[2] O. Hebb, *The Organization of Behavior*. New York: Wiley, 1949.
[3] F. Rosenblatt, "A bibliography of perceptron literature," in *Collected Technical Papers*, vol. 2, *Cognitive Systems Research Program*, Report no. 4, Cornell University, July 1963.
[4] F. Rosenblatt, *Principles of Neurodynamics*. East Lansing, MI: Spartan Books, 1962.
[5] M. Minsky and S. Papert, *Perceptrons, an Introduction to Computational Geometry*. Cambridge, MA: MIT Press, 1st ed. 1969, expanded ed., 1988.
[6] F. Rosenblatt, "Perceptual generalization over transformation groups," in *Self-Organizing Systems*, (Yovits and Cameron, Eds.) Elmsford, NY: Pergamon Press, 1960.
[7] F. Rosenblatt, *A Model for Experiential Storage in Neural Networks* (Computer and Information Sciences, Tou and Wilcox, Eds.). East Lansing, MI: Spartan Books, 1964.
[8] T. H. Barker, "A computer program for simulation of perceptrons and similar neural networks, *Cognitive Systems Research Program*, Report no. 8, Cornell University, July 1966.
[9] H. C. Hay and C. W. Wightman, "The Mark I perceptron, design and performance," in *IRE Nat. Convention Rec.*, part 2, 1960.

Adaptive Nearest Neighbor Pattern Classification

Shlomo Geva and Joaquin Sitte

Abstract—**We describe a variant of nearest neighbor pattern classification (NN) [1] and supervised learning by learning vector quantization (LVQ) [2], [3]. The decision surface mapping method, which we call DSM, is a fast supervised learning algorithm, and is a member of the LVQ family of algorithms. A relatively small number of prototypes are selected from a training set of correctly classified samples. The training set is then used to adapt these prototypes to map the decision surface separating the classes. This algorithm is compared with NN pattern classification, learning vector quantization (LVQ1) [2], and a two-layer perceptron trained by error backpropagation [4]. When the class boundaries are sharply defined (i.e., no classification error in the training set) the DSM algorithm outperforms these methods with respect to error rates, learning rates, and the number of prototypes required to describe class boundaries.**

I. Introduction

The nearest neighbor (NN) method assigns an unclassified sample vector to the class of the nearest of a set of correctly classified prototypes, or codebook vectors. Cover and Hart have shown that in a large sample, the error of this rule is bounded above by twice the Bayes probability of error [1].

Learning vector quantization (LVQ1, LVQ2, LVQ2.1, and LVQ3), described by Kohonen [2], [3], is a nearest neighbor classification method in which a fixed number of prototype vectors are progressively modified to cover the input space. The LVQ family of algorithms is concerned with optimal placement of these prototypes, so as to reflect the probability distribution of the training samples. The adaptive decision surface mapping (DSM) algorithm is a variation of the LVQ method, but we have dropped the requirement that the prototypes reflect the probability distribution of the classes. Instead, the algorithm adapts the prototype vectors to closely map the decision surface separating classes. DSM is described in detail in the next section. In Section III we present comparative results for three classification problems, which indicate a drastic performance improvement over the LVQ and backpropagation.

Manuscript received October 5, 1990.

The authors are with the Faculty of Information Technology, Queensland University of Technology, GPO Box 2434, Brisbane, Queensland, 4001 Australia.

IEEE Log Number 9042019.

II. Adaptive Decision Surface Mapping

The DSM algorithm starts by selecting a small subset of prototypes from the training set. The initial prototypes are selected at random. We have found that good results are obtained when the proportion of prototypes from each class in the initial subset matches the *a priori* probabilities of the classes, and it is indeed the procedure commonly followed with LVQ. This information is usually available in the training set, when random sampling is used, or may have to be provided externally if the training set does not reflect the *a priori* probabilities of the classes.

