Statistical Pattern Recognition with Neural Networks: Benchmarking Studies

Teuvo Kohonen, György Barna, and Ronald Chrisley
Laboratory of Computer and Information Science
Helsinki University of Technology
Rakentajanaukio 2 C
SF-02150 Espoo, Finland

Abstract

Successful recognition of natural signals, e.g., speech recognition, requires substantial statistical pattern recognition capabilities. This is at odds with the fact that the bulk of work on applying neural networks to pattern recognition has concentrated on non-statistical problems. Three basic types of neural-like networks (Backpropagation network, Boltzmann machine, and Learning Vector Quantization), were applied in this work to two representative artificial statistical pattern recognition tasks, each with varying dimensionality. The performance of each network's different approach to solving the tasks was evaluated and compared, both to the performance of the other two networks, and to the theoretical limit. The Learning Vector Quantization was further benchmarked against the parametric Bayes classifier and the k-nearest-neighbor classifier using natural speech data. A novel Learning Vector Quantization classifier (LVQ2) is introduced the first time in this work.

1. Introduction. – The "Bayes Machine"

Some of our colleagues have claimed that the biological neural networks, due to their inherent nonlinearities, mostly implement decision processes (cf., e.g., Hopfield, in press). Whatever the truth about their purpose, if we concentrate on this aspect with artificial neural networks, then evaluation of their decision-making accuracy ought to be based on standard **decision-theoretic analyses**. At least one might speculate that the simple least-square error criteria that are often used would perhaps be unsatisfactory with real statistical problems.

By and large the only generally valid statistical decision theory is based on the average cost or loss in misclassification, formulated in terms of the Bayes expressions for conditional probabilities (briefly called "Bayes classifier") (cf., e.g., Devijver and Kittler 1982, Kohonen 1988a, Patrick 1972). In standard pattern recognition theory, and obviously with most of the problems that occur in neural computing, too, it is reasonably accurate to assume the unit misclassification cost the same for all classes. Assume then that $x \in \mathbf{R}^n$ is the vector of input observables (pattern elements, set of attribute values, etc.), and $\{C_i, i=1,2,...,K\}$ is the set of classes (identities) to which x may belong. Let $p(x|C_i)$ be the probability density function of x in class C_i, and $P(C_i)$ the a priori probability of occurrence of samples from class C_i; in other words, $d_i(x) = p(x|C_i)P(C_i)$ corresponds to the "class distribution" of those samples of x which belong to class C_i. Here the $d_i(x)$, or any monotonically increasing functions of them like the logarithms, are also called **discriminant functions** (cf. section 7.1). The average rate of misclassifications is minimized if x is conclusively classified according to the following rule:

$$x \text{ is assigned to } C_i \text{ iff } d_i(x) > d_j(x) \quad \text{for all } j \neq i.$$

(See the one-dimensional case in Fig. 1.) The main problem, of course, is to obtain analytic expressions for the $d_i(x)$. Notice that even a large number of samples of x, as such, does not define any analytical probability density function. One has to use either *parametric methods*, e.g. by fitting the samples to parametrized Gaussian functions, or *nonparametric methods* whereby some fixed kernel function that is everywhere continuous must be defined around every available sample of x. For general distributions, none of these methods is satisfactory. Either the accuracy may remain low (like in parametric methods), or the computations become heavy (nonparametric methods). The dream about an ideal "Bayes machine" has therefore been around for a while.

The "Boltzmann machine" and "Backpropagation network" are based on a least-square error criterium. If enough training samples and internal parameters are available, it might seem that their input-output transformations can be defined to an arbitrary accuracy. However, since their design is not based on decision-theoretic arguments, it is not obvious how well they perform in **conflicting** cases, i.e., when the "class distributions" intersect. We found this problem of such a fundamental importance in practical applications with stochastic data, that we decided to "benchmark" these models, together with a few others, on certain typical-looking statistical data. In this work we have concentrated on statistical accuracy, although comparison of certain other figures like computing speed would also have been interesting. It seems, however, that it is not so easy to define comparison criteria for the latter, because, for instance, the "sigmoid function" in Boltzmann machines and Backpropagation networks can be computed in many different ways. In the concluding section we are trying to report our general experiences about typical computing times. Roughly speaking, of the methods studied, the LVQ is the fastest, then the Backpropagation network, and finally the Boltzmann machines. The statistically best Boltzmann machine, however, contains circuit elements (plenty of precision level detectors for all input signals) which are very difficult to realize in massive circuits and may thus not be practicable with real problems.

