Handwriting Recognition Systems: An Overview

Avi Drissman
Dr. Sethi
CSC 496
February 26, 1997

Committing words to paper in handwriting is a uniquely human act, performed daily by millions of people. If you were to present the idea of “decoding” handwriting to most people, perhaps the first idea to spring to mind would be *graphology*, which is the analysis of handwriting to determine its authenticity (or perhaps also the more non-scientific determination of some psychological character traits of the writer). But the more mundane, and more frequently overlooked, “decoding” of handwriting is handwriting recognition—the process of figuring out what words and letters the scribbles and scrawls on the paper represent.

Handwriting recognition is far from easy. A common complaint and excuse of people is that they couldn’t read their own handwriting. That makes us ask ourselves the question: If people sometimes can’t read their own handwriting, with which they are quite familiar, what chance does a computer have? Fortunately, there are powerful tools that can be used that are easily implementable on a computer. A very useful one for handwriting recognition, and one that is used in several recognizers, is a neural network.

Neural networks are

> richly connected networks of simple computational elements. The fundamental tenet of neural computation (or computation with [neural networks]) is that such networks can carry out complex cognitive and computational tasks. [9]

In addition, one of the tasks at which neural networks excel is the classification of input data into one of several groups or categories. This ability is one of the main reasons neural networks are used for this purpose. In addition, neural networks fare well with noisy input, making them even more attractive.

Handwriting recognizers are broadly divided into two categories—on-line and off-line. On-line recognizers run, and receive the data, as the user writes. Generally, they have to process and recognize the handwriting in real- to near real-time. Off-line recognizers run after the data have been collected, and the image of the handwriting, for analysis, is given to the recognizer as a bitmap. Thus, the speed of the recognizer is not dependent on the writing speed of the user, but the speed dictated by the specifications of the system, in words or characters per second.

The most common approach, used by most recognizers both on- and off-line, is to use a neural network to distinguish the letters that make up the words, and then to run the possible combinations of letters through a dictionary to find the word that the user most likely intended. This is a very simplistic approach which begs important questions, but it is a useful foundation for comparison.

On-line recognizers

There is a proliferation of on-line recognizers developed as compared to off-line recognizers. There are two main reasons for this disparity. First, on-line recognizers are easier to build [13], because the order of the pen-strokes is known, as well as timing information. Secondly, handwriting recognition can easily be used for input in handheld or PDA-style computers, where there is no room for a keyboard. Since a recognizer in this use is very visible, this visibility spurs on development.

CalliGrapher

The first two commercial PDAs on the market were the Apple MessagePad with the Newton OS, and the Tandy Zoomer, running GEOS. The first incarnation of the Newton OS used the CalliGrapher recognizer from ParaGraph International [11]. As the first mainstream recognizer on the market, CalliGrapher, as implemented on the Newton, took a lot of beating, the most famous of which was a series of Doonesbury cartoon strips lampooning the recognition ability of the Newton devices.

Part of the reason for the ridicule was that the dictionary was of fairly limited scope, and often the wrong word was selected. In addition, expectations were set far too high, and people often gave up before the recognizer adapted to their handwriting. For its release with Newton 2.0 (version 2.1 [8]), there was a significant improvement made to the recognizer, and a larger dictionary was added. With a print recognizer from Apple (see below) taking a lot of the heat off of CalliGrapher, it's been more respected.

CalliGrapher uses a fuzzy-logic matching algorithm to match written characters, both cursive and a mixture of cursive and print, with "prototypes" of letters [7]. The prototypes used are general descriptions of the shapes of the strokes used to draw the character. When a user finishes entering a word, the recognizer attempts to match the strokes to active prototypes to determine the letters. If the dictionary mode is on, then the interpreted word is run through the dictionary in an attempt to find and cope with errors.

Limited training is possible [7]. If the user never uses a particular prototype, then it can be safely deleted, speeding up the recognition process and reducing the possibility of future misinterpretation.

> To know which prototypes a user really uses and which they don't requires the recognizer to know the correct answer. We suppose that if a user corrects some answer by choosing a word from the list of guesses, then this word is assumed to be the correct answer and we will use the word for learning. We also suppose that if a user did not correct a word, and wrote ten words after it, that this word was correct. [7]

Creation of new prototypes is not allowed in the current system as implemented in the Newton OS, because it is time-consuming and is considered to be of little benefit. Adaptation of the prototypes to more closely match the user's handwriting is available with an add-on package [7].

