LeRec: A NN/HMM Hybrid for On-Line Handwriting Recognition

Yoshua Bengio*
bengioy@iro.umontreal.ca

Yann LeCun
yann@research.att.com

Craig Nohl
nohl@research.att.com

Chris Burges
burges@research.att.com

AT&T Bell Laboratories
Rm 4G332, 101 Crawfords Corner Road
Holmdel, NJ 07733

Published in *Neural Computation*, vol. 7, no. 6, pp. 1–5.

*also, Dept. IRO, Université de Montréal, C.P. 6128, Succ. Centre-Ville, Montréal, Qc, H3C-3J7, Canada

Abstract

We introduce a new approach for on-line recognition of handwritten words written in unconstrained mixed style. The preprocessor performs a word-level normalization by fitting a model of the word structure using the EM algorithm. Words are then coded into low resolution "annotated images" where each pixel contains information about trajectory direction and curvature. The recognizer is a convolution network which can be spatially replicated. From the network output, a hidden Markov model produces word scores. The entire system is globally trained to minimize word-level errors.

1 Introduction

Natural handwriting is often a mixture of different "styles", lower case printed, upper case, and cursive. A reliable recognizer for such handwriting would greatly improve interaction with pen-based devices, but its implementation presents new technical challenges. Characters taken in isolation can be very ambiguous, but considerable information is available from the context of the whole word. We propose a word recognition system for pen-based devices based on four main modules: a preprocessor that normalizes a word, or word group, by fitting a geometrical model to the word structure using the EM algorithm; a module that produces an "annotated image" from the normalized pen trajectory; a replicated convolutional neural network that spots and recognizes characters; and a Hidden Markov Model (HMM) that interprets the networks output by taking word-level constraints into account. The network and the HMM are *jointly* trained to minimize an error measure defined at the word level.

Many on-line handwriting recognizers exploit the sequential nature of pen trajectories by representing the input in the time domain. While these representations are compact and computationally advantageous, they tend to be sensitive to stroke order, writing speed, and other irrelevant parameters. In addition, global geometric features, such as whether a stroke crosses another stroke drawn at a different time, are not readily available in temporal representations. To avoid this problem we designed a representation, called AMAP, that preserves the pictorial nature of the handwriting.

In addition to recognizing characters, the system must also correctly segment the characters within the words. To choose the optimal segmentation and take advantage of contextual and linguistic structure, the neural network is combined with a graph-based post-processor, such as an HMM. One approach, that we call INSEG, is to recognize a large number of heuristically segmented candidate characters and combine them optimally with a post-processor (Burges et al., 1992; Schenkel et al., 1993). Another approach, that we call OUTSEG, is to delay all segmentation decisions until after the recognition, as is often done in speech recognition. An OUTSEG recognizer must accept entire words as input and produce a sequence of scores for each character at each location on the input (Matan et al., 1992; Keeler and Rumelhart, 1991; Schenkel et al., 1993). Since the word normalization cannot be done perfectly, the recognizer must be robust with respect to relatively large distortions, size variations, and translations. An elastic word model –e.g., an HMM– can extract word candidates from the network output. The HMM models the long-range sequential structure while the neural network spots and classifies characters, using local spatial structure.

2 Word Normalization

Input normalization reduces intra-character variability, simplifying character recognition. We propose a new word normalization scheme, based on fitting a geometrical model of the word structure. Our model has four "flexible" lines representing respectively the ascenders line, the core line, the base line and the descenders line (see Figure 1). Points $(\mathbf{x}, \mathbf{y})$ on the lines are parameterized as follows:

$$\mathbf{y} = f_j(\mathbf{x}) = k(\mathbf{x} - x_0)^2 + s(\mathbf{x} - x_0) + y_{0j} \tag{1}$$

where k controls curvature, s is the skew, and (x_0, y_0) is a translation vector. The parameters k, s, and x_0 are shared among all four curves, whereas each curve has its own vertical translation parameter y_{0j}. The free parameters of the fit are actually k, s, a (ascenders y_0 minus baseline y_0), b (baseline y_0), c (core line y_0 minus baseline y_0), and d (baseline y_0 minus descenders y_0), as shown in Figure 1. x_0 is determined by taking the average abscissa of vertical extrema points. The lines of the model are fitted to the *extrema of vertical displacement*: the upper two lines to the vertical maxima of the pen trajectory, and the lower two to the minima. The line parameters $\theta = \{a, b, c, d, k, s\}$ are tuned to maximize the joint probability of observed points and parameter values:

$$\theta^* = \underset{\theta}{\arg\max} \quad \log P(Z \mid \theta) + \log P(\theta) \tag{2}$$

where Z represents the extrema y-position when given their x-position.