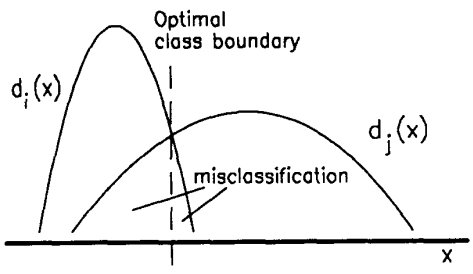

Fig. 1. Minimization of misclassification rate.

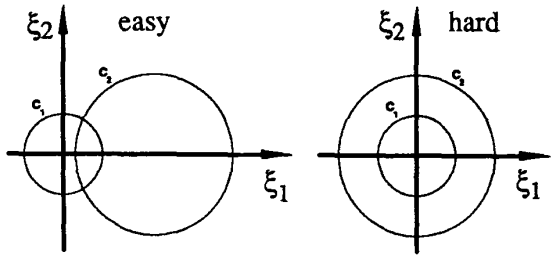

Fig. 2. Illustration of the two tasks. (Square roots of the variances are indicated by circles.)

2. Selection of the tasks

Numerous neural network studies have been performed with discrete, class-separable data. Some benchmarking studies have earlier also been performed on low-dimensional statistical data (Huang and Lippmann, 1987). Based on practical experiences, however, we felt that the nature of the statistical problems might change with higher dimensionality of input vectors. Therefore we introduced a model of artificial data with **dimensionality ranging from 2 to 8.**

Intentionally, we thus defined our benchmarking data to represent statistically difficult cases, where we wanted to have: 1. Heavy intersection of the class distributions. 2. High degree of nonlinearity of the class boundaries. 3. High dimensionality of the input vectors. - To this end we defined two artificial data sets and had two natural data sets available.

As far as the above three properties are taken into account, it is not desirable to load the artificial datum model by any further unnecessary details or complexities. For **benchmarking purposes** we thus decided to use for the artificial data symmetric, heavily overlapping Gaussian distributions (with different variances for the classes) because it is then easy, for reference, to compute the theoretical accuracy limits of the Bayes classifier. With artificial data, we also restricted to a two-class problem, since we believed it yields answers to the most essential questions. Let us denote $x=(\xi_1,\xi_2,...,\xi_n)$. Class C_1 was represented by a multivariate normal distribution with zero mean and square root of variance equal to 1 in all the dimensions, class C_2 by a normal distribution with mean $\bar{x}_1=(\xi_1,0,0,...,0)$ and square root of variance equal to 2 in all dimensions, respectively. In one task the relative offset between classes was $\xi_1=2.32$ ("easy task"), in the other $\xi_1=0$ ("hard task"), respectively (cf. Fig. 2).

We also wanted to make experiments with **natural data**.To that end we had available manually segmented real speech spectra (computed over 30 millisecond intervals) with which it was possible to test the Learning Vector Quantization methods against theoretical limits. These tests were carried out for 18 phonemic classes simultaneously. The different classes had different statistics. We have not yet been able to carry out the corresponding tests for the Backpropagation network and Boltzmann machines, because these experiments are very time-consuming indeed, whereas for LVQ, computing time is no problem.

3. Backpropagation

Backpropagation (BP) is a learning algorithm for multi-layer feedforward networks, as introduced in Werbos (1974) and Parker (1982).

3.1. Particular configuration

A standard, two-layer backpropagation network as described in Rumelhart, Hinton, and Williams (1986) was used. Two, as opposed to three, layers were used since the optimal decision regions for all tasks were known to be convex, and it has been shown in Lippmann (1987) that a two-layer network is sufficient to form convex decision regions. The advantage in convergence time that a three-layer network sometimes has over two-layer networks was irrelevant, since asymptotic error rates were the only basis of comparison. The learning rate was 0.01 and the momentum coefficient was 0.9. There were 8 nodes in the hidden layer, with a number of inputs equal to the dimensionality of the input vectors and the number of output nodes equaling the number of classes, which was 2 in all tasks. This, combined with a bias weight for each node, provided for a total of $8 \cdot d + 26$ weights, where d is the dimensionality of input. Although it is possible that increasing the BP resources per dimension at a faster rate could allow BP to better track the theoretical limit in the higher dimensions, it has also been noted that adding units to a BP network does not always result in improved performance.

During training, it was stipulated that the correct output for a sample of a given class should be 1 for the node corresponding to the sample's class, and 0 for the other node. For recognition and testing purposes, the network was considered to have correctly classified the sample if the output node with the greatest activation from that sample was associated with the sample's class.