Apple-Newton Print Recognizer

The Apple-Newton Print Recognizer (ANPR) was introduced in the release of the Newton OS 2.0 along with the 2.1 version of CalliGrapher. Due to the pent-up frustration due to the low initial accuracy of the version of Calligrapher used in Newton OS 1, users seemed willing to restrict themselves to printing to get relatively accurate recognition without much training.

ANPR uses a three-step process to attempt handwriting recognition [15]. The first step in the process is tentative segmentation [15]. Since many characters comprise more than one stroke, there is no easy or reliable way to correctly divide the strokes that make up the word into letters. So the first task of the recognizer is to segment the strokes into presumed groupings of strokes which might be letters. The segmentation algorithm does not take into account any factors; all it does is generate all groups of strokes that might be part of one character.

The second part of the recognizer is the neural network [15]. The neural network gets passed the stream of segments, and analyzes each segment in an attempt to recognize it as a letter. The neural network is a multi-layer perceptron using error back-propogation as a learning technique. (A neural network is composed of “neurons”, simple units whose output is usually a function of the weighted sums of their inputs. A perceptron uses these units in layers, with each layer feeding forward to the next one. In most perceptrons, and in this one, the neural network does not process the entire image but concentrates on specific areas. Back-propagation is a technique used to train neural networks; when the output is in error, the information is passed backwards through the nodes to allow them to adjust their weights. Details can be found in a good introductory text such as [1] or [4].)

Of particular interest is the construction of the neural network. While on-line recognizers have the advantage of knowing the order of the strokes, it would be beneficial to also attempt to perform recognition on the bitmaps of the letters. Therefore, the neural network is set up to take two sets of inputs: the stroke data and a character bitmap. The two halves of the neural network independently converge on letters, and a final layer merges the two letter decisions together [15]. It could probably be said that this approach uses techniques of both on-line and off-line recognition.

Finally, the context-driven search is given the letters and the groupings, and it has to make the final decisions on which letters and which segments of strokes most likely represent the word the user wrote [15]. It uses a set of language models to achieve this goal. The language set consists of a set of rules; breaking rules incur penalties of differing strengths. The rules range from letter transition rules (‘q’ without ‘u’ following is highly unlikely) to penalizing abrupt case transitions. Also, small dictionaries with common prefixes and suffixes are used.

The search takes the set of interpreted letters and the original groupings of strokes, and figures out the combination of them which racks up the smallest penalty. This combination of letters, which hopefully is the user’s word, is what the recognizer delivers as its final output.

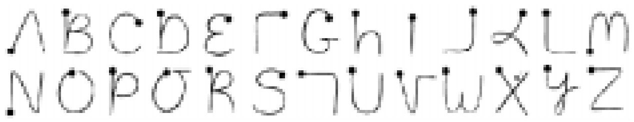
ThinkWrite

IBM's ThinkWrite system, currently available for OS/2 and Windows 3.1, is a printing-only system [8]. According to on-line documents, the ThinkWrite system uses a "hybrid" [5] system combining on-line (stroke and timing based) and off-line (bitmap based) recognizers, suggesting an underlying concept similar to Apple's ANPR. However, it doesn't use a neural network, but algorithms to match the characters [8].

Graffiti

Graffiti [14], a product of Palm Computing (now a division of US Robotics), took the market by storm with the original Newton CalliGrapher debacle. Graffiti, available for almost all handheld platforms, takes a different approach to handwriting recognition than CalliGrapher and ANPR by eschewing handwriting completely.

Graffiti uses a character system in which all letters of the alphabet and all symbols are represented by one stroke. This solves two problems [10]. First, the stroke segmentation problem that the ANPR dealt with in the first stage is eliminated by the fact that each letter is represented by one stroke. This one-to-one correspondence simplifies character separation greatly. Second, since the strokes can be chosen arbitrarily (although some resemblance to the alphabet remains), letters which are very similar in natural handwriting can have strokes assigned which are quite different. This avoids many problems in recognition, and reduces the need for post-processing to determine the appropriate character.



