6-7-1993

# Off-line Cursive Handwriting Recognition Using Style Parameters

Berrin A. Yanikoglu
*Dartmouth College*

Peter A. Sandon
*Dartmouth College*

# OFF-LINE CURSIVE HANDWRITING RECOGNITION USING STYLE PARAMETERS

Berrin A. Yanikoglu
Peter A. Sandon

Technical Report PCS-TR93-192

# Off-line Cursive Handwriting Recognition Using Style Parameters

Berrin A. Yanikoglu
Peter A. Sandon

Department of Mathematics and Computer Science
Dartmouth College
Hanover, NH, 03755

June 7, 1993

**Abstract**

We present a system for recognizing off-line cursive English text, guided in part by global characteristics of the handwriting. A new method for finding the letter boundaries, based on minimizing a heuristic cost function, is introduced. The function is evaluated at each point along the baseline of the word to find the best possible segmentation points. The algorithm tries to find all the actual letter boundaries and as few additional ones as possible. After size and slant normalizations, the segments are classified by a one hidden layer feedforward neural network. The word recognition algorithm finds the segmentation points that are likely to be extraneous and generates all possible final segmentations of the word, by either keeping or removing them. Interpreting the output of the neural network as posterior probabilities of letters, it then finds the word that maximizes the probability of having produced the image, over a set of 30,000 words and over all the possible final segmentations. We compared two hypotheses for finding the likelihood of words that are in the lexicon and found that using a Hidden Markov Model of English is significantly less successful than assuming independence among the letters of a word. In our initial test with multiple writers, 61% of the words were recognized correctly.

# 1 Introduction

Handwriting recognition is the task of interpreting an image of handwritten text. We use the terms *handwriting* and *cursive handwriting* interchangeably, in their most general sense, i.e. successive letters of a word may or may not be joined. Strictly connected cursive handwriting and discretely written handwriting are referred to as *pure cursive handwriting* and *handprinting*, respectively.

The image of the handwriting can be obtained by flat-bed scanning after the text has been written (off-line), or by means of digitizing tablets or stylus pens as the text is being written (on-line). On-line devices can capture dynamic characteristics of the handwriting, such as the number and order of strokes, and velocity and pressure changes, but the information must be processed in real time. Off-line approaches have less information available for recognition, but have no real-time constraints [TSW90].

The system we are developing is for off-line recognition of cursive, handwritten text. The input to the system is the scanned image of a page of cursive handwriting, written roughly along the rulings. Currently, the system is trained to recognize lowercase letters only, but the writing style is not constrained in any other way.

Section 2 of this paper describes the process of finding the text line boundaries. For each text line, we extract parameters that characterize the style of the writing (*style parameters*), as explained in Sections 3 and 4. Sections 5 and 6 describe the algorithms that isolate words on a line and letters in a word. This latter step, called word segmentation, or simply segmentation, involves the use of style parameters to accurately locate letter boundaries and is key to the success of the overall system. Section 7 describes the process of recognizing segmented letters. Two different methods for word recognition are described in Section 8. Experimental results are described in Section 9.

# 2 Finding text line boundaries

To recognize the page, we first find the text line boundaries and then recognize each text line in turn. These boundaries are not always straight lines, due to overlap as shown in Figure 1.

To find the exact boundary between two text lines, we compute the histogram of pixel densities along the horizontal scan lines. Using the smoothed histogram, we roughly locate the baseline (the line where the letters sit) of each text line and apply a contour following

algorithm to find the exact boundaries. The exact boundary between two text lines can be thought of as the path of a bug trying to reach the right hand side of the page, starting from the left hand side of the top baseline, while staying below the text of the top line. It is based on the algorithm that finds the outline of an object, in [DH73]. Since the algorithm employs 8-connectedness, the image is first smoothed using a 3x3 Gaussian mask to avoid problems caused by thin, slanted strokes.

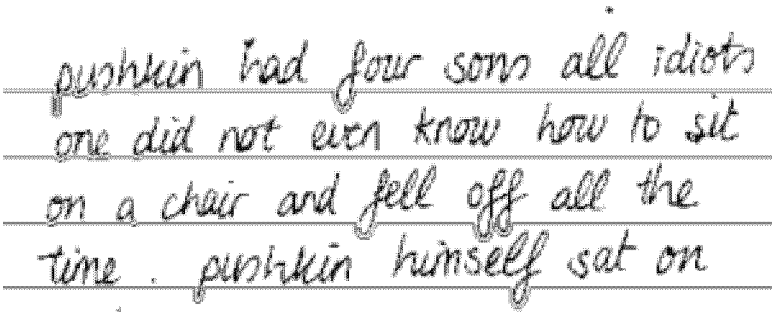


Figure 1: Segmentation of the page into text lines.

