Local Learning Algorithms

Léon Bottou, Vladimir Vapnik
AT&T Bell Laboratories, Holmdel, NJ 07733, USA

Abstract

Very rarely are training data evenly distributed in the input space. Local learning algorithms attempt to locally adjust the capacity of the training system to the properties of the training set in each area of the input space.

The family of local learning algorithms contains known methods, like the k-Nearest Neighbors method (kNN) or the Radial Basis Function networks (RBF), as well as new algorithms. A single analysis models some aspects of these algorithms. In particular, it suggests that neither kNN or RBF, nor non local classifiers, achieve the best compromise between locality and capacity.

A careful control of these parameters in a simple local learning algorithm has provided a performance breakthrough for an optical character recognition problem. Both the error rate and the rejection performance have been significantly improved.

1 Introduction.

Here is a simple local algorithm: *For each testing pattern, (1) select the few training examples located in the vicinity of the testing pattern, (2) train a neural network with only these few examples, and (3) apply the resulting network to the testing pattern.*

Such an algorithm looks both slow and stupid. Indeed, only a small part of the available training examples is used to train our network. Empirical evidence however defeats this analysis. With proper settings, this simple algorithm improves significantly the performance of our best optical character recognition networks.

A few years ago, V. Vapnik devised a theoretical analysis for such local algorithms, briefly discussed in (Vapnik, 1992). This analysis introduces a new component, named *locality*, in the well known tradeoff between the capacity of the learning system and the number of available examples.

This paper attempts to explain, and demonstrates, that such an algorithm might be very efficient for certain tasks, and that its underlying ideas might be used with profit. The voluminous equations of the theoretical analysis, however, will not be discussed in this paper. Their complexity, mostly related to the imperfection of the current generalization theories, would introduce unnecessary noise in this discussion.

In Section 2, we show that handling rejections in some pattern recognition problems requires different properties for a learning device in different areas of the pattern space. In Section 3, we present the idea of a local learning algorithm, discuss related approaches, and discuss the impact of the locality parameter on the generalization performance. In Section 4, we demonstrate the effectiveness of a local learning algorithm on a real size optical character recognition task.

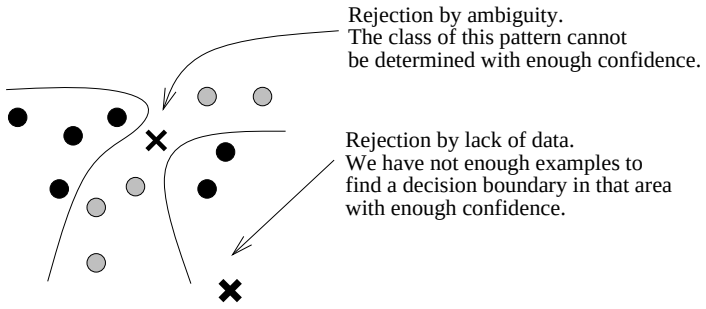

Figure 1: *This is a piece of an imaginary pattern space. Gray and black circles are examples of two classes. Thin lines are the actual bayesian decision boundary between these two classes. Both crosses represent rejected patterns.*

2 Rejection.

An ideal isolated character recognition system always assigns the right symbolic class to a character image. But a real recognizer might commit an error or perform a rejection.

Errors are very expensive to correct. A zipcode recognition system, for instance, might erroneously send a parcel to the other end of the world. Therefore, the system should reject a pattern whenever the classification cannot be achieved with enough confidence. Having this pattern processed by a human being is usually less expensive than fixing an error. Selecting a proper confidence threshold reduces both the cost of handling the rejected patterns and the cost of correcting the remaining errors.

The quality of such a pattern recognition system is measured by its rejection curve (cf. Figure 4.) This curve displays the possible compromises between the number of rejected patterns and the numbers of remaining errors.

Two very different situations reduce the classification confidence and might cause a rejection (cf. Figure 1.)

- Patterns might well be ambiguous. For instance, certain people write their "1" like other people write their "7".