3.2. Experimental results

Table 1 gives the assymptotic error rates for BP's performance on both tasks. Notice that for a constant amount of computational resources (weights), and as dimensionality increased, BP's performance improved at a rate significantly less than the rate improvement of the theoretical limit. Lowering the learning parameter frequently caused the network to get stuck in local minima, while raising the rate often prevented convergence.

4. Boltzmann machines

The Boltzmann machine (BM) is a parallel computing network, consisting of simple processing units which are connected by bidirectional links. The links are represented by real numbers while the units can be in the states 'on' or 'off'. For details, further references and applications see Ackley, Hinton and Sejnowski (1985), Prager, Harrison and Fallside (1986).

4.1. Particular configurations

Although the BM was originally designed for binary units, the tasks described in Section 2 require continuous inputs. Two types of input units were therefore used (as in Prager et al., 1986):

a) continuous input units, allowed by the generalization of the learning rule;
b) groups of binary input units, used with a binary coding scheme.

In case a) the input values were shifted to the positive range. In case b) in each dimension the input range was divided into 20 subranges. Thus, 20 input units must be used for each dimension. All of them were set 'off' except the one which is associated with the subrange containing the input value of the dimension, which is set 'on'. We call this scheme "BM2".

In addition, there were always 2 hidden and 2 output units. This corresponds to the numbers of weights equal to $80 \cdot d + 3$ in case a) and $4 \cdot d + 3$ in case b). When the output units are clamped, the clamping patterns are {on,off} and {off,on} for class C_1 and C_2, respectively. The probabilities for two units being simultaneously 'on' in the unclamped and clamped modes, respectively, were estimated over 10,000 samples.

For updating the w_{ij} weights, only the sign of Δw_{ij} was determined by the gradient descent learning algorithm detailed in Ackley et al. (1985). The absolute value of Δw_{ij} was constant (Δw) for each update and slowly decreased over time.

4.2. Experiments

The experimental results for the above configurations are shown in Table 2. Increasing the number of input or hidden units did not decrease the error significantly. The value of Δw was initially about two times the average of the absolute values of w_{ij} and slowly decreased (to 1/5th of the initial value at about the 500,000th sample). Reaching the minimal error level required about 4-500,000 samples.

5. Learning Vector Quantization

Learning Vector Quantization (LVQ) is a nearest-neighbor method operating explicitly in the input domain (Kohonen 1988a, 1988b). It consists of a **predetermined number** of processing units, each unit having a d-element **reference vector**, and each unit being associated with one of the classes of the input samples. Let c be the processing unit that is the closest to x,

in some appropriate metric; this is then also the classification of x . During learning, unit c is updated. Here t is the discrete-time index (integer). The exact form of this change is:

$$m_c(t+1) = m_c(t) + \alpha(t)(x(t)-m_c(t))$$
if x and the closest unit belong to the same class ,

$$m_c(t+1) = m_c(t) - \alpha(t)(x(t)-m_c(t))$$
if x and the closest unit belong to different class ,

$$m_i(t+1) = m_i(t) \quad \text{for } i \neq c \,, \tag{2}$$

where $0 < \alpha(t) < 1$, and α is decreasing monotonically with time.

5.1. Particular configuration

The number of processing units was chosen to be $5 \cdot d$, thus resulting in $5 \cdot d^2$ weights. For calculating the closest unit the Euclidian metric was used.

Since the LVQ learning algorithm assumes a good initial state, the traditional k-means clustering (cf. Makhoul et al., 1985), was used to initialize the processing units. Then each of the processing units was associated with one of the classes, by running the system without learning and collecting statistics on how frequently each of the processing units was the closest to the samples from each class.

During learning, α decreased from 0.01 to 0 over 100,000 samples.

5.2. Experimental results

The experimental results are in Table 1. We have found that if the number of processing units in LVQ can be increased without limits, we might reach the theoretical limit. Here the number of units was kept comparable to that used in the other methods.

Table 1 **Error percentages for artificial data. All figures in the last four columns are given with 0.1% accuracy.**

	dim.	theoretical limit	BP	BM (continuous)	BM2 (binary)	LVQ
$\bar{\xi}_1$=2.32	2	16.4	16.4	29.2	16.5	17.0
	3	13.7	14.0	26.2	14.0	14.6
	4	11.6	12.5	24.5	11.7	13.1
	5	9.8	11.0	23.9	10.2	12.2
	6	8.4	10.8	23.4	8.7	10.7
	7	7.2	9.72	22.9	8.2	10.1
	8	6.2	11.3	22.7	6.7	10.0
$\bar{\xi}_1$=0	2	26.4	26.3	*	26.5	26.5
	3	21.4	21.5	*	21.6	21.8
	4	17.6	19.4	*	18.0	18.8
	5	14.8	19.5	*	15.2	16.9
	6	12.4	20.7	*	12.7	15.3
	7	10.6	16.7	*	11.0	14.5
	8	9.0	18.9	*	9.4	13.4

* no convergence was observed.