$P(Z|\theta)$ is modeled by a mixture of Gaussians (one Gaussian per curve), whose means are the functions of x given in equation 1:

$$P(y_i \mid x_i, \theta) = \sum_{j=0}^{3} w_k N(y_i; f_j(x_i), \sigma_y) + P_{\text{background}}(y_i) \tag{3}$$

where $N(y; \mu, \sigma)$ is the likelihood of y under a univariate Normal model (mean μ, standard deviation σ). $P_{\text{background}}(y_i)$ is a uniform background model that prevents outlying points (far from the four lines) to disturb the estimation of θ. The w_k are the mixture parameters, some of which are set to 0 in order to constrain the upper (lower) points to be fitted to the upper (lower) curves. They are computed *a priori* using measured frequencies of associations of extrema to curves on a large set of words. Priors $P(\theta)$ on the parameters (modeled here with Normal distributions) are important to prevent the collapse of the curves. They can be used to incorporate *a priori* information about the word geometry, such as the expected position of the baseline, or of the height of the word. These priors are also used as initial values in the EM optimization of the fit function. The prior distribution for each parameter (independently) is a Normal, with the standard deviation controlling the strength of the prior. In our experiments, these priors were set using some heuristics applied to the input data itself. The priors for the curvature (k) and angle (s) are set to 0, while the ink points themselves are preprocessed to attempt remove the overall angle of the word (looking for a near horizontal projection with minimum entropy). To compute the prior for the baseline, the mean and standard deviation of y-position is computed (after rough angle removal). The baseline (b) prior is taken to be one standard deviation below the mean. The core line (c) prior is taken to

be two standard deviations above the baseline. The ascender (descender) line prior is taken to be between 1.8 (-0.9) and 3.0 (-2.0) times the core height prior, depending on the maximum (minimum) vertical position in the word.

The discrete random variables that associate each point with one of the curves are taken as hidden variables of the EM algorithm. One can thus derive an auxiliary function which can be analytically (and cheaply) solved for the 6 free parameters θ. Convergence of the EM algorithm was typically obtained within 2 to 4 iterations (of maximization of the auxiliary function).

3 AMAP

The recognition of handwritten characters from a pen trajectory on a digitizing surface is often done in the time domain (Tappert, Suen and Wakahara, 1990; Guyon et al., 1991). Typically, trajectories are normalized, and local geometrical or dynamical features are sometimes extracted. The recognition is performed using curve matching (Tappert, Suen and Wakahara, 1990), or other classification techniques such as Time-Delay Neural Networks (Guyon et al., 1991). While these representations have several advantages, their dependence on stroke ordering and individual writing styles makes them difficult to use in high accuracy, writer independent systems that integrate the segmentation with the recognition.

Since the intent of the writer is to produce a legible *image*, it seems natural to preserve as much of the pictorial nature of the signal as possible, while at the same time exploit the sequential information in the trajectory. We propose a representation scheme, called AMAP, where pen trajectories are represented by low-resolution images in which each picture element contains information about the local properties of the trajectory.

An AMAP can be viewed as a function in a multidimensional space where each dimension is associated with a local property of the trajectory, such as the direction of motion ϕ, the X position, and the Y position of the pen. The value of the function at a particular location (ϕ, X, Y) in the space represents a smooth version of the "density" of features in the trajectory that have values (ϕ, X, Y) (in the spirit of the generalized Hough transform (Ballard, 1981)).

An AMAP is implemented as a multidimensional array (in our case 5x20x18) obtained by discretizing the continuous "feature density" functions, which varies smoothly with position (X, Y) and other variables such as direction of motion ϕ, into "boxes". Each of these array elements is assigned a value equal to the integral of the feature density function over the corresponding box. In practice, an AMAP is computed as follows. At each sample on the trajectory, one computes the position of the pen (X, Y) and orientation of the motion ϕ (and possibly other features, such as the local curvature c). Each element in the AMAP is then incremented by the amount of the integral over the corresponding box of a predetermined *point-spread function* centered on the coordinates of the feature vector. The use of a smooth point-spread function (say a Gaussian) ensures that smooth deformations of the trajectory will correspond to smooth transformations of the AMAP. An AMAP can be viewed as an "annotated image" in which each pixel is a feature vector.

A particularly useful feature of the AMAP representation is that it makes very few assumptions about the nature of the input trajectory. It does not depend on stroke ordering or writing speed, and it can be used with all types of handwriting (capital, lower case, cursive, punctuation, symbols). Unlike many other representations (such as global features), AMAPs can be computed for complete words without requiring segmentation. In the experiments we used AMAPs with 5 features at each pixel location: 4 features are associated to four orientations (0^o, 45^o, 90^o, and 135^o); the fifth one is associated to local curvature. For example, when there is a nearly vertical segment in an area, nearby pixels will have a strong value for the first ("vertical") feature. Near endpoints or points of high spatial curvature on the trajectory, the fifth ("curvature") feature will be high. Curvature information is obtained by computing the cosine of the angle between successive elementary segments of the trajectory. Because of the integration of the Gaussian point-spread function, the curvature feature at a given pixel depends on the curvature at different points of the trajectory in the vicinity of that pixel.