  This cause of rejection is inherent to the problem. Ambiguities arise because important information, like the contextual knowledge of the writing style, is not provided as input to the system.

  Knowing exactly the probability distribution of the patterns and classes would not eliminate such rejections. A pattern would still be rejected if its most probable class does not win by a sufficient margin.

- Patterns might be unrelated to the training data used for defining the classifier. For instance, many atypical writing styles are not represented in the training database.

  Low probability areas of the pattern space are poorly represented in the training set. The decision boundary of our classifier in such areas are mere side effects of the training algorithm. These boundaries are just irrelevant.

  This second cause of rejection is a direct consequence of the finite nature of the training set. Knowing exactly the probability distribution of the pattern and classes would reveal the exact Bayesian decision boundaries everywhere.

This latter cause of rejection has rarely been studied in the literature[1]. Its mere definition involves non asymptotical statistics closely related to the generalization phenomenon.

- A high capacity learning system is able to model accurately and with high confidence the parts of the decision boundary that are well described by the training examples. In these areas, both rejections and misclassifications are rare.

  The same system however produce unreliable high confidence decision boundaries in the poorly sampled areas of the pattern space: Rejection are rare, but misclassifications are frequent.

- Alternatively, a low capacity learning system builds low confidence boundaries in the poorly sampled areas of the pattern space. This system rejects more atypical patterns, but reduces the number of misclassifications.

  Unfortunately, such a device performs poorly in the well sampled areas. Unable to take profit of the abundant data, it builds poor decision boundaries, and rejects almost everything, because everything looks too ambiguous.

In fact, different properties of the learning algorithm are required in different areas of the input space. In other words, the "local capacity" of a learning device should match the density of training examples.

3 Local learning algorithms.

It is now generally admitted that the generalization performance is affected by a global trade off between the number of training examples and the capacity of the learning system.

Various parameters monotonically control the capacity of a learning system (Guyon et al., 1992), including architectural parameters (e.g. the number of hidden units), preprocessing parameters (e.g. the amount of smoothing), or regularization parameters (e.g. the weight decay).

The best generalization is achieved for some optimal values of these capacity control parameters, which depend on the size of the training set. This fact holds for rejection or for raw performance, in the case of pattern recognition, regression, or density estimation tasks.

Whenever the distribution of patterns in the input space is uneven, a proper local adjustment of the capacity can significantly improve the overall performance. Such a local adjustment requires the introduction of capacity control parameters whose impact is limited to individual regions of the pattern space. Here are two ways to introduce such parameters:

- Our experiment (cf. Section 4) illustrates the first solution. For each testing pattern, we train a learning system with the training examples located in a small neighborhood around the testing pattern. Then we apply this trained system to the testing pattern itself.

  The parameters of the locally trained system *de facto* affect only the capacity of the global system in the small neighborhood defined around each testing pattern.

  We shall show below that the k-Nearest-Neighbor (kNN) algorithm is just a particular case of this approach. Like kNN, such systems are very slow. The recognition speed is penalized by

[1] A Bayesian approach has been suggested in (Denker and Le Cun, 1991) for estimating error-bars on the outputs of a learning system. This useful information affects the interpretation of the outputs, and might improve the rejection performance, as suggested by our reviewer. This method could as well improve the rejection of local algorithms.

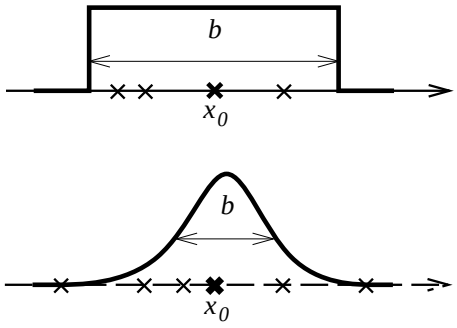

Figure 2: *A square kernel selects only those examples located in a specific neighborhood. A smooth kernel gives a different weight to all examples, according to their position with respect to the kernel. The locality parameter, b, measures the "size" of the neighborhood.*

the selection of the closest training patterns and by the execution of the training algorithm of the local learning system.