In the learning stage, the training set is used to modify the prototypes in order to gradually adapt the decision surface they define to that defined by the entire training set and reduce the classification error rate. Samples from the training set are cyclically or randomly presented for classification. When a training sample is correctly classified, that is, the training sample is of the same class as the nearest prototype, no modifications are applied. When misclassification occurs, modifications take place to apply both punishment and reward.

The punishment step takes the nearest neighbor prototype, which, in this case, is of the wrong class, and moves it away from the training sample, along the line connecting the two vectors

$$\vec{m}_w(t+1) = \vec{m}_w(t) - \alpha(t)\left[\vec{x}(t) - \vec{m}_w(t)\right]. \quad (1)$$

The reward step searches for the nearest correct prototype and moves it towards the training sample, along the line connecting the two vectors

$$\vec{m}_c(t+1) = \vec{m}_c(t) + \alpha(t)\left[\vec{x}(t) - \vec{m}_c(t)\right]. \quad (2)$$

The term α is a scalar gain factor, monotonically decreasing with time. For the cases discussed below, we have found that very good results are obtained when α starts from a value of 0.3 or less, and linearly decreases to 0, at a rate consistent with the desired training limit (number of presentations). The algorithm is not very sensitive to initial values of α, but if alpha starts too small, training takes longer.

In the earlier stages of the training process α is relatively large; therefore the process is allowed to rapidly modify prototypes to remove large classification errors caused by the initial conditions. In later stages, as α decreases, a more refined adaptation takes place to correct smaller classification errors or to arrive at a compromise configuration where errors are minimized.

The algorithm modifies prototypes only on misclassification, and since errors are more likely to occur with samples near class boundaries, it rearranges prototypes, in pairs, on each side of a class boundary, to correct or at least reduce the magnitude of these errors.

It is possible that a configuration eliminating all classification errors on the training set could be arrived at before α reaches a value of 0. In that instance training is complete.

DSM is different from all the variants of LVQ. In LVQ1 modifications are applied at each presentation, either to punish an incorrect classification or to reward a correct one. LVQ2 modifies the nearest and next-to-nearest neighbors whenever the nearest neighbor is of a different class, and the next nearest neighbor is of the same class, as the training sample. Furthermore, LVQ2 requires the training vector to fall within a window which is determined by the relative distances of the training sample from the pro-

1045-9227/91/0300-0318$01.00 © 1991 IEEE

totypes. LVQ2.1 applies modifications upon incorrect, as well as correct, classifications. Finally, LVQ3 is a combination of the earlier LVQ algorithms.

III. Experimental Results

To demonstrate the performance of the DSM algorithm we have conducted experiments with three classification problems with different degrees of complexity, all with nonlinearly separable classes. The problems involve the classification of two-dimensional real valued vectors, taken from within a rectangle. The classes were defined by partitioning the rectangles into several disjoint regions.

For each of the problems we have used a training set of 6400 samples and a test set of 6400 samples, all taken at random. All the comparisons are related to the same training set, initial conditions, and test set. We compare the performance of DSM, LVQ1, NN, and a two-layer perceptron trained by backpropagation. It turns out that DSM not only out performs the other methods, but also is the only method that produces results that are consistently better than NN and at the same time is the most economical.

Because LVQ1, backpropagation, and DSM are based on random processes, we have repeated each experiment many times and averaged the results.

A. Straight Line Class Boundaries

The classification problem here is adapted from Hart [5], and is depicted in Fig. 1. We have tested DSM, NN, LVQ1, and a two-layer perceptron, trained by backpropagation, with varying numbers of prototypes/nodes. The nearest neighbor classification error produced by the training set, when used to classify the test set, was 1.14%. The error rates for the other methods are summarized in Table I.

Fig. 1 shows a set of 24 random prototypes which were used to initiate both DSM and LVQ1. Figs. 2 and 3 show the final configurations arrived at by LVQ1 and DSM respectively. LVQ1 tends to generate a uniform spread of prototypes within class boundaries. The error rate for this configuration is 3.06%. DSM, on the other hand, tends to locally rearrange prototypes so as to follow the decision surface on either side. It does not modify prototypes well inside a class distribution, only those in the vicinity of the class boundary. The prototypes' spread is therefore nonuniform, and inspection of Figs. 1 and 3 reveals that DSM does not utilize all 24 prototypes; yet it produces significantly better results. The error rate for the DSM configuration is 0.41%, significantly better than the NN error rate of 1.14%.

