Shortest Path Segmentation: A Method for Training a Neural Network to Recognize Character Strings

C. J. C. Burges
O. Matan*
Y. Le Cun
J. S. Denker
L. D. Jackel
C. E. Stenard
C.R. Nohl
J.I. Ben

AT&T Bell Laboratories, Room 2G-235, Crawford's Corner Road, Holmdel, NJ 07733

ABSTRACT

We describe a method which combines dynamic programming and a neural net recognizer for segmenting and recognizing character strings. The method selects the optimal consistent combination of cuts from a set of candidate cuts generated using heuristics. The optimal segmentation is found by representing the image, the candidate segments, and their scores, as a graph in which the shortest path corresponds to the optimal interpretation. The scores are given by the neural net outputs for each segment. A significant advantage of the method is that the labor required to manually segment images is eliminated: the dynamic programming stage both performs the segmentation and provides inputs and desired outputs to the neural network during training. The training examples thus generated contain examples of both characters and "non-characters".

The system was trained on approximately 7000 unsegmented handwritten ZIP Codes provided by the United States Postal Service. The system has achieved a per-ZIP Code raw recognition rate of 81% on a 2,368 hand written ZIP Code test set, and gets 3.6% error (as a percent of those accepted) at 60% correct.

---

* Current address: Department of Computer Science, Stanford University, Stanford, CA 94305

0-7803-0559-0/92 $3.00

1. INTRODUCTION

Recognition of strings of (possibly connected) numerals is a problem of both commercial and academic interest. Several workers have recently presented results on this problem[1] [2] [3] [4] [5]. Of the approaches considered, ones that combine the recognition and segmentation process seem most promising[6]. In this paper we present a dynamic programming based method for recognition driven segmentation of handwritten strings of numerals. The method is designed to be generalizable to other related problems (such as recognition of arbitrary length digit strings, or recognition of alpha strings). The method selects the optimal consistent combination of cuts from a set of candidate cuts generated using heuristics. The optimal segmentation is found by representing the image, the candidate segments, and their scores, as a graph in which the shortest path corresponds to the optimal interpretation. A significant advantage of the method is that it allows us to eliminate the labor intensive manual segmentation of the images prior to neural network training. In addition, the method can be used to train a network "dynamically". By this we mean that the data presented to the net can change as training proceeds: "counterexamples" (mistakes the net makes) can thus be generated, and trained on, automatically. Finally, once the net has been trained, the method can be easily extended to generate the K rank ordered top choices (where the choices may differ in the segmentation, the recognition results, or both), and can also be used to attach scores to entries in a lexicon (and thus rank the lexicon).

The data used for this project consists of approximately 11,000 212 dpi binary United States Postal Service ZIP Code images. Approximately 7,000 were used for training, 1,000 for validation, and 3,000 for testing. The neural network used is a 20 by 20 input pixel version of the one described in Reference[7].

In the following, we use the word "cut" to mean a line (which may or may not be straight) drawn from the top to the bottom of the image, "cell" to denote a piece of the image between two adjacent cuts, and "segment" to mean a combination of one or more adjacent cells. A "definite cut" is a cut that is assumed to be correct.

In Section 2, we briefly describe the image preprocessing we perform, and the methods we use to construct the set of cells. We then describe how particular segmentations of the image can be represented as paths through a directed acyclic graph. In Section 3 we describe how we used these techniques to train our network, and in Sections 4, 5 and 6 we give results and draw conclusions.

2. SHORTEST PATH SEGMENTATION

Shortest Path Segmentation (SPS) rests on the assumption that the (gray scale or binary) image has been cut into a set C of cells in such a way that the set S of correct segments (those corresponding to the individual characters) can be constructed from combinations of cells in the set C. For example, the digit string in Figure 1 has been cut into 7 cells, of which the first two combined, the second, the next two combined, and the last two constitute the correct set of segments. In addition, SPS requires a recognizer that not only attaches a classification for each segment, but that also gives a score.

The methods we describe below are applicable to any method of construction of C.

2.1 IMAGE PREPROCESSING

The image preprocessing, prior to cut generation, follows that described in[3]. Underlines are removed, the image is deslanted and deskewed, and then flyspecks (small connected components) and intruding strokes from other fields are removed. Then, the image is normalized to 20 by P pixels, where P is chosen so that the aspect ratio of the image is unchanged. This is done so that images can be sent to the recognizer without further renormalization. Note that this resampling often results in an effectively gray scale image, so all techniques described from here on are equally applicable to gray scale images. The upper and lower contours are then used to clip long tails of digits, in both horizontal and vertical directions.