4 Convolutional Neural Networks

Image-like representations such as AMAPs are particularly well suited for use in combination with Multi-Layer Convolutional Neural Networks (MLCNNs) (LeCun et al., 1989; LeCun et al., 1990). MLCNNs are feed-forward neural networks whose architectures are tailored for minimizing the sensitivity to translations, rotations, or distortions of the input image. They are trained to recognize and spot characters with a variation of the Back-Propagation algorithm (Rumelhart, Hinton and Williams, 1986; LeCun, 1986).

Each unit in an MLCNN is connected only to a local neighborhood in the previous layer. Each unit can be seen as a local feature detector whose function is determined by the learning procedure. Insensitivity to local transformations is built into the network architecture by constraining sets of units located at different places to use identical weight vectors, thereby forcing them to detect the same feature on different parts of the input. The outputs of the units at identical locations in different feature maps can be collectively thought of as a local feature vector. Features of increasing complexity and scale are extracted by the neurons in the successive layers.

Because of weight-sharing, the number of free parameters in the system is greatly reduced. Furthermore, MLCNNs can be scanned (replicated) over large input fields containing multiple *unsegmented* characters (whole words) very economically by simply performing the convolutions on larger inputs. Instead of producing a single output vector, such an application of an MLCNN produces a sequence of output vectors. The outputs detect and recognize characters at different (and overlapping) locations on the input. These multiple-input, multiple-output MLCNNs are called Space Displacement Neural Networks (SDNN) (Matan et al., 1992; Keeler and Rumelhart, 1991; Schenkel et al., 1993).

One of the best networks we found for character recognition has 5 layers arranged as illustrated in figure 2; layer 1: convolution with 8 kernels of size 3x3, layer 2: 2x2 subsampling, layer 3: convolution with 25 kernels of size 5x5, layer 4 convolution with 84 kernels of size 4x4, layer 5: 2x1 subsampling, classification layer: 95 radial basis function (RBF) units (one

per class). The subsampling layers are essential to the network's robustness to distortions. Hidden units of a sumbsampling layer apply the squashing non-linearity to a scaled and offset sum of their inputs (from the same feature map at the previous layer). For each feature map, there are two learned parameters in a subsampling layer: the scaling and bias, which control the effect of the non-linearity. The output layer is one (single MLCNN) or a series of (SDNN) 95-dimensional vectors, with a distributed target code for each character corresponding to the weights of the RBF units.

The choice of input field dimension was based on the following considerations. We estimated that at least 4 or 5 pixels were necessary for the *core* of characters (between baseline and core line). Furthermore, very wide characters (such as "w") can have a 3 to 1 aspect ratio. On the vertical dimension, it is necessary to leave room for ascenders and descenders (at least one core height each). In addition, extra borders allow outer edges of the characters to lie at the center of the receptive field of some units in the first layer, thereby improving the accuracy. Once the number of subsampling layers, and the sizes of the kernels are chosen, the sizes of all the layers, including the input, are determined unambiguously. The only architectural parameters that remain to be selected are the number of feature maps in each layer, and the information as to what feature map is connected to what other feature map. In our case, the subsampling rates were chosen as small as possible (2x2), and the kernels as small as possible in the first layer (3x3) to limit the total number of connections. Kernels sizes in the upper layers are chosen to be as small as possible while satisfying the size constraints mentioned above. The last subsampling layer performs a vertical subsampling to make the network more robust to errors of the word normalizer (which tends to create variations in vertical position). Several architectures were tried (but clearly not exhaustively), varying the type of layers (convolution, subsampling) the kernel sizes, and the number of feature maps.

Larger architectures did not necessarily perform better and required considerably more time to be trained. A very small architecture with half the input field also performed worse, because of insufficient input resolution. Note that the input resolution is nonetheless much less than for optical character resolution, because the angle and curvature provide more information than a single grey level at each pixel.

Training proceeded in two phases. First, we kept the centers of the RBFs fixed, and trained the network weights so as to maximize the logarithm of the output RBF corresponding to the correct class (maximum log-likelihood). This is equivalent to minimizing the mean-squared error between the previous layer and the center of the correct-class RBF. This bootstrap phase was performed on isolated characters. In the second phase, all the parameters, network weights and RBF centers were trained globally to minimize a discriminant criterion at the word level. This is described in more detail in the next section.

5 Segmentation and Post-Processing

The convolutional neural network can be used to give scores associated to characters when the network (or a piece of it corresponding to a single character output) has an input field, called a *segment*, that covers a connected subset of the whole word. A *segmentation* is a

sequence of such segments that covers the whole word. Because there are often many possible segmentations, sophisticated tools such as hidden Markov models and dynamic programming are used to search for the best segmentation.

In this paper, we consider two approaches to the segmentation problem called INSEG (for input segmentation) and OUTSEG (for output segmentation). In both approaches, the post-processors can be decomposed into two levels: 1) character-level scores and constraints obtained from the observations, 2) word-level constraints (e.g., from a grammar or dictionary). The INSEG and OUTSEG systems share the second level. The INSEG and OUTSEG architectures are depicted in Figure 3.