This classification problem has a family of perfect solutions which require only ten prototypes. One such solution is depicted in Fig. 4. When DSM was initialized with ten randomly selected prototypes, it produced this very configuration consistently, and an average error rate of 0.43%. LVQ1 produced poor results (12.34% error), while backpropagation used with a perceptron having ten nodes in the hidden layer produced reasonable results (1.66% error), but not better than nearest neighbor.

While backpropagation training required 1000 presentations of each training sample, LVQ1 and DSM were trained with only ten presentations. It is possible that slightly better results for the perceptron could be obtained by reducing the learning rate and increasing the number of presentations, but the training times involved were already orders of magnitude longer than those required by DSM and LVQ1.

It is also possible to reduce the number of modifications in backpropagation, by accumulating the changes and applying them after each epoch, but it is still a global modification affecting every weight.

The results clearly show that DSM produces consistently better results than LVQ1 and does not display the same performance degradation as the number of prototypes is reduced. For this classification problem, any configuration having fewer than ten prototypes is bound to produce some nearest neighbor classification error, regardless of the size of the training set. Even with eight prototypes, DSM produced an error rate of only 3.82%, while LVQ1 produced an error rate of 19.55%. A perceptron trained by backpropagation produced somewhat erratic results when having a small number of nodes at the hidden layer. Table I shows that it is sensitive to the number of nodes, producing a good result (1.53% error) when eight nodes are used and a poor result (9.27%) when nine nodes were used. Again, it is possible that a better result is obtainable with the perceptron, but there is no way of telling in what way parameters should be adjusted, which is an old problem associated with backpropagation.

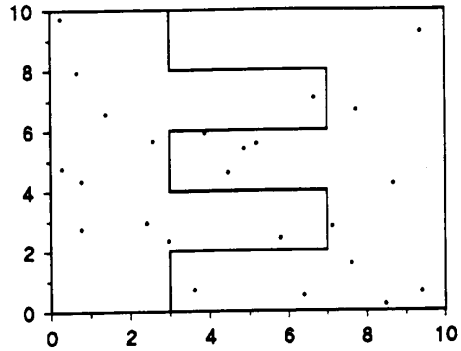

Fig. 1. Straight line class boundaries. Depicted is the initial random set of 24 prototypes, which were subsequently modified by adaptive learning using DSM and LVQ1.

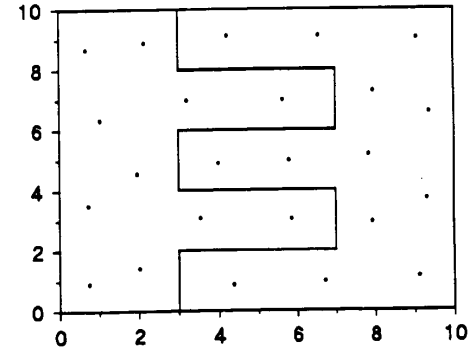

Fig. 2. LVQ1 solution: prototypes are uniformly spread within class boundaries and follow class boundaries on either side.

TABLE I
Error Rates (%) Produced by DSM, LVQ1, and Backpropagation for Classification Problem 1 (Corresponding NN Error Rate Is 1.14%)

Number of Prototypes	DSM	LVQ1	Backpropagation
6	7.14	19.00	9.42
8	3.82	19.55	1.53
9	1.86	14.64	9.27
10	0.43	12.34	1.66
20	0.45	4.44	1.53
24	0.41	3.06	1.38
50	0.49	2.51	1.56
250	0.79	1.84	1.55

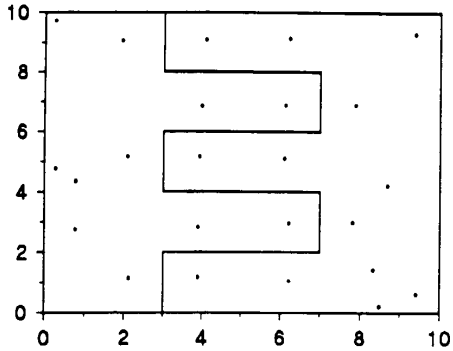

Fig. 3. DSM solution: prototypes are adjusted to follow class boundaries. Not all of the prototypes are actually utilized.