- In the second solution, the structure of the learning device ensures that each parameter affects the capacity of the system in a small neighborhood only.

  For example, we might use a separate weight decay per Radial Basis Function (RBF) unit in a RBF network. Each weight decay parameter affects the capacity of the network only locally. Similarly, architectural choices in a modular network (Jacobs et al., 1991) have a local impact on the capacity of the global system.

  Since such a system is trained at once, the recognition time is not affected by the local nature of the learning procedure.

In the next subsections, we shall describe a general statement for local learning algorithms, discuss related approaches, and explain how the locality parameter affects the generalization performance.

3.1 Weighted cost function.

We present here a general statement for local learning algorithms. Let us define $J(y, f_w(x))$ as the loss incurred when the network gives an answer $f_w(x)$ for the input vector x, when the actual answer is y.

The capacity control parameters γ directly or indirectly define a subset W_γ of the weight space (Guyon et al., 1992). A non local algorithm searches this subset for the weight vector $w^*(\gamma)$ which minimizes the empirical average of the loss over a training set $(x_1, y_1), \cdots, (x_l, y_l)$.

$$w^*(\gamma) \;=\; \underset{w \in W_\gamma}{\text{Arg Min}} \; \frac{1}{l} \sum_{i=1}^{l} \; J(y_i, f_w(x_i)) \tag{1}$$

For each neighborhood defined around a point x_0, a local algorithm searches for a weight vector $w^*(x_0, b, \gamma)$ which minimizes a weighted empirical average of the loss over the training set.

$$w^*(x_0, b, \gamma) \;=\; \underset{w \in W_\gamma}{\text{Arg Min}} \; \frac{1}{l} \sum_{i=1}^{l} \; K(x_i - x_0, b) \; J(y_i, f_w(x_i)) \tag{2}$$

The weighting coefficients are defined by a kernel $K(x - x_0, b)$ of width b centered on point x_0. Various kernels might be used, including square kernels and smooth kernels (cf. Figure 2).

Since a separate minimization is performed in each neighborhood, the capacity control parameters γ and the kernel width b can be adjusted separately for each neighborhood.

3.2 Related approaches.

This formulation allows for many variations concerning the class of function $f_w(x)$, the number of neighborhoods, the shape of the kernels $K(x - x_0, b)$, the scaling laws for the kernel width b or the parameters γ.

Selecting the class of constant functions with respect to the input vectors x, and using a quadratic loss $J(y, \hat{y}) = (y - \hat{y})^2$ leads to several popular algorithms, like the kNN method or the RBF networks. In each specific neighborhood, such algorithms try to find a constant approximation $\hat{y}^*$ of the desired output:

$$\hat{y}^* \;=\; \underset{\hat{y}}{\text{Arg Min}} \;\; \frac{1}{l} \sum_{i=1}^{l} \; K(x - x_0, b) \; (y_i - \hat{y})^2 \tag{3}$$

For instance, consider a pattern recognition problem. If the pattern $\hat{x}_i$ belongs to the n^{th} class, the n^{th} coefficient of the corresponding desired output $\hat{y}_i$ is equal to 1; the others coefficients are equal to 0.

- For each testing pattern x_0, we consider a square kernel whose width is adjusted to contain exactly k examples. The optimum $\hat{y}$ of (3) is the mean of the desired outputs of the k closest patterns. Its highest coefficient, then, corresponds to the most represented class among the k closest patterns to x_0. This is the *k-Nearest Neighbors* algorithm (kNN).

  If we use a smooth kernel instead of a square kernel, minimizing (3) for each testing pattern x_0 computes estimates of the posterior probability of the classes. This is the *Parzen windows* algorithm.

- We consider now R fixed neighborhoods, defined by the centers x_r^+ and the standard deviation b_r^+ of their gaussian kernels. Minimizing (3) in each neighborhood computes the weighted average $\hat{y}_r^+$ of the desired values of the training examples.