In an INSEG system, the network is applied to a number of heuristically segmented candidate characters. A *cutter* generates candidate *cuts*, which represent a potential boundary between two character segments. It also generates *definite cuts*, which we assume that no segment can cross. A *combiner* then generates the candidate segments, based on the cuts found.

The cutter module finds candidate cuts in cursive words (note that the data can be cursive, printed, or mixed). A superset of such cuts is first found, based on the pen direction of motion along each stroke. Next, several filters are applied to remove incorrect cuts. The filters use vertical projections, proximity to the baseline, and other similar characteristics. Horizontal strokes of "T"s that run into the next character (with no pen up) are also cut here.

Next, the combiner module generates segments based on these cuts. Heuristic filters are again used to significantly reduce the number of candidate segments down to a reasonable number. For example, segments falling across definite cuts, or that are too wide, or that contain too many strokes, are removed from the list of candidates; and segments that contain too little ink are forcibly combined with other segments. Finally, some segments (such as the horizontal or vertical strokes of T's, other vertical strokes that lie geometrically inside other strokes, etc) are also forcibly combined into larger segments.

The network is then applied to each of the resulting segments separately. These scores are attached to nodes of an *observation graph* in which the connectivity and transition probabilities on arcs represent segmentation and geometrical constraints (e.g., segments must not overlap and must cover the whole word, some transitions between characters are more or less likely given the geometrical relations between their images). Each node in the observation graph thus represents a segment of the input image and a candidate classification for this segment, with a corresponding score or cost.

In an OUTSEG system, all segmentation decisions are delayed until after the recognition (Matan et al., 1992; Keeler and Rumelhart, 1991; Schenkel et al., 1993), as is often done in speech recognition (Bengio et al., 1992). The AMAP of the entire *word* is shown to an SDNN, which produces a sequence of output vectors equivalent to scanning the single-character network over all possible pixel locations on the input. The Euclidean distances between each output vector and the targets are interpreted as log-likelihoods of the output given a class. To construct an *observation graph*, we use a set of character HMMs, modeling the sequence of network outputs observed for each character. We used three-state HMMs for each character, with a left and right state to model transitions and a center state for the character itself. The

observation graph is obtained by connecting these character models, allowing any character to follow any character.

On top of the constraints given in the observation graph, additional constraints that are independent of the observations are given by what we call a *grammar graph*, which can embody lexical constraints. These constraints can be given in the form of a dictionary or of a character-level grammar (with transition probabilities), such as a trigram. Recognition searches the best path in the observation graph that is compatible with the grammar graph. When the grammar graph has a complex structure (e.g. a dictionary), the product of the grammar graph with the observation graph can be huge. To avoid generating such a large data structure, we define the nodes of this product graph procedurally and we only instantiate nodes along the paths explored by the graph search (and pruning) algorithm.

With the OUTSEG architecture, there are several ways to put together the *within-character constraints* of the HMM observation graph with the *between-character constraints* of the grammar graph. The approach generally followed in HMM speech recognition system consists in taking the product of these two graphs and searching for the best path in the combined graph. This is equivalent to using the costs and connectivity of the grammar graph to connect together the character HMM models from the observation graph, i.e., to provide the transition probabilities *between* the character HMMs (after making duplicates of the character models for each corresponding character in the grammar graph). Variations of this scheme include pruning the search (e.g. with beam search) and separating the search in the observation graph and the grammar graph.

A crucial contribution of our system is the joint training of the neural network and the post-processor with respect to a single criterion that approximates word-level errors. We used the following *discriminant* criterion: minimize the total cost (sum of negative log-likelihoods) along the "correct" paths (the ones that yield the correct interpretations), while maximizing the costs of *all* the paths (correct or not). The discriminant nature of this criterion can be shown with the following example. If the cost of a path associated to the correct interpretation is much smaller than all other paths, the criterion is very close to 0 and almost no gradient is back-propagated. On the other hand, if the lowest cost path yields an incorrect interpretation but differs from a path of correct interpretation on a sub-path, then very strong gradients will be propagated along that sub-path, whereas the other parts of the sequence will generate almost no gradient. Within a probabilistic framework, this criterion corresponds to maximizing the mutual information (MMI) between the observations and the correct interpretation (Nadas, Nahamoo and Picheny, 1988). The mutual information $I(C,Y)$ between the correct interpretation C (sequence of characters) and the transformed observations Y (sequence of outputs of the last layer of the neural net before the RBFs) can be rewritten as follows, using Bayes' rule:

$$I(C,Y) = \log \frac{P(Y,C)}{P(Y)P(C)} = \log \frac{P(Y|C)}{P(Y)} \tag{4}$$

where $P(Y|C)$ is the likelihood of transformed observations Y constrained by the knowledge of the correct interpretation sequence C, $P(Y)$ is the unconstrained likelihood of Y (i.e., taking all interpretations possible in the model into account) and $P(C)$ is the prior probability of the sequence of characters C. Interestingly, when the class priors are fixed, maximizing $I(C,Y)$ is equivalent to maximizing the posterior probability of the correct sequence C, given the

observations Y (also known as the Maximum A Posteriori, or MAP, criterion):

$$P(C|Y) = \frac{P(Y|C)P(C)}{P(Y)} \tag{5}$$

Both the MMI and MAP criteria are more discriminant than the maximum likelihood criterion (maximizing $P(Y|C)$) because the parameters are used not to *model* the type of observations corresponding to a particular class C, but rather to *discriminate* between classes. The most discriminant criterion is the number of classification errors on the training set but, unfortunately, it is computationally very difficult to directly optimize such a discrete criterion.

