Appeared in IEEE Pattern Analysis and Machine Intelligence Vol. 20 No. 3 March 1998 pp. 309–321




# An Off-line Cursive Handwriting Recognition System

Andrew W. Senior and Tony Robinson.

*Abstract*— **This paper describes a complete system for the recognition of off-line handwriting. Preprocessing techniques are described, including segmentation and normalization of word images to give invariance to scale, slant, slope and stroke thickness. Representation of the image is discussed and the skeleton and stroke features used are described. A recurrent neural network is used to estimate probabilities for the characters represented in the skeleton. The operation of the hidden Markov model that calculates the best word in the lexicon is also described. Issues of vocabulary choice, rejection, and out-of-vocabulary word recognition are discussed.**

*Keywords*— **Offline cursive handwriting recognition, optical handwritten character recognition, preprocessing, feature extraction, recurrent neural networks, hidden Markov models, out of vocabulary word models.**

## I. Introduction

Off-line handwriting recognition is the automatic transcription by computer of handwriting, where only the image of the handwriting is available. Off-line handwriting is thus distinguished from on-line handwriting where the path of the pen is measured by a device such as a digitizing tablet. A host of applications of off-line handwriting can be envisaged, including document transcription, automatic mail routing, and machine processing of forms, cheques and faxes.

Other systems to recognize off-line handwriting have been produced, but most are limited to digit recognition or small vocabulary transcription problems such as the postal or cheque-reading applications where the context or additional knowledge, *e.g.* the zip code, limit the vocabulary considerably [1].

The system described in this paper has been designed to tackle the large-vocabulary task of text transcription. The work described has been carried out on a publicly-available database of writing from a single author. As such the envisaged application is for the transcription of documents by one writer — either for personal notes, off-line data entry, or potentially for historical document transcription. All of the techniques used here would be applicable to a writer-independent transcription system that could be used to transcribe incoming mail, or faxes, address blocks or cheques. Similar tasks have been attempted by other authors [2], [3].

A.W. Senior is with the IBM TJ Watson Research Center, NY 10598, USA. E-mail: aws@watson.ibm.com

A.J. Robinson is with Cambridge University Engineering Department, CB2 1PZ, UK. Email: ajr4@cam.ac.uk

### A. Overview

This paper describes the operation of a complete handwriting recognition system intended for general text transcription, from the scanning of the initial image from paper to the output of machine-readable text. The system can be conveniently divided into the same broad sections of preprocessing, recognition and post-processing, as are found in most other handwriting recognition systems. The processing starts with data acquisition and proceeds in a bottom-up manner, as shown in Fig. 1. Smaller amounts of data are processed at successively higher levels of representation to arrive at a word identity which is output in ASCII code.

To capture data from a handwritten document, a conventional flat-bed scanner is used. The scanned image must be *segmented* into separate words (section I-B) and then a series of image processing operations is carried out to normalize the image, to make it invariant to some of the distortion processes affecting handwriting. Normalization is described in section II. The subsequent section discusses the best way of representing the useful information contained in the image.

A recurrent neural network, described in section IV is used to estimate data likelihoods for each frame of data in the representation. The likelihoods are combined in a hidden Markov model (HMM) (section V) which finds the best choice of word for the observed data. This system allows the natural incorporation of prior information about the lexicon of permitted words and about the grammar of a language. The system also allows for the recognition of words not in the lexicon, effectively giving an unlimited vocabulary, and for the rejection of words where the classification is not confident. Some of the work described in this paper has previously been published [4], [5], and for further details the reader is referred to the previous publications.

### B. Image acquisition and corpus choice

The system processes data captured from a flatbed scanner. In the database collected for this research, words were written by a single author on plain, white paper. 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 at 300 dots per inch resolution, in 256 levels of grey to produce one file per page.

The next task is to segment each page into its component words. The segmentation algorithm takes a simple approach, looking for the gaps between lines and words.

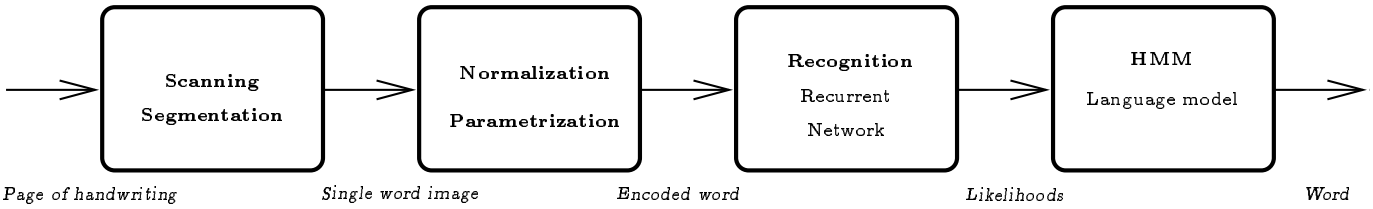



Fig. 1. A schematic of the recognition system.

Fig. 2 shows a section of a page of data with the automatic segmentation displayed. More complex segmentation algorithms are available [6] but study of segmentation is beyond the scope of this paper. Words for training are automatically labelled by alignment with the machine readable file used to prompt the writer.

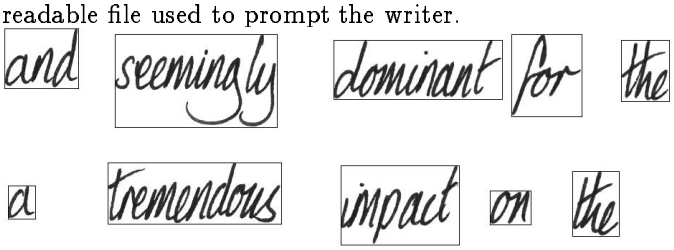


Fig. 2. A section of a page of the database, showing the automatically-detected bounding boxes.

