two ‘e’ models. However, with the more complex duration models, those with double letters will have different probabilities to those with single letters. In the ‘reed/red’ example, ‘red’ will have a higher probability than ‘reed’ if the number of frames with high ‘e’ probabilities is around the mean duration of an ‘e’, lower if there are more than double the mean.

8.3 Target re-estimation

Having trained the network for some time, it has a good estimate of the probability of each frame belonging to any letter. Given the correct word, the best state sequence S^* for this word represents a segmentation giving a new label for each frame. For a network which models the probability distributions well, this segmentation will be better than the automatic segmentation of section 7.1.2 since it takes the data into account. Finding the most probable state sequence S^* is termed a *forced alignment*. Since only the correct word model need be considered, such an alignment is faster than

Duration model	Number of states	Error rate (%)		Recognition time per word (s)
		$\hat{\mu}$	$\hat{\sigma}$	
Geometric	1	18.2	0.97	0.42
Geometric	2	16.6	0.92	0.62
Geometric	3	17.1	1.00	0.91
Geometric	4	26.1	0.79	0.91
Poisson	2	16.5	0.94	0.55
Poisson	3	16.4	0.82	0.83
Poisson	4	16.3	0.76	0.91
Poisson	6	16.1	0.82	1.43
Poisson	8	16.2	0.86	1.65
Poisson	10	15.9	0.79	2.14
Poisson	12	15.7	0.74	2.49
Gamma	2	16.5	0.92	0.55
Gamma	3	16.4	0.90	0.83
Gamma	4	15.9	0.69	0.91
Gamma	6	15.7	0.78	1.43
Gamma	8	15.6	0.72	1.65
Gamma	10	15.5	0.77	2.14
Gamma	12	15.5	0.81	2.49

Table 8.2: Sample performance figures for the different duration models.

the search through the whole lexicon required for recognition. Training on this automatic segmentation gives a better recognition rate, but still avoids the necessity of manually segmenting any of the database.

Figure 8.7 shows three different segmentations of the word '*butler*'. First (a) shows the segmentation arrived at by taking the most likely state sequence when using an 8 state gamma distribution Markov model, but with an untrained network, so the graphic data has no effect on the segmentation. This is similar to the 'equal length' segmentation used to bootstrap the system. (b) shows the effect of removing the duration model. There is now nothing to distinguish between the state sequences, except slight differences in the network's probability estimates due to initial asymmetry, so a poor segmentation results. After training the network (c), the durations deviate from the prior assumed durations to match the observed data. This re-estimated segmentation represents the data more accurately, so gives better targets towards which to train.

Having trained one network, the segmentations can be stored with the data files and used to train new networks, avoiding the less-accurate, equal-length segmentations and speeding up training. However, after completing training on these fixed targets, a further small improvement in recognition accuracy can be obtained by using the targets determined by the new network's own re-estimation of the segmentation.

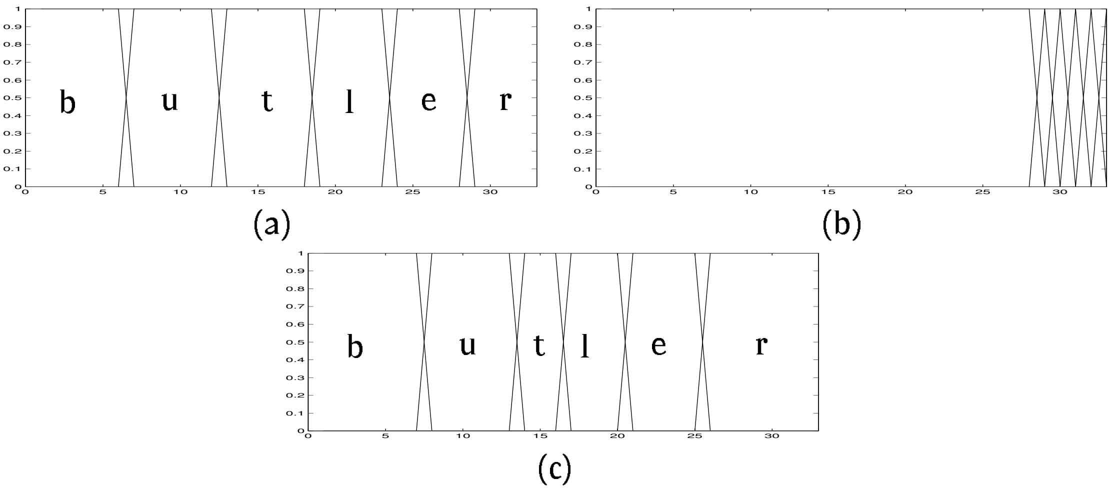


Figure 8.7: Viterbi segmentations of the word '*butler*'. Each line represents one letter Λ_i and is high for the frames t when $S_t^* = \Lambda_i$. (b) is a segmentation with an untrained network and no duration model. (a) shows the effect of adding an eight state gamma distribution duration model, and is similar to the 'bootstrap' segmentation. (c) is the segmentation re-estimated with a fully trained network and a duration model. For clarity, the segments are not labelled in (b).

The effects of this can be seen in the graph of relative entropy against number of epochs (figure 7.7). After a plateau indicating convergence, training on the fixed targets is stopped according to the stopping criterion. Training on the network's segmentation re-estimation is then begun and a steeper drop in relative entropy is seen. The relative entropy falls significantly because the new segmentation is that which is closest (within the constraints of the duration modelling, and the correct word model) to that indicated by the network's output probabilities. Thus the relative entropy of the output and target distributions will immediately be lower when the new segmentation is adopted. Thereafter, a new segmentation is calculated at every epoch and the network adapts its parameters in accordance with this segmentation. The relative entropy continues to fall. Similar effects can be seen in the graph of error rate against number of epochs (figure 7.6), but the effect is largely masked by noise.

Table 8.3 shows word recognition error rates for three 80-unit networks trained towards fixed targets estimated by another network, and then retrained, re-estimating the targets at each iteration. The retraining improves the recognition performance ($T(2) = 3.91, t_{.95}(2) = 2.92$).

Training method	Error (%)	
	$\hat{\mu}$	$\hat{\sigma}$
Fixed targets	21.2	1.73
Retraining	17.0	0.68

Table 8.3: Error rates for 3 networks with 80 units trained with fixed alignments, then retrained using individually re-estimated alignments.

8.3.1 Forward-backward retraining

The system described above performs well, but examining the speech recognition literature, a potential method of improvement can be seen. Viterbi frame alignments have so far been used to determine targets for training. These assign one class to each frame, based on the most likely state sequence, but a better approach might be to allow a distribution across all the classes indicating which are likely and which are not, avoiding a 'hard' classification at points where a frame may indeed represent more than one class, or none (as in a ligature). A 'soft' classification would give a more accurate portrayal of the frame identities.

Such a distribution can be calculated with the *forward-backward* algorithm (Rabiner and Juang 1986). To obtain the distribution $\gamma_p(t) = P(S_t = q_p|\mathsf{x}_0^\tau, W)$, the forward probabilities $\alpha_p(t)$ must be combined with the *backward* probabilities $\beta_p(t)$ which represent the probability of observing frames x_{t+1}^τ when starting in state p at time t. The backward probabilities are calculated similarly to the forward probabilities of equation 8.16:

$$\beta_p(t-1) \quad = \quad \sum_r \beta_r(t) P(\mathsf{x}_t|S_t = q_r) a_{p,r}. \tag{8.32}$$

A suitable final distribution $\beta_r(\tau) = \rho_r$ is chosen, *e.g.* $\rho = 1$ for the last character only. The likelihood of observing the data x_0^τ and being in state q_p at time t is then given by:

$$\xi_p(t) \quad = \quad \alpha_p(t) P(\mathsf{x}_t|S_t = q_p) \sum_r a_{p,r}\beta_r(t+1). \tag{8.33}$$

Then the probabilities $\gamma_p(t)$ of being in state q_p at time t are obtained by normalization:

$$\gamma_p(t) \quad = \quad \frac{\xi_p(t)}{\sum_r \xi_r(t)}.$$

These probabilities are used as targets for the recurrent network outputs.

Figure 8.8a shows the initial estimate of the class probabilities for a sample of the word '*butler*'. The probabilities shown are those estimated by the forward-backward algorithm when using an untrained network, for which the $P(\mathsf{x}_t|S_t = q_p)$ will be independent of class. Despite the lack of information, the probability distributions can be seen to take reasonable shapes. The first frame must belong to the first letter, and the last frame must belong to the last letter, of course, but it can also be seen that half way through the word, the most likely letters are those in the middle of the word. Several class probabilities are non-zero at a time, reflecting the uncertainty caused since the network is untrained. Nevertheless, this limited information is enough to train a recurrent network, because as the network begins to approximate these probabilities, the segmentations become more definite. In contrast, using Viterbi segmentations with no duration model for an untrained network, the most likely alignment can be very different from the true alignment (figure 8.7b). The segmentation is very definite though, and the network is trained towards the incorrect targets, reinforcing its error.

The process of training a network can be speeded up by enforcing a strong duration model, as shown in figure 8.8b, which gives more pronounced peaks in the probabilities for individual letters, because the duration model reduces the uncertainty in their length and location. Figure 8.8 c,d shows the effect that dividing by the class prior probability has on the segmentation. With no duration model, the segmentation is distorted, but when the duration model is imposed, the segmentation is better (stronger peaks, which overlap less) than before dividing by the class prior.