# 3 Finding the reference lines

The *reference lines* of a text line [SR87] are the four horizontal lines that mark the top of the ascenders, the top of the main bodies of letters, the baseline and the bottom of the descenders. They will be referred to as l1, l2, l3 and l4, from top to bottom. Figure 2 and Figure 3 show the reference lines of two text lines as found by the system. Note that l1 and l4 are the points where the ascenders and descenders would approximately reach, if there were any. The algorithm that finds the reference lines is as follows:

1. Compute the horizontal pixel density histogram, $h_0$, of the text line.
2. Compute the smoothed histogram, $s_0$, as

$$s_0(x) = \sum_{i=-2}^{i=2} h_0(x+i)$$

3. Compute the "derivative" of the smoothed histogram, $d_0$, as

$$d_0(x) = s_0(x) - s_0(x-5)$$

   This represents the pixel density difference of successive blocks of 5 scan lines.

4. Starting from the top of the line, let $l1$ be the first point where $s_0$ is non-zero.
5. Starting from the bottom of the line, let $l4$ be the first point where $s_0$ is non-zero.
6. From the midpoint of $l1$ and $l4$ up to $l1$, redefine $l1$ to be the first point where $s_0$ becomes zero.
7. From the midpoint of $l1$ and $l4$ down to $l4$, redefine $l4$ to be the first point where $s_0$ becomes zero.
8. Let the peak $p$ be the point where $s_0$ has its maximum.
9. Let $l3$ be the point of local minimum of $d_0$ between $p$ and $l4$.
10. Let $l2$ be the point of local maximum of $d_0$ between $l1$ and $p$.
11. If $l1$ and $l2$ are very close, set $l1$ to $(l2 - max((l4 - l3), (l3 - l2)))$.
12. If $l3$ and $l4$ are very close, set $l4$ to $(l3 + max((l2 - l1), (l3 - l2)))$.

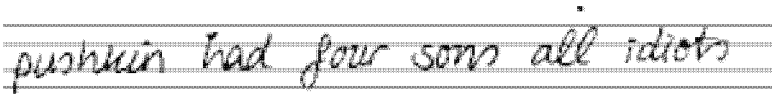

Figure 2: Reference lines l1, l2, l3 and l4, from top to bottom.

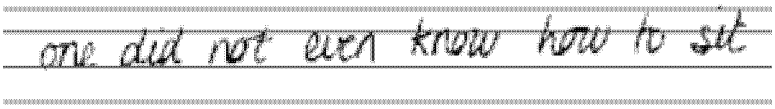

Figure 3: l4 is adjusted since there were no descenders.

If the page was not properly positioned on the scanner, or if the writing does not follow the rulings, the horizontal pixel density histogram does not show a significant peak. In that case, we compute histograms angled at -10 and 10 degrees from the horizontal and interpret the angle corresponding to the histogram that shows the most significant peak as the line skew. We then correct the skew by shearing, before applying the above algorithm.

# 4 Extracting style parameters

We extract parameters that characterize the style of the writing and use them in decisions throughout the system. Parameterized decisions are essential for writer independent recognition. Some of these parameters are the dominant slant of the writing, the thickness of the pen and, the average values of inter-word spaces and letter widths.

Most of the style information, such as the average heights of ascenders, descenders and letter bodies, is obtained from the reference lines. These are used, in particular, in the size normalization step to decide whether a letter has an ascender, a descender or neither, and to scale it accordingly.

We estimate the thickness of the pen and the average inter-word space by analyzing the run-length histogram of the text. Then, using the vertical pixel density histogram, the average body height as an initial estimate, and the estimated pen thickness, we estimate the average letter width. Finally, we find the dominant slant of the writing by computing the slant histogram using edge operators and finding the most common slant within -30 and +30 degrees from the vertical. The dominant slant and the average letter width are particularly useful in finding the letter boundaries.

For most of these parameters, it would also be useful to know not only the average values, but also the variances in order to assess the reliability of our decisions. For example, in handwriting such as that in Figure 4, it is important to know that there is a large variance in letter slants so as not to rely on the dominant slant which is not well defined.

Figure 4: Writing without a consistent slant.

# 5 Finding word boundaries

Before applying the segmentation algorithm, we find the connected components of the text line, using a region growing algorithm. A distance greater than the average letter width

between two components is considered to be a word boundary. Finding word boundaries in this way is not completely reliable but works quite well.