The character set of Graffiti. The dots indicate the starting point. [14]

Jeff Hawkins, in an interview with Pen-Based Computing, explained that Graffiti uses a pattern-matching algorithm that

> "…was inspired by my [Hawkins's] neural theory, although it's not exactly based on it. It's a very simple and clever algorithm for doing pattern matching. However, it's a very different approach than other people have taken." [6]

The theory that people will learn a new way to write the letters of the alphabet to achieve fast, consistent recognition may be true, but as the algorithms and networks to recognize normal handwriting improve, then the need for Graffiti decreases, as happened with the Newton with the transition to version 2.0. It has been used as the sole input recognizer for several devices with mixed records—the Tandy Zoomer which flopped, and the US Robotics Pilot, which is currently enjoying a fair amount of success.

Off-line recognizers

Off-line recognizers haven't had the attention—both good and bad—that their relatives, the on-line recognizers, had. This is quite unfortunate because they can play, and are playing, an essential role in a nation which is buried in paper. Several problem areas exist where off-line handwriting recognizers can be of use because of large quantities of hand-written data. Also interesting is that because the task of off-line recognition is harder, more of the research is performed in universities than for on-line recognizers.

One such area is postal address recognition [12]. While performing OCR on printed addresses is relatively simple, recognizing the addresses that are handwritten is far more difficult, and a tedious task ripe for assumption by recognizers. This task is made easier because addresses have redundancy—ZIP codes specify the city and state. Also, many parts of address blocks have a limited range of values—ZIP codes are all digits, and are either five or nine digits long. There are only fifty states, each of which has a two letter code, plus a few codes for territories.

Another area where off-line recognizers are finding use is in check reading [12]. Millions of checks pass through clearinghouses monthly, and each one must have the amount written on it machine coded on the bottom. This, again, is an ideal task for recognizers. Checks have great redundancy, because the dollar amount is written both in figures and out in words. Secondly, the character set is limited—digits for the number field, and a limited vocabulary ("one", "sixty", "hundred") of words for the spelled-out field.

It is important to remember that in both of these target applications, a 100%

recognition rate isn't essential. Since these fields already have a large staff to currently process the items, recognizers can flag any item as unreadable and set it aside for manual processing, while taking care of the clear-cut cases. A good example is a check-reading system sponsored by the French post office, which has a goal of a 0.01% error rate which it hopes to accomplish by allowing 50% of the checks to be rejected by the system (and routed to humans) as unreadable [13].

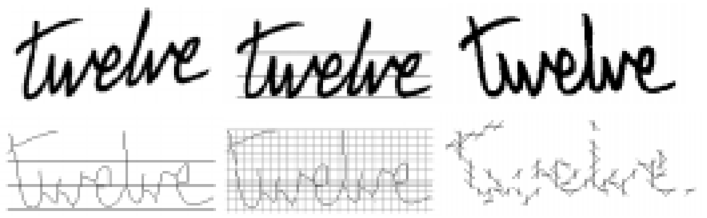
A. W. Senior

A. W. Senior, a researcher at Cambridge University in England, describes in [13] a fairly typical off-line handwriting recognition system. The system uses words written with plenty of separation between them from a group of words typically used on checks. Because of the great difficulty involved in deciphering handwriting without stroke timing information, as available in on-line recognizers, most off-line recognizers use neural networks to recognize the letters, while many on-line recognizers employ other methods.

On-line recognizers get the strokes input by the user as their direct input, ready to be processed and recognized. Off-line recognizers don't have that luxury. Several steps of pre-processing are needed to ensure that the image is normalized, all strokes are a consistent width, and the data are changed into a form that the neural network can deal with [13].

Senior's system employs several steps of pre-processing and data massaging [13]. First, it attempts to find the baseline of the written words, and rotate the image to a horizontal position. Then it finds vertical strokes in the words, and skews the images until those identified strokes are vertical. Then, since we want the system to be independent of pen thickness, all strokes are thinned to a one-pixel thickness.

Since a bitmap is rather unwieldy to manipulate, and since the strokes of the letters are more important, the bitmap is "parameterized" according to a grid [13]. If a stroke passed through a box, the box is marked according to the direction of the stroke—horizontal, vertical, or at the 45° angle. If the stroke falls between those directions then it counts as both.