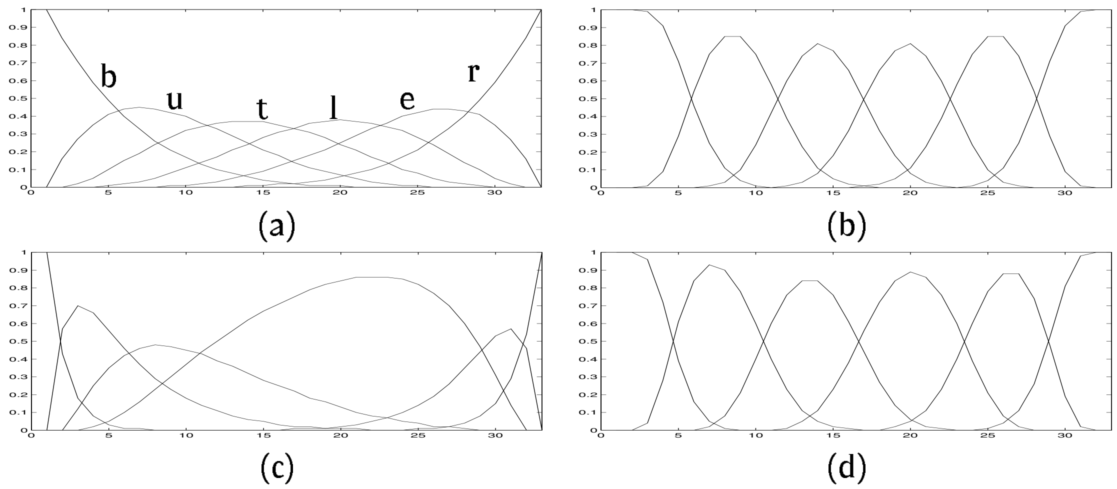
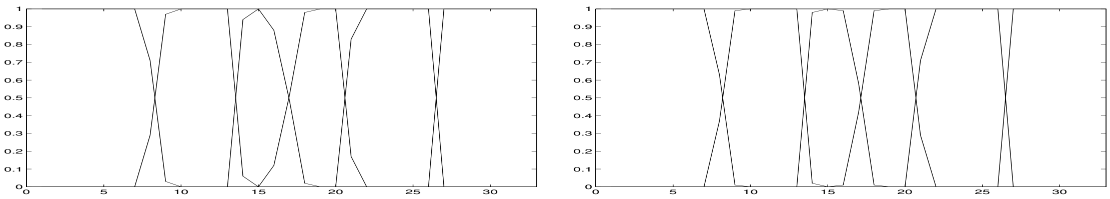


Figure 8.8: Baum-Welch segmentations of the word 'butler' with an untrained network. (a) is the segmentation using no duration model, and a uniform class prior. (b) shows the effect of adding an eight state gamma distribution duration model. (c) shows the effect of dividing by the prior class probability (equation 8.15). (d) shows the same with a duration model.

Finally, a trained network (especially when a strong durational model is used) gives a much more rigid segmentation (figure 8.9 a,b), with most of the probabilities being zero or one, but with a boundary of uncertainty at



(a) (b)

Figure 8.9: Baum-Welch segmentations of the word '*butler*' using trained networks. (a) has the geometric duration model and (b) has an eight-state gamma distribution duration model.

the transitions between letters. This uncertainty, where a frame might truly represent parts of two letters, or a ligature between two, allows the network trained with the forward-backward algorithm and tested using full forward probabilities to give improved recognition results over a network using Viterbi alignments and testing. The improvement is shown in table 8.5. The final probabilistic segmentation can be stored with the frames of data in the same way as the Viterbi segmentation was, and used when subsequent networks are trained on the same data. Training is then significantly quicker than when training towards the approximate bootstrap segmentations.

Table 8.4 shows word recognition error rates for 80-unit networks trained towards fixed Baum-Welch targets estimated by another network, and then retrained, re-estimating the targets at each iteration. As with the corresponding Viterbi alignments (figure 8.3) the retraining improves the recognition performance ($T(4) = 3.11, t_{.975}(4) = 2.78$).

Training method	Error (%)	
	$\hat{\mu}$	$\hat{\sigma}$
Fixed targets	16.9	0.75
Retraining	15.6	0.72

Table 8.4: Error rates for 5 networks with 80 units trained with Baum-Welch alignments, then retrained using re-estimated alignments.

Table 8.5 shows a comparison between the use of Viterbi and full probabilities when training and decoding. It can be seen that the error rates for the networks trained with Baum-Welch targets are lower than those trained on Viterbi targets ($T(2) = 5.24, t_{.975}(2) = 4.30$). As seen in table 8.1, the error is lowest if the system is tested with Viterbi rather than full decoding. For Baum-Welch targets, the difference is smaller but still significant ($T(4) = 4.94, t_{.995}(4) = 4.60$).

Baum-Welch retraining is the standard method of retraining the discrete Markov model, and the tables in section 7.3.3 refer to models retrained with

Training method	Error (%)			
	Viterbi decode		Full decode	
	$\hat{\mu}$	$\hat{\sigma}$	$\hat{\mu}$	$\hat{\sigma}$
Viterbi	17.0	0.68	20.4	0.82
Baum-Welch	15.4	0.74	15.6	0.72

Table 8.5: Error rates for networks with 80 units trained with Viterbi (3 networks) or Baum-Welch (5 networks) alignments, then tested using Viterbi or full probability decoding.

Baum-Welch. The network estimations used to prime the training are generally better than those of the discrete HMM, so only a small improvement is seen by retraining.

8.4 Language modelling

I am not yet so lost in lexicography ...

Samuel Johnson.

One area where great gains in recognition accuracy can be made is by language modelling, as can be seen from the wealth of literature on this area from the field of speech recognition (Waibel and Lee 1990:ch.8). The system as described so far has a language model built in in the form of a fixed lexicon which limits the search to a set $\mathcal{L}$ of permitted words.

8.4.1 Vocabulary choice

The lexicon used so far was chosen to be the union vocabulary of the training, test and validation sets, so that any word in the corpus would be in the lexicon. In practice, the lexicon size would be dictated by the task to be dealt with. In an application such as reading cheques, the vocabulary size would be around 35 words, comprising numbers, currency units, '`and`' and so forth. On the other hand, for transcribing longhand documents, the vocabulary would need to be tens of thousands of words, to cover nearly all the words likely to occur. The size of the vocabulary affects the performance of any recognition system because when it is large, words similar to the correct word are more likely to be permitted. For instance, in a cheque application the word '`hundred`' is unlike all the other words, but '`hounded`' might be necessary in a large vocabulary system, increasing the likelihood of confusion.

In postal applications, the potential vocabulary is large, containing all street, city, county and country names, but a system might be required to identify only the city or only the state name, these having been segmented from the address block. The vocabulary is now much smaller, making the task easier. In fact, the main reason for using cursive script in address reading is to disambiguate confusions in reading the zip code. If the zip code is reliably read, the city will be known, but if one, two or three digits are uncertain, the vocabulary will reflect this uncertainty and rise to ten, a hundred or a thousand potential city names. (If the correspondence between zip codes and cities is not one-to-one, the vocabulary size will vary, but this is a rough guide.) Thus these are reasonable vocabulary sizes for testing a postal system, with the vocabulary being dynamically chosen from a longer list according to the cities matching the known digits of the zip code.

Lexicon size	Error rate (%) $\hat{\mu}$	$\hat{\sigma}$	Time per word (s)
501	13.3	0.72	0.22
1048	16.1	0.73	0.33
2155	18.3	0.73	0.61
4554	20.8	0.72	1.27
9733	23.7	0.72	2.83

Table 8.6: Error rates from testing five 80-unit networks on lexica of different sizes.

To test the effect on error rate that the lexicon size has, experiments have been conducted with lexica containing different numbers of words. Table 8.6 and figure 8.12 show the results of these experiments. The lexica are created by taking the vocabulary of the test-set (447) and adding to that the most frequent words from the LOB corpus that were not already included. This ensures that the correct word is always in the lexicon, but allows lexica from 447 to 10,000 words to be tested. In practice, the lexica were made from approximately 500, 1000, 2000, 4000 and 8000 words, but including all words sharing the lowest frequency needed to make up the total, meant that these figures were exceeded in each case. This experiment corresponds to one done by Schenkel *et al.* (1994) who similarly construct lexica including all the test-set words. They make up the total with words chosen randomly from a large dictionary which will tend to be longer, and thus less confusable than the most frequent words. The 501 word error rate is lower than those quoted before, because of the smaller lexicon size, but later lexica give more errors because of the increase in similarity between the permitted words. Because the most common words were added, and since these are the shorter words which the system tends to confuse, the results are worse with this 1048 word lexicon than with the usual 1334 word lexicon (15.6%).

8.4.2 Grammars

After considering the vocabulary of the system, the next level of complexity in language modelling is to impose a grammar on the words, to limit which words are permissible in a given context or to account for the frequencies of different words. The simplest form is termed a '*unigram*' grammar, and simply involves determining the probability of a word occurring, and using that as the $P(W)$ term of equation 8.1. The probabilities are determined by frequency counts in a corpus of data, for instance in the whole LOB corpus (less the training set) or just on the training set. One problem with defining stochastic grammars is that words in the grammar may not occur in the database available for training the grammar. Complex smoothing techniques exist, but here the simple expedient of assigning a frequency count of one to unobserved words is adopted.

