KNOWLEDGE INTEGRATION IN TEXT RECOGNITION

Sargur N. Srihari and Jonathan J. Hull
Department of Computer Science
State University of New York at Buffalo
Amherst, New York 14226

ABSTRACT

The paper describes an algorithm based on AI techniques for recognizing words of printed or hand-written text--with the technique developed also applicable to correcting substitution spelling errors. The algorithm effectively integrates bottom-up information in the form of letter shapes, letter transitional probabilities and letter classification-error probabilities together with top-down knowledge in the form of a lexicon of legal words represented as a letter trie. Experimental results with the algorithm are reported for the combined top-down and bottom-up approach and for each of the two approaches individually.

I INTRODUCTION

The recognition of text that is machine printed with multiple fonts, hand-printed, or written as cursive script finds many applications including that of office automation. Some present generation optical character readers [OCRs] accept single font machine print or text hand-printed under certain constraints. Any deviation from these constraints results in rejection or a highly garbled representation of the text.

Human beings perform better than present OCRs by at least an order of magnitude in error rate although their performance when viewing a letter in isolation does not significantly differ from OCR error rate. This is attributed to effective use of contextual factors like letter sequences, vocabulary, word-dependency, sentence-structure and phraseology, style and subject matter as well as the associated processes of comprehension, inference, association, guessing, prediction and imagination, all of which take place very naturally during the process of reading. The example of Fig. 1 illustrates some aspects of this process. Although the letters 'H' and 'A' in the words 'THE' and 'PAPER' are identically printed--thereby leading to identical feature vectors--they are easily distinguished by the human reader due to the presence of surrounding letters in the respective words. The last word of the sentence is either 'CLIP' or 'CUP' which can be disambiguated by more global knowledge, e.g., if the next sentence were 'I NEED SOME COFFEE' then the word in doubt is probably 'CUP'.

It is clear that if computer programs are to reach expert human ability in text recognition then they need to be able to effectively integrate diverse contextual knowledge sources about the text, as well as knowledge about the kinds of textual errors that are likely, i.e., characteristics of the text transmission channel that introduces errors. A number of programs that utilize only a few knowledge sources in text recognition are described in the literature; tutorial surveys of these methods have been made [1],[2]. Some of these methods, viz., text recognition algorithms, are directly applicable to a set of image vectors representing characters of text and others, viz., text error correction algorithms, are applicable only to previously decoded text. A majority of these methods can also be characterized as those that are data-driven or bottom-up, and those that are concept-driven or top-down.

Data-driven algorithms proceed by refining successive hypotheses about an input string. An example is a program that utilizes a statistical (Markovian) representation of contextual knowledge in the form of a table of transitional probabili-

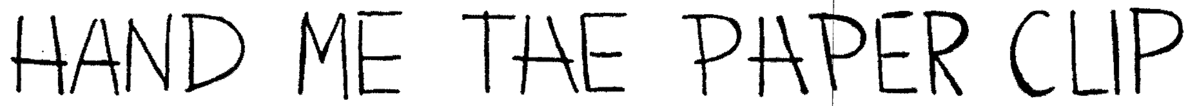


Fig. 1. Identical patterns have different interpretations in different contexts.

*This work was supported by the National Science Foundation Grant IST-80-10830.

ties, i.e., the probability of each letter given that a letter sequence has previously occurred. Concept-driven algorithms proceed with an expectation of what the input string is likely to be and proceed to fit the data to this expectation. Examples are algorithms that use an implicit or explicit representation of a lexicon.

This paper describes an algorithm that effectively merges a bottom-up refinement process that is based on the utilization of transitional probabilities and letter confusion probabilities, known as the Viterbi Algorithm [VA], together with a top-down process based on searching a lexicon that is applicable to text containing an arbitrary number of character substitution errors such as that produced by OCR machines. The work is part of a larger ongoing effort on the text recognition problem at SUNY/Buffalo.

II THE BOTTOM-UP APPROACH

The VA is a method of finding the word that maximizes likelihood over all possible letter combinations and not necessarily those in a lexicon; it is based on a dynamic programming formulation which leads to a recursive algorithm [3]. The method utilizes the characteristics of the OCR channel in the form of a table of confusion probabilities. Each entry of this table represents the probability that the OCR channel assigns a given letter to another (possibly the same) letter due to ambiguities in the shape features used to classify shapes into character classes.