  To evaluate the output $\hat{y}_{\text{Global}}(x)$ of the complete system for an input pattern x, we merge these weighted averages according to the values of the R kernels on x.

$$\hat{y}_{\text{Global}}(x) \;=\; \sum_{r=1}^{R} \; \hat{y}_r^+ \; K(x - x_r^+, b_r^+) \tag{4}$$

  This is a *Radial Basis Functions* (RBF) network (Broomhead and Lowe, 1988),(Moody and Darken, 1989).

3.3 Locality and capacity.

Theoretical methods developed for non local algorithms apply to each local optimization. In particular, the best value for the capacity control parameters (Guyon et al., 1992) depend on the number of training examples.

In the context of a local algorithm, however, the effective number of training examples is modulated by the width b of the kernels. For instance, a square kernel selects from the training set a subset, whose cardinality depends on the local density of the training set and on the kernel width b.

The classical tradeoff between capacity and number of examples must be reinterpreted as a tradeoff between capacity and locality. If we increase the locality by reducing b, we implicitly reduce the effective number of training examples available for training the local system.

kNN and the RBF networks use small kernels and a class of constant functions. There is no reason, however, to believe that the best results are obtained with such a low capacity device. Conversely, big multi-layer networks are non local ($b = \infty$), but have a high capacity.

Modular networks (Jacobs et al., 1991) sit somewhat between these two extremes. The kernel functions are embodied by a "gating network", which selects or combines the outputs of the modules, according to the input data and sometimes the outputs of the modules themselves.

Another phenomenon makes the situation slightly more complex: The capacity control parameters can be adjusted separately in each neighborhood. This local adjustment is more accurate when the kernel width is small. On the other hand, there is little to adjust in a very low capacity device.

4 Experiments.

This section discusses experiments of a simple local learning algorithm on a real size pattern recognition problem. Comparisons have been carried out (a) with a back-propagation network, (b) with the kNN and Parzen windows algorithms.

- A back-propagation network is a non local algorithm, with a comparatively high capacity. Comparison (a) shows that introducing locality and reducing the capacity improves the resulting performance of such a network.
- A kNN classifier is an extremely local algorithm, with a very low capacity. Comparison (b) shows that reducing the locality and increasing the capacity again improves the resulting performance.

4.1 A simple local learning algorithm.

We have implemented a simple local algorithm:

For each testing pattern x_0, a linear classifier is trained on the k closest training examples, $(x_1, y_1), \ldots, (x_k, y_k)$, for the Euclidian distance. This trained network then is applied to the testing pattern x_0.

The effective number of examples, k, is much smaller than the number of weights in the linear classifier. Therefore, a strong weight decay γ is required to reduce the capacity of the linear classifier. A weight decay, however, pulls the weights toward some arbitrary origin. For isotropy reasons, the origin of the input space is translated on the testing pattern x_0, by subtracting x_0 from all the selected training patterns. This also has the advantage of reducing the eigenvalue spread of the hessian matrix.

The training procedure computes the explicit minimum of a quadratic cost incorporating a weight decay term. The positive weight decay γ ensures that the required matrix inversion is possible.

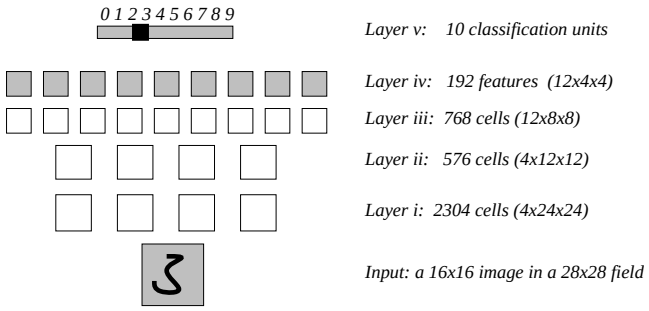

Figure 3: *"LeNet": Layers i,ii,iii, and iv compute 192 features. Layer v performs the classification. In this paper, we replace layer v by a local algorithm.*

$$w^*(x_0,k,\gamma) \;=\; \operatorname*{Arg\,Min}_{w} \left\{ \frac{1}{k}\sum_{i=1}^{k}(y_i - f_w(x_i - x_0))^2 \;+\; \gamma|w|^2 \right\} \tag{5}$$