During global training, the MMI criterion was optimized with a modified stochastic gradient descent procedure that uses second derivatives to compute optimal learning rates (LeCun, 1989) (this can be seen as a stochastic version of the Levenberg-Marquardt algorithm with a diagonal approximation of the Hessian). This optimization operates on *all* the parameters in the system, most notably the network weights and the RBF centers.

Experiments described in the next section have shown important reductions in error rates when training with this word-level criterion instead of just training the network separately for each character. Similar combinations of neural networks with HMMs or dynamic programming have been proposed in the past, for speech recognition problems (Bengio et al., 1992).

6 Experimental Results

In the first set of experiments, we evaluated the generalization ability of the neural network classifier coupled with the word normalization preprocessing and AMAP input representation. All results are in *writer independent* mode (different writers in training and testing). Initial training on isolated characters was performed on a database of approximately 100,000 printed characters (upper, lower, digits, and punctuation). Tests on a database of isolated characters were performed separately on the four types of characters: upper case (2.99% error on 9122 patterns), lower case (4.15% error on 8201 patterns), digits (1.4% error on 2938 patterns), and punctuation (4.3% error on 881 patterns). Experiments were performed with the network architecture described above. The enhance the robustness of the recognizer to variations in position, size, orientation, and other distortions, additional training data was generated by applying local affine transformations to the original characters.

The second and third set of experiments concerned the recognition of lower case words (writer independent). The tests were performed on a database of 881 words. First we evaluated the improvements brought by the word normalization to the INSEG system. For the OUTSEG system we *have* to use a word normalization since the network sees a whole word at a time. With the INSEG system, and before doing any word-level training, we obtained without word normalization 7.3% and 3.5% word and character errors (adding insertions, deletions and substitutions) when the search was constrained within a 25461-word dictionary. When using the word normalization preprocessing instead of a character level normalization, error rates dropped to 4.6% and 2.0% for word and character errors respectively, i.e., a relative drop of 37% and 43% in word and character error respectively.

In the third set of experiments, we measured the improvements obtained with the joint training of the neural network and the post-processor with the word-level criterion, in comparison to training based only on the errors performed at the character level. After initial training on individual characters as above, global word-level discriminant training was performed with a database of 3500 lower case words. For the OUTSEG system, without any dictionary constraints, the error rates dropped from 38% and 12.4% word and character error to 26% and 8.2% respectively after word-level training, i.e., a relative drop of 32% and 34%. For the INSEG system and a slightly improved architecture, without any dictionary constraints, the error rates dropped from 22.5% and 8.5% word and character error to 17% and 6.3% respectively, i.e., a relative drop of 24.4% and 25.6%. With a 25461-word dictionary, errors dropped from 4.6% and 2.0% word and character errors to 3.2% and 1.4% respectively after word-level training, i.e., a relative drop of 30.4% and 30.0%. Even lower error rates can be obtained by drastically reducing the size of the dictionary to 350 words, yielding 1.6% and 0.94% word and character errors.

The AMAP preprocessing with bi-dimensional multilayer convolutional networks was also compared with another approach developed in our laboratory (Guyon et al., 1991), based on a time-domain representation and a one-dimensional convolutional network (or Time-Delay Neural Network). The networks were not trained on the same data, but were both tested on the same database of 17858 isolated characters provided by AT&T Global Information Solutions (AT&T-GIS, formerly NCR) for comparing a variety of commercial character recognizers with the recognizers developed in our laboratory. Error rates for the AMAP network were respectively 2.0%, 5.4%, 6.7% and 2.5% on digits, upper case, lower case and a reduced set of punctuation symbols. On the same categories, the Time-Delay Neural Network (based on a temporal representation) obtained 2.6%, 6.4%, 7.7% and 5.1% errors, respectively. However, we noticed that the two networks often *made errors on different patterns*, probably because they are based on different input representations. Hence we combined their output (by a simple sum), and obtained on the same classes 1.4%, 3.9%, 5.3% and 2.2% errors, i.e., a very important improvement. This can be explained because these recognizers not only make errors on different patterns, but also have good rejection properties: the highest scoring class tends to have a low score when it is not the correct class.