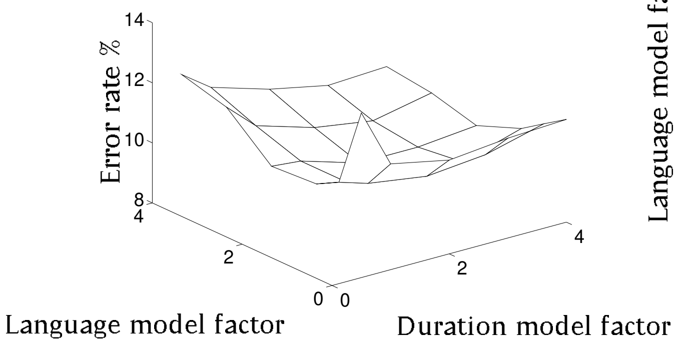


Figure 8.10: A mesh plot showing the effect on error rate of weighting the language and duration model probabilities.

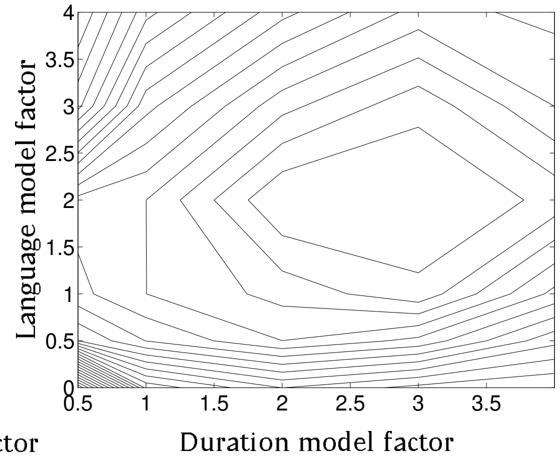


Figure 8.11: The corresponding contour plot, showing the minimum at (3,2).

In practice, it has been found in speech recognition that weighting the language model and the duration model with respect to the acoustic model gives better recognition. This is equivalent to rewriting equation 8.14 as

$$L(W|\mathrm{x}_0^\tau) \quad = \quad P(W)^\lambda \sum_{S \in \mathcal{S}(W)} P(\mathrm{x}_0^\tau|S)P(S)^\kappa. \tag{8.34}$$

Varying the weights κ and λ affects the recognition rate, and is a method of indicating the relative degree of confidence in the accuracy of the three probability estimates. Figures 8.10 and 8.11 show the variation in error rate when testing a single network as the weights are altered, keeping the weighting of the graphic data probability equal to one. The optimum values found are 3 for the language model weight, λ and 2 for the duration model weight, κ.

Much research has been done into using more complex language models which use context to determine which words are possible in the next position — such as word pair grammars which simply limit the vocabulary according to the previous word — or *bigram* grammars which assign a probability to a word, conditioned on the previous word. By determining statistics on a large corpus of text, the frequency of occurrence of pairs of words can be determined, giving the bigram grammar $P(W_t|W_{t-1})$. For pairs of words not observed in the corpus, the unigram grammar must be used instead. More context can be used, as in the general *n-gram* grammar $P(W_t|W_{t-1},\ldots,W_{t-n})$, and parsing sentences during recognition can give information about what parts of speech are possible or likely in the next word. Kuhn and de Mori (1990) describe a method of *caching* recently used words as these are more likely in the following text, and Jelinek (1991) discusses other methods of language modelling. The present system considers each word in isolation, so none of these more complex schemes has been implemented, though they would be appropriate for a system transcribing sentences. Cheque amounts and postal addresses have a simple structure for which a restrictive grammar can be written to significantly reduce the number of words that need to be considered at the next stage.

Grammar	Entropy	Perplexity
No grammar	10.38	1334
Grammar based on training set only	8.96	500
Grammar based on whole of LOB corpus	9.72	845

Table 8.7: Entropy and perplexity of grammars for the LOB corpus.

All grammars are used to limit the choice of words, and so improve the recognition rate. A crude method of quantifying how effective a grammar $\mathcal{G}$ is, is to measure its *perplexity* $Q(\mathcal{G})$ (Lee 1989:p.145). This is the average over all words of the number of permitted successor words. For a unigram grammar, this is simply two to the power of the entropy $H(\mathcal{G})$ of the unigram probability distribution, measured in bits:

$$H(\mathcal{G}) = -\sum_{W\in\mathcal{L}} P(W)\log_2 P(W) \tag{8.35}$$

$$Q(\mathcal{G}) = 2^{H(\mathcal{G})}. \tag{8.36}$$

Lee notes that this "does not reflect the uncertainty encountered when decoding." If the grammar does not reflect the actual frequencies of the words in the test set, then the perplexity is a poor guide to the grammar's utility. A better measure is the *test set perplexity* $Q_{\text{test}}(\mathcal{G})$ calculated from the cross entropy of the test set, given the grammar (Charniak 1993:p.34):

$$H_{\text{test}}(\mathcal{G}) = -\sum_{W\in\mathcal{L}} P_{\text{test}}(W)\log_2 P(W) \tag{8.37}$$

$$Q_{\text{test}}(\mathcal{G}) = 2^{H_{\text{test}}(\mathcal{G})}, \tag{8.38}$$

where $P_{\text{test}}(W)$ is the proportion of the test set that word W represents, not the unigram probability $P(W)$. (Where test set words are not in the lexicon, as in section 8.4.4, $P_{\text{test}}(W)$ is calculated as a proportion of the in-vocabulary

Lexicon size	Perplexity	Error rate (%)	
		No grammar	Unigram
1334	500	15.6	14.5
1334	845	15.6	15.6
501	742	13.3	13.8
1048	921	16.1	15.5
2155	1029	18.3	16.8
4554	1119	20.8	17.7
9733	1188	23.7	18.7

Table 8.8: Error rates from testing five 80-unit networks on lexica of different sizes. The 1334 word lexicon is tested with the training set grammar and the LOB corpus grammar.

words.) This perplexity measure indicates how useful the grammar is at limiting the choice of words *to those in the test set*, which is the function that the grammar should perform.

Sample test-set perplexities are seen in tables 8.7 and 8.8. The unigrams for the lexica with lengths other than 1334 are estimated on the LOB corpus excluding the test set. Because of the mismatch between the test set distribution and the unigram probabilities, the perplexity for the 501 word vocabulary is higher than the lexicon size, and the 1334 word grammar estimated on the LOB corpus has a higher perplexity than that estimated on the training set. The effect of using these grammars for recognition is shown in table 8.8. It can be seen that using a grammar decreases the error rate in all cases except with the 501 word lexicon when the perplexity of the grammar is higher than the lexicon size (figures 8.12 and 8.13). The test set perplexity can be seen to indicate the effectiveness of the grammar reasonably well. This is highlighted in figure 8.14 where the recognition rate is seen to be proportional to the log perplexity for each of the types of lexicon and grammar used, though the slopes differ between the grammar types.

8.4.3 Experimental conditions

At this point, the whole of the standard test system has been described, and it is now possible to summarize the conditions used for earlier experiments. These conditions are used everywhere except as noted in individual experiments. The typical conditions are as follows:

Normalization Slope correction; Srihari and Božinović's slant estimate; Zhang and Suen's thinning algorithm.

Representation Uniform horizontal quantization; 7 band vertical quantization; skeleton coding at four angles; turn, endpoint, junction and dot features; eleven snake features.

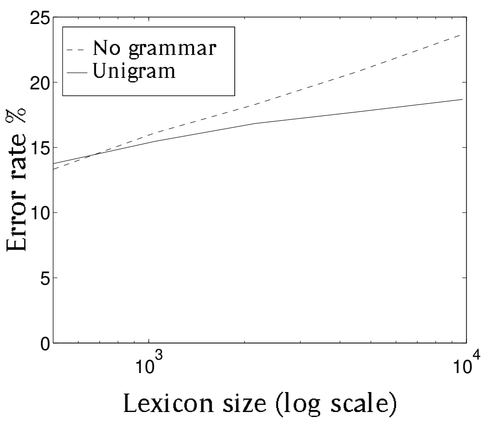


Figure 8.12: A graph of error rates averaged over five 80-unit networks. Error rates are shown when testing with and without a unigram grammar.

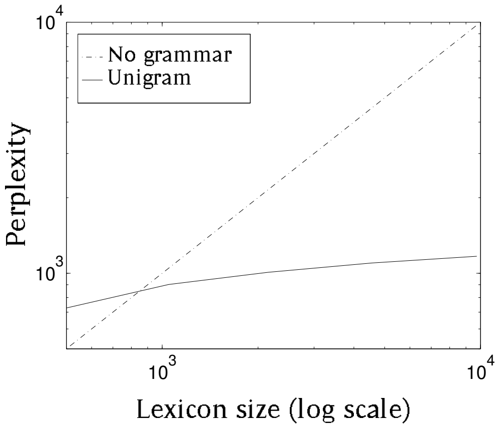


Figure 8.13: The test-set perplexities for the unigram grammars plotted against lexicon size. The lexicon size (the perplexity with no grammar) is also plotted for comparison.