The database was made by transcribing part of the Lancaster–Oslo/Bergen (LOB) corpus [7] of modern English. The LOB handwritten database contains images of 2360 training words, 675 validation words and 1016 test words. 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.[1]

### C. A note on results

This paper presents results for a number of experiments testing the usefulness of methods described. Since there is usually no direct, objective measure of the effectiveness of one method compared with another, methods are compared, and optimal parameter values determined, by training a complete system for each of the possible conditions and testing on an unseen test set. The final results obtained are word error rates showing the percentage of words in the test-set incorrectly classified by the whole system. The standard experimental conditions for each part of the system are described in the following sections as those parts are described, but some 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% error rate, and guessing the most likely word ('`the`') all the time would give a 93.2% error rate. The best result presented in the paper is a 6.6% error rate.

[1] The database described is publicly available by anonymous ftp: `ftp://svr-ftp.eng.cam.ac.uk/pub/data/`

Because the training of recurrent networks is found to be dependent on initial network conditions, results are subject to a certain amount of variation. Where possible, several networks have been trained under conditions identical except for the random initial values of the weights. For these runs, an estimate $\hat{\mu}$ of the mean percentage error rate and $\hat{\sigma}$, the standard error of the mean are quoted.

### D. Notation

Throughout this paper, the distinction is made between a handwritten word, and the *idea* of that word. To represent a *handwritten* word or letter, the following font is used: '*abcdef...*'; and to denote the letters or words as *concepts*, this font is used: '`abcdef...`'. The purpose of a recognition system is to transcribe '*words*' into '`words`'. The set {`a`, ..., `z`} is denoted $\Lambda$ and an arbitrary individual letter is shown $\Lambda_i$.

## II. Normalization

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 compact and informative manner. The processing to be carried out before recognition consists of two major parts — normalization and representation. The first of these attempts to remove some of those variations in the images which do not affect the identity of the word, and the second expresses the salient information in the image in a concise way, suitable for processing by a pattern recognition system.

### A. Sources of variation

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 acquiring invariance to some of these noise processes is to identify certain parameters of the handwriting that may vary between different instances of 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.

*Slant* The slant is the deviation of strokes from the vertical, varying between words and between writers.

*Slope* This is the angle of the base line of a word if it is not written horizontally.

*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. In this system, rotation is assumed to be small and is removed by a combination of slant and slope-correction transforms.

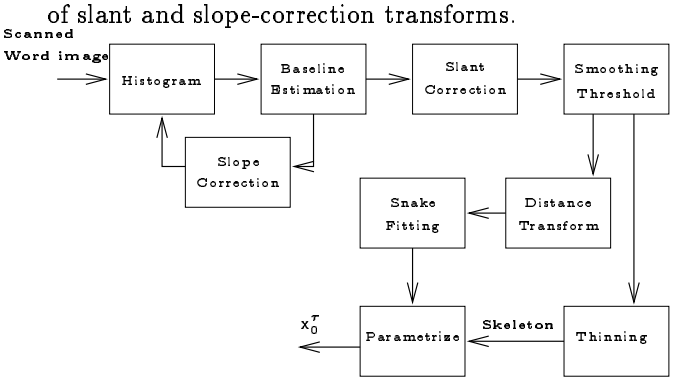



Fig. 3. A schematic of the to normalization and parametrization operations.

This system 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. Fig. 3 shows a schematic of these normalization operations, which are explained below. The processes described are illustrated for a sample word in Fig. 5.

### *B. 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 Fig. 4 — the upper and lower base lines respectively, with a centre line between the two. With these lines, the ascenders and descenders which are used by human readers in determining word shape [8] can also be identified.

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

1 Calculate the vertical density histogram, shown in Fig. 4, by counting the number of black pixels in each horizontal line in the image.

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. Even for slanted words this gives a sufficient idea of the location of descenders so that slanted base lines can be estimated by the remaining steps.

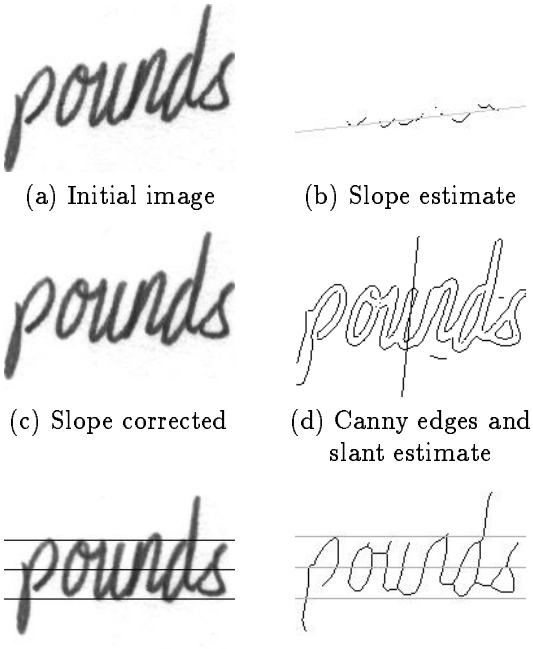

Fig. 4. Histograms, centre line and base lines of a deslanted word.

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 (Fig. 5b).

6 Reject the points far from the line of best fit and re-estimate. This is now considered to be the base line of the character.



(a) Initial image (b) Slope estimate

(c) Slope corrected (d) Canny edges and slant estimate

(e) Slant corrected with base line estimates. (f) Skeleton

Fig. 5. Successive stages in the normalization.

Given the estimate of the lower base line, the writing can be straightened to make the base line horizontal by application of a shear transform parallel to the $y$ axis (Fig. 5c). 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 estimated using a similar procedure, though this is less robust, because of the presence of '*t*' strokes, which are harder to separate from the body of text than are descenders.

### *C. Slant correction*

The slant of a word is estimated by finding the average angle of near-vertical strokes. This is calculated by finding the edges of strokes, either by finding the contour of the thresholded image or by using an edge detection filter. Both of these techniques give a chain of connected pixels representing the edges of strokes. The mode orientation of those edges close to the vertical is used as an overall slant estimate (Fig. 5d). Edge orientations are estimated with a Canny [9] edge detector.

### *D. Smoothing and thinning*

To remove noise from the image, originating from the original document, scanning defects, or applying shear transforms to discrete images, it is smoothed by convolution with a 2-dimensional Gaussian filter.