AT&T-GIS conducted a test in which such a combined system was compared with 4 commercial classifiers on the printable ASCII set (isolated characters, including upper and lower case, digits, and punctuations). On this benchmark task, because characters are given in isolation without baseline information, there are inherent confusions between many sets of characters such as (“O”,“o”), (“P”,“p”), (“2”,“z”,“Z”), (“l”,“i” with no dot, “1”), (“;”, “i”), etc... Our recognizer obtained a character error rate 18.9%. The error rates obtained by the commercial recognizers were, in decreasing order of performance, 30.8%, 32.5%, 34.0%, and 39.0%. These results are illustrated in the bar chart of figure 4. Note however that the results are slightly biased by the fact that we are comparing a laboratory prototype to established, commercial systems with real-time performance.

7 Conclusion

We have demonstrated a new approach to on-line handwritten word recognition that uses word or sentence-level preprocessing and normalization, image-like representations, convolutional neural networks, graph-based word models, and global training using a highly discriminant word-level criterion. Excellent accuracy on various writer-independent tasks were obtained with this combination.

Acknowledgments

We would like to thank Isabelle Guyon for the fruitful exchanges of ideas and information on our approaches to the problem. Mike Miller and his colleagues at AT&T-GIS are gratefully acknowledged for providing the database and running the benchmarks. We would also like to acknowledge the help of other members of our department, in particular, Donnie Henderson, John Denker, and Larry Jackel. Y.B. would also like to acknowledge the support of the National Research and Engineering Research Council of Canada.

References

Ballard, D. (1981). Generalizing the hough transform to detect arbitrary shapes. *Pattern Recognition*, 13(2):111–122.

Bengio, Y., De Mori, R., Flammia, G., and Kompe, R. (1992). Global optimization of a neural network-hidden Markov model hybrid. *IEEE Transactions on Neural Networks*, 3(2):252–259.

Burges, C., Matan, O., LeCun, Y., Denker, J., Jackel, L., Stenard, C., Nohl, C., and Ben, J. (1992). Shortest path segmentation: A method for training a neural network to recognize character strings. In *Proc. International Joint Conference on Neural Networks*, volume 3, pages 165–172, Baltimore.

Guyon, I., Albrecht, P., Le Cun, Y., Denker, J. S., and W., H. (1991). design of a neural network character recognizer for a touch termin al. *Pattern Recognition*, 24(2):105–119.

Keeler, J. and Rumelhart, D.and Leow, W. (1991). integrated segmentation and recognition of hand-printed numerals. In Lippman, R. P., Moody, J. M., and Touretzky, D. S., editors, *Neural Information Processing Systems*, volume 3, pages 557–563. Morgan Kaufmann Publishers, San Mateo, CA.

LeCun, Y. (1986). Learning processes in an asymmetric threshold network. In Bienenstock, E., Fogelman-Soulié, F., and Weisbuch, G., editors, *Disordered Systems and Biological Organization*, pages 233–240. Springer-Verlag, Berlin, Les Houches 1985.

LeCun, Y. (1989). Generalization and network design strategies. Technical Report CRG-TR-89-4, Department of Computer Science, University of Toronto.

LeCun, Y., Boser, B., Denker, J., Henderson, D., Howard, R., Hubbard, W., and Jackel, L. (1989). Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1:541–551.

LeCun, Y., Boser, B., Denker, J., Henderson, D., Howard, R., Hubbard, W., and Jackel, L. (1990). Handwritten digit recognition with a back-propagation network. In Touretzky, D., editor, *Advances in Neural Information Processing Systems 2*, pages 396–404, Denver, CO. Morgan Kaufmann, San Mateo.

Matan, O., Burges, C., LeCun, Y., and Denker, J. (1992). Multi-digit recognition using a space displacement neural network. In Moody, J., Hanson, S., and Lipmann, R., editors, *Advances in Neural Information Processing Systems 4*, pages 488–495, San Mateo CA. Morgan Kaufmann.

Nadas, A., Nahamoo, D., and Picheny, M. (1988). On a model-robust training method for speech recognition. *IEEE Transactions on Acoustics, Speech and Signal Processing*, ASSP-36(9):1432–1436.

Rumelhart, D., Hinton, G., and Williams, R. (1986). Learning representations by back-propagating errors. *Nature*, 323:533–536.

Schenkel, M., Weissman, H., Guyon, I., Nohl, C., and Henderson, D. (1993). Recognition-based segmentation of on-line hand-printed words. In Hanson, S. J., Cowan, J. D., and Giles, C. L., editors, *Advances in Neural Information Processing Systems 5*, pages 723–730, Denver, CO.

Tappert, C., Suen, C., and Wakahara, T. (1990). The state of the art in on-line handwriting recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(12).