The algorithm can be viewed as that of finding a maximum cost path through a directed graph called a trellis. The log-transitional probabilities are associated with the edges of the trellis and the log-confusion probabilities are associated with the nodes. The cost of a path is then the sum of all the edge and node values in the path. We use a computationally improved version of the VA where the number of alternatives per letter is variable--these alternatives are determined by the letters that have the highest confusion probability.

This method represents a purely bottom-up approach whose performance may be unacceptable due to the fact that the resulting strings do not necessarily belong to a lexicon. Our approach to improve performance is to use top-down contextual information, in the form of a lexicon of allowable input words, to aid the bottom-up performance of the VA.

III LEXICAL REPRESENTATION

The lexical data structure and method of access is critical to the efficiency of any text correction algorithm. Several alternative structures are available--the choice has to be based on the search strategy of the algorithm and the memory available.

A data structure that is suitable for determining whether a given string is an initial substring, or prefix, of a lexical entry is known as the trie [4]. Since the VA proceeds by computing for a given length the most likely prefix, the trie is an attractive data structure. Essentially, the trie considers words as ordered lists of characters, elements of which are represented as nodes in a binary tree. Each node has five fields: a token, CHAR; a word-length indicator array of bits, WL; and end of word tag bit, E; and two pointers labelled NEXT and ALTERNATE (see Fig. 2).

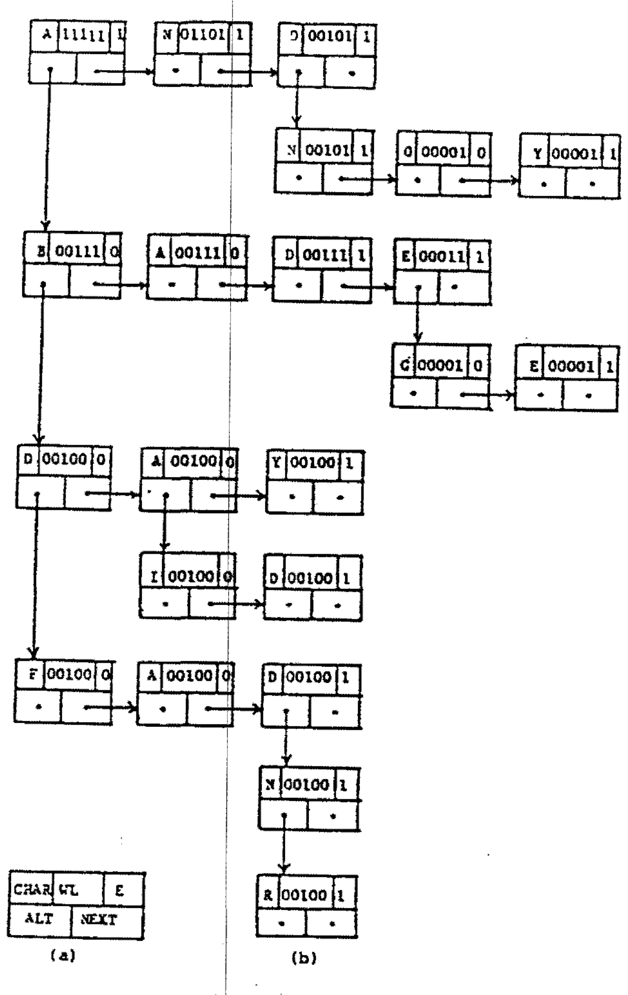


Fig. 2. Trie structure: (a) the fields of a typical record, and (b) trie of the lexicon: A, AN, AND, ANN, ANNOY, BAD, BADE BADGE, DAY, DID, FAD, FAN, FAR.

A node is a NEXT descendent if its token follows the token of its father in the initial substring of a lexical word. It is an ALTERNATE descendent if its token is an alternative for the father's given the initial substring indicated by the most immediate ancestor which is a NEXT descendent. Without loss of generality it is required that the lexical value of the token of each ALTERNATE descendent be greater than that of its father. The end of word bit is set if its token and the initial substring given to reach the token comprise a complete dictionary word. The mth bit of the word length indicator array is set if the token is on the path of an m letter word in the trie.