After normalization and smoothing, the image is thresholded to leave every pixel black or white. The threshold is found by simply looking for a minimum between two maxima in the grey level histogram of the image. Next an iterative, erosive thinning algorithm [10, p.153] 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 Fig. 5f.

## III. Representation

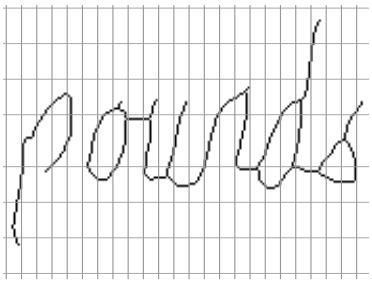
Now that the image has been reduced to a standard form, which highlights invariants of the words and suppresses spurious variations, the normalized image needs to be parametrized in an appropriate manner for input to the network which is to carry out the recognition process. From the original scanned image, all that is ultimately desired is the identity of the words on the page. 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.[2] This section 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.

### *A. 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.

In the skeleton coding scheme, the area covered by the word is first divided into a grid of rectangles. (Fig. 6a.) The vertical strips (*frames*) are of a fixed width for the whole word, a length determined by the height estimate of the character. Thus there is a variable number of these

[2] A system operating on the grey-scale image was tested [4], but was found to work less well than the system with skeleton features.



(a) Skeleton with grid

(b) A representation of the parametrized line segment data

(c) Features superimposed on the above line segment data

Fig. 6. Successive stages in the parametrization.

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. Typically there are 6 frames in a horizontal space as wide as the distance between the two baselines. 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. Section III-B describes a technique that avoids this assumption.

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 II-B 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, vertical strokes. The middle region between these two lines captures important information about the short, vertical 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 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 hooks of ascenders and descenders. Higher vertical resolutions have been tried, but performance was slightly lower because generalization of the recognition system was impaired; the storage requirement of the networks and the training data also increased.

For each of these rectangles in the grid, four bins are allocated to represent different line segment 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 a new box in the grid is entered, the section in the previous box is coded according to its angle. The box associated with this segment's $(x, y, \theta)$ values is now 'filled' (set to one). Segments which are not perfectly aligned with the discrete angles of the bins contribute to the bins representing the two closest orientations. This representation can be said to resemble the Hubel and Wiesel cells [11] which code information early in the visual cortex. These are similarly tuned to a particular spatial location and angle, but also respond to edges or bars with similar parameters. Caesar *et al.* [12] and Bengio *et al.* [13] 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.

Fig. 6b 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 '$d$' ascender which is between the vertical and 45 degrees so both these lines are shown in the corresponding boxes in Fig. 6b. Each vertical strip is referred to as a frame of data. The $t^{\text{th}}$ frame of a word is denoted $\mathsf{x}_t$; the $\tau+1$ frames representing a whole word are written $\mathsf{x}_0^\tau$.

### B. Non-uniform quantization

The above representation codes all the frames to be of equal width, and the frames are chosen by blindly drawing a grid on the word image. The width of the frames is chosen in proportion to the character height. In practice though, the ratio of writing height to width varies 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 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.

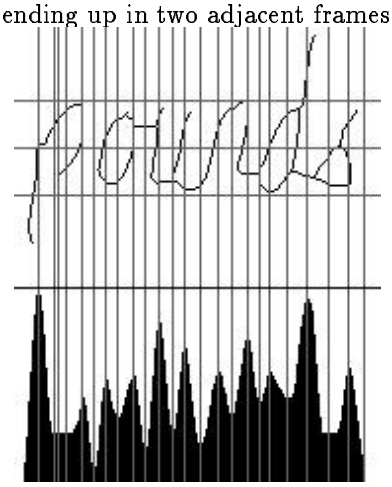


Fig. 7. The non-uniform horizontal quantization scheme superimposed on the histogram of the original word and its skeleton.

To correct these two problems, the following system has been devised. 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. Fig. 7 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, but results in better accuracy (a 17% reduction in the error rate).

### C. 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 represents low-level information about the word, and codes 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. 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 potential dots.

*Junctions* Junctions occur where two strokes meet or cross, and are easily found in the skeleton as points with more than two neighbours.

*End points* End points are points in the skeleton with only one neighbour and mark the ends of strokes, though some are artefacts of the skeletonization algorithm.

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

*Loops* Loops are found from connected-component analysis on the smoothed 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.

Each of these features can be encoded by a single number. However, 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 line segment features for each horizontal strip. Thus instead of four line segment features at each vertical position, ten features are encoded (four line segment angles, four turning points, junction and endpoint) and an extra two features (loop and dot) are associated with the whole frame. With seven horizontal bands, this increases the size of a frame from 28 bytes

$(7 \times 4)$ to 72 $(7 \times 10 + 2)$, but the additional information improves the recognition performance. Some of these features are shown in Fig. 6c, superimposed on the line segment features. Endpoints are indicated by '■' shapes, turning points by '<' and junctions by '×'.

Adding these extra features to the handwriting representation has been found to reduce the error rate significantly, as shown in table I. A further method of finding handwriting features has been developed, based on dynamic contour ('snake') fitting methods. This enables large-scale stroke-like features to be identified in the handwriting. The procedure for identifying these features is described in more detail elsewhere [4], [5].

| Features | Error rate (%) | |
|---|---|---|
| | $\hat{\mu}$ | $\hat{\sigma}$ |
| Line orientation | 60.4 | 2.7 |
| Lines and basic features | 30.1 | 1.2 |
| Lines, basic features & snakes | 23.1 | 0.9 |

TABLE I

ERROR RATES WHEN TRAINING AND TESTING NETWORKS WITH DIFFERENT TYPES OF FEATURES.

## IV. RECOGNITION

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 section. A variety of pattern recognition methods is available, and many have been used for handwriting recognition by other authors.

There are several established methods of estimating a sequence of probabilities from a sequence of data which have been applied in the fields of both speech and handwriting recognition. From the literature, two main methods emerge. Discrete or continuous hidden Markov models [14], [15] and neural network/HMM hybrids [16], [17] have been successfully applied in speech, on-line handwriting, and off-line handwriting recognition. For off-line recognition, where time information is not available, the $x$-axis is generally divided up and processed left-to-right over time. Among the neural-network approaches, feed-forward and recurrent network approaches can be distinguished. The latter have been successful in speech recognition [18], but have not previously been applied in handwriting recognition.

