Experimental Results on Improved Handwritten Word Recognition using the Levenshtein Metric

Philippe GENTRIC
Laboratoires d'Electronique Philips
22, av. Descartes
94453 LIMEIL-BREVANNES
FRANCE
Tel: (33) 45 10 68 12
Fax: (33) 45 10 67 43
E-mail: gentric@lep.lep-philips.fr

Abstract

In the context of dynamic handwriting recognition, i.e. handwriting acquired from the pen motion, we focus on the recognition of words written in the discrete mode i.e. characters should not overlap. The word recognition is made of a character recognizer and a lexical resource based on a Levenshtein-like distance. We demonstrate that the word recognition rate can be greatly improved by enhancing the nature of the information provided by the character recognition classifier to the lexical processor. A Radial Basis Function (RBF) classifier is used to provide accurate substitution costs in the Levenshtein metric lexical search scheme. We report experimental results that demonstrate a clear advantage of this method over the traditional use of the classifier confusion matrix for substitution cost estimation. We conjecture that this is probably related to the highly multi-modal and ambiguous nature of the handwritten character classification problem.

keywords:

Handwriting recognition, lexical processing, classification, radial basis function, neural networks

1 Introduction

In the context of dynamic handwriting recognition (i.e. handwriting acquired from the pen motion) we focus on the recognition of words of a given lexicon. In such systems the word recognition is built on a character recognizer assisted by a lexical resource. A segmentation module proposes *segmentation hypothesis* consisting of a list of *candidate characters*. Each candidate character is submitted to the character recognizer and the resulting string of recognition results is passed to the lexical resource. Hypothesis may differ because of the segmentation i.e. different pieces of the input signal are gathered in order to build characters, or because of the character recognition i.e for each candidate character the recognizer provides a list of possible results. Here, the lexical module is used to find the best hypothesis. It is not just a check that a given word belongs the lexicon; the lexical module finds the word of the lexicon that matches best the segmentation and the character recognition. In order to do this we need a quantitative estimation of how "good" is a segmentation and recognition result, this is done by using a Levenshtein metric [1]. This metric has to be computed for every word in the lexicon except if heuristics are used to discard non-pertinent words, for example the length of the word can be estimated and only words of neighboring length are tested. As this is computationally expensive the implementation is accelerated using a Dynamic Programming search [2]. The Levenshtein metric requires an estimation of the costs of the three basic operators: substitution, deletion and insertion. This estimation is generally based on statistical considerations. For example in Bozinovic and Srihari [3] a probabilistic finite state machine is used to model the segmentation errors (merging and splitting errors) and the substitution errors (recognition errors). In other words, the system is supposed to make errors and these errors are modeled in order to compute probability of alternatives in terms of Bayes' decision theory.

We focus here on the problem of computing substitution costs.

The most simple method for computing substitution cost consists in giving for each recognized character a zero substitution cost for all known alternatives and an infinite cost for all other characters. This method has the advantage of being very simple to implement in software as well as in hardware, leading to great execution speed. It is the base of most "spelling checker" programs. We will show however that in handwriting recognition the performance is poor.

Another widely used method consists in estimating the substitution costs using the class error probabilities -or classifier confusion matrix- as in [3]. For a given character X the substitution cost with another character Y is computed by counting how many times in the past the classifier gave the answer Y when a prototype labelled with the class X was presented.

We propose here a method based on computing the substitution costs using the class likelihood. This method is not based on a compilation of the previous classifier errors. Instead we use the fact that a classifier can be used not only for producing one class but for producing a ranked list of classes. Providing that the classifier has some basic properties that we will detail, this has the advantage that the alternatives are more *relevant* of the input character pattern, which gives a remarkable improvement of the word recognition performances.

We first briefly describe the two building blocks of the handwriting recognition system, the character recognition on one hand and the lexical processor on the other hand.