6. Another Learning Vector Quantization (LVQ2)

The method discussed in this section is being introduced the first time here.

The basic LVQ can easily be modified to better comply with Bayes' philosophy. Consider Fig. 3 which represents a one-dimensional two-class case. Assume that the neighboring reference values m_i and m_j are initially in a wrong position. The (incorrect) discrimination surface, however, is always defined as the midpoint ("midplane") of m_i and m_j. Let us define a symmetric *window* with nonzero width around the midpoint and stipulate that *corrections to m_i and m_j shall only be made if x falls into the window, on the wrong side of the midpoint.* If the corrections are made according to Eq. (3), it will be easy to see that for vectors falling into the window, the corrections of both m_i and m_j, on the average, have such a direction that the midplane moves towards the crossing point of the class distributions (cf. Fig. 1), and thus asymptotically coincides with the Bayes decision border.

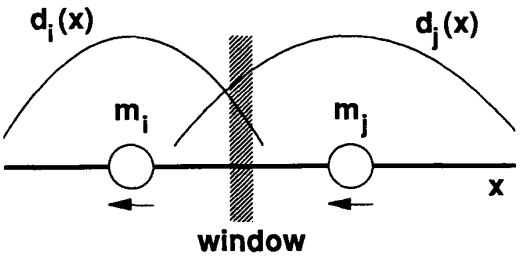

Fig. 3.

$$m_i(t+1) = m_i(t) - \alpha(t)(x(t)-m_i(t)) \quad ,$$

$$m_j(t+1) = m_j(t) + \alpha(t)(x(t)-m_j(t)) \quad ,$$

if C_i is the nearest class, but x belongs to $C_j \neq C_i$ where C_j is the next-to-nearest class; furthermore x must fall into the "window". In all the other cases,

$$m_k(t+1) = m_k(t) \; . \qquad (3)$$

7. Experiments with speech data

Benchmarking studies for LVQ and LVQ2 were also made on real 15-channel speech spectra which were manually picked up from stationary regions of (Finnish) speech waveforms. We applied two independent data sets, each consisting of 1550 phonemic samples, using one data set for training, and the other for testing, respectively. All results of this section have been collected to Table 2.

7.1. Parametric Bayes classifier

For a reference we used the familiar parametric Bayes classifier, in which each class is described by a multivariate normal distribution (in reality, two times the logarithm of the class distribution, from which a constant term has been dropped)

$$d_i(x) = 2 \log P(C_i) - \log |\psi_i| - (x-\bar{x}_i)^T \psi_i^{-1} (x-\bar{x}_i) \qquad (4)$$

where ψ_i is the covariance matrix of those x values that belong to class C_i, and $|\psi_i|$ its determinant, respectively; $\bar{x}_i$ the class mean of $x \in C_i$. In the case that ψ_i^{-1} does not exist for the set of samples from which ψ_i is computed, it may be replaced by the **pseudoinverse** ψ_i^+ (cf. Albert 1972, Kohonen 1988a).

After the ψ_i and $\bar{x}_i$ were computed from the training data set, classification was performed on the test data set like in Eq. (1).

7.2. kNN (k-nearest-neighbor) classifier

In this classification algorithm, a large number of training or reference data is collected for each class, and each of the test vectors x is compared against all of them. A majority voting over k nearest reference vectors is performed in order to decide to which class x belongs. This method is computationally heavy, and it approximates the Bayes classifier only if the number of reference vectors is large.

In our tests we compared all the 1550 test vectors against all the 1550 training vectors, using k=5 or k=6 in voting (this seemed to be the optimum in the present case).

7.3. LVQ and LVQ2

The same training and test data sets as above were used in this method. Because LVQ and LVQ2 need an appreciable number of training cycles for asymptotic convergence, the training data were applied reiteratively. It turned out that half a dozen iterations over the training data were sufficient for final accuracy. There were 117 nodes in the network.

7.4. The results

Table 2 represents the results of Secs. 7.1 through 7.3. On the first row, data set 1 was used for training and data set 2 for testing; on the second row, the roles of the data sets were switched.

Table 2 **Speech recognition experiments; error percentages for independent test data**

	Parametric Bayes	kNN	LVQ	LVQ2
Test 1	12.1	12.0	10.2	9.8
Test 2	13.8	12.1	13.2	12.0

8. Discussion

Notice that for both tasks with artificial data, the relative offset of the two class distributions was constant, whereby it is a natural phenomenon that the theoretically minimal error rate decreases with increasing dimensionality, since the relative value of the "probability mass" of the intersection is then a decreasing function of dimensionality.