All three methods have been investigated in this system, as methods of estimating the data likelihoods $P(\mathrm{x}_0^\tau|\Lambda_i)$ which are used to find word likelihoods. A recurrent neural network/HMM hybrid was found [4] to perform better than a time delay neural network (TDNN) [17] hybrid or a discrete HMM. This section describes the recurrent neural networks that were used.

### A. Recurrent Neural Networks

This section describes the recurrent error propagation network which has been used as the probability distribution estimator for the handwriting recognition system. A recurrent network is well suited to the recognition of patterns, such as speech, occurring in a time-series because the same processing is performed on each section of the input stream. Here the time axis is replaced by the horizontal displacement through the word. 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. The complexity and non-linearity of a recurrent network gives a more general model than a pure HMM.

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 (section IV-B) 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 $P(\Lambda_i|\mathrm{x}_0^t)$.

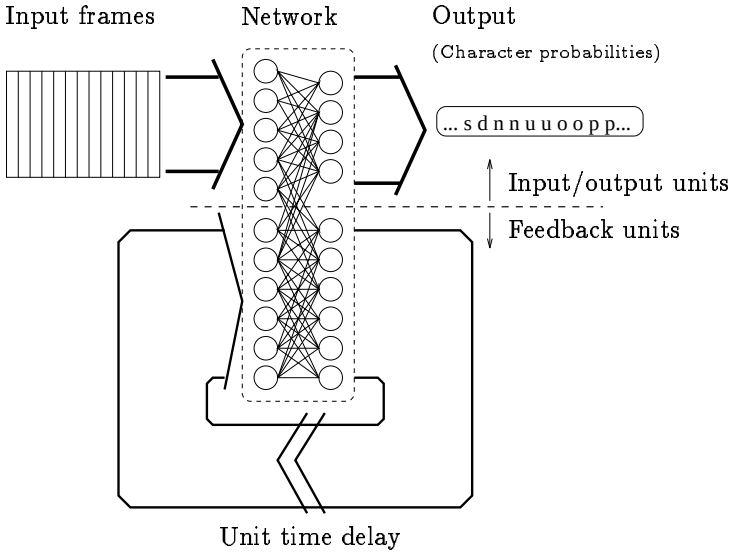



Fig. 8. A schematic of the recurrent error propagation network. For clarity only some of the units and links are shown.

The recurrent network architecture used here is a single layer of standard perceptrons with nonlinear activation functions [19]. 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 $b_i$:

$$o_i = f_i(\{\sigma_j\}), \tag{1}$$

$$\sigma_i = b_i + \sum a_j w_{ij}. \tag{2}$$

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 (Fig. 8). The remaining input units accept one frame of parametrized input and the remaining output units estimate letter probabilities for each of the 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 $f_i(\{\sigma_j\}) = \frac{e^{\sigma_i}}{\sum_j e^{\sigma_j}}$.

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 1 and 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 [16, 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 $P(\Lambda_i|\mathrm{x}_0^t)$. It will be seen later (section V) how these probabilities can be combined to obtain word likelihood estimates in a Markov model framework.

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.

Further assimilation of context information is made possible by presenting several frames of data to the network at each time step (as in a TDNN). This gives a 'sliding window' effect highlighting short-term correlations. The effect of this can be seen in table II.

| Frames | Error rate (%) | |
|---|---|---|
| | $\hat{\mu}$ | $\hat{\sigma}$ |
| 1 | 23.1 | 0.9 |
| 2 | 19.7 | 1.0 |
| 3 | 16.3 | 0.4 |

TABLE II

EFFECT OF PRESENTING MORE THAN ONE FRAME TO THE NETWORK AT EACH TIME STEP.

### B. Training

The network is trained using standard connectionist techniques, in particular the generalized delta rule [19]. This determines how the weights should be changed to improve the network approximation to a set of desired target outputs. Since the network is recurrent, 'back-propagation through time' [18] is used, which treats the network at successive instants as successive layers in a multilayer network, with as many layers as there were frames of input.

### C. Network targets

For training, target values must be given, against which the network output can be compared. This allows calculation of the error in the outputs and thus of the weight updates. Target values are 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. The database already has a word label associated with each word image (section I-B). However, the labelling of individual frames with the corresponding class is difficult. Unlike the segmentation problem of many other handwriting systems, 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 can only be truly carried out after accurate recognition. For some problems, such as speech recognition, hand-labelled data has been used as an initial training set. This has been avoided here by using a 'bootstrap' scheme which derives an approximate segmentation from a simple 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 and would give better results, but would be laborious and would need to be done again for new databases.

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 are 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 others. These targets are only used for preliminary training. Re-estimated targets are used to achieve greater performance. The re-estimation process is described in section V-B.

### D. Generalization

One problem with network training is to obtain the optimum solution to the trade-off between training and generalization. This well-known problem can 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,

or the original points were not generated by such a polynomial, the values on the curve between will correspond badly to the values of any subsequently observed datapoints. The curve is over-fitted and generalization to the new data 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 [16, 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, using a technique like weight decay or adding noise to weights. 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 (minimum word error rate) as will ultimately be used when testing the network. If a network is trained on a data set, 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 data set tasks, *e.g.* large-vocabulary speech recognition [16].

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 training has been stopped, the network with the lowest error rate is reloaded, and tested on the test set — a third set which consists of data not previously presented to the network.

### *E. Network size*

Adding more feedback units to the network increases its capacity, but the error rate of the system is seen to fall as the number of feedback units is increased, as shown in table III. Thus it can be seen that early stopping ensures that generalization does not suffer when the network size is increased. Because of the increased training time associated with larger networks, no network above 160 feedback units has been trained, though it is likely that the recognition rate would be still higher for larger networks. Elsewhere in this work, we present results only for networks with 80 feedback units.

| Number of feedback units | Error rate (%) | |
|---|---|---|
| | $\hat{\mu}$ | $\hat{\sigma}$ |
| 0 | 29.8 | 0.6 |
| 10 | 25.1 | 0.6 |
| 20 | 26.8 | 3.4 |
| 40 | 24.5 | 2.5 |
| 80 | 23.1 | 0.9 |
| 160 | 18.5 | 3.3 |

TABLE III

THE EFFECT ON ERROR RATE OF CHANGING THE NUMBER OF FEEDBACK UNITS IN THE NETWORK. (ONE FRAME OF INPUT AT EACH TIME STEP.)