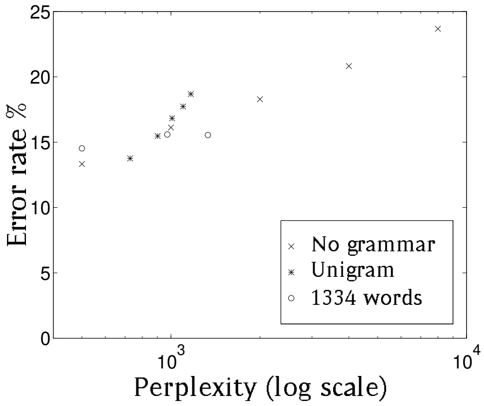


Figure 8.14: A graph of error rate against perplexity for lexica of different lengths with and without use of the unigram grammar. Figures for two different grammars are shown for the 1334 word lexicon (based on the training set or LOB corpus).

Recognition Recurrent network; 80 feedback units; 26 softmax output units.

Training Back-propagation through time with the modified delta bar-delta scheme; training towards fixed Baum-Welch targets until stopping criterion; retraining towards re-estimated targets.

Testing 1334 word vocabulary; no unigram grammar; 8 state gamma distribution duration model; duration model weighting of 2; full forward probability calculation. Tests with a unigram grammar use the grammar based on the training and validation sets, and a grammar weighting factor of 3.

It will be noted that these conditions are not the optimal conditions found so far. Improvements can be made by: using the Canny slant estimate; increasing the number of feedback units; using the unigram grammar and using non-uniform quantization. Subsequent experiments described in this chapter use all of these enhancements, but the network size is limited to 160 feedback units — the largest network trained on the non-uniformly quantized data set. Error rates for this network are shown in table 8.9.

Conditions	Error rate (%)
Before retraining, no grammar, full decoding	11.6
After retraining, no grammar, full decoding	9.6
After retraining, perplexity 500 grammar, full decoding	9.2
After retraining, perplexity 500 grammar, Viterbi	8.8

Table 8.9: Error rates when testing a 160-unit network on the 1334 word vocabulary.

8.4.4 Coverage

In most applications, there is a chance that the recognizer will be asked to identify a word that is not in the lexicon. A cheque amount could be filled in incorrectly, or a large vocabulary system might be presented with a proper name or neologism which would not be in the lexicon. Thus a system must be able either to recognize words not in the vocabulary (the next section describes one method of doing this), be condemned to incorrectly classify these non-words or flag that there was an out-of-vocabulary word for human proof-reading.

In the case where out-of-vocabulary words are not errors, and the system should be able to classify them, the vocabulary is termed 'open', in contrast to the 'closed' vocabulary task assumed above. For an open vocabulary task, the issue of *coverage* must be addressed — the proportion of words in a text which are in a recognizer's lexicon. If there is no method of recognizing out-of-vocabulary words, then this figure is an upper bound on the proportion of words that the recognizer can classify correctly. Some sample coverages for the LOB corpus with lexica of different sizes are shown in table 8.10. In each case, the lexicon is made of the n most frequent words from the LOB corpus. It should be noted that the coverage figures for the larger lexica are artificially high because the lexica are derived from the corpus on which coverage is assessed. On any other corpus, coverage would flatten off more for larger lexica. The coverage proportions are compared with the performance of the 160-unit network of section 8.4.3.

These results are shown graphically in figure 8.15. It can be seen that, as the lexicon size increases, the recognition rate increases, though it does not rise as fast as the test set coverage rate which is the optimal performance. As a measure of how well the system is performing compared to this upper

Lexicon size n	Coverage (%) LOB	Test	Error rate (%) Test set	In lexicon	Test-set perplexity	Decoding time per word (s)
2	9.9	10.9	89.1	0.0	1.9	0.76
4	15.5	14.9	85.1	0.6	3.5	0.76
8	21.9	22.1	78.6	3.6	7.2	0.77
16	28.4	28.7	73.7	8.6	12.6	0.77
32	36.5	36.1	66.5	7.4	22.0	0.78
64	44.6	43.7	59.8	8.1	37.2	0.80
125	51.8	51.9	53.2	10.0	61.9	0.85
250	58.6	58.3	47.5	10.1	94.8	0.96
500	65.4	66.8	39.8	9.9	156.6	1.19
1000	72.6	72.5	34.3	9.5	226.1	1.68
2000	79.7	81.0	27.4	10.3	369.7	2.70
4000	86.6	88.5	20.4	10.1	571.6	4.93
10000	93.8	94.1	16.0	10.9	822.8	11.8
20000	97.5	97.7	13.9	11.9	1048.2	23.9
30000	99.0	99.3	12.6	12.0	1179.4	36.8

Table 8.10: Coverage rates for lexica composed of the n most frequent words from the LOB corpus, on the LOB corpus as a whole, or on the LOB test set. The latter figure is the upper bound on the number of words correct. Error rates are shown as a percentage of words incorrect in the test set and as a percentage of the maximum potential words correct. Recognition times per test word are shown.

bound, the in-lexicon error rate is also plotted. This is the proportion of in-vocabulary words (which the system could have correctly identified with that lexicon) which are misclassified. This rises from 0% with two words (all words `the` and `of` are correctly classified) to 12% with a 30,000 word vocabulary.

8.4.5 Search issues

In the system described here, which has not been optimized for speed, with a large lexicon the majority of the recognition time is spent calculating the α probabilities in the hidden Markov model rather than estimating the posteriors in the recurrent network. Since there is one model per word, the search time increases linearly with the length of the lexicon (as can be seen in table 8.10 where the recurrent network takes approximately 0.76s per word, plus 10^{-3}s per lexicon item). For a development system with a 1000 word vocabulary this is tolerable, but for larger vocabularies there are a number of strategies which must be implemented to increase the speed. None of these has yet been implemented in the system, but all could be added simply. Patience was the only strategy adopted for the few large-vocabulary tests described here.

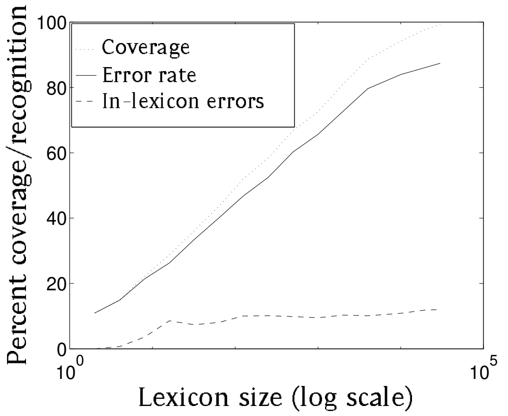


Figure 8.15: A graph of test-set coverage rate for lexica of different sizes. Recognition rates for a 160-unit network are shown, and the failure rate is also plotted. Failure is the proportion of in-vocabulary words that are wrongly classified.

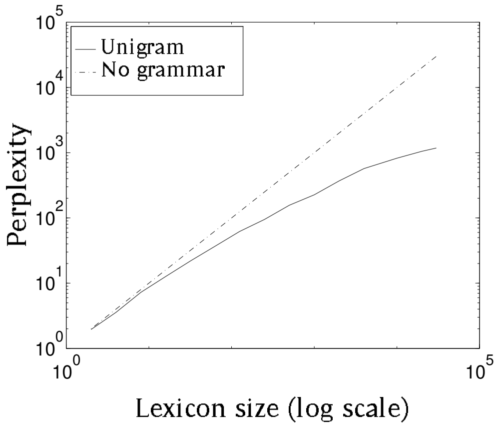


Figure 8.16: A graph of the test set perplexity of the unigram grammar for in-vocabulary words.

The first saving, which does not affect the performance of the recognizer, would be to organize the lexicon according to a tree structure. Since the words '`proud`' and '`proof`' share the first three letters, the calculations for these letters are being repeated. By storing the lexicon in a tree, this labour can be saved, at the cost of a small organizational overhead (figure 8.17).

Further time savings can be introduced by pruning the search path. If a state is found to be much less likely than the other states the search along paths leading from that state is terminated. Similarly, only the *n-best* paths at each time step need be retained, reducing the number of operations required. Renals and Hochberg (1994) have shown that the search can be very effectively pruned by examining the posterior probabilities estimated by the network.

Speed might also be improved by restricting the vocabulary using a simple, crude recognition method. The method does not need to be very accurate if it can reject a reasonable proportion of the vocabulary but rarely reject the correct word. Potential methods might include running a cut-down recognizer with one-state-per-letter-models, or a technique as simple as considering the height:width ratio of a word, or recognizing just the first letter with an isolated character recognizer. After reducing the vocabulary with this simple method, the full recognizer can be run with the smaller vocabulary. Systems for on-line recognition already use this fast match approach (Schenkel *et al.* 1994).

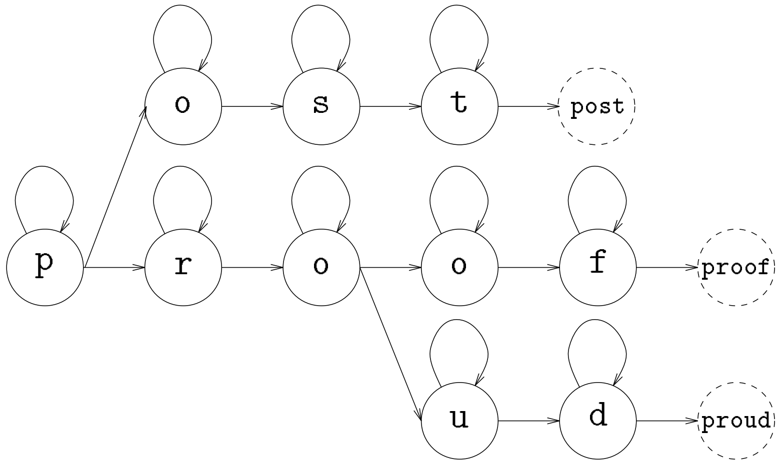


Figure 8.17: Three words from a lexicon stored as a tree to reduce the calculation time in decoding.