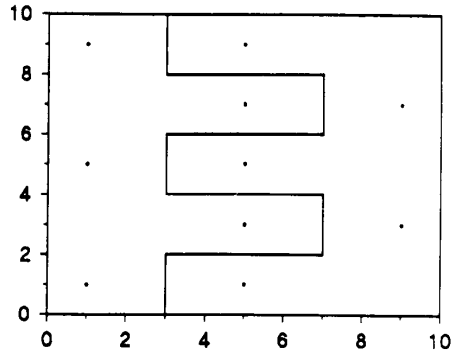

Fig. 4. A perfect solution to the problem, using only ten prototypes.

B. Curved Class Boundaries

The second classification problem we have used is depicted in Fig. 5. It is more complicated, involves four classes, and none of the class boundaries are straight lines. The error rate of nearest neighbor classification by the entire training set was 1.85%. The error rates for the other methods are summarized in Table II.

The error rate DSM produces is significantly below that of NN, while it is considerably more economical than LVQ1 for similar error rates. Experiments with more complicated class structures and with spaces of higher dimensionality support these results. The performance of DSM is consistently better than that of LVQ1, while it proved impractical to simulate backpropagation with complex problems that require large training sets and a large hidden layer, on a sequential computer.

An additional advantage of DSM is that many training sessions can be attempted, in a short time, to find a good solution, simply by increasing the number of prototypes used. Backpropagation is too slow to be repeated many times, and in any case it is not always clear what modifications are necessary to improve the performance. For example, backpropagation produced better results with 40 hidden layer neurons than it did with 50 when trained with the same learning parameters. It is very sensitive to the initial conditions and the shape of the error surface it minimizes, allowing it to be trapped in false minima.

Fig. 5 shows an initial set of 50 random prototypes that were used to initiate LVQ1 and DSM. Figs. 6 and 7 show the final configurations arrived at by LVQ1 and DSM rspectively. In this particular case DSM produced an average error rate of 2.2% while LVQ1 produced an average error rate of 8.8%. Inspection of Figs. 5 and 7 reveals that the DSM solution utilizes only 40 prototypes, not 50, and yet results in considerably lower error rates than LVQ1.

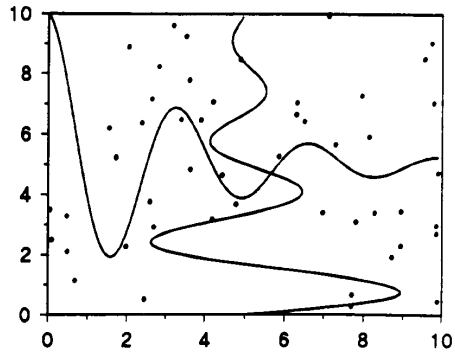

Fig. 5. Curved class boundaries. Depicted is the initial random set of 50 prototypes, which were subsequently modified by adaptive learning using DSM and LVQ1.

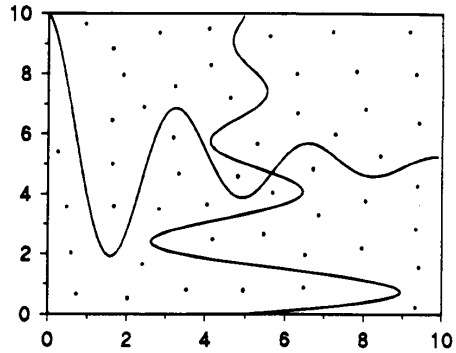

Fig. 6. LVQ1 solution: prototypes are uniformly spread within class boundaries and follow class boundaries on either side.