## V. HIDDEN MARKOV MODELLING

The neural network gives an estimate of the character posterior probability given the data, $P(\Lambda_i|\mathrm{x}_t)$ for each frame of input $\mathrm{x}_t$ and for each character class $\Lambda_i$. This section deals with the process of deriving the best word choice from a sequence of these frame probability distributions by the use of HMMs. 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.

### *A. 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 represented by the data. 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, the cursively-written word '*day*' may also be read '`clay`', so a good recognition system should return high probabilities for both, and much smaller probabilities for dissimilar words. The probability distribution to be determined is $P(W|\mathrm{x}_0^\tau)$ across all words $W$ in the lexicon $\mathcal{L}$, given the input data $\mathrm{x}_0^\tau$.

The individual frame probabilities are combined to produce word probabilities using a HMM [20]. A separate HMM is created for each word in the known lexicon, with one state representing each letter. Fig. 9 shows a model for the word '`one`'. If there are $N$ states, the set of states

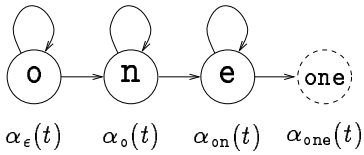



Fig. 9. A simple Markov model for the word 'one' with one state per letter.

is $Q = \{q_r : r = 0, \ldots, N-1\}$, corresponding to the letters $\mathtt{L}(q_r)$. 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 HMM can allow transitions between any pair of states, but in handwriting, the order of the letters is known, so the model is restricted to allow only the two transitions shown in Fig. 9. To use the model, transition probabilities, assumed to be independent of the time, are assigned to each of the permitted transitions: $a_{p,r} = P(S_{t+1} = q_r | S_t = q_p)$, $a_{p,r} = 0$ except when $r = p$ or $r = p + 1$. For the model to be a true Markov model, the transition probabilities are dependent only 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 frames representing the letters $\mathtt{L}(S) = (\mathtt{o},\mathtt{o},\mathtt{o},\mathtt{o},\mathtt{n},\mathtt{n},\mathtt{n},\mathtt{e},\mathtt{e},\mathtt{e},\mathtt{e},\mathtt{e})$. The model is a *hidden* Markov model because $S$ is not directly observable, only inferred.

In the generation process which is to be modelled, the system produces a frame of handwriting data $\mathtt{x}_t$ at each time step. The data are part of the representation of the letter signified by the current state. The data are assumed to occur according to a probability distribution $P(\mathtt{x}_t | \mathtt{L}(S_t))$. With this information, an expression can be derived for the probability of a word, given a particular observation sequence $\mathtt{x}_0^\tau$.

The posterior probability of a word $W$ can be rewritten using Bayes' rule, deriving the denominator from the fact that the word must be in the vocabulary:

$$P(W|\mathtt{x}_0^\tau) = \frac{P(\mathtt{x}_0^\tau|W)P(W)}{\sum_{W \in \mathcal{L}} P(\mathtt{x}_0^\tau|W)P(W)}. \quad (3)$$

Summing over all state sequences, $S$, representing the word $W$,

$$P(\mathtt{x}_0^\tau|W) = \sum_{S \in \mathtt{S}(W)} P(\mathtt{x}_0^\tau|S)P(S), \quad (4)$$

where, according to the Markov model, 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:

$$P(S) = \pi_{S_0} \prod_{t=0}^{\tau-1} a_{S_t, S_{t+1}}. \quad (5)$$

Here, $\pi_r = 0$ for all states except the first ($\pi_0 = 1$), so each state sequence is constrained to start with the first letter of a word. Expanding the other term in equation 4,

$$P(\mathtt{x}_0^\tau|S) = P(\mathtt{x}_0|S) \prod_{t=1}^{\tau} P(\mathtt{x}_t|S, \mathtt{x}_0^{t-1}). \quad (6)$$

If it is assumed that the emission probability is conditionally independent of preceding or following states, given the current state, and indeed is dependent only on the letter that state represents, this reduces to

$$P(\mathtt{x}_0^\tau|S) = \prod_{t=0}^{\tau} P(\mathtt{x}_t|\mathtt{L}(S_t), \mathtt{x}_0^{t-1}). \quad (7)$$

By applications of Bayes' rule, it can be seen that:

$$P(\mathtt{x}_t|\mathtt{L}(S_t), \mathtt{x}_0^{t-1}) = \frac{P(\mathtt{L}(S_t)|\mathtt{x}_0^t)P(\mathtt{x}_0^t)}{P(\mathtt{L}(S_t), \mathtt{x}_0^{t-1})}. \quad (8)$$

Now $P(\mathtt{L}(S_t)|\mathtt{x}_0^t)$ is the posterior character probability estimated by the recurrent network, with dependence on the observation history $\mathtt{x}_0^{t-1}$. $P(\mathtt{x}_0^t)$ is the probability of the first few frames of data, which is the same for all words. $P(\mathtt{L}(S_t), \mathtt{x}_0^{t-1})$ is assumed to be approximately proportional to $P(\mathtt{L}(S_t))$, the prior probability of a frame belonging to the class $\mathtt{L}(S_t)$. This probability can be estimated by counting the number of frames in each class according to the labels of the training set.