In general, any algorithm that uses a threshold distance is prone to failure since two words may be arbitrarily close to each other, while the letters of a single word may be significantly apart. This problem implies that the task of finding the word boundaries should be extended to the word recognition stage where, for example, matching the image to prefixes and suffixes in the dictionary would solve the second part of the problem.

Connected component analysis also segments consecutive, disconnected letters, as a first step in the letter segmentation process.

# 6 Finding letter boundaries

We use *separator lines* in one of six fixed angles (-20, -10, 0, 10, 20 and 30 degrees clockwise from the vertical[1]) as letter boundaries, as shown in Figure 5. The gaps and ligatures between letters are detected using pixel density histograms of the text line along these six angles.

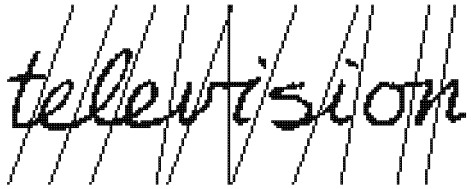

Figure 5: Segmentation of the word *television* using separator lines.

Note that although this sample word is oversegmented, the segmentation is *potentially correct* in the sense that all the actual letter boundaries are found. The oversegmentation is partly due to the ambiguity of cursive script segmentation: some letter pairs (digraphs) or triples (trigraphs) are indistinguishable from a single letter at the image level, as with the letter **w** and the digraphs **ui** and **iu**. Since we are using a strictly bottom-up approach, our algorithm tries to find all possible letter boundaries and thus oversegments these letters. Section 6.2 describes how we handle oversegmentation.

[1] All angles are clockwise from the vertical, unless otherwise specified.

Depending on the dominant slant of the writing, we choose a subset of four of the separator line angles to use in segmenting the word. For example, if the writing has a 20 degree slant, we use 0, 10, 20 and 30 degree angles. Using each of the corresponding angled histograms, we segment the word using separator lines at the angle of that histogram, as explained below. Three segmentations of the above sample, as computed by the segmentation algorithm, are shown in Figure 6.

Figure 6: Three segmentations of the word *television* using separator lines with 0, 10 and 20 degree angles.

The beginning of the first word in the text line constitutes the first segmentation point for all four angles. Given one segmentation point, we find the next one by evaluating a cost function, described in Section 6.1, at each point along the baseline to the right of that point and choosing the point with the minimum cost. The cost at a point depends on the distance of that point to the previous segmentation point, the average letter width, the number of text pixels cut by the separator line and the height of the cut. This last feature helps to

locate the ligatures between letters since they are usually connected at their bases. We make use of the style parameters in this process. For example, we normalize the amount of text cut by dividing it by the average pen thickness.

For determining the final letter boundaries, we first take the union of these four sets of segmentation points. This is necessary since a letter boundary may be detected in only one of the segmentations. On the other hand, the union of the segmentation points usually contains more than one segmentation point, corresponding to the same letter boundary. We use a threshold of half the average letter width to identify segmentation points that are too close and might correspond to the same letter boundary. From among these, we identify the ones that do correspond to the same boundary and choose the one with the lowest cost as the final segmentation point. The details of the algorithm is described below.

1. Find the next segmentation point $p$.
2. Let threshold $\theta$ be half the average character width.
3. Find the set of points $P_1$ that are within $\theta$ of $p$.
4. If there are two points in $P_1$ with the same segmentation angle, exclude all the points after the second one.
5. From the remaining points in $P_1$, find the set of points $P_2$ that do not span much text. (The number of text pixels should be less than the size of a small **i**.)
6. From among the points in $P_2$, choose the one with the lowest cost as the final segmentation point.

The output of the algorithm for a sample word is shown in Figure 7. Note that although some of the letter boundaries are not found in some of the segmentations, the final segmentation is potentially correct and that there are only 3 extraneous segmentation points.

Using this algorithm, we were able to segment all but one of the words in our test set of 111 words, as explained in the results section.

## 6.1 Derivation of the cost function

We use a cost function to find the segmentation points in the text. Given a segmentation point we find the next one by assigning a cost to all the points to the right of it and choosing the point that has the minimum cost. The first segmentation point is the left end of the text that can easily be found.