TABLE II
ERROR RATES (%) PRODUCED BY DSM, LVQ1, AND BACKPROPAGATION FOR CLASSIFICATION PROBLEM 2 (CORRESPONDING NN ERROR RATE IS 1.85%)

Number of Prototypes	DSM	LVQ1	Backpropagation
24	3.17	10.55	3.16
40	2.11	8.77	2.33
50	2.02	8.55	3.83
100	1.53	5.49	3.13
200	1.54	3.64	3.44

C. The Generalized XOR Problem

The third problem we have used is that of the two-dimensional binary XOR problem, generalized to real-valued two-dimensional vectors. The problem is depicted in Fig. 8. The results are given in Figs. 9 and 10.

LVQ1 produces eratic error rates when a very small number of prototypes are used, as Fig. 9 shows. LVQ1 is sensitive to the geometry of the problem, and the classification error rate fluctuates in the range of 2% to 6% as the number of prototypes increases from 5 to 32. As the number of prototypes is further increased, LVQ1 performance improves gradually, with ever decreasing fluctuations. However, even with 640 prototypes the error rate is 0.95%, which is still worse than the 0.84% of NN classification.

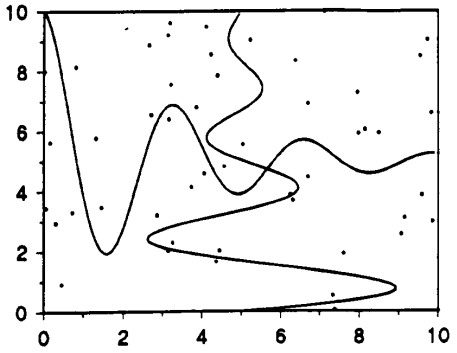

Fig. 7. Prototype selection of DSM. Points are adjusted to follow class boundaries.

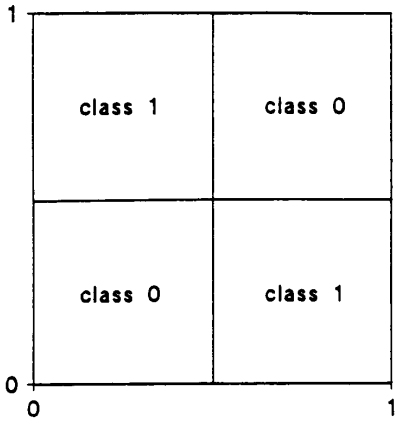

Fig. 8. Classification problem 3. A generalized two-dimensional XOR classification.

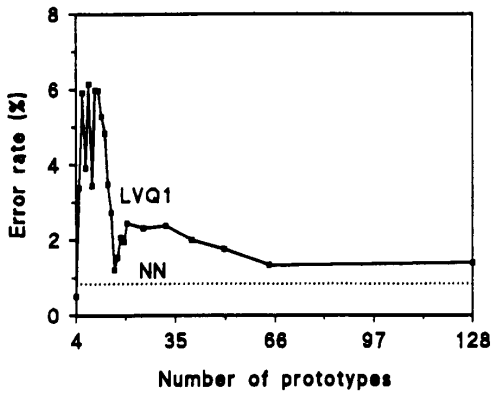

Fig. 9. LVQ1 error rates as a function of the number of prototypes used.

When only four prototypes are used LVQ1 produces an error of 0.50% and outperforms NN classification (0.84% errors). The algorithm places one prototype at the center of each of the disjoint

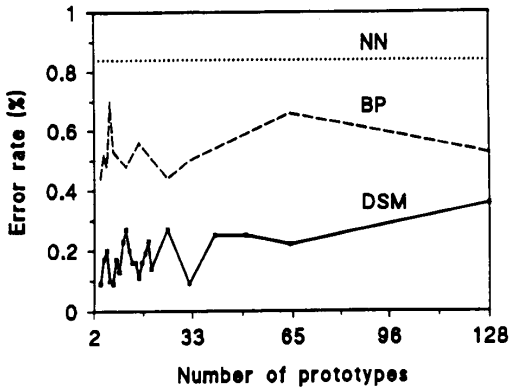

Fig. 10. DSM and backpropagation error rates as a function of the number of prototypes used.