In the next step, definite cuts in the image are identified and the image is accordingly shrunk. A cut is taken to be definite if it lies within sufficiently many contiguous columns of background; the image is shrunk by removing excess columns of background. Definite cuts are important in that they can be used to reduce the number of calls to the recognizer for a given image.

2.2 CUT GENERATION

In generating the cuts, the object is to construct a segmentation of the image with as few cells as possible, yet such that the correct segmentation is contained in some combination of the cells. Cuts are constructed by first identifying a set of "start points" along the top and bottom of the image. The start points are computed from minima of the upper contour (or maxima of the lower contour). To construct the cuts given the start points, we use an algorithm, which we call the Modulated Gradient Hit or Deflect algorithm, which is based on the binary Hit or Deflect algorithm[8]. Modulated Gradient Hit or Deflect extends the basic Hit or Deflect algorithm to grey scale images. It uses the ink gradient to determine the path the cut takes: in addition, the sign of the gradient is changed if the current ink value is above a threshold (hence "Modulated Gradient"). This has the effect of keeping the cut *within* ink if it gets deep enough inside some ink; this addresses the problem that sometimes the cut must separate two heavily touching digits.

Some simple heuristics are then applied to clean up the set of cuts (for example, if cuts collide, only the straightest is kept).

2.3 GRAPHICAL REPRESENTATION of the SEGMENTATION PROCESS

The number of cuts generated by the above process is not known beforehand, but depends on the image. The method described here assumes that the expected number of characters in the answer is known; however, if this is not the case, the method could be applied for each of a range of possible word lengths, and the overall (suitably normalized) best score taken.

Figures 1 and 2 show the process for one particular ZIP Code. The image after preprocessing is shown at the top of Figure 1. The (six) candidate cuts are shown in the middle of the Figure. Since we are expecting 5 digits, we know that two of the cuts must be incorrect, and therefore that up to three cells must be combined into segments. If N is the expected number of characters in the image and C the number of cuts resulting from the segmentation process, then the maximum segment width (in cells) is $C-N+2$.

In general, an acceptable segmentation of the image may contain overlapping segments, or parts of the image may be skipped (for example, if the image contains several columns of noise). Given a set of candidate cuts and a set of rules defining how many cells two segments are allowed to overlap by, and how many cells are allowed to be skipped between adjacent segments, one can construct the set of all possible "legal" segments of the image. (In general, if W_{Max} (W_{Min}) is the maximum (minimum) allowed segment width in cells, and if C is the number of cuts resulting from the segmentation process, then the number of legal segments is given by $Numseg = (W_{Max} - W_{Min} + 1)(C + 2 - (W_{Max} + W_{Min})/2)$. Note W_{Min} is usually equal to one).

Now associate with each such segment a node in a graph. Connect nodes with directed arcs such that two nodes are connected if and only if the segments they represent are legal neighbors. If we consider the graph to be arranged from left to right (as in Figure 2), then the left (right) part of the graph corresponds to the left (right) part of the image. Thus there is a one-to-one correspondence: every path through the graph corresponds to a particular legal segmentation of the image, and every possible legal segmentation of the image corresponds to a particular path through the graph. Note that the number of nodes does not depend on the number of skip or overlap cells allowed, but the number of arcs does. In the current application, we take the number of allowed skip cells, and the number of allowed overlap cells, to be both equal to zero. Figure 2 represents the graph corresponding to the segmentation shown in Figure 1. In Figure 2 (in which for clarity only a few arcs are drawn), an arbitrary node X, which corresponds to segment S in Figure 1, is chosen for illustration. The graph is drawn so that all nodes in a column correspond to segments with the

same left hand edge, and all nodes in a row correspond to segments with the same right hand edge. Thus node *X* is of width 2 cells, and overlaps (by one cell) with all nodes in the column to its immediate right. The two small boxes at the right and left hand edges of the graph are added to ensure that all paths start (and end) on one node.

In Figure 1, one of the cuts (between the "0" and the "1") was found to be definite. This allows us to recursively prune the graph: any node which corresponds to a segment which falls across a definite cut can be removed from the graph. For example, the node labeled "Y" in Figure 2 corresponds to the segment consisting of the last two cells in the image, which straddles a definite cut; accordingly, all arcs leaving or terminating on that node can be removed from the graph. Since all arcs terminating on that node are also removed, further nodes upstream may also become "sterile" (have no child nodes) and can be removed (hence the recursion).