Then we report experimental results that demonstrate a clear advantage of our method over the traditional use of the classifier confusion matrix for substitution cost estimation. We show why RBF-based classifiers, having localised receptive fields, are a major asset in this respect.

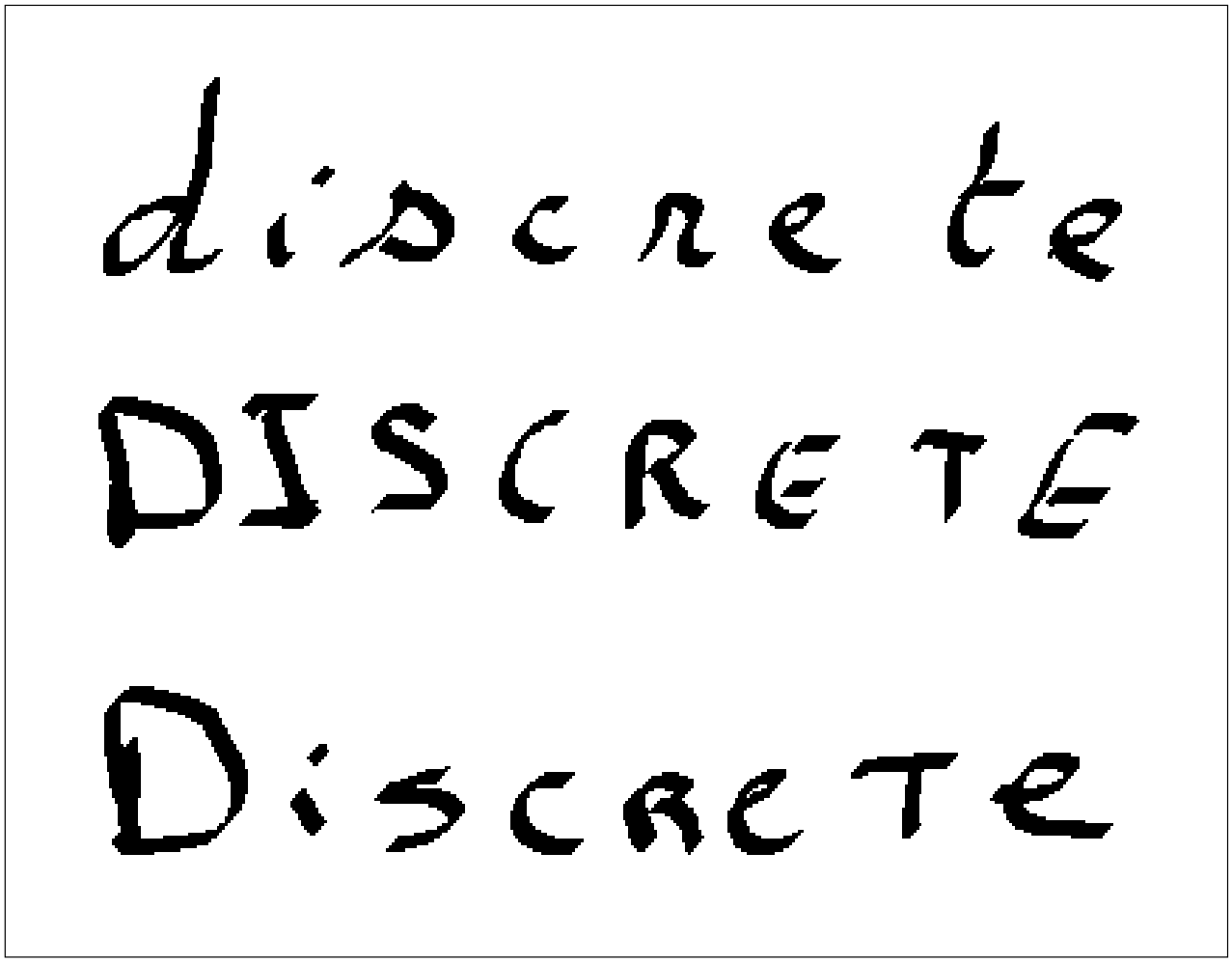

Figure 1: *A few examples of a words written using the spaced-discrete mode*

2 Handwriting recognition system

The first stage of our handwriting recognition system is a "segmentation" stage which consists in the identification of a list of *candidate characters* in a time-dependent signal that we assume to be a written word. We assume here that the segmentation is almost error free because the writer is asked to write well spaced characters as in figure 1.