The four preprocessing steps of Senior [13]. The top left image is the original. From left to right, top to bottom, the steps are: slope correction, slant correction, thinning, parameterization boxes, and parameterization. Illus. from [13].

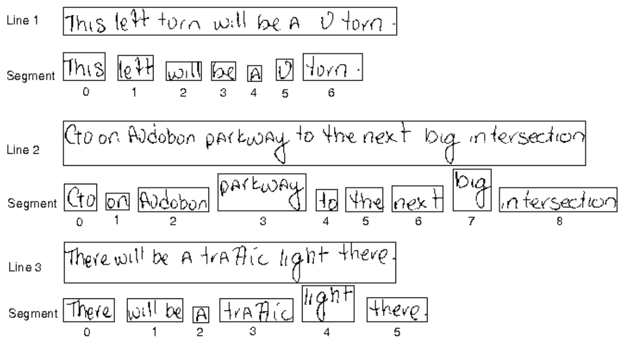
The neural network used by Senior is a recurrent perceptron [13]. A recurrent neural network is simply one which has a loop where some of the outputs are fed back into the network as inputs. This allows the network some context in which to work. The neural network itself is a standard perceptron.

CEDAR Penman

Penman [2] takes a different approach to off-line recognition by attempting to read multiple lines of naturally-written handwriting. Here too, just as in Senior's system [13], the task of recognizing the handwritten words of the user is overshadowed by the preprocessing required. The preprocessing is similar to Senior's, but more tasks need to be done as this system does not assume that the words are isolated on the page amidst white space.

The first step in this system is the separation of lines [2]. Penman does this by analyzing the outlines of the characters, and finding the "extreme points" (places at which the shape of the outline bends considerably) on them. From the points, it estimates the line height. Then it starts going through the points, grouping them into lines. If a group of points is too far away from a line (based on the calculation of the line height) then that group of characters is considered a new line.

The next task is the separation of lines into individual words [2]. The creators of Penman considered this as a difficult task, especially considering the fact that people often leave little to no space in-between words. Using an algorithm based on visual clues, they extracted data to use in a neural network trained to discern words.


Line 1
This left turn will be A U turn.
Segment
0 1 2 3 4 5 6
Line 2
Go on Audobon parkway to the next big intersection
Segment
0 1 2 3 4 5 6 7 8
Line 3
There will be A traffic light there.
Segment
0 1 2 3 4 5

The result of letting the word segmentation neural network loose on an example set of data. [2]

The word recognizer in Penman is based on feature extraction [2], in which the loops, lines and distinguishing characteristics of the scanned data are extracted and analyzed to determine the letters. The sequences of recognized letters is compared against words in a dictionary to aid accuracy.

A post-processing step is useful in determining which choices of words are the ones in the sentence. Penman uses post-processing models based on knowledge of the language is is recognizing [2]. Not much more detail is available in [2] about the post-processing stage.

Guillevic and Suen

Guillevic and Suen, researchers at Concordia University, developed a system [3] to read the legal amount (the value written out in words) on checks. They classified all attempts to read handwriting into two categories: recognizing the word as a whole, or trying to break it up into its constituent letters and recognizing them individually. Eventually, it was decided to use both methods—first trying to identify it according to its overall shape, and if that was insufficient to then try breaking it up into characters. Their paper [3] only describes the first method.

Their system starts with the usual preprocessing in order to normalize the input data [3]. Interestingly enough, their first step is connection of broken images. Since conversion from a gray-scale image to a black-and-white image might cause drop-outs in the lettering, they use a routine which checks if two unconnected strokes are close enough to each other, and if they are, the routine connects them. They did not say how they defended against spurious connections; while the word recognition described in [3] wouldn't be terribly affected, it is unclear what affect it might have on a letter-by-letter recognition.

The next step in preprocessing is the usual slant correction [3]. This simplifies recognition by eliminating some of the natural variation of people's handwriting. This recognizer, unlike other recognizers, doesn't bother with slope correction. This is probably due to the fact that checks have a line to write on, and most people stay on the line.