Thus the probability $P(\mathtt{x}_0^\tau|W)$ of the data given a word can be calculated to within a constant $K(\mathtt{x}_0^t)$ which does not depend on $W$, and normalized with the word prior, to give posterior word probabilities:

$$L(\mathtt{x}_0^\tau|W) \approx K(\mathtt{x}_0^t) \sum_S \left( \prod_{t=0}^{\tau} \frac{P(\mathtt{L}(S_t)|\mathtt{x}_0^t)}{P(\mathtt{L}(S_t))} \right) \left( \pi_{S_0} \prod_{t=0}^{\tau-1} a_{S_t, S_{t+1}} \right) \quad (9)$$

$$P(W|\mathtt{x}_0^\tau) = \frac{P(W)L(\mathtt{x}_0^\tau|W)}{\sum_W P(W)L(\mathtt{x}_0^\tau|W)}. \quad (10)$$

These calculations can be performed efficiently by recursive evaluation of intermediate, 'forward' probabilities. 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:

$$\alpha_r(t+1) = \sum_{p \in Q} \alpha_p(t) \frac{P(\mathtt{L}(S_t)|\mathtt{x}_0^t)}{P(\mathtt{L}(S_t))} a_{p,r} \quad (11)$$

until all have been processed. At this point the final state (dashed in Fig. 9) contains $\alpha_n(\tau+1) = L(\mathtt{x}_0^\tau|W)$, the likelihood that the data $\mathtt{x}_0^\tau$ represented the word of this model.

The maximum likelihood word, $\text{argmax}_W P(W)L(\mathrm{x}_0^\tau|W)$, is returned as the recognition result.

In practice, most of the state sequences $S$ are improbable, contributing little to the probability of the word, and there will be a small number of similar state sequences which are much more likely than all the others. 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 4 is:

$$P(\mathrm{x}_0^\tau|W) \quad \propto \quad P(\mathrm{x}_0^\tau|S^*)P(S^*). \tag{12}$$

Taking only the most likely state sequence when decoding is called *Viterbi decoding*. In this case, decoding is simpler and faster; a different set of likelihoods, $\alpha'$, is stored:

$$\alpha'_r(t+1) \quad = \quad \max_p \alpha'_p(t) P(\mathrm{x}_t|\mathrm{L}(q_p))a_{p,r}. \tag{13}$$

### B. Targets

Using the $\alpha_r(t)$ it is possible to calculate a probability distribution across the states of the model for each frame — an average of all the state sequences. This is in effect a label, describing which state the frame belongs to, but it is a soft label, reflecting the ambiguity of some frames which are unclear, contain no information or actually contain parts of more than one character. Nevertheless, this soft label can be used as a target for the recurrent network and enables the iterative retraining of the hybrid system towards progressively better targets [21].

### C. Multistate letter models

It will be seen that the above model, with only one state per letter, loses much of the discrimination necessary to distinguish between characters because the model is not attempting to discriminate between the different parts of a character, but only determine to which character a frame belongs. Thus a set of frames fed into the model would give the same probability regardless of the order (except for the context-assimilation methods described before). Instead, if each letter is now considered as the horizontal concatenation of two or more letter parts, the network can be trained to estimate probabilities for each of these letter parts for every frame, and each letter in the HMM can be represented by one state for each letter part. This has two beneficial effects. Firstly, the network no longer has to associate the same class output with such dissimilar frames as, say the left and right halves of a '$d$', and secondly a time ordering is forced upon these frames, increasing the difference between '$d$' and '$b$', say. The results of table IV demonstrate the reduced error given by this improvement.

### D. Rejection

In many applications, there may be a high cost associated with mis-recognizing a word, and an alternative to machine recognition may be available. In these cases, we would like the system to be able to identify which words have been well recognized, where the answer can be trusted. Poorly recognized words could be dealt with in a different way. Thus in a document transcription system, the user may be prompted to review the words whose classification was uncertain, or in a postal sorting application, the error rate could be reduced to an acceptable value by having all unclear addresses sent to human sorters.

| Number of parts per letter | Error rate (%) | |
|---|---|---|
| | $\hat{\mu}$ | $\hat{\sigma}$ |
| 1 | 16.3 | 0.4 |
| 2 | 10.7 | 0.5 |
| 3 | 9.8 | 1.4 |

TABLE IV

A COMPARISON OF ERROR RATES FOR HYBRIDS BREAKING EACH LETTER INTO A NUMBER OF SEPARATELY-RECOGNIZED PARTS. (3 FRAMES PRESENTED AT EACH TIME STEP)

In this system, words are rejected on the basis of the likelihood ratio between the top two words. If these likelihoods are similar, small perturbations of the input could reverse the ranking, showing that the classification is not certain. Thus, if the log likelihoods (divided by the number of frames in the word for normalization) of the top two words are closer than some threshold, the classification is rejected as uncertain. By varying the threshold chosen, the error rate can be traded off against the rejection rate, as shown in Fig. 10. An error rate of 1% can be achieved when rejecting 15.5% of the words (high threshold), or 5.2% when rejecting 5.0% (moderate threshold).

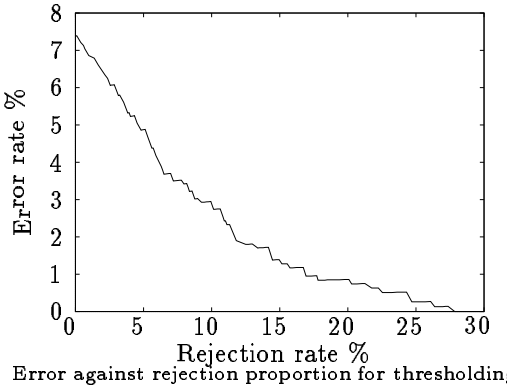



Fig. 10. Error against rejection proportion for thresholding on difference in normalized log likelihood.

## VI. LANGUAGE MODELLING

One area where great gains in recognition accuracy can be made is by modelling the structure of the language being processed [22, 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 of permitted words. This section investigates the choice of vocabulary, and techniques to improve performance and to recognize words not in the vocabulary.

### A. 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 any application the lexicon is dictated by the task. For instance,

when reading cheques the vocabulary would be around 35 words, comprising numbers, currency units, 'and' and so forth. When transcribing longhand documents, the vocabulary would need to be tens of thousands of words, to cover 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. However, 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 or more digits are uncertain, the vocabulary will reflect this uncertainty and increase to include other city names with zip codes that match the digits that were reliably read. A postal system can reasonably be tested with a few hundred words, the vocabulary for each word being dynamically chosen from a longer list by zip code. It should also be noted that the resulting vocabularies consist of longer, less-confusable words than vocabularies of similar size designed for transcribing more general texts.