In terms of applications, this is a realistic assumption because while this mode of writing is not as fast as the cursive mode it has the big advantage of being reliable. Indeed, the error rate is very low, providing that the writer follows only one very simple "writing rule": letters must not overlap. The system has a "trace" mode where the results of the segmentation are displayed so that the user can see that when the writing rule is not respected it causes segmentation errors (typically two letters are attached), which in turns results in word recognition errors. From the user point of view, this is very important: when following the rule, the recognition rate is very high. Failure occurs only when several letters in the word are very badly shaped. In comparison, cursive handwriting recognition systems are crippled by catastrophic performance degradations for reasons that are most often absolutely impossible to figure for the user because all the causes of error are mixed. Consequently such systems are rejected as being not reliable. In fact the problem is that even a cooperative user cannot find out what to do in order to be absolutely certain to have his (her) handwriting recognized.

The rationale of using the discrete mode is thus that with a minimal effort from the user we can use handwriting recognition for text entry with a very high word recognition rate in real time using a microcomputer or Personal Digital Assistant (PDA).

Consequently, we can restrict ourselves to the study of lexical assistance with only substitutions.

2.1 Character recognition

The character recognition used here has been described elsewhere [4]. It is based on building a feature vector from the pen signals i.e. pen tip position and contact with the writing surface as a function of time using classical normalisation (size, translation, re-sampling) and feeding this feature vector to a Constructive Tree Radial-Basis-Function (CTRBF) classifier that will be briefly described. This classifier produces not only a class but a list of all possible classes ordered by decreasing likelihood estimation which is fundamental for the efficient subsequent processing as we will show.

2.2 CTRBF

The CTRBF algorithm is based on a Radial Basis Function Network, and has the basic properties of RBF. However, a CTRBF network is much faster than a classical RBF network, for evaluation as well as for training, with no degradation of the performances [5].

Radial-Basis-Function Networks are known to be capable of universal approximation [6] [7] and the output of a RBF network can be related to Bayesian properties.

A RBF net has 2 layers. The hidden layer is fully connected to the input units $X = (X_i)$ of size N_{input}. A hidden unit j has an input vector of synaptic weights $Win^j = (Win^j_i)$ and is evaluated using a metric, for example in the following formula the Euclidean metric, and a nonlinear function f(x) such as exp(-x) or 1/(x+a). σ_j is an adjustable parameter. As this process has a radial symmetry of center Win^j, the output of a hidden unit j will increase when the input pattern vector X comes "closer" (according to the metric) to the synaptic weights vector Win^j:

$$OUT_j = f(||Win^j - X||) = f(\sum_{i=0}^{N_{input}-1} (X_i - Win^j_i)^2/\sigma_j)$$

The next layer gathers the activities of the hidden units with the purpose of taking a decision on the class of the input prototype. For classification tasks with C classes, this output layer will have C output units. The classification result is obtained from this layer on a "winner-take-all" (WTA) basis: the class of the input pattern will be given by the most active output unit. Then the second most active unit gives the best alternative etc...

Note that the synaptic vector Win^j *stores a location in the problem space*, in other words, it stores an input pattern. The important properties of RBF networks derive from this fact. For example RBF networks provide intrinsically a very reliable rejection of "completely unknown" patterns, because these patterns being "far away" produce weak activities.