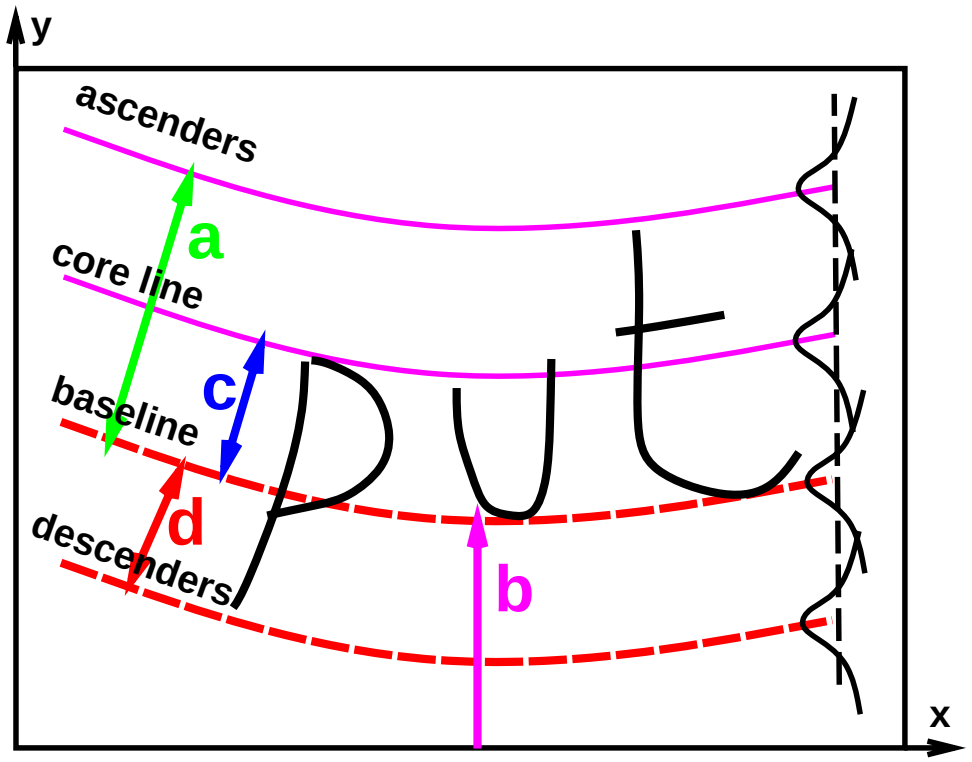

Figure 1: Word Normalization Model: Ascenders and core curves fit *y-maxima* whereas descenders and baseline curves fit *y-minima*. There are 6 parameters: **a** (ascenders curve height relative to baseline), **b** (baseline absolute vertical position), **c** (core line position), **d** (descenders curve position), **k** (curvature), **s** (angle).

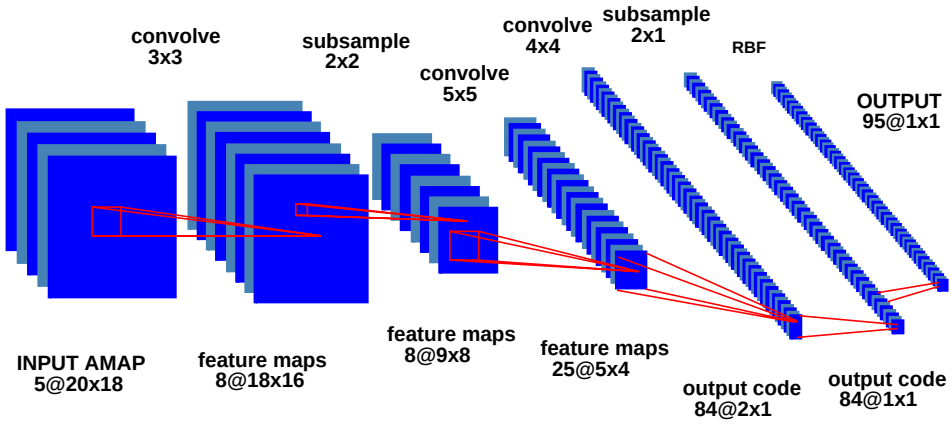

Figure 2: Convolutional Neural Network character recognizer. This architecture is robust to local translations and distortions, with subsampling, shared weights and local receptive fields.