class regions, hence the low error rate. As soon as a fifth prototype is added, the balance is disturbed, and the region which has two prototypes introduces a significant error (3.38%). An additional prototype brings the error rate to 5.91% and it continues to fluctuate up or down, depending on the geometric configurations generated by LVQ1. When 16 prototypes are used, the geometry is again favorable. There are four prototypes in each region, which are symmetrically placed, and the error rate drops to 1.20%. It is useful to have an equal number of prototypes in each disjoint class region in this particular problem, but in general one does not know the class boundaries. LVQ1 produces the same results, consistently, even when different sets of initial prototypes are used. This erratic behavior is less obvious when a large number of prototypes are used.

The behavior of DSM is somewhat similar, but at a much smaller scale, and is due to a different cause. As Fig. 10 shows, the error rate does fluctuate as the number of prototypes is increased from 5 to 32, but the range is now between 0.08% and 0.27%, always smaller than the 0.84% NN classification.

To explain these fluctuations we must recall that it is possible for DSM to use only a subset of the available prototypes to describe the decision surface; therefore different initial configurations may lead to different numbers of prototypes being actually used, and to different results. In general, DSM performs better with fewer prototypes, provided that the number is not too small to effectively describe the decision surface.

As the number of prototypes is increased, the performance of DSM degrades somewhat. With 640 prototypes it reaches 0.67%, still better than NN. This behavior is to be expected: DSM trains the prototypes to reduce the error rate on the training set. As prototypes are added, DSM is able to find a decision surface that is closer to the decision surface defined by the training set, and the error rate moves toward that of NN classification. The conclusion from this is that, provided that DSM is not initialized with too few prototypes, it should find a configuration better than NN. This is indeed observed in all experiments. LVQ1, on the other hand, produces results that are consistently worse than NN, although it is possible that by a careful, or lucky, choice of prototypes it will outperform NN.

With the generalized XOR problem, as one might expect, the perceptron trained with backpropagation produced good results and outperformed NN when the number of nodes in the hidden layer exceeded 4. However, as Fig. 10 shows, it did not outperform DSM.

The reason that DSM is able to produce better results than NN is the averaging out of the random fluctuations of samples in the training set, near class boundaries.

IV. Discussion

Kohonen, in introducing LVQ1 [2], suggests that "with pattern recognition problems it is the decision surface between classes and not the inside of the class distribution which should be described most accurately." Yet LVQ1 generates a configuration that utilizes the prototype vectors to describe not only the decision surface, but also the insides of class distributions. Indeed, if one is to construct a probability function to describe the class distributions, based on the prototype vectors, then LVQ is more appropriate than DSM.

DSM, on the other hand, is a much more effective algorithm for describing the class boundaries. It is worth noting here that although we have not compared the performance of DSM with all the flavors of the LVQ, these do not vary much in performance [3].

DSM is concerned only with an accurate description of the class boundaries; rather than move prototypes away from regions where classification errors occur, it rearranges prototypes, in pairs, on each side of the boundary to reduce the error rates. In fact, it is because DSM is not concerned with the insides of class distributions that it is capable of mapping the decision surface more economically. DSM "draws" prototypes from inside class distributions toward the class boundaries to produce a more accurate description.

The results presented here show that DSM outperforms backpropagation with respect to error rates and training times. In some instances, e.g. the XOR problem, backpropagation does produce competitive error rates, but not in all cases; it is certainly impractical for very complex problems, as it requires the adjustment of many more configuration parameters and requires unrealistic training times. It is hard to predict how backpropagation will perform on a new problem, and it typically requires a considerable effort to fine-tune the learning parameters and perceptron structure. DSM is much more predictable and requires no modifications for different problems, and it is easy to find an economical DSM configuration to outperform NN classification.