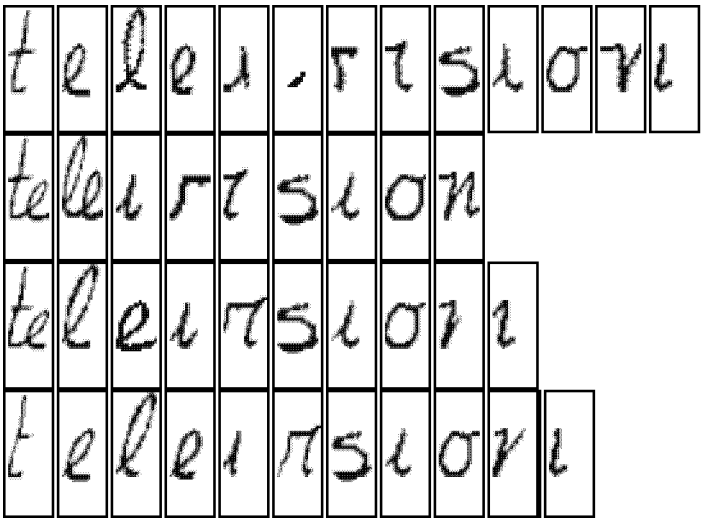

Figure 7: Segments found using 0, 10 and 20 degree angles are shown on the second, third and fourth lines, respectively. The output of the segmentation algorithm is shown at the top.

The cost function is defined for the pair $(A, P)$ where $A$ is the angle of the separator line (segmentation angle) passing through the point $P$ on the baseline. For a fixed angle $A$, it is computed as follows:

$$cost = w_1 \left(\frac{P - PP}{EW}\right)^2 - w_2 \left(\frac{P - PP}{EW}\right) + w_3(TC) + w_4(HC),$$

where $PP$ is the previous segmentation point, $EW$ is the average letter width, $TC$ is the number of text pixels cut by the separator line normalized by the average pen thickness and $HC$ is the height of the highest point cut normalized by the total height of the line. The $w_i$s are the relative weights of the cost terms.

We used linear programming to find a set of weights that would correctly segment a set of digraphs, representing all types of connections between letters. Since digraphs are the image primitives with respect to segmentation, an algorithm that correctly segments all digraphs would also correctly segment all words. For each digraph, we chose a correct segmentation point and a wrong one that is likely to be chosen, and constrained that the cost of the correct segmentation point be smaller than that of the wrong one. For example, we used

the digraph **bi** where the correct segmentation point was chosen between the letters **b** and **i** and the incorrect one was after the letter **i**. Note that once these two points are chosen, the only unknowns are the weights. Satisfying all the constraints simultaneously is achieved using linear programming, while minimizing the sum of the weights.

### 6.2 Handling oversegmentation

At this stage, some words are oversegmented as shown in Figure 8. To find the correct segmentation of the word, the extraneous segmentation points, the second and third ones in Figure 8, need to be identified and removed. Since we have no way of knowing which of those points are extraneous with certainty, we evaluate all possible final segmentations, generated by either keeping or removing the points that are likely to be extraneous. Figure 9 and 10 shows two possible final segmentations for Figure 8. In the word recognition stage, we arbitrate the segmentation by rating each word in the lexicon for its fit to each final segmentation and choosing the word that has the best fit.

There are $2^{n-1}$ possible segmentations of a word that is segmented into $n$ segments if every segmentation point is considered as extraneous. We try to identify the ones that are likely to be extraneous by analyzing the size and shape of the segments and the segmentation cost. For example, the two segmentation points around a narrow segment or a segmentation point with a high segmentation cost are likely to be extraneous. Also, our segmentation algorithm segments a letter into at most three segments, which also reduces the number of possible final segmentations, since we do not combine more than three segments into one. Since only $k$ of the $n-1$ segmentation points are identified as likely to be extraneous, we must consider $2^k$ segmentations in the word recognition stage. For a six letter word, we identify 3 such segmentation points on the average, requiring 8 segmentations to be considered.

## 7 Letter recognition

### 7.1 Slant correction

We correct the slant of a letter by shearing the letter along the x-axis by the amount of its estimated slant. We estimate the slant of a letter using two different methods. The first estimate is the dominant orientation of all edges in the interval $[-30^o, 30^o]$, found using edge detection techniques, followed by a histogram calculation. The second is the angle of the

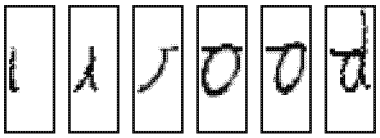

Figure 8: The output of the segmentation algorithm for the word *wood*.

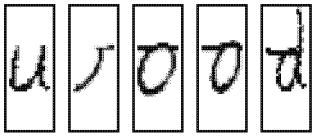

Figure 9: One possible final segmentation.