The last two steps in preprocessing [3] are noise removal and line removal. Noise removal identifies all the connected components of the image and removes all of them under a given size. Line removal identified the long horizontal lines many people write before and after amounts and removes it. Both of these steps help the recognizer by removing as much non-word data as possible.

The word is identified as a whole by locating features on it [3]—namely, number and position of ascenders and descenders, positions of loops, and word length. Then it compares the features of the input word to the words it knows, and comes up with its best guess.

The system is reported [3] to do well for words with distinctive ascender/descender patterns, but not so well in words which are similar. Nevertheless, since this is a first step in a system that goes into more depth, this performance is promising.

Conclusion

Several years ago, people who used computers took for granted the notion that they would have to adapt to their style of input to something computer friendly—whether in typing, or filling out forms with letters neatly boxed. But now, computers whose sole input method is handwriting are doing well, and computers are taking on tasks once thought beyond their abilities. Handwriting recognition is, without doubt, changing the way people relate to computers.

References

1. Aleksander, Igor and Helen Morton. *An Introduction to Neural Computing*. Second Edition. Thompson Computer Press, London, 1995.

2. CEDAR. “Penman: Handwritten Text Recognition Project Description”. Available from http://www.cedar.buffalo.edu/Penman/description.html

3. Guillevic, Didier and Ching Y. Suen. “Cursive Script Recognition: A Sentence Level Recognition Scheme”, In *Int. Workshop on the Frontiers of Handwriting Recognition* (IWFHR), Taipei, Taiwan, p. 216-223, December 1994. CENPARMI, Concordia University, Montréal, Canada. Available from http://www.cenparmi.concordia.ca/publications/PS/guillevic_iwfhr94.ps.Z

4. Hertz, John, *et al*. *Introduction to the Theory of Neural Computation*. Addison-Wesley Publishing Company, Redwood City, California, 1991.

5. IBM Corporation. “Handwriting Recognition for Pen Computers”. Available from http://www.almaden.ibm.com/cs/showtell/handwriting/Initpage.html

6. Jerney, John. Executive View: A Conversation with Palm Computing’s Jeff Hawkins. *Pen-Based Computing*. Available from http://www.volksware.com/pbc/article/hawkins.htm

7. Lossev, Ilia. Characters from Russia: ParaGraph’s CalliGrapher Handwriting Recognition. *Pen Computing Magazine*. June 1996, 34–35.

8. MacNeill, David. Handwriting Recognizer Specification Matrix. *Pen Computing Magazine*. August 1995, 32–33.

9. Mehra, Pankaj and Benjamin W. Wah. *Artificial Neural Networks: Concepts and Theory*. IEEE Computer Society Press, Los Alamitos, California, 1992.

10. Palm Computing. “Graffiti White Paper”. Available from

http://www.iicm.edu/0x811b9908_0x0012144a;internal&sk=ROBOT/

11. ParaGraph International. http://www.us.paragraph.com/

12. Rawson, C. Edward. “Breaking Down the Last Document Automation Barriers!” Available from http://www.infotivity.com/hwr.htm

13. Senior, A. W. “Off-Line Handwriting Recognition: A Review and Experiments”, Technical Report CUED/F-INFENG/TR 105, Cambridge University Engineering Department, Cambridge, England, Dec. 1992. Available from ftp://svr-ftp.eng.cam.ac.uk/pub/reports/senior_tr_105.ps.Z

14. US Robotics. “Graffiti”. Available from http://www.usr.com/palm/5036.html

15. Yaeger, Larry, Brandyn Webb, and Richard Lyon. “Combining Neural Networks and Context-Driven Search for On-Line, Printed Handwriting Recognition”. For the Fifth International Workshop on the Frontiers of Handwriting Recognition (University of Essex, England, September 1996). Available from ftp://ftp.apple.com/pub/larryy/ANHR/Yaegeretal.AppleRecog.bin.2.ps.Z

Yes, this paper is mine—I wrote it myself without assistance of any kind. Despite the pain, suffering, and anguish endured during its creation, I did not plagiarize *anything at all*, and all material I used from outside sources is properly attributed to the best of my knowledge and ability. No animals were harmed in the production of this paper. Those who appreciate quality enjoy it responsibly.

____

Avi Drissman