IV THE COMBINED APPROACH

Simultaneous search of the VA trellis using a variable number of alternatives per input letter and the trie structure is controlled by a binary array A. This may be regarded as a blackboard through which the top-down and bottom-up processes communicate [5]. Element A[j,i] is set to 1 if the jth letter of the alphabet is a possible correction for the ith letter of the input word, i.e., log-confusion probability exceeds a threshold t, and 0 otherwise. Thus the paths of the trellis that need to be evaluated are only those that begin at the 1's of the first column of A and proceed through the 1's of the subsequent columns. Before evaluating a path that proceeds from one column of A to the next column, that path is determined to be legal with respect to the trie. The computational complexity of the resulting algorithm is of the same order as the VA.

V EXPERIMENTAL RESULTS

To determine the performance and efficiency of the algorithm with actual text and to compare this with variations of the VA, a data base was established and experiments were conducted.

English text in the Computer Science domain (Chapter 9 of Artificial Intelligence, P.H. Winston, Addison Wesley, 1977) containing 6372 words was entered onto a disk file. Unigram and first order transitional probabilities were estimated from this source. A model reflecting noise in a communications channel was used to introduce substitution errors into a copy of this text and confusion probabilities were estimated from this source. A lexicon of 1724 words containing 12231 distinct letters was extracted from the correct text and a trie was constructed. There were 6197 nodes in the trie and the average number of alternates for all nodes was 1.62. The storage required to load the program and its knowledge data structures were, in terms of CDC Cyber 174 words: program (10K), trie (18K), confusion and transitional probability tables (1.5K).

An example of garbled text is given in Fig. 3 and its correction produced with t=-11) is given in Fig. 4. It can be observed that the corrected text is significantly better than the text input to it. We also note the shortcomings that the words "lomputer" and "tayfr" were rejected and the garbled words "bm" and "bes" were erroneously corrected to "by" and "but" (instead of "be" and "few) respectively, and the lexical word "come" was not corrected to "some". Rejections could be eliminated by decreasing the alternative selection threshold t, thereby allowing more possibilities for each letter.

If we looi at what has prodused lomputer imtelligence qo far, we see multiple lamers, each of which rests on primitives of the naxd tayfr dowm, forminc a hierarcfical structure with a great deal interposed between the intelligent prphvem and the transistors which ultimatelu suppodt it. Figure 9-8 illustratms.

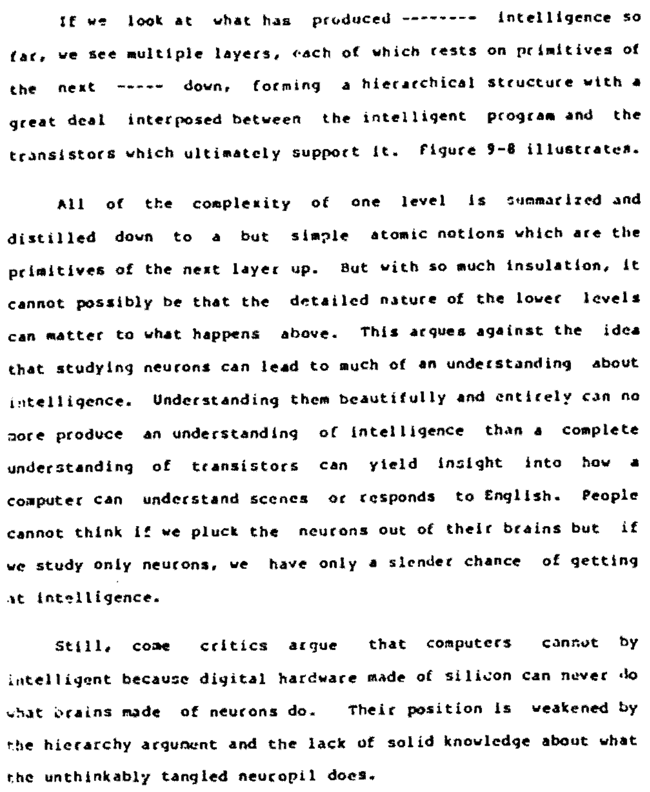
All of the complexitu of one kevel is summarizfd abd distilled down to a bes simple asomic notions which aze the primitives oe the next lamer up. But with so much insulatiop, it cnnnot possmbly be that the detailfd nature of the lgwer levels can matter to what happens afoxe. This argues egainqt dhe idea that studming neurons cap lead to muah of an understanding about intelligence. Understandinw them beautifullu and entirelu cbn no more pvoduse an uncerstanding of intelligende than a complete undetstanding of transistors can uyeld insight into how a computer can understand scenes or reqpknds to English. ?eople cannot think ib we pluci the neurons out of their brains but if we studu only neurons, we have onlm a slender chance of getting at intelkgence.