2.3 Lexical processor

The lexical resource we use is based on a modified Levenshtein [1] distance computation: for a given candidate word we search the closest word in the dictionary using a Levenshtein distance. It is a classical dynamic programming scheme: the distance between two words can be computed by finding the "cheapest" sequence of edition operations (substitution, deletion, insertion) that can be used to transform one word into the other. "Cheapest" means here the smallest cost where the

cost of a sequence of edition operations can be computed as the sum of the costs of each edition operation divided by the number of operations [8].

Here we restrict ourselves to substitution (we assume a perfect segmentation, or rather we extracted the segmentation errors in order to study only the substitution cost estimation). In the most simple case the costs of the edition operations are empirical constants. We will show that while this provides extremely cheap implementation (especially for hardware implementation) the gain in word recognition rate due to the lexical processing is not very big. In fact, important improvements can be obtained by using costs that are adapted to each word comparison.

In [9] the experimental results are reported on garbled words produced using a random generator, with a 1000 word dictionary. But classification errors are not random. It is important to stress that the key improvements are obtained when the substitution errors are correctly modeled, i.e. we want to have the best possible alternative list for each character recognized.

The rationale is that for example the substitution cost of "B" with "R" should be lower than the substitution cost of "B" with "I", because the patterns "B" and "R" are more difficult to tell apart.

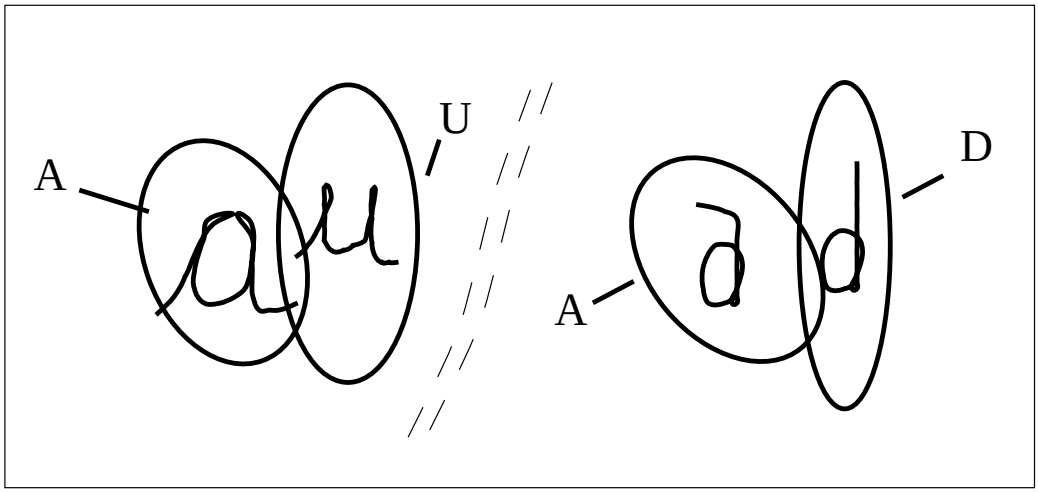

Figure 2: *We assume that we can represent each pattern used for character classification as a two-dimensional vector (or find a projection of a high dimension vector on 2 directions). In the case of handwriting recognition, the data is multi-modal (a character may have different shapes) and highly ambiguous (the same shape may be interpreted as several characters). Thus it is important not to rely on a class-based statistical error accounting for the estimation of substitution costs but on a pattern-based method. Here, a "A" may be confused with a "D" or with a "U" but not both at the same time.*