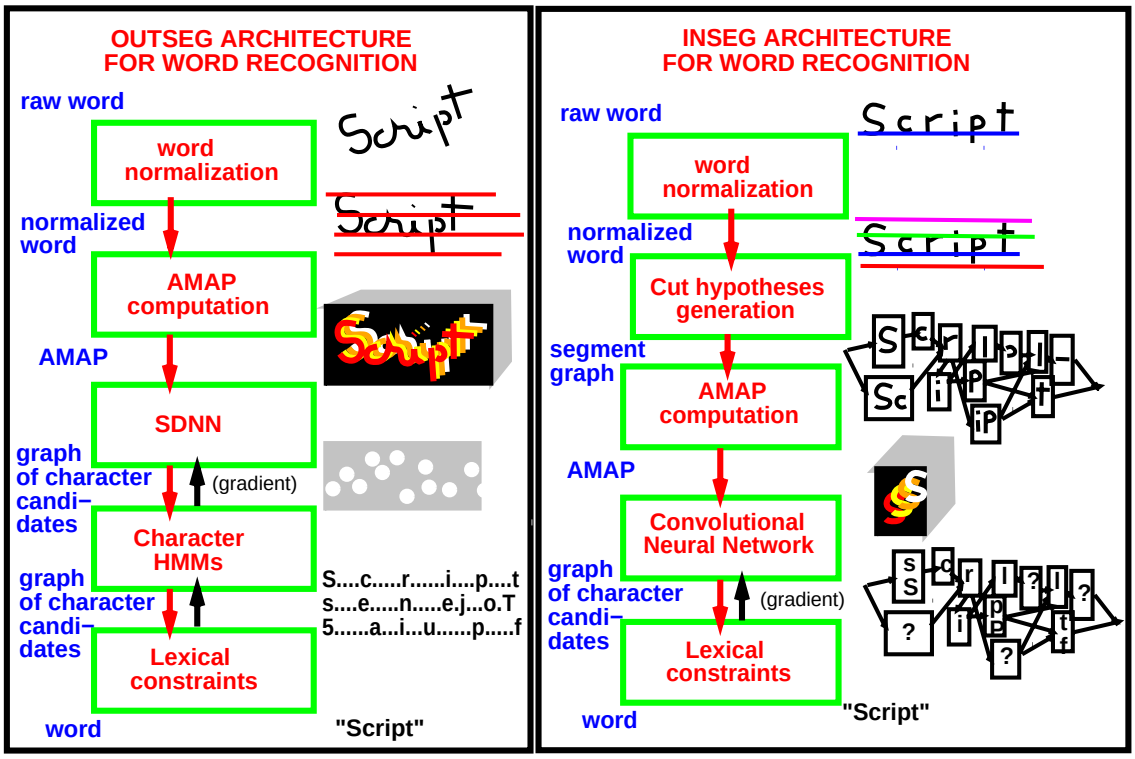

Figure 3: INSEG and OUTSEG architectures for word recognition.

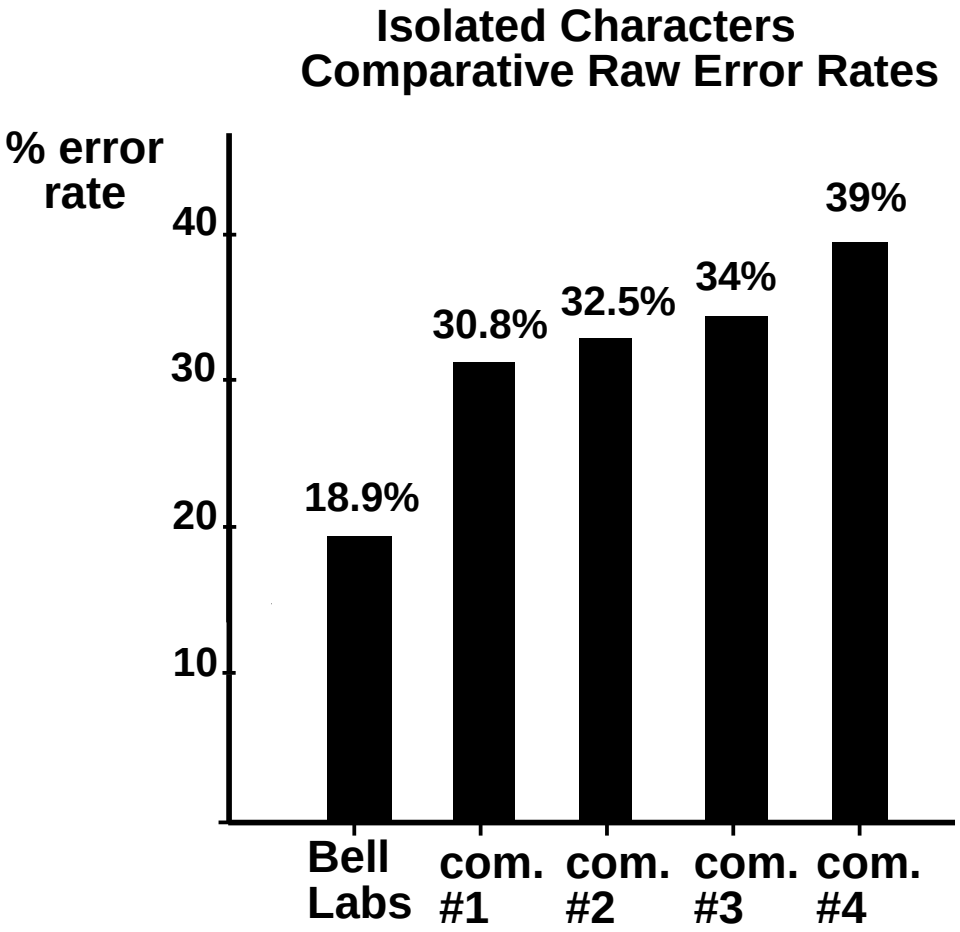

Figure 4: Comparative results on a benchmark test conducted by AT&T-GIS on isolated character recognition (uppers, lowers, digits, symbols). The last four bars represent the results obtained by five competing commercial recognizers.