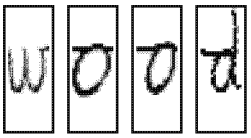

Figure 10: The correct segmentation.

vector joining the center of gravities of the top and bottom halves of the letter, assuming uniform weight distribution [Bur80].

We combine the two methods since there are problems associated with each of them. For example, the edge detection method may estimate the slant of a non-slanted **x** as 30 degrees due to its diagonal stroke. On the other hand, the center of gravity method estimates a positive slant for a non-slanted **d**. When the two estimates are within 20 degrees of each other, we use the first estimate as the overall estimate. Otherwise, we assume a slant of 0 degrees.

We are using shearing, as opposed to rotating, since usually only the near-vertical strokes of a letter are slanted.

## 7.2 Size normalization

To normalize the size of a segment and to make it fit in the 20 by 50 pixel input layer of the letter recognition network, multiple linear mappings are used. Figures 11 show two training alphabets and their letters after slant and size normalization.

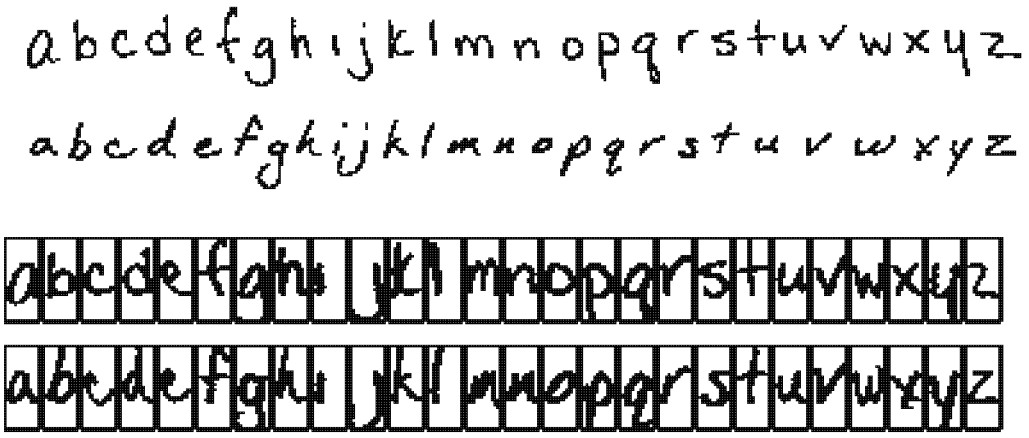

Figure 11: Two of the alphabets used for training the letter recognition network and their letters after size and slant normalizations.

We first decide, using the position of the letter in the text line and its horizontal pixel density histogram, whether a letter is an ascender, a descender or is short, and where its body starts and ends. If a letter is classified in one of these 3 categories with confidence, its body is mapped to the middle zone of the input layer and its ascender or descender is scaled to map to the top or bottom zone of the input layer. Otherwise, the part of the letter that overlaps with the middle zone of the text line is mapped to the middle zone and the remaining portions of the letter are mapped to the top and bottom parts of the input layer, respectively. This method prevents problems that occur when the ascender or descender of a letter is too long compared to the body, as for the example in Figure 12. Since the body is scaled independently, it does not get negligibly small during the scaling.

By applying the same idea in the other direction as well, thus mapping the width of the body to extend to the width of the center zone[2], we also prevent problems that occur when the ascender or descender are too wide compared to the body. More specifically, we linearly

[2] With the exception of narrow letters, for which the width to height ratio is preserved.

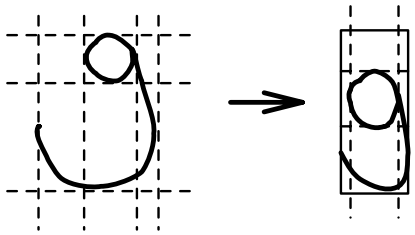

Figure 12: A sample letter before and after size normalization.

map the 9 regions on the left of Figure 12 to the corresponding 9 regions in the input layer. Much of the ascender and descender variation is removed with this method.

## 7.3 Network architecture

After normalization, we use the one hidden layer feedforward neural network, shown in Figure 13, to classify the extracted letters.