When the graph has been constructed and pruned, each remaining node corresponds to a segment which must be sent to the recognizer for classification and scoring. Thus the number of calls to the recognizer equals the number of nodes in the pruned graph.

Each node in the graph is then assigned a "length" which is derived from the recognizer score for the segment corresponding to that node. In our case the recognizer score was converted to a probability using the Softmax algorithm[9], and the probabilities are converted to log likelihoods by taking their negative log. The shortest path through the graph thus corresponds to the best overall recognition and segmentation of the image. The shortest path in this case is shown by the white line in Figure 2, and corresponds to the "Segments Chosen by System" shown in Figure 1. The "Score" in Figure 1 is simply the product of the probabilities associated with each of the five segments chosen. In general this score is then used as a threshold to accept or reject answers.

3. NEURAL NETWORK TRAINING

The method described above is reminiscent of techniques used in speech recognition[10], and in fact the Viterbi algorithm[11] provides a convenient means for rapidly finding the shortest path which traverses *N* nodes (corresponding to an interpretation for *N* characters). In this Section we describe a modification to the Viterbi algorithm which allows us to automatically segment the training images.

We call the modified algorithm the "Constrained" Viterbi algorithm. The idea is to restrict the search of the paths in the graph to just those paths that give a particular, chosen solution. For example, suppose that the ZIP Code shown in Figure 1 was a training image, and that we therefore knew that it corresponded to the string "37501". Then the constrained Viterbi algorithm will give us the highest scoring segmentation which gives the answer "37501". The associated probability is necessarily less than or equal to that for the unconstrained answer (which in this case may or may not be "37501").

The constrained Viterbi algorithm is executed as follows: consider the trellis that is constructed during the execution of the Viterbi algorithm[11]. The nodes in the first time step of the trellis correspond to all nodes that can be reached after traversing just one arc of the graph, similarly for the second time step, etc. Then instead of taking the highest probability associated with a given node, we take the probability corresponding to the given answer. Thus, in the above example, the nodes in the first time step of the trellis would be scored according to the neural networks' output for "3", the nodes for the second time step of the trellis would be scored according to the nets' output for "7", and so on. *Note that the same node can appear at more than one time step*, hence we need to keep all ten outputs from the network for each node in the graph. In Figure 3 we give a formal statement of the algorithm, following the notation of reference[11]. In the Constrained version, the value associated with a given node becomes a function of the time step *k*.

The net result is that the "Constrained" shortest path corresponds to the best segmentation which gives the known answer. Using this, we can create segmented images automatically, for training.

We now give a high level description of the training techniques used. We started with a "bootstrap network", i.e. a network which has been trained on a small set of single digits. To describe the training process, we now follow the process for just one image. First, a test is done on the image as though it were a test image; the resulting answer and chosen (*Un*constrained) segmentation are stored. Next, the same process is followed, but we use the Constrained Viterbi algorithm, using the known "truth", to do the segmentation. We assume that the segmentation thus found is correct. The segmented digits found from the Constrained Viterbi segmentation are then trained on positively (in the following, by "positive training" we mean training where the appropriate net output is set to +1, all others set to -1; negative training corresponds to all outputs set to -1). Then, the solution found from the *Un*constrained Viterbi algorithm is examined. If the segmentation is the same as that found in the Constrained case, but one or more of the digit answers differ, the images that the net is getting wrong are set aside for extra *positive* training later. If the two segmentations differ, the *Un*constrained answer is assumed to be incorrect, and the incorrectly segmented images are trained on *negatively*, and are also set aside, for extra *negative* training later. The extra training ("over training") feature is implemented with three First-In, First-Out buffers of different lengths, which results in three extra training passes after three different delays (depending on the size of the buffers). One result of this training scheme is that the net is always being trained on its current mistakes.

4. USE of a LEXICON

The above methods can be extended easily to incorporate a lexicon into the recognition process. If we wish to rank order the entries in a lexicon, we can populate the graph and then simply apply the Constrained Viterbi algorithm for each lexicon entry. Clearly this will only be practical for sufficiently limited lexicon sizes. For larger lexicon sizes, we can use information from the K shortest paths in the graph (for example, by taking the shortest path which also corresponds to an entry in the lexicon) to improve recognition rates. The results of applying these techniques for ZIP Codes are given below.