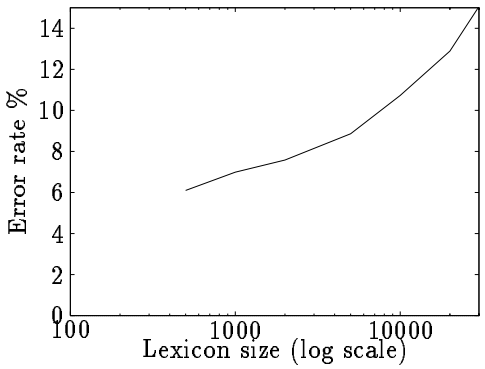



Fig. 11. A graph of error rates for different lexicon sizes with full coverage, for a sample network.

Experiments were conducted to test the effect of lexicon size on error rate. Fig. 11 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. (The most frequent words tend to be short and thus most easily confusable.) Thus each of these lexica fully *covers* the test set vocabulary cf. section VI-B. This experiment is similar to one described by Schenkel *et al.* [17] who supplement the test-set lexicon with randomly chosen words. The 500 word error rate is lower than those quoted before, because of the smaller lexicon size, but larger lexica give more errors because of the increase in similarity between the permitted words.

### *B. Coverage and out-of-vocabulary words*

In most applications, there is a chance that the recognizer will be asked to identify a word that is not in the lexicon. Thus a system must be able either to recognize words not in the vocabulary (as described in the next section), flag that there was an out-of-vocabulary word for human proof-reading using a rejection criterion as above, or be condemned to incorrectly classify these non-words.

In the case where out-of-vocabulary words are not errors, and the system should be able to identify 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 V and figure 14. In each case, the lexicon is made of the $n$ most frequent words from the LOB corpus, and thus no longer contains all the words in the test set. The coverage proportions are compared with the performance of an 80-unit network.

These results are shown by the solid line in Fig. 14. 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 bound, the in-lexicon error rate is also shown in table V. This is the proportion of in-vocabulary words — those words that the system could have correctly identified given the lexicon — which are misclassified. This rises from 0% with three words, since all words 'the', 'of' and 'and' are correctly classified, to 11.7% with a 30,000 word vocabulary.

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.

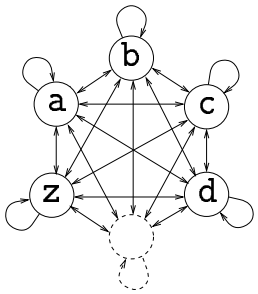



Fig. 12. A non-word Markov model showing the connections between some of the 26 letter models.

One such strategy is to create a non-word Markov model, as shown in Fig. 12. Each circle represents a letter model, with one or more states. The initial distribution $\pi$ is uniform across the first states of each letter model. The probabilities are combined to find the $\alpha'$ probabilities as before,

| Lexicon size $n$ | Coverage (%) | | Error rate (%) | |
|---|---|---|---|---|
| | LOB | Test | Test set | In lexicon |
| 3 | 9.9 | 13.1 | 86.9 | 0.0 |
| 10 | 15.5 | 23.4 | 77.5 | 3.8 |
| 30 | 28.4 | 36.0 | 65.8 | 4.9 |
| 100 | 44.6 | 50.4 | 55.0 | 9.8 |
| 300 | 58.6 | 60.4 | 44.6 | 8.3 |
| 1000 | 72.6 | 72.5 | 34.1 | 9.1 |
| 3000 | 86.6 | 85.6 | 22.0 | 9.9 |
| 10000 | 93.8 | 94.2 | 16.1 | 11.0 |
| 30000 | 99.0 | 99.3 | 87.7 | 11.7 |

TABLE V

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. ERROR RATES FOR A TESTING NETWORK ARE SHOWN AS PERCENTAGES OF WORDS IN THE TEST SET AND OF THOSE TEST SET WORDS IN THE LEXICON.

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 by the Viterbi algorithm and the letters corresponding to its state sequence can be printed out. A letter *bigram* can be created, detailing the probability of making a transition from one letter to another, as observed on a corpus of text by counting how often each possible pair of adjacent letters occurs. These probabilities can be multiplied in to the state sequence probability, biassing the answers towards word-like strings of characters. Table VI 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, Ullman and Flash [2].

| Bigram | Error rate (%) |
|---|---|
| No | 49.7 |
| Yes | 41.1 |

TABLE VI

ERROR RATES FOR THE NON-WORD MODEL.

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 been found that the normalized likelihood used as a rejection criterion gives a good measure of separation. Fig. 13 shows the normalized likelihoods from the word and non-word models plotted for words classified according to whether they were correctly or incorrectly classified by each model. It is necessary to discriminate between the two outcomes where one model is right and the other wrong. If both are right or both wrong, the decision is immaterial. A straight line decision boundary separates the two cases well, and corresponds to a non-word penalty of 0.22.

Fig. 14 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. 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. As the lexicon size increases and the coverage rises toward 100%, the advantage of using the out-of-vocabulary model becomes less significant but the recognition rate remains above the lexicon-only recognition rate until 30,000 words (87%).

## VII. CONCLUSIONS

This paper 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. An 87% 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. Single author error rates for other systems include (for various lexicon sizes): 52% [23], 50% [2] and 30% [3].

Enhancements in normalization and in the detection and representation of features have led to reduced error rates. A hybrid system using recurrent neural networks and HMMs was found to perform better than a discrete probability HMM system or a TDNN system. The system performance was increased by allowing more frames of context in the network inputs and by 'untying' the output distributions, to distinguish between the different parts of each character.