Since the new coordinate system is centered on the testing pattern, the output of the network on this testing pattern is equal to the bias vector. The highest output determines which class is selected for pattern x_0. If however the difference between the highest output and the second highest output is less than a certain threshold, the pattern is rejected.

A simple heuristic rule controls the capacity versus locality tradeoff within each neighborhood. It adjusts the locality and leaves the capacity constant. In other words, the same value of the weight decay parameter γ is used in all neighborhoods. The locality is usually controlled by a kernel size b which should increase when the density of training example decreases.

In fact, selecting the k closest training examples is equivalent to having a square kernel whose size is somewhat adjusted according to the local density of training examples. We just use the same value for k in all neighborhoods.

Although this system is extremely inefficient, it implements a wide range of compromises between locality and capacity, according to the values of only two parameters, a locality parameter k and a regularization parameter γ. We have found that this quality was attractive for a first experiment.

4.2 Results and comparisons.

We trained several system on the same training set composed of normalized 16x16 images of 7291 handwritten digits and 2549 printed digits. Performance has been measured on a test set of 2007 handwritten digits. The same database was used in (Le Cun et al., 1990).

Table 1 gives the raw error and the rejection at 1% error for various systems. The "raw error" is the percentage of misclassifications when no rejection is performed. The "rejection for 1% error" is the percentage of rejected pattern when the rejection threshold is adjusted to allow less than 1% misclassification on the remaining patterns.

The 2.5% human performance on the segmented and preprocessed digits provides a reference point (Säckinger & Bromley, personal communication).

The nickname "LeNet" designates the network described in (Le Cun et al., 1990). This five layer network performs both the feature extraction and the classification of 16x16 images of single handwritten digits. Four successive convolutional layers extract 192 translation invariant features;

		Raw Error	Rejection for 1% Error
Human	(on segmented digits)	$\approx 2.5\%$	n.a.
LeNet	(on segmented digits)	5.1%	9.6%
kNN	(on LeNet features)	5.1%	n.a.
Parzen	(on LeNet features)	4.7%	10.8%
Local	(on LeNet features)	**3.3%**	**6.2%**

Table 1: *Results on a optical character recognition task.*

a last layer of 10 fully connected units performs the classification (cf. Figure 3). This network has achieved 5.1% error and 9.6% rejection for 1% error[2].

The 192 features computed by LeNet are used as inputs to three other pattern recognition systems. These systems can be viewed as replacements for the last layer of LeNet. Therefore, results can be directly compared.

The best kNN performance, 5.1% raw error, was obtained by using the three closest neighbors only. These few neighbors allow no meaningful rejection strategy.

The Parzen system is similar to kNN. We just replace the square kernel by a gaussian kernel, whose standard deviation is half the distance of the 4^{th} closest pattern. Several variations have been tested; we report the best results only: 4.7% raw error and 10.8% rejection for 1% error.

Finally, we have tested the simple local learning algorithm described above, using the 200 closest patterns and a weight decay of 0.01. This weight decay is enough to train our 1920 weights using 200 patterns. At this time, both the 3.3% raw error and the 6.2% rejection rate for 1% error were the best performances reached on this data set.

A derivation reported in the appendix shows that this performance improvement is statistically significant, Figure 4 compares the rejection curve of the local system with the rejection curve of "LeNet." The local system performs better for all values of the threshold. At 17% rejection, the single remaining error is a mislabeled pattern.

With a proper adjustment of its locality and capacity parameters, this simple algorithm outperforms both (a) a non local algorithm (i.e. the last layer of LeNet), and (b) two extremely local algorithms (i.e. KNN or Parzen windows). Figure 5 shows how the raw error of the local system changes around the best values of k and of the weight decay γ.