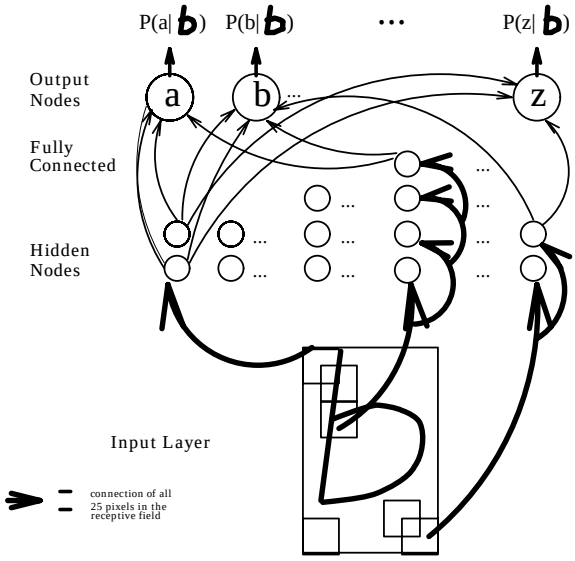


Figure 13: The architecture of the neural network used for letter recognition.

The network has 26 output nodes corresponding to the 26 lower case letters, and 70 hidden

nodes. The hidden nodes have 10x10 receptive fields which are more densely distributed towards the center of the input layer. Each hidden node is fully connected to each output node.

The input to the network is a 20 by 50 pixel gray scale image of the letter, after slant and size normalization has been applied. The network is trained, using the backpropagation learning algorithm, with 28 handprinted alphabets written by 28 different writers. It learns to correctly classify the training set after 18 epochs. We experimented with different network topologies and connection complexities, including fully connected networks, and found this architecture to yield the best generalization. The activation level of a letter node in the output layer is interpreted by the word recognition stage as the conditional probability of the letter given the input image. Hampshire and Pearlmutter, and others have shown that a network with enough connections approximates the Bayesian discriminant function such that the output values approximate the *a posteriori* probabilities, if trained with sufficient training data and using mean squared error estimation [HP90], [RRK+90].

# 8 Word recognition

To recognize the image of a word, we first assume that it is in the lexicon[3] and rate each word to find the one that best matches the image. Two hypotheses for finding the likelihood of words are compared for performance. The first assumes the words exhibit the digraph statistics of English, for which a Hidden Markov Model is used. The second assumes the small sampling of words in the training and test sets are equally probable, and do not reflect those digraph statistics. We found that the second method works significantly better, as we describe below.

## 8.1 Using Hidden Markov Models

Hidden Markov Models(HMMs) have proved to be useful in speech recognition, cryptanalysis and handwriting recognition, because of their ability to deal with statistical and sequential aspects of these problems. A Markov process is a stochastic process such that the probability of going into a state depends only on the current state [RJ86] [KS60] [Kon82]. In other words, knowledge of all the previous states does not give any more information than that of the previous state. A HM process is a doubly stochastic process with an unobservable underlying

[3]The letter strings that are not in the lexicon are referred to as non-words.

Markov process that generates a sequence of observations as it moves from state to state, where an observation is a stochastic function of the current state.

The states of the HMM correspond to the 26 letters of English, the observations to the letter images, and words to sequences of states. The probability of observing $o_i$ in a state $l_k$ corresponds to the probability of generating the image $o_i$ when writing the letter $l_k$. In other words, the output $o_i$ corresponds to what we see, the image of the letter, and the state corresponds to the letter $l_k$ that has produced it. We use the 26 outputs of the neural network when the input is $o_i$ as the posterior probabilities $p(l_j|o_i), j = 1..26$. The transition probabilities between states correspond to the transition probabilities (digraph statistics) of English which we estimate by counting number of occurrences of every letter pair in a large combined text of around 3 million words[4].

Finding the word that is most likely to have produced the image is equivalent to finding the most likely sequence of states, given the sequence of letter images and the fixed parameters of the HMM. Assuming the word is in our lexicon of 30,000 words, we find that probability for each possible segmentation of the image and choose the word that maximizes this probability over all possible segmentations.

For a word $w = l_1 l_2 ... l_n$, and the sequence of letter images,$s = o_1 o_2 ... o_n$,

$$p(w|s) = \frac{t_{\_l_1} p(o_1|l_1) t_{l_1 l_2} p(o_2|l_2) ... p(o_n|l_n) t_{l_n \_}}{p(o_1) p(o_2) ... p(o_n)} \quad (1)$$

$$= \frac{t_{\_l_1} t_{l_n \_} \prod_{i=1}^{i=n} t_{l_i l_{i+1}}}{\prod_{i=1}^{i=n} p(o_i)} \prod_{i=1}^{i=n} p(o_i|l_i) \quad (2)$$

where an underscore represents a blank, and $t_{l_i l_k}$ is the transition probability between letter $l_i$ and letter $l_k$.