8.5 Rejection

And none can read the text — not even I.
Merlin in Tennyson's Idylls of the King.

The results quoted so far have all been error rates, where each word is classified by the recognizer and, according to its label, determined to be correct or incorrect. This is the performance measure which must be used for any non-interactive text transcription system, for it is the number of errors that is significant. For an application that allows some human intervention, however, a mechanism for rejection can be used. If a measure of confidence for the system's classifications can be formulated, then those words which are classified with low confidence can be rejected. With a good measure of confidence, many more incorrect words than correct words would be rejected, so the proportion of accepted words which are correct would be higher than the raw recognition rate. For a text transcription system, rejected words can be highlighted in the transcription and the user prompted for correct classification, reducing the effort needed to proof-read and correct the transcribed text. Similarly, in a post-office sorting situation, if those envelopes whose addresses are classified with low confidence are rejected and manually sorted, the number of machine sorted mail pieces incorrectly routed will be reduced. Projects designed to tackle commercial problems have specified accuracy and rejection goals that the classifiers must meet. Because recognition must be good to make automation cost-effective, the accuracy figure is usually very high, but, acknowledging the difficulty of handwriting recognition, the permitted rejection rates are high (section 2.2.1).

Three rejection measures have been evaluated for this system, based on the word likelihoods and posterior word probabilities for the most likely word, W_{best}, and the second most likely word, W_{second}. In the decoder, the likelihoods $L(W_{\text{best}}|\mathsf{x}_0^\tau)$, $L(W_{\text{second}}|\mathsf{x}_0^\tau)$and probabilities $P(W_{\text{best}}|\mathsf{x}_0^\tau)$, $P(W_{\text{second}}|\mathsf{x}_0^\tau)$ are already calculated, and it can be seen that if the graphic data matches a word model very well, then $P(W_{\text{best}}|\mathsf{x}_0^\tau)$ will be close to one and $L(W_{\text{best}}|\mathsf{x}_0^\tau)$, $\frac{P(W_{\text{best}}|\mathsf{x}_0^\tau)}{P(W_{\text{second}}|\mathsf{x}_0^\tau)}$ and $\frac{L(W_{\text{best}}|\mathsf{x}_0^\tau)}{L(W_{\text{second}}|\mathsf{x}_0^\tau)}$ will all be high.

$L(W_{\text{best}}|\mathsf{x}_0^\tau)$ is the product of a variable number of probabilities (depending on the number of frames $(\tau+1)$ in the word). To obtain a threshold applicable to words of any length, the log likelihood is scaled to be independent of these factors and the variable thresholded is the normalized likelihood $\hat{L}(W_{\text{best}}|\mathsf{x}_0^\tau)$:

$$\log \hat{L}(W_{\text{best}}|\mathsf{x}_0^\tau) = \frac{\log L(W_{\text{best}}|\mathsf{x}_0^\tau)}{\tau+1}. \tag{8.39}$$

Alternative scaling factors have been tested, incorporating the weights κ and λ, but this simple normalization was found to be most effective.

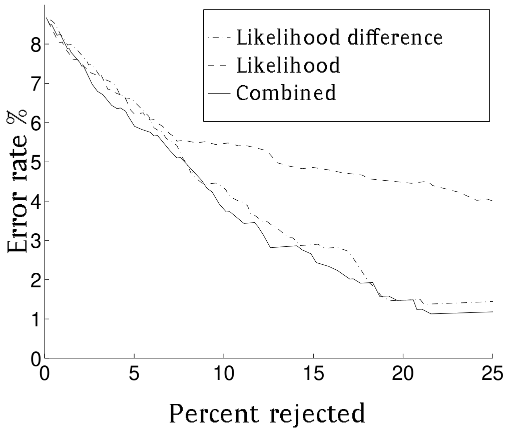


Figure 8.18: Error against rejection proportion for thresholding on normalized maximum log likelihood, difference in normalized log likelihood and a combined scheme.

By varying a threshold on any of these dimensions, and rejecting words which fall beyond the threshold, the error rate $\left(\frac{\text{incorrect words accepted}}{\text{total words accepted}}\right)$ and rejection rate can be found. The error rate can be plotted against the rejection rate for a variety of threshold values, to show the trade-off between rejection and accuracy. Figure 8.18 shows these curves when the threshold is on the normalized log of the maximum likelihood, and on the difference in normalized log of the best two words' likelihoods. The likelihood difference method works better than the likelihood method, since the error rate is lower for a given rejection rate. This graph also shows the performance of a combined method which thresholds on a linear combination of the two measures. Methods based on the posterior probability gave similar results (understandably, since the posteriors are closely related to the likelihoods, by equation 8.15). Using such a rejection criterion increases the accuracy of the system, *e.g.* giving error rates as low as 1.2% when rejecting 20.8% of the words, or 4.9% when rejecting 8.0%.

8.6 Out-of-vocabulary word recognition

Words and wordlessness. Between the two....

Tony Harrison. Wordlists.

If the vocabulary is not inherently limited by the task (in which case an out of vocabulary word is an error), the system should be able to detect that the word is poorly recognized and, if possible, should then use an alternative strategy to recognize the word.

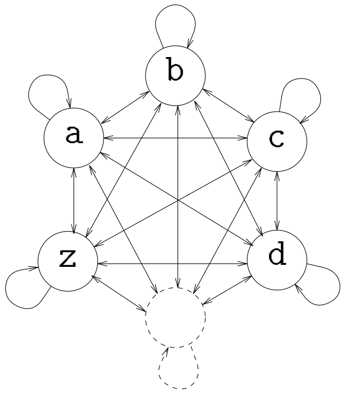


Figure 8.19: A non-word Markov model showing some of the 26 letter models.

One such strategy is to create a non-word Markov model, as shown in figure 8.19. Each circle represents a letter model, with one or more states. The initial distribution π is uniform across the first states of each letter model. The probabilities are combined to find the α' probabilities as before, but after each letter is complete, a transition to any of the letters is permitted. As the data are accumulated, a path is traced between successive letters.

When the final frame is processed, the most likely path is found and the letters corresponding to its state sequence can be printed out. Viterbi decoding is used, since finding the best sequence of letters when calculating full probabilities is much more difficult than in the fixed-vocabulary task. Just as with a word bigram, a letter bigram can be created detailing the probability of making a transition from one letter to another, and these probabilities can be multiplied into the state sequence probability. Table 8.11 shows the recognition rates for the non-word model when it is used instead of a lexicon. These results compare favourably with the single-author non-word error rates of 78–92% of Edelman *et al.* (1990).

Decoding with the non-word model is faster than when using a lexicon. (0.76s compared to 2.21s when using a 1334 word lexicon, both with the 160-unit network.) The non-word model could be used as a fast alternative to the lexicon-based decoder. It is possible to find the most likely letter sequence by this method, and then, if a lexicon is available, the best in-lexicon match

Bigram weight	Error rate (%)
0	60.3
1	53.9
2	54.2
3	55.1

Table 8.11: Error rates for the non-word model with different weightings (with the duration model weight $\kappa = 3$), using Viterbi decoding.

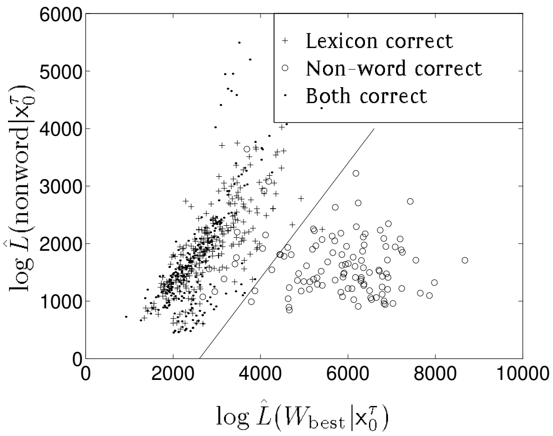


Figure 8.20: Words plotted with non-word normalized likelihood against lexicon normalized likelihood.

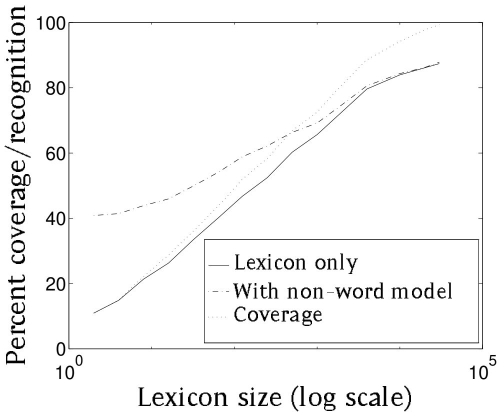


Figure 8.21: A graph of recognition rate against lexicon size, with and without modelling out-of-vocabulary words. The coverage of the lexica is also shown.

is determined by finding the word with the minimum edit distance from this sequence.[1] Several of the closest words could be identified and used as the vocabulary for a slower, more accurate recognition.