Finally, no significant performance changes have been obtained by using smooth kernels or fancy heuristic for controlling the kernel width and the weight decay.

4.3 Recognition speed.

This simple system, however, spends 50 seconds for recognizing a single digit. Training a network for each testing pattern is certainly not a practical approach to the optical character recognition problem.

In Section 3, however, we have presented two solutions for building local learning algorithms. We have deliberately chosen to implement the simpler one, which leads to very slow recognizers.

We could as well design systems based on our second solution, i.e. using a network structure which allows a local control of the capacity of the system. Such a system would be slightly more

[2]This is slightly worse than the 4.6% raw error and 9% rejection for 1% error reported in (Le Cun et al., 1990). This is due (*i*) to a slightly different definition of the rejection performance (1% error on the remaining patterns vs. 1% error total), and (*ii*) to a more robust preprocessing code.

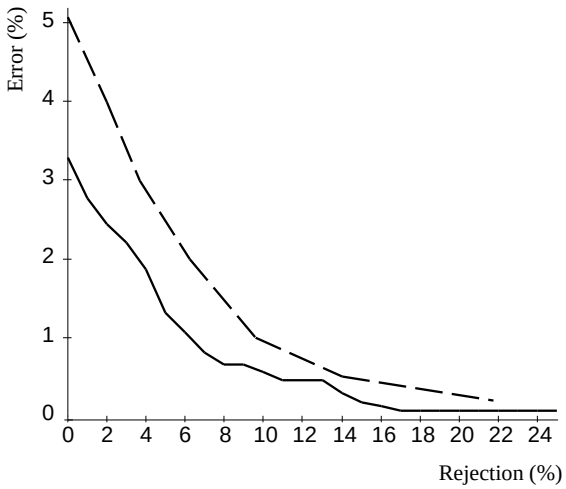

Figure 4: *Punting curve. This curve shows the error rates at various rejection rates for plain LeNet (dashed curve) and our simple local algorithm operating on the features computed by LeNet (plain curve).*

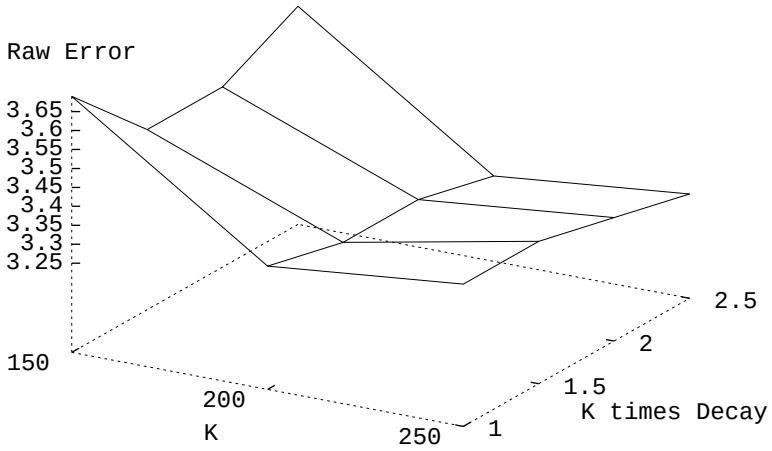

Figure 5: *Evolutions of the raw error for the local system around the best values of k and of the weight decay γ. (In fact, the decay axis displays the product γk.)*

complex to handle, but would have a much smaller recognition time.

5 Conclusion.

No particular architectural change is responsible for this performance breakthrough. In fact, this system is linear, and replaces a linear decision layer in LeNet. The change concerns the training procedure. The performance improvement simply results from a better control of the basic tradeoffs involved in the learning process.

Although much remains to be understood about learning, we have some practical and theoretical knowledge of these basic tradeoffs. Understanding how these tradeoffs affect a specific learning problem often allows us to take profit from their properties for practical applications.