2.3.1 Estimating substitution costs using the confusion matrix

One approach consists in building the substitution costs table from the confusion matrix of the character recognizer: knowing that the classifier is not perfect we can record its errors in order to predict alternatives. For example, when the input pattern is a "R", if the recognizer makes an error, the probability that the erroneous answer is "B" is bigger than "I". This is done by building the classifier confusion matrix. After training the classifier on the first part P1 of the data, it is tested on P2 in order to build the confusion matrix. The confusion matrix is of size NxN if N is the number of classes. In this work N = 26 because we assume that upper-case lower-case ambiguities have to be processed at a different level -application or syntactico-semantic levels- this matrix is

```
A, 130: D 22,T 21,O 15,U 8,G 8,F 7,E 6,K 6,R 5,Q 4,H 4,C 4,M 4,I 3,X 3,N 3,P 3,W 1,V 1,Z 1,J 1
B, 107: H 50,D 19,L 8,F 8,K 5,S 4,A 3,P 3,R 2,N 2,T 1,E 1,G 1
C, 41:  O 14,E 11,L 6,S 3,Z 3,P 2,G 1,A 1
D, 76:  B 15,P 13,A 6,H 6,R 6,U 4,L 3,M 3,S 3,E 2,G 2,N 2,O 2,V 2,F 1,J 1,Q 1,X 1,Y 1,C 1,T 1
E, 190: L 86,F 27,A 13,T 13,C 12,K 9,I 6,O 5,R 4,H 4,G 4,Z 3,M 1,P 1,X 1,U 1
F, 66:  T 17,E 16,B 10,J 5,A 4,I 3,P 2,G 2,H 2,D 1,K 1,R 1,Y 1,L 1
G, 89:  Q 40,C 11,Y 10,O 9,A 5,T 3,Z 3,B 2,F 2,K 1,H 1,J 1,E 1
H, 52:  N 18,B 9,K 7,L 5,A 4,U 4,F 1,T 1,M 1,Y 1,C 1
I, 85:  E 20,J 17,T 11,P 6,D 5,F 4,Y 4,L 3,M 3,R 2,G 2,X 2,A 2,U 1,Q 1,Z 1,K 1
J, 29:  I 13,T 6,L 4,P 2,S 2,G 1,D 1
K, 56:  H 28,F 5,T 4,B 3,A 2,Q 2,U 2,E 2,R 2,L 1,D 1,G 1,W 1,O 1,J 1
L, 187: E 122,I 17,C 14,B 14,S 3,Z 3,H 3,T 2,R 2,D 2,F 2,O 1,Y 1,U 1
M, 47:  N 26,U 7,H 7,R 3,B 1,K 1,W 1,A 1
N, 230: M 53,U 50,R 29,H 27,W 27,X 16,P 7,A 5,D 4,V 4,B 4,Q 1,I 1,Z 1,G 1
O, 70:  U 13,V 10,G 9,A 8,R 5,E 5,W 5,D 3,C 3,T 2,P 2,Z 1,B 1,N 1,L 1,Y 1
P, 46:  R 19,N 5,D 4,T 4,G 2,Y 2,M 2,O 2,U 2,A 1,H 1,V 1,W 1
Q, 32:  G 24,S 3,K 3,A 2
R, 199: V 58,N 17,Z 15,U 12,K 11,T 10,B 10,M 10,E 9,D 8,L 8,S 8,Y 7,H 5,A 4,W 2,X 2,J 1,Q 1,F 1
S, 33:  J 8,G 7,R 6,B 4,Q 2,L 2,E 1,Z 1,D 1,T 1
T, 108: F 19,J 17,Z 14,I 7,R 6,A 6,E 6,Y 6,G 5,B 4,P 4,O 3,C 3,Q 2,K 2,L 1,S 1,W 1,D 1
U, 195: V 77,N 51,O 19,A 14,R 5,W 5,K 5,M 5,H 4,D 3,Y 2,G 1,Q 1,I 1,C 1,E 1
V, 122: U 81,R 21,N 8,W 8,O 2,P 2
W, 29:  N 10,V 6,M 6,K 2,O 2,R 1,T 1,A 1
X, 14:  N 2,D 2,H 1,E 1,L 1,P 1,A 1,K 1,R 1,Y 1,F 1,U 1
Y, 125: G 80,Z 12,J 7,F 7,Q 6,E 3,C 2,P 2,R 1,D 1,L 1,T 1,S 1,U 1
Z, 26:  L 10,R 8,G 2,C 1,T 1,I 1,E 1,Y 1,F 1
```