The error rate has been reduced to 9.8%. (Averaged over four 80-unit networks. An error rate of 6.6% has been achieved with a 160-unit network.) It has been shown that the system can recognize 59% of words without any use of a lexicon, and that a model for words not in the system's vocabulary can increase the recognition rate beyond that otherwise obtained. Lower error rates can be achieved by

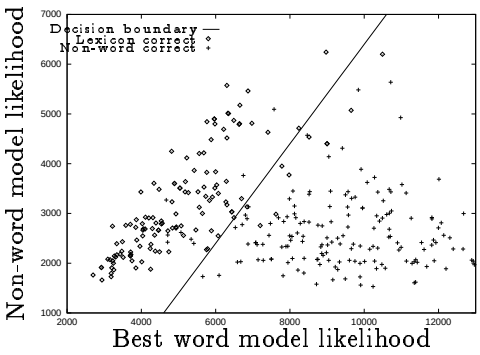



Fig. 13. Words recognized either by 300 word lexicon or non-word model plotted with non-word normalized likelihood against lexicon normalized likelihood.

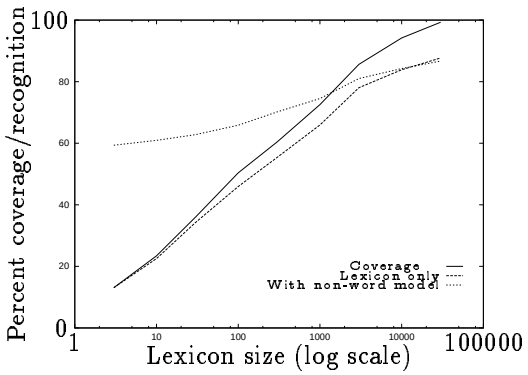



Fig. 14. A graph of recognition rate against lexicon size, with and without modelling out-of-vocabulary words. The test set coverage of the lexica is also shown.

the use of rejection criteria.

Further recognition rate improvements could be expected from techniques such as context-dependent modelling which have benefitted a similar recurrent-network hybrid speech recognition system [24]. A larger training set would also bring a slight improvement because there is still a gap between training and test-set performance. The use of probabilistic grammars, not discussed here, has also been found to give better results.

## References


[1] M. Gilloux, J.-M. Bertille, and M. Leroux, "Recognition of handwritten words in a limited dynamic vocabulary," in *Third International Workshop on Frontiers in Handwriting Recognition*, pp. 417–422, CEDAR, SUNY Buffalo., May 1993.

[2] S. Edelman, S. Ullman, and T. Flash, "Reading cursive script by alignment of letter prototypes," *International Journal of Computer Vision*, vol. 5, no. 3, pp. 303–331, 1990.

[3] B. A. Yanikoglu and P. A. Sandon, "Off-line cursive handwriting recognition using style parameters," Tech. Rep. PCS-TR93-192, Dartmouth College, NH, April 1993.

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

[5] A. W. Senior and F. Fallside, "Using constrained snakes for feature spotting in off-line cursive script," in *International Conference on Document Analysis and Recognition*, pp. 305–310, IEEE, October 1993.

[6] R. G. Casey and E. Lecolinet, "Strategies in character segmentation: A survey," in *International Conference on Document Analysis and Recognition*, vol. 2, pp. 1028–1031, August 1995.

[7] S. Johansson, E. Atwell, R. Garside, and G. Leech, "The tagged LOB corpus," tech. rep., Norwegian Computing Centre for The Humanities, Bergen, 1986.

[8] H. Bouma, "Visual recognition of isolated lower case letters," *Vision Research*, vol. 11, pp. 459–474, 1971.

[9] J. F. Canny, "A computational approach to edge detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 8, pp. 679–698, 1986.

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

[11] D. H. Hubel and T. N. Wiesel, "Receptive fields,binocular interaction and functional architecture in the cat's visual cortex," *Journal of Physiology*, vol. 160, pp. 106–154, 1962.

[12] T. Caesar, G. Joachim, A. Kaltenmeier, and E. Mandler, "Recognition by handwritten word images by statistical methods," in *Third International Workshop on Frontiers in Handwriting Recognition*, pp. 409–416, CEDAR, SUNY Buffalo., May 1993.

[13] Y. Bengio, Y. Le Cun, and D. Henderson, "Globally trained handwritten word recognizer using spatial representation, convolutional neural networks and hidden Markov models," in *Advances in Neural Information Processing Systems* (J. D. Cowan, G. Tessauro, and J. Alspector, eds.), no. 6, pp. 937–944, Morgan Kaufmann, 1994.

[14] P. C. Woodland, J. J. Odell, V. V. Valtchev, and S. J. Young, "Large vocabulary continuous speech recognition using HTK," in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 2, pp. 125–128, Apr. 1994.

[15] J. B. Bellegarda, D. Nahamoo, K. S. Nathan, and E. J. Bellegarda, "Supervised hidden Markov modeling for on-line handwriting recognition," in *International Conference on Acoustics, Speech and Signal Processing*, vol. 5, pp. 149–152, 1994.

[16] H. Bourlard and N. Morgan, *Connectionist Speech Recognition: A Hybrid Approach*. Kluwer, 1993.

[17] M. Schenkel, I. Guyon, and D. Henderson, "On-line cursive script recognition using time delay neural networks and hidden Markov models," in *International Conference on Acoustics, Speech and Signal Processing*, vol. 2, pp. 637–640, 1994.

[18] A. Robinson, "The application of recurrent nets to phone probability estimation," *IEEE Transactions on Neural Networks*, vol. 5, Mar. 1994.

[19] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition* (D. E. Rumelhart and J. L. McClelland, eds.), vol. 1, ch. 8, pp. 318–362, Bradford Books, 1986.

[20] L. R. Rabiner and B. H. Juang, "An introduction to hidden Markov models," *IEEE ASSP magazine*, vol. 3, pp. 4–16, January 1986.

[21] A. Senior and T. Robinson, "Forward-backward retraining of recurrent neural networks," in *Advances in Neural Information Processing Systems*, no. 8, pp. 743–749, MIT Press, 1996.

[22] A. Waibel and K.-F. Lee, eds., *Readings in Speech Recognition*. Morgan Kaufmann, 1990.

[23] R. M. Božinović and S. N. Srihari, "Off-line cursive word recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 11, pp. 68–83, January 1989.

[24] D. Kershaw, T. Robinson, and M. Hochberg, "Context-dependent classes in a hybrid recurrent network–HMM speech recognition system," in *Advances in Neural Information Processing Systems* (D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, eds.), no. 8, MIT Press, 1996.