It is interesting to note that one of the methods, the BM2 with the dynamic range of each of its inputs divided into 20 subranges, almost followed the theoretical accuracy limit. This method, however, is a brute-force approach, and it is questionable how the precision input discrimination levels can be implemented in massive networks. It also needs a much higher number of units and weights than the other methods. A typical learning time for the BM models, even in this simple two-class problem, was five hours of CPU time on our Masscomp MC 5600 computer, although it had the fast (3000-Whetstone) "Lightning" floating point processor!

The BP network, while being reasonably accurate at low dimensionality (e.g., 2) was significantly inferior with higher dimensionalities, especially in the "hard task" (ξ_1=0). Typical learning time for this particular BP network would be somewhat less than an hour on Masscomp. The BP results seem to be much more unstable than those obtained by the other methods.

With regard to accuracy, the Learning Vector Quantization models fell between these extremes. Typical learning time was a couple of tens of minutes on the Masscomp. On the other hand, LVQ has previously been programmed for microprocessors, and in speech recognition experiments, re-learning all the phonemic classes then only required about 10 minutes on an IBM PC/AT! Notice that the "sigmoid function" in BMs and BP networks is computationally much heavier than the distance computations in Learning Vector Quantization methods; the latter can even be based on integer arithmetic. On the other hand, it seems that comparable accuracy needs roughly similar amounts of nodes and interconnects in any of the models studied.

As we were not yet able to make comparative studies on speech data using Boltzmann machines or BP networks, we do not dare to speculate anything about their relative merits in this task. The very good accuracies yielded by the LVQ methods for speech may be explained by the fact that vector quantization of the pattern space is very advantageous, if the class regions are "piled up" such that there are neighboring classes on all sides of most classes. This was not the case for the artificial two-class problem, where the reference vectors of class C_2 in particular had to surround those of class C_1 on all sides, which is difficult if the dimensionality is high.

In any event, these results show that in statistical pattern recognition, the Learning Vector Quantization is a very viable alternative to the comparatively older BM and BP, and is statistically superior to those BM and BP approaches which are realistic in practice.

Acknowledgement. The work of one of the authors, Ronald Chrisley, was made possible via a Fulbright Graduate Study Grant under the Fulbright-Hayes Exchange Program.

References

Ackley, D.H., Hinton, G.E., and Sejnowski, T.J. (1985): A learning algorithm for Boltzmann machines. Cognitive Science **9**, 147-69.

Albert, A. (1972): *Regression and the Moore-Penrose Pseudoinverse*. Academic, New York.

Devijver, P.A. and Kittler, J. (1982): *Pattern recognition: A statistical approach*. Prentice Hall, London.

Hopfield, J.J. (in press): Neural computations and neural systems. *Proc. of Conference on Computer Simulation in Brain Science, Copenhagen, Denmark, Aug. 20-22, 1986*. Cambridge University Press.

Huang, W.Y. and Lippmann, R.P. (1987): Comparisons between neural net and conventional classifiers. *Proceedings of the 1st IEEE International Conference on Neural Networks*, Vol. IV, 485-94.

Kohonen, T. (1988a): *Self-organization and associative memory*. (2nd ed.) Springer, Berlin-Heidelberg-New York-Tokyo.

Kohonen, T. (1988b): An introduction to neural computing. Neural Networks **1**, 3-16.

Lippmann, R. (1987): An introduction to computing with neural nets. IEEE ASSP Mag. **4**, 4-22.

Makhoul, J., Roucos, S., and Gish, H. (1985): Vector quantization in speech coding. Proc. IEEE **73**, No. 11, 1551-88.

Parker D.P. (1982): Learning logic. Invention Report, S81-64, File 1, Office of Technology Licensing, Stanford University.

Patrick, E.A. (1972): *Fundamentals of Pattern Recognition*. Prentice-Hall, Englewood Cliffs, NJ.

Prager, R.W., Harrison, T.D., and Fallside, F. (1986): Boltzmann machines for speech recognition. Computer Speech and Language **1**, 3-27.

Rumelhart, D.E., Hinton, G.E., and Williams, R.J. (1986): Learning internal representations by error propagation. in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations*. MIT Press, Cambridge. pp. 318-62.

Werbos, P.J. (1974): Beyond regression: New tools for prediction and analysis in the behavioral sciences. Thesis in applied mathematics, Harvard University, Aug. 1974.