A system has been created which uses both the lexicon and the non-word model, finding the most likely word in the lexicon and the most likely letter string respectively. The problem then is to decide which of these hypotheses to choose. It has already been seen that the normalized likelihood is a good confidence measure for the classification of the lexicon-based system. A similar measure can be defined for the non-word model, based on the likelihood of the most likely state sequence, $L(\mathrm{nonword}|\mathsf{x}_0^\tau)$.

$$\log \hat{L}(\mathrm{nonword}|\mathsf{x}_0^\tau) = \frac{\log \left(P(W_{\mathrm{best}})^\lambda L(\mathrm{nonword}|\mathsf{x}_0^\tau)\right)}{\tau+1} \tag{8.40}$$

[1]The edit distance is calculated by comparing the letter string with each lexicon word, and penalties are accumulated for deletion, insertion and substitution of letters. This comparison is faster than calculation of the α probabilities for each word.

Note that, to correct for the effect of the unigram grammar on $\hat{L}(W_{\text{best}}|\mathsf{x}_0^\tau)$, the same prior, $P(W_{\text{best}})$ must be included in the non-word normalization to make the figures comparable. Now, plotting $\log \hat{L}(\text{nonword}|\mathsf{x}_0^\tau)$ against $\log \hat{L}(W_{\text{best}}|\mathsf{x}_0^\tau)$ for each word (figure 8.21) shows that there is a clear boundary separating the out-of-vocabulary words which the non-word model correctly identifies from the in-vocabulary words which the lexical approach gets right but the non-word model gets wrong. These are the two sets of words for which the decision between methods is critical. Words for which both methods are right or both are wrong can be ignored here as the choice between strategies does not affect the accuracy of these classifications.

Since the two groups of words hardly overlap, a threshold P_{nw}, can be chosen to give a decision boundary on the line:

$$\log \hat{L}(\text{nonword}|\mathsf{x}_0^\tau) \quad = \quad \log P_{\text{nw}} + \log \hat{L}(W_{\text{best}}|\mathsf{x}_0^\tau). \tag{8.41}$$

Figure 8.20 shows one such boundary $-\log_b P_{\text{nw}} = 2600$. This threshold can be interpreted as the log of the probability of transition into the non-word model within a global model which encompasses the non-word model and all the lexicon word models. In fact $P_{\text{nw}} = 0.33$. The base b of the logarithm was chosen to permit numerically accurate calculations with the probabilities stored as integers, if desired, so in fact b is little more than one.

Figure 8.21 shows the error rates when using this decision boundary. The error rates are compared to the coverage and the error rate using only the lexicon, as in figure 8.15. This time the recognition rate is higher than the coverage for small lexica, showing the power of the non-word model for recognizing out-of-vocabulary words. With larger lexica, the recognition rate falls below the coverage, but remains above the lexicon-only recognition rate. Thus a non-word model always improves the recognition rate, though the effect is small when the lexicon is large.

8.7 Summary

This chapter has described the final stage in the process of recognizing handwritten words: deriving word probabilities from the frame likelihoods of the previous chapter. From the simple models with one state per letter, a number of enhancements have been described. By modelling the duration distributions of letters, the system accuracy has been improved. The problem of vocabulary size has been addressed and its effect on the error rate shown, for both closed and open vocabulary tasks. A simple unigram grammar has been implemented, and it has been shown how this reduces the error rate. A scheme for rejecting poorly recognized words has been described and a system for recognizing words not in the lexicon implemented. Combining these has given increased recognition on the open vocabulary task when many test words are not in the lexicon.

The most significant results from this chapter are the final error rates of 8.8% with a lexicon and grammar, 53.9% using no lexicon and 12% on the open vocabulary task. Lower error rates can be achieved by applying a rejection criterion.

Chapter 9

Conclusions

I saw infinite processes that formed one single felicity and, understanding all, I was able to understand the script of the tiger.
Jorge Luis Borges. The God's Script.

This thesis has described a complete handwriting recognition system which has been implemented and tested on a database of cursive script. The results show that the method of recurrent error propagation networks can be applied successfully to the task of off-line cursive script recognition and perform better than a comparison hidden Markov model system. An 88% recognition rate has been achieved on an open-vocabulary task. Comparison of results with other researchers is difficult because of differences in experimental details, the actual handwriting used and the method of data collection. The results which have been published for similar problems are noted in section 2.3.2. The single author recognition rates for other systems are (for various lexicon sizes): 48% by Božinović and Srihari (1989), 50% by Edelman *et al.* (1990) and 70% by Yanikoglu and Sandon (1993).

The recognition performance of the system has been improved in a number of ways. The successive improvements are summarized in table 9.1. This shows the relative reduction in error rate that each of the techniques has brought about.

Enhancements in normalization and in the detection and representation of features have led to reduced error rates. The hybrid system, which was found to perform better than the discrete probability HMM system, was improved by retraining with re-estimated frame labels. Baum-Welch retraining of the recurrent network has been described here and has also brought about an improvement in recognition rates compared to using Viterbi targets. Better performance still can be hoped for from training larger networks, but the training time is problematic for such large networks without specialist hardware.

Language modelling has been found to improve the recognition performance, both by incorporating a model of the duration of each letter, and by adding a unigram word grammar. It has been shown that the system can recognize 46% of words without restriction to a lexicon, and that a model

Method	Proportional error rate reduction (%)
Skeleton vs. undersampling	36
Features	11
Non-uniform quantization	15
Snakes	14
Hybrid vs. discrete HMM	30
Baum-Welch vs. Viterbi targets	9
Retraining	9
Duration model	14
Unigram grammar	7

Table 9.1: The proportional reduction in error rate achieved by the incorporation of the techniques described in previous chapters. The discrete HMM is compared to a hybrid with the same number of parameters.

for words not in the system's vocabulary can increase the recognition rate beyond that otherwise obtained.

The training time of the recurrent network has been investigated and has been reduced by choosing an effective weight update scheme, by using softmax outputs, by specifying the training schedule and by initialization of the weight matrix. Preliminary work to investigate the operation of the network has been carried out, giving a greater understanding of the weights and feedback units. Much more could be done in this area with the hope of greater understanding and improved performance.

9.1 Further work

In writing a complete handwriting recognition system, one must face the problem of where effort can be most effectively applied to increase the performance. It is felt that in this system, the effort has been evenly distributed, but with a slight emphasis on the work described in chapter 8. In distributing the effort, potential improvements in every aspect of the system have necessarily been left without being investigated. As a result, further work could be carried out, with reasonable hope of return, on any of the techniques that have been described.

The preprocessing used could be improved upon, for example by extracting a better skeleton from the raw image. Doermann (1993) tackles this particular problem with a model-based approach, and derives a representation of the off-line strokes with inferred temporal information. His technique has not yet been applied to a recognition task. Normalization of a skeleton in the form derived by Doermann could be carried out using the procedures of Singer and Tishby (1994) which use a model of handwriting production to guide normalization. The non-uniform quantization scheme could also be made more stable, and the snake feature models could be extended as described at the end of chapter 6.

This system has been tested on the problem of single-writer handwriting recognition, though the design has been made open to accepting any style of handwriting, with normalization against scale, slope, slant and stroke width. It is hoped that future work will include the incorporation of algorithms which will allow the system to be tested on the CEDAR database.

The pure HMM system could be improved by experimenting with other quantization schemes using alternative metrics or dividing the input space into several spaces to be individually quantized. The HMM could also be given a probability distribution for each state, instead of tying the distributions across all states representing the same letter. This would be simple for the pure HMM, but might be computationally intensive for the hybrid system. Context-dependent models might also be used.

Better recognition rates for the hybrid system could be expected from the technique of connectionist model merging (Robinson *et al.* 1994). The imposition of a more complex, task-dependent grammar which further restricts the choice of words can also be expected to yield higher accuracy.

Bibliography

Abbink, G. H., Teulings, H. L. and Schomaker, L. R. B. (1993) Description of on-line script using Hollerbach's generation model. In (IWFHR 1993), pp. 217–224.

Aldus and Microsoft, (1988) *Tiff Standard Definition*, 5.0 edition.

Alimi, A. and Plamondon, R. (1993) Performance analysis of handwritten strokes generation models. In (IWFHR 1993), pp. 252–261.

Arcelli, C. and Sanniti di Baja, G. (1985) A width independent fast thinning algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **7** (4): 463–474.

Bellegarda, J. B., Nahamoo, D., Nathan, K. S. and Bellegarda, E. J. (1994) Supervized hidden Markov modeling for on-line handwriting recognition. In *International Conference on Acoustics, Speech and Signal Processing*, volume 5, pp. 149–152.

Bengio, Y., Le Cun, Y. and Henderson, D. (1994a) Globally trained handwritten word recognizer using spatial representation, convolutional neural networks and hidden Markov models. In (Cowan *et al.* 1994), pp. 937–944.

Bengio, Y., Simard, P. and Frasconi, P. (1994b) Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks* **5** (2): 157–166.

Bos, B. and van der Moer, A. (1993) The Bakunin project and optical character recognition. In (OCRHD 1993), pp. 11–15.

Boser, B. E. (1994) Pattern recognition with optimal margin classifiers. In (Impedovo 1994), pp. 147–171.

Bouma, H. (1971) Visual recognition of isolated lower case letters. *Vision Research* **11**: 459–474.

Bourlard, H. and Morgan, N. (1993) *Connectionist Speech Recognition: A Hybrid Approach*. Kluwer.

Božinović, R. M. and Srihari, S. N. (1989) Off-line cursive word recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **11** (1): 68–83.