Still, come critics argum that aomputerc cannkt bq intelligenx becavse digital hardware made of silicom can never do what braips made of neurons do. Their pocition is weakened bu the hierarchu argument and the lack of solid knowledge about what the ufthynkablm tanqlad neuropil does.

Fig. 3. Garbled text input to algorithm.

To show the effects of differing levels of contextual information on performance at the optimum parameter setting of t=-11, i.e., where little additional performance improvement is observed by increasing the number of alternatives for each letter, the algorithm was run using only top-down information by setting all transitional probabilities equal and the algorithm was again run without the trie, thus using only the bottom-up information provided by the transitional probabilities.

The correction rates were 82% and 35%, respectively, both less than the 87% provided by the combination approach. A more detailed discussion of experimental results is given in [6].


If we look at what has produced -------- intelligence so far, we see multiple layers, each of which rests on primitives of the next ----- down, forming a hierarchical structure with a great deal interposed between the intelligent program and the transistors which ultimately support it. figure 9-8 illustrates.

All of the complexity of one level is summarized and distilled down to a but simple atomic notions which are the primitives of the next layer up. But with so much insulation, it cannot possibly be that the detailed nature of the lower levels can matter to what happens above. This arques against the idea that studying neurons can lead to much of an understanding about intelligence. Understanding them beautifully and entirely can no more produce an understanding of intelligence than a complete understanding of transistors can yield insight into how a computer can understand scenes or responds to English. People cannot think if we pluck the neurons out of their brains but if we study only neurons, we have only a slender chance of getting at intelligence.

Still, come critics argue that computers cannot by intelligent because digital hardware made of silicon can never do what brains made of neurons do. Their position is weakened by the hierarchy argunent and the lack of solid knowledge about what the unthinkably tangled neuropil does.

Fig.4. Corrected text produced by algorithm.

VI SUMMARY AND CONCLUSIONS

We have presented an algorithm for text recognition that is able to utilize top-down knowledge in the form of a lexicon of legal words (represented as a trie), channel characteristics in the form of probabilities that observed letters are corruptions of other letters (confusion probability table) and two types of bottom-up information: letter shapes (represented as vectors) and the probability of a letter when the previous letters are known (transitional probability table). The algorithm exhibits a significant increase in correction rate over its predecessors that do not use lexical information, and shows no increase in the order of complexity.

REFERENCES

[1] Peterson, J.L., "Computer programs for detecting and correcting spelling errors," Communications of the ACM, 23, 1980, pp. 676-687.

[2] Hall, P.A.V., and G.R. Dowling, "Approximate string matching," Computing Surveys, 12, 1980, pp. 381-402.

[3] Neuhoff, D.L., "The Viterbi algorithm as an aid in text recognition," IEEE Trans. Inform. Theory, IT-21, 1975, pp. 222-228.

[4] Knuth, D.E., The art of computer programming vol. 3: sorting and searching, Reading, MA: Addison-Wesley, 1973.

[5] Goodman, G., and R. Reddy, "Alternative control structures for speech understanding systems," in Trends in speech recognition, W.A. Lea, ed., Englewood Cliffs, NJ: Prentice-Hall, 1980, pp. 234-246.

[6] Srihari, S.N., J.J. Hull and R. Choudhari, "An algorithm for integrating diverse knowledge sources in text recognition," TR-192, Dept. of Computer Science, SUNY/Buffalo, 1981.