Figure 3: *CTRBF recognizer confusion matrix: the first letter designates the class, the following figure is the total error number for that class, then for each error we have the error count. The total error record of our classifier for 51079 mixed upper-case and lower-case test characters is thus of 2384 errors (4.7%) distributed as shown.*

plotted is table 3. Note that for example the high confusion rate between E and L or A and D are due to the lower-case confusions. This matrix can be seen as giving for each possible class the probability that the result is true. Typically the diagonal of this matrix should be made of terms very close to 1, all other terms should be very small (a perfect classifier confusion matrix is the identity matrix). However in handwriting recognition we have a very difficult pattern recognition problem especially in terms of confusions: the same shape can be used to represent different characters (see for example figure 2) so that for almost all classes there are several ambiguities. Note that these ambiguities are not only due to classifier imperfections, most are intrinsic.

Using the confusion matrix has the following drawbacks:

- Some patterns that are not ambiguous at all will be processed as ambiguous. In our example a perfectly shaped "B" will be interpreted in the lexical process as having a non-zero probability of being a "R" which introduces an artificial ambiguity.
- Some patterns that are very ambiguous (a character that resembles both a "B" and a "R") will be processed as only slightly ambiguous, because the error rate is an average rate.
- The part P2 of the data base (typically P1 and P2 have the same size) is devoted to the constitution of the confusion matrix. It is almost paradoxical that this data, instead of

being used for a better training of the classifier, is used for recording the *errors* since we have to suspect that the data base (in our case approximately 102000 characters) is barely big enough. This is especially true for a single-pass constructive algorithm like CTRBF where, contrary to Error Back Propagation and other iterative methods, we do not have to reserve test data (P2) for the determination of when to stop the training.

We demonstrate here that more accurate and useful information can be extracted from the character recognition process providing that the classifier uses local information like RBF-based classifiers do.

2.3.2 Estimating substitution costs using the recognition likelihood

The CTRBF classifier provides an ordered list of classes according to the "activities" of the output neurons of the CTRBF network. These activities are non-normalized class probabilities estimate, i.e. only the relative values are useful because the RBF hidden unit density is not a quantitative estimation of the pattern density (due to the fact that the error function during training is not computed using a class probability density estimation but only a symbolic error on classification performance). Providing a class probability estimate would be better but would require a Bayesian classifier that would be more expensive to implement. Instead we compute a per-character substitution cost based on the RBF "activity" normalized using the winner (top choice) activity and the simple formula:

$$cost(i) = a(w)/a(i) - 1$$

where i is the class number (0-25 for characters), $a(w)$ is the RBF winner activity and $a(i)$ is the RBF activity for class i. Note that, as the RBF works with localized perceptive fields, $a(i)$ is a local estimation: it is a per-pattern view instead of a per-class view as with the confusion matrix.

3 Experimental results

The results reported here are based on a data set of 18331 "discrete" words written by 37 writers and containing approximately 92000 characters (the average word length is 5). We assume that the segmentation system is "perfect", i.e. segmentation errors are not accounted.

The CTRBF classifier used has a maximum recognition rate of 95.3 % (zero rejection) and treats upper-case and lower-case forms as a single class (thus we have 26 classes).

There was 47891 English words in the reference dictionary based on the "ispell" public domain dictionary. Of course all words of the data are in the dictionary.

Typical average recognition speed was approximately 13 words per second on a Sun Sparc 10 workstation, including segmentation, character recognition and lexical processing.