Breuel, T. M. (1994) A system for the off-line recognition of handwritten text. Technical Report 94–02, IDIAP, CP 609, 1920 Martigny, Switzerland.

Bridle, J. S. (1990) Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. *Neurocomputing* **F 68**: 227–236.

Brown, P. F. (1987) The acoustic-modelling problem in automatic speech recognition. Technical report, Carnegie Mellon Computer Science Department. CMU-CS-87-125.

Browning, J. (1992) Artificial intelligence survey. *The Economist* **322** (7750): 21.

Caesar, T., Gloger, J. M. and Mandler, E. (1993a) Preprocessing and feature extraction for a handwriting recognition system. In (ICDAR 1993), pp. 408–411.

Caesar, T., Joachim, G., Kaltenmeier, A. and Mandler, E. (1993b) Recognition by handwritten word images by statistical methods. In (IWFHR 1993), pp. 409–416.

Canny, J. F. (1986) A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **8**: 679–698.

Charniak, E. (1993) *Statistical Language Learning*. MIT Press.

Cheriet, M. and Suen, C. Y. (1993) Extraction of key letters for cursive script recognition. *Pattern Recognition Letters* **14**: 1009–1017.

Cipolla, R. and Blake, A. (1990) The dynamic analysis of apparent contours. In *Third Int. Conf. Computer Vision*, pp. 616–623.

Cootes, T. F. and Taylor, C. J. (1992) Active shape models — 'smart snakes'. In *Proceedings of the British Machine Vision Conference*, ed. by D. Hogg and R. Boyle. Springer Verlag.

Cootes, T. F., Taylor, C. J., Cooper, D. H. and Graham, J. (1992) Training models of shape from sets of examples. In *Proceedings of the British Machine Vision Conference*, ed. by D. Hogg and R. Boyle. Springer Verlag.

Cowan, J. D., Tessauro, G. and Alspector, J. eds. (1994) *Advances in Neural Information Processing Systems*, number 6. Morgan Kaufmann.

Davies, E. R. (1990) *Machine Vision: Theory, Algorithms, Practicalities*. Microelectrics and signal processing Number 9. London Academic.

Doermann, D. S., (1993) *Document Image Understanding: Integrating Recovery and Interpretation*. University of Maryland Ph.D. thesis.

Downing, J. and Leong, C. K. (1982) *Psychology of Reading*. Macmillan.

Edelman, S., (1988) *Reading and Writing Cursive Script: A Computational Study*. Dept. of Applied Math, Weizman Institute of Science Ph.D. thesis.

Edelman, S., Ullman, S. and Flash, T. (1990) Reading cursive script by alignment of letter prototypes. *International Journal of Computer Vision* **5** (3): 303–331.

Eldridge, M. A., Nimmo-Smith, I., Wing, A. M. and Totty, R. N. (1984) The variability of selected features in cursive handwriting: Categorical measures. *Journal of the Forensic Science Society* **24** (3): 179–219.

Elliman, D. G. and Banks, R. N. (1991) A comparison of two neural networks for hand-printed character recognition. In *IEE 2nd Neural Networks*, number 349 in IEE, pp. 224–228.

Fahlman, S. E. (1988) An empirical study of learning speed in backpropagation neural networks. Technical Report CMU-CS-88-162, CMU.

Fontaine, T. and Shastri, L. (1992) Character recognition using a modular spatiotemporal connectionist model. Neuroprose, University of Pennsylvania, Philadelphia PA 19104-6389.

Fukushima, K. (1980) Neocognitron: A self-organising neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics* **36**: 193–202.

Garris, M. D., Blue, J. L., Candela, G. T., Dimmick, D. L., Geist, J., Grother, P. J., Janet, S. A. and Wilson, C. L., (1994) NIST form-based handprint recognition system. Document Understanding Mailing List.

Geake, E. (1992) Letters to a computer. *New Scientist* pp. 30–33.

Giles, C. L., Hanson, S. J. and Cowan, J. D. eds. (1993) *Advances in Neural Information Processing Systems*, number 5. Morgan Kaufmann.

Gilloux, M., Leroux, M. and Bertille, J.-M. (1993) Strategies for handwritten words recognition using hidden Markov models. In (ICDAR 1993), pp. 299–304.

Govindan, V. K. and Shivaprasad, A. P. (1990) Character recognition – a review. *Pattern Recognition* **23** (7): 671–683.

Gray, R. M. (1984) Vector quantization. In (Waibel and Lee 1990), chapter 3.3, pp. 75–100.

Haber, R. N. and Haber, L. R. (1981) Visual components of the reading process. *Visible Language* **XV** (2): 147–182.

Hepp, D. J. (1991) An application of backpropagation to the recognition of handwritten digits using morphologically derived features. *Proceedings of the SPIE* **1451**: 228–233.

Hinton, G. E., Williams, C. K. I. and Revow, M. D. (1992) Adaptive elastic models for hand-printed character recognition. In (Moody *et al.* 1992), pp. 512–522.

Hochberg, M. M., (1992) *A Comparison of State-Duration Modelling Techniques for Connected Speech Recognition.* Division of Engineering, Brown University Ph.D. thesis.

Hollerbach, J. M. (1981) An oscillation theory of handwriting. *Biological Cybernetics* **39**: 139–156.

Huang, Y. S. and Suen, C. Y. (1993) Combination of multiple classifiers with measurement values. In (ICDAR 1993), pp. 598–601.

Hubel, D. H. and Wiesel, T. N. (1962) Receptive fields,binocular interaction and functional architecture in the cat's visual cortex. *Journal of Physiology* **160**: 106–154.

Hull, J. J. (1993) A database for handwritten text recognition research. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

ICDAR. (1991) *First International Conference on Document Analysis and Recognition*, St. Malo, France.

ICDAR. (1993) *Second International Conference on Document Analysis and Recognition*, Tsukuba, Japan. IEEE Computer Society Press.

Idan, Y. and Chevalier, R. C. (1991) Handwritten digits recognition by a supervised Kohonen-like learning algorithm. *IJCNN 91* **3**: 2576–2581.

Impedovo, S. ed. (1994) *Fundamentals in Handwriting Recognition*, volume 124 of *NATO ASI Series F: Computer and Systems Sciences*. Springer Verlag.

Impedovo, S., Dimauro, G. and Pirlo, G. (1990) A new decision tree algorithm for handwritten numerals recognition using topological features. *Proc. SPIE* **1384**: 280–284.

IWFHR. (1993) *Third International Workshop on Frontiers in Handwriting Recognition*. CEDAR, SUNY Buffalo.

Jacobs, R. A. (1988) Increased rates of convergence through learning rate adaptation. *Neural Networks* **1**: 295–307.

Jelinek, F. (1991) Up from trigrams! The struggle for improved language models. In *Proceedings European Conference on Speech Communication and Technology*, pp. 1037–1039.

Johansson, S., Atwell, E., Garside, R. and Leech, G. (1986) The tagged LOB corpus. Technical report, Norwegian Computing Centre for The Humanities, Bergen.

Kass, M., Witkin, A. and Terzopoulos, D. (1987) Snakes: Active contour models. In *Proc. 1st Inter. Conf. Computer Vision*, pp. 259–268. IEEE.

Kimura, F. and Shridhar, M. (1991) Handwritten numerical recognition based on multiple algorithms. *Pattern Recognition* **24** (10): 969–983.

Kimura, F., Shridhar, M. and Chen, Z. (1993a) Improvements of a lexicon directed algorithm for recognition of unconstrained handwritten words. In (ICDAR 1993), pp. 18–22.

Kimura, F., Shridhar, M. and Narasimharmurthi, N. (1993b) Lexicon directed segmentation-recognition procedure for unconstrained handwritten words. In (IWFHR 1993), pp. 122–131.

Kuhn, R. and de Mori, R. (1990) A cache-based natural language model for speech recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **12** (6): 570–583. Corrected: PAMI, **14**(6): 691,1992.

Lam, L., Lee, S. and Suen, C. Y. (1992) Thinning methodologies — A comprehensive survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **14** (9): 869–885.

Lanitis, A. (1992) Applications of point distribution models in handwritten optical character recognition and face recognition. Transfer report, Dept. of Medical Biophysics, University of Manchester.

Lanitis, A., Taylor, C. J. and Cootes, T. F. (1993) A generic system for classifying variable object using flexible template matching. In *British Machine Vision Conference*, ed. by J. Illingworth, volume 1, pp. 329–338. BMVA Press.

Le Cun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. and Jackel, L. D. (1989) Backpropagation applied to handwritten zip code recognition. *Journal of Neural Computation* **1**: 541–551.

Lecolinet, E. and Baret, O. (1994) Cursive word recognition: Methods and strategies. In (Impedovo 1994), pp. 235–263.

Lecolinet, E. and Crettez, J.-P. (1991) A grapheme-based segmentation technique. In (ICDAR 1991), pp. 740–748.

Lee, K.-F. (1989) *Automatic Speech Recognition: The Development of the SPHINX System*. Kluwer.

Leroux, M., Salomé, J. C. and Badard, J. (1991) Recognition of cursive script words in a small lexicon. In (ICDAR 1991), pp. 774–782.

Leymarie, F. (1990) Tracking and describing deformable objects using active contour models. Technical Report TR-CIM-90-9, McGill University.

Linde, Y., Buzo, A. and Gray, R. M. (1980) An algorithm for vector quantizer design. *IEEE Transactions on Communications* **COM-28** (1): 84–95.