5. RESULTS

On a test set of 2398 5-digit ZIP Code images, the system currently attains a raw recognition rate (per ZIP Code) of 81%. If we reject sufficient images such that, of those we accept, 60% (as a percentage of the total number of images) are correct, then we get an error rate of 3.6% (as a percentage of those accepted). If we combine information from the five shortest paths and the ZIP Code lexicon (approximately 41% of all possible 5-digit ZIP Codes are legal), the error rate drops to 2.1% at 60% correct.

We can combine these results with those of a previous system[3], which is also neural network based and which is designed specifically to solve the ZIP Code problem. For that system, the ZIP Codes in the training set had to be manually segmented, and counterexamples generated and classified manually after each training session. The system obtains 77% raw recognition rate and also 3.6% error at 60% correct. Using a simple voting scheme, the combined systems achieve 83% raw recognition rate and 0.7% error at 60% correct.

6. CONCLUSIONS

We have shown how dynamic programming techniques can be used to rapidly find the best amongst a (possibly large) set of possible segmentations of an image. In addition we have shown how such a system can be used to automatically segment images for neural network training. The dynamic programming approach lends itself easily to generating a ranked list of answers, and to scoring a given lexicon. The approach described above is very general and we are currently studying how to extend it to reading cursive script.

7. ACKNOWLEDGEMENTS

The authors are greatly indebted to Dr. C. O'Connor of the Office of Advanced Technology of the United States Postal Service, and to Dr. J. Tan and Dr. G. Houles of Arthur D. Little Inc., for their support and guidance of this work, and would also like to thank Dr. J. Hull of SUNY Buffalo, for providing the ZIP Code data used.

REFERENCES

1. R. Fenrich, *Segmentation of Automatically Located Handwritten Words*, International Workshop on Frontiers in Handwriting Recognition, September 1991
2. M. Shridhar, Session Chairman, *Session on Hand Written Digit Recognition*, United States Postal Service Advanced Technology Conference, November 5-7, 1990
3. O. Matan et al, *Reading Handwritten Digits: A ZIP Code Recognition System*, to appear in IEEE Computer magazine
4. G.L. Martin, *Recognizing Overlapping Hand-Printed Characters by Centered-Object Integrated Segmentation and Recognition*, Neural Information Processing Systems Conference Proceedings, 1991
5. J.D. Keeler and D.E. Rumelhart, *Integrated Segmentation and Recognition of Hand-Printed Numerals*, Neural Information Processing Systems Conference Proceedings, 1990 and 1991
6. E. Lecolinet and J. Moreau, *A New System for Automatic Segmentation and Recognition of Unconstrained Handwritten ZIP Codes*, 6th Scandinavian Conference on Image Analysis, Finland, June 1989
7. Y. LeCun et al, *Handwritten ZIP Code Recognition with MultiLayer Networks*, in Proceedings of the 10th International Conference on Pattern Recognition, IEEE Computer Society Press, 1990
8. See, for example, R. Fenrich and S. Krishnamoorthy, *United States Postal Service Advanced Technology Conference*, November 1990, Volume 1
9. J.S. Bridle, *Probabilistic Interpretation of Feedforward Classification Network Outputs with Relationships to Statistical Pattern Recognition*, in F. Fougelman-Soulie and J. Herault, editors, Neuro-computing: Algorithms, Architectures and Applications, Springer Verlag, 1989
10. L.R. Rabiner, *A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition*, Proceedings of the IEEE, Vol. 77, No. 2, February 1989
11. G. David Forney, Jr., *The Viterbi Algorithm*, Proceedings of the IEEE, Vol. 61, No. 3, March 1973