3.1 Substitution cost estimation

The strategy used for computing the substitution costs is based on the idea that the RBF classifier provides a winner class (best result) and an ordered list of alternatives. We can either process this full list or only use the top of this list. Using only the top of the list may be an important alternative for better speed and implementation cost (software as well as hardware). We will also show that experimentally only the top of the list is useful anyway, which could be suspected considering the number of classes (26 here). When using only the top of the list, we define the "marginal cost" as being the substitution cost for classes (characters) that are "out-lier" i.e not in the top of the list. As will be seen in the experimental results what is important is to specify if

method / candidates	exact match	arbitrary increasing costs	computed costs Cost = (winner)/(RBF act.) -1	statistical costs
best	80.6 / 94.1 (0)	80.6 / 94.1 (0)	81.0 / 94.46 (0)	
+ second best	85.2 / 89.4 (0,0)	92.2 / 97.2 (0,1)	92.3 / 97.3	
+ third best	75.2 / 77.3 (0,0,0)	94.2 / 97.4 (0,1,3)	94.4 / 97.4	
all			- / 97.3	- / 94.7

Figure 4: *For each variant two different marginal costs are tested, results are separated by a slash: on the left, infinite marginal cost, of the right non-infinite marginal coast (typically: 10). Between parenthesis figures are the actual top alternatives substitution costs, when relevant*

this cost is infinite or not. An infinite substitution cost means that the whole word will be refused because the distance to this word is infinite. Infinite costs are useful in order to speed up the search because as soon as an infinite cost arises the current reference word is abandoned and the next reference word is tried. This is conceptually equivalent to using a tree structure [10] where one starts at the beginning of the word and at each character a choice is made, if the character does not match the word is refused without having to go deeper in the tree: this is equivalent to our infinite cost. It has the advantage of being faster but if an error is made in the beginning it will not be recovered.

A non-infinite marginal cost means that no word will be eliminated because it has one letter that has not been matched well. This allows to recognize words with one or even several characters that are completely wrong, very badly shaped or not recognized at all.

3.2 Exact match

For comparison purpose we provide the results using an exact match strategy: By exact match we mean that the distance used is a Hamming distance: when computing the distance between two words if two letters are the same we add nothing, otherwise we add 1. This is implemented by making a test or using a *OR*, which is extremely fast and economical for a hardware implementation. However, this strategy leads to 80.6 % word recognition rate which is barely better than the average statistical rate of 78.6 % (computed from the character recognition rate).

If one tries to improve this by providing the second (and eventually) third alternative given by the classifier one can see in the table 4 that the performance increases a little (85.2 %)and then actually decreases (75.2 %) if the third possibility is also taken into account. It means that the top 3 choices with equal likelihood are not discriminating enough in such a large dictionary.

The most interesting result is that the best result (94.1 %) is obtained with a non-infinite

marginal cost but using only the classifier top choice (all other characters are "equally possible"). This clearly indicates that providing a list of alternatives *without ranking them* is a bad strategy: it adds confusion. This is especially clear when the top 3 choices are considered, indeed then the performance goes below the statistical rate !

3.3 Arbitrary increasing costs

When considering the top N choices given by the CTRBF classifier for each character, instead of giving a zero substitution cost for all N top choices as before we give arbitrary increasing costs (typically, 0 for the best, 1 for the second best, 2 for the third best, etc...). While being slightly more complicated than a Hamming distance, this is still easy and cheap to implement in VLSI [11], this is a important point as the performance is equivalent to computed costs (see below) for which the VLSI implementation is much more complex.

Two important remarks can be made: first the performance improves when considering an increasing number of alternatives, second, using a non-infinite marginal cost is also a better alternative.

3.4 Computed costs

The substitution costs are computed using the formula given above, note that alternatives that have a zero activity produce infinite substitution costs and that the winner has a zero substitution cost. In the case of a very ambiguous pattern the second best answer will have an activity very close to the winner and will thus produce a very small substitution cost, which is an exact translation of the idea that ambiguous patterns should be seen by the lexical processor as a list of almost equivalent alternatives.

One can see in table 4 that computing *all* costs is equivalent to computing the substitution costs for the top 3 choices only using a non-infinite marginal cost for all the others. This may be an indication that the CTRBF classifier gives a good list of alternatives only down to the top 3 alternatives, beyond that the class probability estimation is probably drowned in noise. However, it may also due to the dictionary size and the subsequent high word density.