Therefore, we find the word $w$ that maximizes

$$p(w|s) = \frac{t_{\_l_1} t_{l_n \_} \prod_{i=1}^{i=n} t_{l_i l_{i+1}}}{\prod_{i=1}^{i=n} p(l_i)} \prod_{i=1}^{i=n} p(l_i|o_i)$$

since

$$p(o_k|l_i) = \frac{p(l_i|o_k).p(o_k)}{p(l_i)}$$

[4]The text of 3 books from Lewis Carroll and the World FactBook.

Note that $p(w|s)$ can be written as a product of two terms, emphasizing the bias (input independent factor) in the model:

$$p(w|s) = \frac{t_{\_l_1} t_{l_n \_} \prod_{i=1}^{i=n} t_{l_i l_{i+1}}}{\prod_{i=1}^{i=n} p(l_i)} \prod_{i=1}^{i=n} p(l_i|o_i)$$

## 8.2 Assuming independence among letters

In our second approach, the posterior probability of a word, for a given segmentation of the image is found as

$$\begin{aligned} p(w|s) &= Pr(w)p(l_1|o_1)p(l_2|o_2)...p(l_n|o_n) & (3) \\ &= Pr(w) \prod_{i=1}^{i=n} p(l_i|o_i) & (4) \end{aligned}$$

where $w = l_1 l_2 ... l_n$, $s = o_1 o_2 ... o_n$ is the sequence of letter images and Pr(w) is the probability of occurrence of the word $w$ in the English language.

Note that this scheme favors more likely words over less likely ones and thus maximizes the performance over a large set of test words. For the experiments in the next section, we used uniform distribution for the probability of occurrence of the words, since the test data we used was not large enough. The reason this scheme works better than the previous one for words in the lexicon is because the system's performance is good enough that the bias introduced by the HMM is often disadvantageous. For example if the image is *dip* , which does not have a high bias term, the HMM may choose the word *die* which does have a high bias term, even if the letter **p** gets a significantly higher probability than the letter **e**.

## 8.3 Recognizing non-words

If none of the words gets a probability over some threshold, the system will decide that the image corresponds to a non-word. We use the Viterbi algorithm to find the most likely non-word, which corresponds to finding the most likely state sequence for the HMM, without constraining it to be a word in the lexicon.

We have not analyzed the scores of the words that are correctly recognized by the system to find the lower bound to use as that threshold. However, in our preliminary tests, the

output of the Viterbi algorithm has been the correct word only for the most common words, which suggests that we use a low threshold.

# 9 Experimental results and discussion

We have experimented with several different network architectures to find the best letter recognition network in terms of generalization performance. The one hidden-layer network with local connections described in Section 7.3 is used in the experiments here, due to its superior performance. The letter recognition network is trained using 28 alphabets written by 28 different writers. The test set consists of 7 other such alphabets written by 7 other writers. Both the training and test alphabets consist of the 26 lower case letters, where only one form of each of the letters **a**,**s** and **z** is represented, in order to reduce the training effort. The recognition rate for the letters of the test set is 75%, when the network is trained to recognize all of the training alphabets. This low performance is due mainly to the very small size of the training data.

In the remainder of this section we report the performance results of the whole system, using word scores computed as described in Section 8.2. We asked several writers to write the same test page the way they usually write, but more or less following the rulings of the page. The page contains 8 text lines and 37 words, where the average number of letters per word is 5.95[5].

In the first experiment, we tested the performance of the system on recognizing handprinting. The sample page was written discretely by a writer (W1) who also wrote a training set. The recognition rates were 81% for letters and 93% for words. These results are similar to the best handprinted word recognition rates [KHB89] [Bur87].

In the second experiment, we replicated the first experiment, except that the page was written cursively this time. 70% of the words were recognized correctly and 76% were in the top three word choices. All of the words were correctly segmented. In order to understand the performance degradation compared to the first test, we analyzed the letter recognition performance, for the correct segmentations of the test words. Only 50% of the letters of the words in the sample page were recognized correctly, compared to 81% in the first test. This is mainly due to not having cursive letters in the training alphabets. Recognizing cursively written letters is also harder than recognizing handprinted letters, since the ligatures increase

[5]The average number of letters per word is 6.22 in the long combined text that we used to compute the transition probabilities for the HMM.

the variance in letter shapes.

In the third and fourth experiments, we replicated the second experiment, but with two other writers (W2 and W3). Writer W2 did not write a training alphabet. 28% of the words written by writer W2 were recognized correctly and 47% of them were in the top three word choices. One word was missegmented, but correctly recognized. The poor performance in this case is partly due to bad writing style (letters are extended at the end of words as in the second *eliminate* in Figure 14 and are inconsistent in size) and to the dissimilarities of the letter shapes compared to the samples in the training set. 83% of the words written by writer W3 were recognized correctly and 92% of them were in the top three word choices. All the words were correctly segmented.