Lu, S. W., Ren, Y. and Suen, C. Y. (1991) Hierarchical attributed graph representation and recognition of handwritten Chinese characters. *Pattern Recognition* **24** (7): 617–632.

Manke, S. and Bodenhausen, U. (1994) A connectionist recognizer for on-line cursive handwriting recognition. In *International Conference on Acoustics, Speech and Signal Processing*, volume 2, pp. 633–6.

Marr, D. (1982) *Vision*. Freeman.

Martins, W. and Allinson, N. M. (1991) Visual search of postal codes by neural networks using human examples. In *IEE 2nd Conference on Neural Networks*.

Matan, O., Burges, C. J. C., Le Cun, Y. and Denker, J. S. (1992) Multi-digit recognition using a space displacement network. In (Moody *et al.* 1992), pp. 488–495.

McGraw, G., Rehling, J. and Goldstone, R. (1994) Roles in letter perception: Human data and computer models. Technical Report CRCC-TR 90, Center for Research on Concepts and Cognition. Indiana University.

McVeigh, A. (1993) The Irish database project: a case for OCR. In (OCRHD 1993), pp. 29–37.

Moody, J. E., Hanson, S. J. and Lippmann, R. P. eds. (1992) *Advances in Neural Information Processing Systems*, number 4. Morgan Kaufmann.

Moreau, J.-V., Plessis, B., Bougeois, O. and Plagnaud, J.-L. (1991) A postal cheque reading system. In (ICDAR 1991), pp. 758–766.

Mori, S., Suen, C. Y. and Yamamoto, K. (1992) Historical review of OCR research and development. *Proceedings of the IEEE* **80** (7): 1029–1058.

Mori, Y. and Yokosawa, K. (1988) Neural networks that learn to discriminate similar Kanji characters. *Neural Information Processing Systems*.

Nag, R., Wong, K. H. and Fallside, F. (1986) Handwritten script recognition using hidden Markov models. In *ICASSP '86*. IEEE.

Nellis, J. and Stonham, T. J. (1991) A fully integrated hand-printed character recognition system using artificial neural networks. In *IEE 2nd Conference on Neural Networks*, number 349 in IEE, pp. 219–223.

OCRHD. (1993) *Optical Character Recognition in the Historical Discipline*, number A18 in VIII International Conference of the Association for History and Computing. Netherlands Historic Data Archive.

Olsen, M. (1993) Scanning, keyboarding, and data verification: Factors in selecting data collection technologies. In (OCRHD 1993), pp. 93–112.

Palumbo, P. W., Srihari, S. N., Soh, J., Sridhar, R. and Demajenko, V. (1992) Postal address block location in real time. *IEEE Computer* **25** (7): 34–42.

Paquet, T. and Lecourtier, Y. (1991) Handwriting recognition: Application on bank cheques. In (ICDAR 1991), pp. 749–757.

Paquet, T. and Lecourtier, Y. (1993) Recognition of handwritten sentences using a restricted lexicon. *Pattern Recognition* **26** (3): 391–407.

Pavlidis, T. (1992) Application of splines to shape description. In *Visual Form*, ed. by Arcelli, Cordella, and S. di Baja, pp. 431–441. Plenum.

Pavlidis, T. (1993) Recognition of printed text under realistic conditions. *Pattern Recognition Letters* **14**: 317–326.

Pearlmutter, B. A. (1990) Dynamic recurrent neural networks. Technical Report CMU-CS-88-191, Carnegie Mellon University, School of Computer Science, Pittsburgh, PA15213.

Pettier, J. C. and Camillerapp, J. (1993) Script representation by a generalised skeleton. In (ICDAR 1993), pp. 850–853.

Plamondon, R., Bordeau, M., Chouinard, C. and Suen, C. Y. (1993) Validation of preprocessing algorithms: A methodology and its applications to the design of a thinning algorithm for handwritten characters. In (ICDAR 1993), pp. 262–269.

Plamondon, R. and Lorette, G. (1989) Automatic signature verification and writer identification — The state of the art. *Pattern Recognition* **22** (2): 107–129.

Plessis, B., Sicsu, A., Heutte, L., Menu, E., Lecolinet, E., Debon, O. and Moreau, J.-V. (1993) A multi-classifier combination strategy for the recognition of handwritten cursive words. In (ICDAR 1993), pp. 642–645.

Rabiner, L. R. and Juang, B. H. (1986) An introduction to hidden Markov models. *IEEE ASSP magazine* **3** (1): 4–16.

Rayner, K. and Pollatsek, A. (1989) *The Psychology of Reading.* Prentice-Hall.

Renals, S. and Hochberg, M. (1994) Decoder technology for connectionist large vocabulary speech recognition. Technical Report CUED/F-INFENG/TR.186, Cambridge University Engineering Department, UK.

Robinson, A. (1994) The application of recurrent nets to phone probability estimation. *IEEE Transactions on Neural Networks.*

Robinson, A. and Fallside, F. (1991) A recurrent error propagation network speech recognition system. *Computer Speech and Language* **5**: 259–274.

Robinson, A., Hochberg, M. and Renals, S. (1994) IPA: Improved phone modelling with recurrent neural networks. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing,* volume 1, pp. 37–40.

Robinson, A. J., (1989) *Dynamic Error Propagation Networks.* Cambridge University Engineering Department Ph.D. thesis.

Rumelhart, D. E., Hinton, G. E. and Williams, R. J. (1986) Learning internal representations by error propagation. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, ed. by D. E. Rumelhart and J. L. McClelland, volume 1, chapter 8, pp. 318–362. Bradford Books.

Sasanuma, S. (1984) Can surface dyslexia occur in Japanese. In *Orthographies and Reading.* Lawrence Erlbaum Associates.

Schenkel, M., Guyon, I. and Henderson, D. (1994) On-line cursive script recognition using time delay neural networks and hidden Markov models. In *International Conference on Acoustics, Speech and Signal Processing,* volume 2, pp. 637–640.

Senior, A. W. (1993) A recurrent network approach to the automatic reading of handwriting. In (OCRHD 1993), pp. 59–65.

Senior, A. W. (1994) Normalisation and preprocessing for a recurrent network off-line handwriting recognition system. In (Impedovo 1994), pp. 360–365.

Senior, A. W. and Fallside, F. (1993a) Off-line handwriting recognition by recurrent error propagation networks. In (IWFHR 1993), pp. 132–141.

Senior, A. W. and Fallside, F. (1993b) Using constrained snakes for feature spotting in off-line cursive script. In (ICDAR 1993), pp. 305–310.

Simard, P., Le Cun, Y. and Denker, J. (1993) Efficient pattern recognition using a new transformation distance. In (Giles *et al.* 1993), pp. 50–58.

Simon, J.-C. (1992) Off-line cursive word recognition. *Proceedings of the IEEE* **80** (7): 1150–1161.

Singer, Y. and Tishby, N. (1993) Decoding of cursive scripts. In (Giles *et al.* 1993).

Singer, Y. and Tishby, N. (1994) Dynamical encoding of cursive handwriting. *To appear in Biological Cybernetics*.

Srihari, S. N. and Božinović, R. M. (1987) A multi-level perception approach to reading cursive script. *Artificial Intelligence* **33**: 217–255.

Srihari, S. N., Govindaraju, V. and Shekhawat, A. (1993) Interpretation of handwritten addresses in US mailstream. In (ICDAR 1993), pp. 291–294.

Starner, T., Makhoul, J., Schwartz, R. and Chou, G. (1994) On-line cursive handwriting recognition using speech recognition techniques. In *International Conference on Acoustics, Speech and Signal Processing*, volume 5, pp. 125–128.

Suen, C. Y., Berthod, M. and Mori, S. (1980) Automatic recognition of handprinted characters— the state of the art. *Proc. IEEE* **68** (4): 469–487. 244 references.

Suen, C. Y., Legault, R., Nadal, C., Cheriet, M. and Lam, L. (1993) Building a new generation of handwriting recognition systems. *Pattern Recognition Letters* **14**: 303–315.

Taylor, I. and Taylor, M. M. (1983) *The Psychology of Reading*. New York Academic.

Teulings, H.-L. (1994) Invariant handwriting features useful in cursive-script recognition. In (Impedovo 1994), pp. 179–198.

Waibel, A. and Lee, K.-F. eds. (1990) *Readings in Speech Recognition*. Morgan Kaufmann.

Wang, C.-H. and Srihari, S. N. (1988) A framework for object recognition in a visually complex environment and its application to locating address blocks on mail pieces. *International Journal of Computer Vision* **2**: 125–151.

Winston, P. H. (1984) *Artificial Intelligence*. Addison Wesley, second edition.

Woodland, P. C., Odell, J. J., Valtchev, V. V. and Young, S. J. (1994) Large vocabulary continuous speech recognition using HTK. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 2, pp. 125–128.

Yamadori, A. (1975) Ideogram reading in alexia. *Brain* **98**: 231–238.

Yanikoglu, B. A. and Sandon, P. A. (1993) Off-line cursive handwriting recognition using style parameters. Technical Report PCS-TR93-192, Dartmouth College, NH.

Yuille, A. L., Hallinan, P. W. and Cohen, D. S. (1992) Feature extraction from faces using deformable templates. *International Journal of Computer Vision* **8** (2): 99–111.

Zhang, T. Y. and Suen, C. Y. (1984) A fast parallel algorithm for thinning digital pictures. *Communications of the ACM* **27** (3): 236–239.