3.5 Statistically estimated costs

Results based on this statistical error estimation for computing the substitution cost are presented in table 4 in the right hand column. Considering that this close to 100 % each percent of improvement is very hard to get, this is very inferior to a local estimation of the substitution cost (94.7 % versus 97.4 %).

The reason for that is best explained considering figure 2. In the case depicted in figure 2 a statistical substitution cost estimation will give "U" *and* "D" as possible alternatives for "A" because both ambiguities occur (when processing a large set of characters). On the other hand both ambiguities *cannot* occur for the same pattern of a "A": a pattern may be confused with "U" *or* "D". In other words, the class ("A" in this example) is not a rich enough information for the best estimation of possible alternatives, the *localisation* of the pattern itself has to be used and for this purpose a RBF-based classifier is a major asset.

4 Conclusion

In handwriting recognition of words that are assumed to belong to a given lexicon, the lexical processor performance is enhanced by providing *relevant* alternatives for each character in the

candidate word. By relevant, we want to stress the difference between this approach and the classical statistical processing of the confusion matrix where the alternatives for each character are estimated on a per-class basis from the class error probabilities of the classifier. Here we provide to the lexical processor the correct possible alternatives for each character in the recognized word, thanks to the use of a RBF classifier which has localised receptive fields and thus provides per-pattern alternatives. Our experimental results demonstrate a major improvement in system performance, from 94.7 % to 97.4 %. We conjecture that this is due to the highly multi-modal and ambiguous nature of handwriting.

References

[1] V.I. Levenshtein, Binary codes capable of correcting deletions, insertions and reversals", Sov. Phys, Dokl. 10, pp. 707-710, Fev. 1966.

[2] L. R. Bahl, F. Jelinek, Decoding for channels with insertions, deletions and substitutions with applications to speech recognition, IEEE Trans. Information Theory, Vol 21, pp 404-410, 1975.

[3] R. Bozinovic, S. Srihari, A String Correction Algorithm for Cursive Script Recognition, IEEE Trans. Pattern Analysis and Machine Intelligence, vol 4, No 6, Nov. 1982.

[4] P. Gentric, On-line handwriting recognition for small computer. Sixth International Conference on Handwriting and Drawing, Paris July 5-6-7, 1993.

[5] P. Gentric, Constructive Methods For A New Classifier Based On A Radial-basis-function Neural Network Accelerated By A Tree,Proc. International Workshop on Artificial Neural Networks, Barcelona, 1993.

[6] J.Park, I.W. Sandberg, Universal Approximation Using Radial-Basis-Function Networks, *Neural Computation* 3,246-257, 1991.

[7] J. Moody and C. Darken, Learning with localized receptive fields, *Proceedings of the 1988 Connectionist Models Summer School*, ed. D. Touretzky, Morgan Kaufmann Publishing, San Mateo, CA, pp. 133-143, 1988.

[8] A. Marzal, E. Vidal, Computation of Normalized Edit Distance and Applications, IEEE Trans. Pattern Analysis and Machine Intelligence, Vol 15, No 9, Sept. 1993.

[9] T. Okuda, E. Tanaka, T. Kasai, A Method for the Correction of Garbled Words Based on the Levenshtein Metric, IEEE Trans. on computers, vol C-25, No 2, Feb. 1976.

[10] D. E. Knuth, The Art of Computer Programming, Vol. 3 - Sorting and Searching. Reading, MA: Addison-Wesley, 1973.

[11] M. Hervieu, J. Y. Brunel , The PHRASES Neuro-Coprocessor Cost Effective Handwriting Recognition for Personal Digital Assistant, Extended Abstracts of the 1994 International Conference on Solid State Devices and Materials, PC-4-8, pp 391-393, Yokohama, 1994.