Local learning algorithms are just a successful example of this strategy.

Appendix: Confidence intervals.

This section presents the derivations establishing the significance of the results presented in this paper. We first derive a non asymptotical formula for computing confidence when comparing two classifiers *on a same test set* of N independent examples.

Each classifier defines certain decision boundaries in the pattern space. It is enough to compare our classifiers on only those cases where one classifier is right and the other one is wrong. Let us call p_1 and p_2 the conditional probabilities of error of each classifier, given that one classifier only gives a wrong answer. Similarly, let us define n_1 and n_2 as the numbers of errors that each classifier makes which the other classifier classifies correctly, and n_{12} as the number of common errors.

According to the large number inequality (2.7) in (Hoeffding, 1963),

$$P\left(p_1 - p_2 \geq \frac{n_1}{n_1+n_2} - \frac{n_2}{n_1+n_2} - \epsilon\right) \;\geq\; 1 - e^{-(n_1+n_2)\epsilon^2} \tag{6}$$

Furthermore, if we name ν_1 and ν_2 the measured error rates on our test set, we have

$$n_1 - n_2 = (n_1 + n_{12}) - (n_2 + n_{12}) = N(\nu_1 - \nu_2) \tag{7}$$

By solving for ϵ when the right hand side of inequality (6) is $1-\eta$, we can compute the minimum difference $\nu_1 - \nu_2$ which ensures that $p_1 - p_2$ is larger than 0 with probability $1-\eta$. Since comparing p_1 and p_2 is enough to decide which classifier is better, the following result is valid:

$$\begin{aligned} &\textit{If} \quad \nu_1 - \nu_2 > \frac{1}{N}\sqrt{-(n_1+n_2)\;\ln\eta} \\ &\textit{then classifier 2 is better than classifier 1 with probability } 1-\eta. \end{aligned} \tag{8}$$

In our case, all systems achieve less than 5.1% error on a test set of size $N = 2007$. The quantity $n_1 + n_2$ is thus smaller than 10.2% of N, probably by a large margin. If we choose $\eta = 5\%$, we get a minimal significative error difference of 1.2%.

Measuring the actual value of $n_1 + n_2$, would further reduce this margin. The significance of the results presented in this paper, however, is established without such a refinement.

Acknowledgements.

We wish to thank Larry Jackel's group at Bell Labs for their continuous support and useful comments. We are especially grateful to Yann Le Cun for providing networks and databases , to E. Säckinger, J. Bromley for providing the human performance results.

References

Broomhead, D. S. and Lowe, D. (1988). Multivariate functional interpolation and adaptive networks. *Complex Systems*, 2:321–355.

Denker, J. S. and Le Cun, Y. (1991). Transforming neural-net output levels to probability distributions. In Lippmann, R., Moody, J., and Touretzky, D., editors, *Advances in Neural Information Processing Systems 3 (NIPS*90)*, Denver, CO. Morgan Kaufman.

Guyon, I., Vapnik, V. N., Boser, B. E., Bottou, L., and Solla, S. A. (1992). Structural risk minimization for character recognition. In *Advances in Neural Information Processing Systems*, volume 4, Denver. Morgan Kaufman.

Hoeffding, W. (1963). Probability inequalities for sums of bounded random variables. *Journal of American Statist. Ass.*, 58:13–30.

Jacobs, R. A., Jordan, M. I., Nowlan, S. J., and Hinton, G. E. (1991). Adaptive mixture of local experts. *Neural Computation*, 3(1).

Le Cun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1990). Handwritten digit recognition with a back-propagation network. In Touretzky, D., editor, *Advances in Neural Information Processing Systems*, volume 2, Denver. Morgan Kaufman.

Moody, J. and Darken, C. J. (1989). Fast learning in networks of locally-tuned processing units. *Neural Computation*, 1(2).

Vapnik, V. N. (1992). Principles of risk minimization for learning theory. In *Advances in Neural Information Processing Systems*, volume 4, Denver. Morgan Kaufmann. To appear.