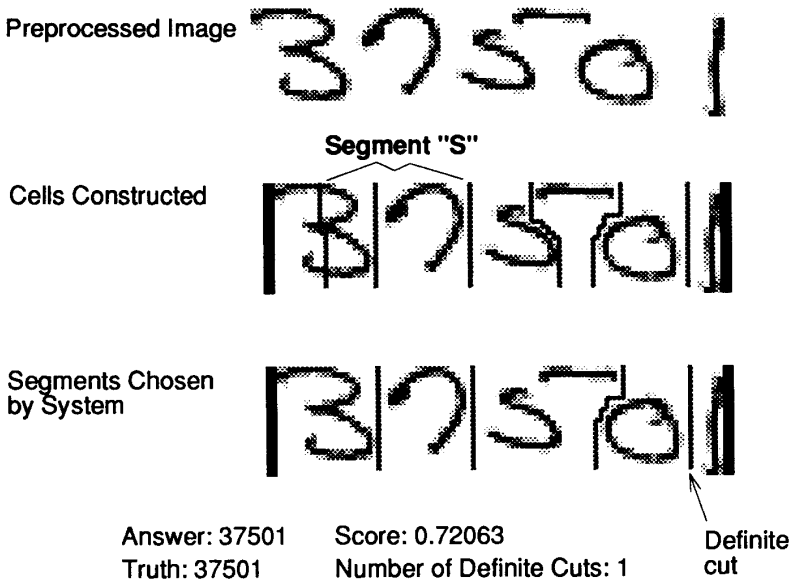

Figure 1

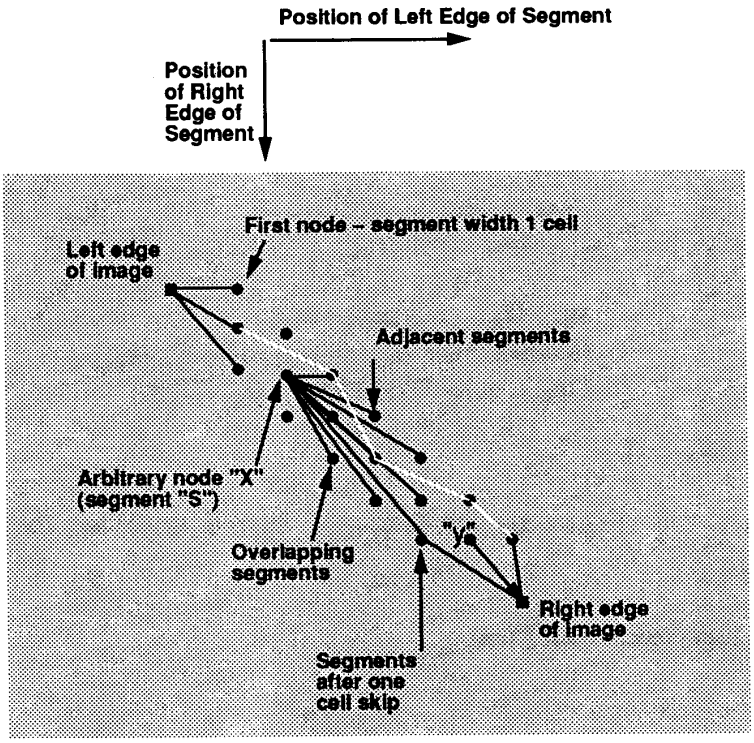

Figure 2

Viterbi Algorithm (adapted from Reference 11)

Definitions:

M	Number of Nodes
$\mathbf{x}(x_k)$	Path terminating at x_k
$\lambda(\mathbf{x}_k)$	Length of path $\mathbf{x}_k$
V_{x_k}, $1 \le x_k \le M$	Value (length) associated with node x_k
ξ	An arc in the trellis

Storage:

k	(Time index)
$\hat{\mathbf{x}}(x_k)$, $1 \le x_k \le M$	(Survivor terminating in x_k)
$\Gamma(x_k)$, $1 \le x_k \le M$	(Survivor length)

Initialization:

$k = 0$;

$\hat{\mathbf{x}}(x_0) = x_0$; $\hat{\mathbf{x}}(m)$ arbitrary, $m \neq x_0$;

$\Gamma(x_0) = 0$; $\Gamma(m) = \infty$, $m \neq x_0$.

Recursion:

Set $\lambda[\xi_k = (x_{k+1}, x_k)] = V_{x_k}$

(Modified: Set $\lambda[\xi_k = (x_{k+1}, x_k)] = V_{x_k}(k)$)

$\Gamma(x_{k+1}, x_k) \equiv \Gamma(x_k) + \lambda[\xi_k = (x_{k+1}, x_k)]$

for all $\xi_k = (x_{k+1}, x_k)$.

Find $\Gamma(x_{k+1}) = \min_{x_k} \Gamma(x_{k+1}, x_k)$ for each x_{k+1}; store $\Gamma(x_{k+1})$ and the corresponding survivor $\hat{\mathbf{x}}(x_{k+1})$.

Set $k = k+1$ and repeat until $k = K$.

Figure 3