Overall, in the experiments 2 through 4, 61% of the words were correctly recognized and 71 % of them were in the top three choices. Some of the correctly recognized words, from each of the three writers, are shown in Figure 14. The first 6 words are from writer W1, the next 5 words are from writer W2 and the next 3 words are from writer W3. Note that W1 writes almost pure cursive whereas W2 and W3 write cursive, but with occasional touching letters. Over the whole test set of 111 words, only one word was missegmented. The segmentation results and the recognition performance on the second test set which was almost pure cursive handwriting shows the success of our segmentation algorithm. Our word recognition results compare favorably to those of Srihari and Božinović [SR87] and to the lexical word recognition performance of the system by Edelman et al [EFU90], although an exact comparison is not possible (see [YS93] for a summary of related work).

We are currently training the letter recognition network with an expanded training set that includes some cursive letters and collecting more test data to test the system thoroughly.

# References


[Bur80] D.J. Burr. Designing a handwriting reader. pages 715–722, 1980.

[Bur87] D.J. Burr. Experiments with a connectionist text reader. In *IEEE First International Conference on Neural Networks*, volume 4, pages 717–724, 1987.

[DH73] R.O. Duda and P.E. Hart. *Pattern Classification and Scene Analysis*. Wiley, New York, 1973.

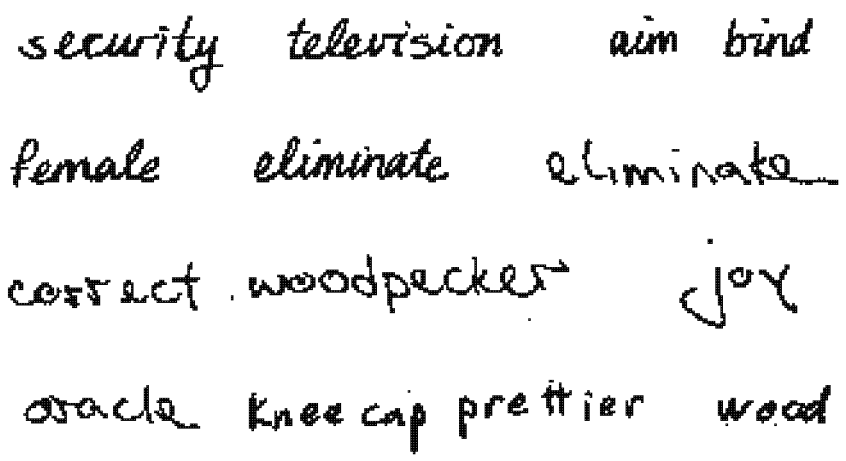


Figure 14: Some of the words that are correctly recognized.

[EFU90] S. Edelman, T. Flash, and S. Ullman. Reading cursive handwriting by alignment of letter prototypes. *International Journal of Computer Vision*, 5(3):303–331, 1990.

[HP90] J. Hampshire II and B. PearlMutter. Equivalence proofs for multi-layer perceptron classifiers and the bayesian discriminant function. In D. Touretzky, J. Elman, T. Sejnowski, and G. Hinton, editors, *Connectionist Models*, pages 159–172. Morgan Kaufmann, San Mateo, 1990.

[KHB89] A. Kundu, Y. He, and P. Bahl. Recognition of handwritten word: First and second order hidden markov model based approach. *Pattern Recognition*, 22:283–297, 1989.

[Kon82] A.G. Konheim. *Cryptography: A Primer*. John Wiley and Sons, NewYork, 1982.

[KS60] J. Kemeny and L. Snell. *Finite Markov Chains*. Cambridge University Press, 1960.

[RJ86] L.R. Rabiner and B.H. Juang. An introduction to hidden markov models. *AASP magazine*, 3(1):4–16, 1986.

[RRK+90] D. W. Ruck, S. K. Rogers, M. Kabrisky, M. E. Oxley, and B. W. Suter. The multilayer perceptron as an approximation to a Bayes optimal discriminant function. *IEEE Transactions on Neural Networks*, 1(4):296–298, 1990.

[SR87] S. Srihari and R. Božinović. A multi-level perception approach to reading cursive script. *Artificial Intelligence*, 33:217–255, 1987.

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

[YS93] Berrin A. Yanikoglu and Peter A. Sandon. Off-line cursive handwriting recognition using neural networks. In *SPIE Conference on Applications of Artificial Neural Networks*, 1993.