DSM shares with the LVQ's a clear advantage over backpropagation, which relates to retraining. If the decision surface changes, DSM requires short retraining, as the performance will degrade only in the areas where change occurred, and the existing configuration is a good starting point. On the other hand, a perceptron trained by backpropagation may require complete retraining, as a solution to one problem is not always a good starting solution to another problem, even if it is similar. Therefore, DSM is more suitable for dynamic, real-time classification problems.

When the decision surface to be described is smooth it requires fewer points to describe adequately than when it is rough. Random selection of prototypes from the training set will work well as long as there is an adequate number of prototypes, so that even an unlucky initial spread of prototypes will leave sufficient numbers in the difficult to map areas, to allow for accurate decision surface mapping. As long as the number of prototypes we use is not so small that it contains severe random fluctuations in the spatial spread of prototype vectors, we can expect good results.

While we have shown that for the problems described DSM consistently outperforms LVQ1, it must be stressed that in cases where the training set contains a probabilistic error, or fuzzy class boundaries, DSM exhibits instabilities. Our preliminary results show that there is no clear advantage to DSM, and LVQ may even be superior owing to its stability.

However, DSM can be used to improve a solution previously obtained with LVQ1. A classifier obtained by LVQ1 is used to generate a new training set, with no probabilistic errors. With this consistent training set DSM is once again stable and may be used. DSM then provides an accurate description of the LVQ1 decision surface and, importantly, requires substantially fewer prototypes.

References

[1] T. M. Cover and P. E. Hart, "Nearest neighbor pattern classification," *IEEE Trans. Inform. Theory*, vol. IT-13, pp. 21-027, Jan. 1967.
[2] T Kohonen, in *Self-Organization and Associative Memory*, 2nd ed. 1988, pp. 199-202.
[3] T. Kohonen, "Statistical pattern recognition revisited," in *Advanced Neural Computers*, R. Eckmiller, Ed. 1990, pp. 137-144.
[4] D. E. Rummelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing: Explorations in the Microstructures of Cognition*, vol. 1, *Foundations*, D. E. Rummelhart and J. L. McClelland, Eds. Cambridge, MA: MIT Press, pp. 318-362.
[5] P. E. Hart, "The condensed nearest neighbor rule," *IEEE Trans. Inform. Theory*, vol. IT-14, pp. 515-516, May 1968.

Performance and Generalization of the Classification Figure of Merit Criterion Function

Etienne Barnard

***Abstract*—A new criterion function for training neural networks, introduced by Hampshire and Waibel [1], is studied. It is shown that this criterion function has some highly desirable properties. However, these properties are not directly related to generalization (as was suggested in [1]); it is shown that systematic improvement of generalization involves a different class of modifications.**

I. Introduction

The training of various neural nets can be viewed as the optimization of criterion functions (also known as objective functions or energy functions). Of these functions, by far the most popular is the mean-square-error (MSE) criterion function, which is commonly used in conjuction with backpropagation [2], but an infinite variety of other possibilties exists [3]. It has long been known that the choice of criterion function can play an important role in the performance of classifiers [4], and this issue has recently been the focus of renewed attention [1], [3].

In assessing the performance of a classifier such as a neural network, a number of related metrics should carefully be distinguished from one another. Conceptually the simplest metric of performance is the numerical value of the criterion function. Since training is accomplished by optimizing this function, it is obvious to think of the criterion function as a measure of the classifier's performance. However, in almost all applications the success of the classifier will be measured not by the value of the criterion function obtained but by the error rate of the classifier. A somewhat more relevant metric of performance is therefore the classifier's error rate on the *training set*. In [3] it is shown that these two metrics of performance are generally not equal and, indeed, that they can differ appreciably for certain criterion functions. Training-set performance, on the other hand, is also not a true measure of the value of a classifier; how well it fares on new, unseen test data is usually the real issue. Thus, the ability to generalize to a disjoint test set is the most important measure of a classifier's performance.

Manuscript received October 29, 1990.
The author is with the Department of Electronics and Computer Engineering, University of Pretoria, Pretoria, 0002 South Africa.
IEEE Log Number 9042021.